EFFORT: Efficient and Effective Bot Malware Detection

Seungwon Shin, Zhaoyan Xu, and Guofei Gu
SUCCESS Lab, Texas A&M University
Email: {swshin, z0x0427, guofei}@cse.tamu.edu

Abstract—**To detect bots, a lot of detection approaches have been proposed at host or network level so far and both approaches have clear advantages and disadvantages. In this paper, we propose *EFFORT*, a new host-network cooperated detection framework attempting to overcome shortcomings of both approaches while still keeping both advantages, i.e., effectiveness and efficiency. Based on intrinsic characteristics of bots, we propose a multi-module approach to correlate information from different host- and network-level aspects and design a multi-layered architecture to efficiently coordinate modules to perform heavy monitoring only when necessary. We have implemented our proposed system and evaluated on real-world benign and malicious programs running on several diverse real-life office and home machines for several days. The final results show that our system can detect all 15 real-world bots (e.g., Waledac, Storm) with low false positives (0.68%) and with minimal overhead. We believe EFFORT raises a higher bar and this host-network cooperated design represents a *timely effort* and a *right direction* in the malware battle.**

I. Introduction

Botnets (networks of bot malware controlled machines) are considered as one of the most serious threats to current Internet security[6], [22]. To eradicate threats posed by bots/botnets, a lot of research has been proposed so far, and they fall into two main categories: (i) network-level detection systems which focus on network behaviors of bots/botnets [7], [6], [8], [22] and (ii) host-level detection systems which investigate bot runtime behaviors in the host [13], [14], [19].

Both detection approaches have their own advantages and disadvantages. Network-level approaches can detect different types of bots without imposing overhead on the hosts. However, their limitations appear when they need to detect a bot communicating through encrypted messages or randomized traffic [20]. Host-level approaches, on the contrary, analyze suspicious runtime program behaviors, so that they can detect a bot even if it uses an encrypted or evasive communication channel. However, they typically suffer from performance overhead because they need to monitor all invoked system calls [13] and/or taint memory touched by the program [19].

Observing their clear advantages and disadvantages motivates us to consider a new system with merits from both approaches: (i) effectiveness and (ii) efficiency. As a promising step toward such a system, we propose *EFFORT*, a new detection framework with high accuracy and low overhead. *EFFORT* considers both host- and network-level features that are helpful to enhance strong points of each other and complement weak points of each other, and it coordinates these features to achieve the main goal (i.e., detecting bots *effectively and efficiently*).

To build EFFORT, We start with investigating several notable intrinsic characteristics of recent popular botnets. First, bots are usually automatic programs without requiring human-driven activities. Second, bots highly rely on DNS for flexible and agile C&C (command and control). Third, bots access system resources anomalously to steal system information or to launch themselves automatically. Finally, unlike normal networked client applications mainly designed to gain information through communicating with external sites, bots are more likely to distribute/leak local information outside and they tend to minimize incoming C&C command communication to reduce the exposure/detection probabilities.

Based on their characteristics, we find several useful features at host and network level. For efficiency, we perform lightweight human-process-network correlation analysis. We correlate interactions between human and process, and record correlated clues between processes and outgoing DNS connections. Thus, we can filter majority of benign programs, and focus on very few suspicious automatic programs contacting DNS servers and start further investigation.

To detect bots effectively, we employ three interesting new modules. First, we monitor *system resource exposure patterns* of the suspicious process. Second, we build a *reputation engine* to characterize the reputation of a process through examining a process and its contacting surfaces. Our intuition is that the reputation of a process could be approximately inferred by the reputation of its social contact surfaces. Third, we analyze network information trading rate for any network process to infer how likely the program is information gaining oriented or information leaking/outgoing oriented.

II. System Design

The overall architecture of EFFORT, which consists of five modules, is shown in Figure 1. The overall operation of EFFORT is summarized as follows. First, *Human-process-network correlation analysis module* finds a process producing bot-driven network connections and notifies other three modules - *Process reputation, System resource exposure, and Network information trading analysis modules* - about the found process. Second, these three modules investigate the found process in detail and each module issues a detection result (but not a final decision) by its own analysis engine. Finally, *Correlation engine* collects the detection results from

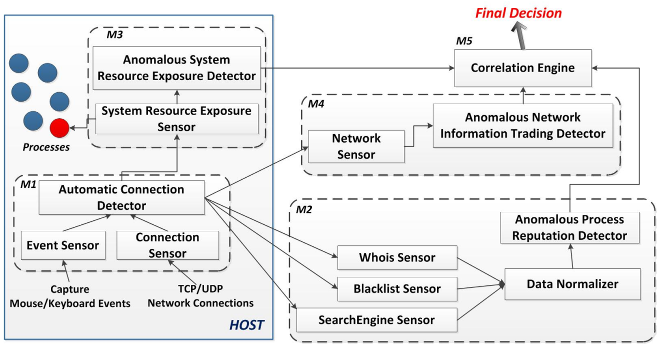
Final Decision
M5
Correlation Engine
M3
Anomalous System Resource Exposure Detector
System Resource Exposure Sensor
Processes
M4
Network Sensor
Anomalous Network Information Trading Detector
M1
Automatic Connection Detector
Event Sensor
Connection Sensor
Capture Mouse/Keyboard Events
TCP/UDP Network Connections
M2
Anomalous Process Reputation Detector
Whois Sensor
Blacklist Sensor
Data Normalizer
SearchEngine Sensor
HOST

Fig. 1: EFFORT Design Architecture, (M1 for *human-process-network correlation analysis module*, M2 for *process reputation analysis module*, M3 for *system resource exposure analysis module*, M4 for *network information trading analysis module*, and M5 for *correlation engine*

the three modules and finally decides whether the process is malicious or not (final decision).

A. Human-Process-Network Correlation Analysis

Since most running processes are benign, it is relatively inefficient to monitor **all** of them in *fine-grained* detail (e.g., monitor system call level activities) **all** the time. Thus, our *human-process-network correlation analysis* module is designed to sift benign programs out.

Human-Process Interactions Monitoring: We monitor keyboard/mouse events of the host to understand which program has human activity/interaction. To do this, our *event sensor* hooks Windows system calls related to keyboard/mouse events, and also determines the program generating them. At this time, we also consider that whether the events are produced by real physical devices or not. Our sensor only trusts the events from physical devices.

Note that in current implementation, we trust operating system and we believe it provides true information, a **common** assumption widely used in this line of research [13], [4]. Of course, some malware (e.g., rootkit) may infect operating system to deliver fake information or even kill our system. In the future, this issue could be solved by employing hardware/TPM [9] or Hypervisor-based introspection and protection [5], [11] (thus out of the scope of this paper).

Process-Network Interactions Monitoring: A *connection sensor* records outgoing network connections from processes in a host. In particular, it cares about one special network connection - DNS query, because botnets heavily rely on using DNS for flexible, efficient, and evasive C&C rallying [1].

Interaction Model Generation and Automatic Connection Detection: Combining information from the *event sensor* and the *connection sensor*, we create a model to describe which process has which network correlations. The model employs three metrics: (i) time difference between the time when a process issues a DNS query and the prior time when a process produces a mouse/keyboard event, (ii) the source of the events, and (iii) whether a process is running foreground or not at the time. We consider that a process event is generated from human, if (i) the time difference is very small, (ii) the event is from actual physical devices, and (iii) the process is running foreground.

B. Detecting Malicious Processes

With the help of the previous module for filtering, we can focus on a few suspicious processes. To further investigate whether they are malicious or not, we perform a set of independent and parallel checks using the following three modules.

1) Process Reputation Analysis Module:

A quick intuitive observation is that we could determine the process reputation by looking at not just "who you are", but also "whom you have talked to". Bots are likely to contact some "bad" servers/peers *automatically* in order to be controlled. On the contrary, benign programs are relatively unlikely to connect to "bad" targets *automatically*.

Domain Information Collection: We collect reputation information of the domain by employing three types of sensors. First, we employ a *whois sensor* to detect some anomaly features in its *registration information*. Second, we use a *blacklist sensor* to investigate its *previous records* in well-known blacklists, which give us relatively clear clues whether the domain has a history of malicious activity or not. Finally, since blacklists might not be complete, we apply a *search engine sensor* to get another heuristic to leverage *community-based* knowledge and intelligence, i.e., asking a search engine to infer the reputation of given domain names, which is motivated by the googling idea in [21].

Feature Extraction: We extract several features from data collected in each sensor. From the *whois and blacklist sensor*, we consider five features: (i) time difference between current date and domain expiration date (most malicious domains are registered very recently), (ii) time difference between domain expiration date and creation date (most malicious domains have short life time), (iv) number of domain registration (malicious domains are typically registered to few name servers), and (v) whether a target domain can be found on blacklist or not (malicious domains are likely on blacklist).

In the case of the *search engine sensor*, we consider the following two features. First, we check whether the domain and the process (contacting the domain) names are well-indexed (thus returning many results). Our intuition on this is that if a domain and a process name can be clearly found on search results, they might be benign. Second, we investigate whether the domain name and the process name are frequently used in a malicious context in the top returned web pages, e.g., they are surrounded by malicious keywords such as bot, botnet, DDoS, spam, and identity theft. Our intuition is that bad domain names are likely associated with some malicious keywords (e.g., as already reported at other places).

Process Reputation Model Creation: We employ a Support Vector Machine (SVM) classifier [3] for the *process reputation model*. In this model, we consider the extracted features, which are mentioned above, as training examples. In addition, we define that there are two classes - benign and malicious - in the model, thus the extracted features will represent one of the two classes.

Anomalous Process Reputation Detection: It is very frequent that a process contacts several different domains during a certain period. Thus, we examine all contacted domains using our trained SVM model, and determine whether "bad" domains (i.e. classified as malicious domains) exist or not. If there exists at least one, we consider the process reputation as bad (malicious), otherwise it is good (benign).

2) System Resource Exposure Analysis: If a bot infects a host, it usually tries to do something useful for its master (to make profit), e.g., stealing information, sending spam, and launching DDoS attacks [10]. These operations consume system resources - memory, cpu, and network - of the host, read/modify files or registries [14]. If we monitor how system resources are exposed to a process (and to what degree), we could infer its anomalous access patterns.

System Resource Exposure Patterns Monitoring: A *system resource exposure sensor* monitors resource access activities of a suspicious process. It monitors how critical resources such as files, registries, and network sockets are exposed to the target process.

System Resource Exposure Model Creation: To build this model, we use the following heuristics: (i) typically normal processes rarely access files in other user's folders and system directories, (ii) typically normal processes do not modify critical registries (with a few exceptions), and (iii) typically normal processes do not create a large number of sockets in a short time period. These heuristics are not perfect, i.e., some normal processes might have some of these patterns. Our goal of this module is not to have zero false positive, instead, we want to detect most of these system-resource-consuming malware. Thus, we believe these heuristics are reasonable.

To build a system resource exposure model, we employ a one-class SVM (OCSVM) classifier [17]. We use one-class instead of two-class SVM because we will only use benign programs in training. To get the ground truth information of the system resource usages of malware is tricky, e.g., some malware may refuse running or behave normally. Thus, even if we obtain the information of malware, it may not represent its behavior clearly. To address this issue, we only use the system resource access patterns of known **benign** processes (i.e., one side of data).

3) Network Information Trading Analysis: Typically, most user programs will act as clients rather than servers, and clients will try to gather information rather than distributing information. However, a bot will behave differently. Usually, the data that a bot receives is a command from a botmaster, therefore the amount of the data may be small (to minimize the chance of being detected). However the data sent out by the bot could be relatively large as it performs malicious operations in the network. Information theft, DDoS attack, and massive spam sending are good examples.

Lightweight Network Traffic Monitoring: To observe network information trades, a *network sensor* captures network flows between a process and a target address and stores them. An important thing here is that this sensor monitors network traffic generated by the specific process not by the host. It could give us more fine-grained observations of network information trading. Our sensor is very simple and lightweight, because it does not need to analyze payload contents and it is robust against encryption used by bots. In addition, we monitor the host level network connections to obtain an aggregated view of network information trading. At this time, the sensor only measures the number of outgoing connection trials (i.e. TCP SYN packets and first UDP packets).

Network Information Model Creation: We use a simple method to model the network information trade rate, i.e., the ratio of incoming and outgoing packets/bytes exchanged between a process and a remote site in a certain period. We define the number of incoming and outgoing packets as θ_1 and θ_2, and the number of incoming and outgoing bytes as δ_1 and δ_2. Thus, each ratio can be represented as $\frac{\theta_1}{\theta_2}$ and $\frac{\delta_2}{\delta_2}$.

To observe an aggregated view, we employ a time window w_i for each host i. We measure how many network connection trials happen in the time window.

Anomalous Information Trading Detection: In the case of the fine-grained view, if one or both of the predefined ratio values of a process is (are) smaller than some threshold γ_1 (for packet) and γ_2 (for bytes), we consider the process anomalous, otherwise normal. Also, we consider the network behavior of the host is anomalous, if a host creates network connection trials larger than a threshold τ in w_i.

4) Correlation Engine: After each module makes its own decision, the *correlation engine* will combine these results and make a final decision using a weighted voting system. It is worth noting that as any intrusion detection system, most of our individual modules might be evaded by very carefully designed bots. However, when combining all modules together, we can achieve much better results, as demonstrated in Section III. We also note that even individual evasion is possible, it will compromise the utility and efficiency of bots.

At the correlation stage, we should determine the weights of the decision of each module. We also employ the SVM technique to determine which element (i.e. decision result of the module) is more important (i.e. should have more weight).

III. Evaluation

In this section, we explain how we collect real-world data of both benign and malicious programs and we show the detection results, false positive analysis, and performance overhead.

A. Data Collection and Usage

1) Benign Data Collection: We have installed our modules into 11 different real-life hosts to collect the information of both process activities and network behaviors for several days. These 11 machines are used in real-life operations by diverse users (including some lab members, friends in different majors, housewives). The collection has been done in working hours on business days. We carefully examine to make sure that there are no malicious programs (especially bots) in the hosts, thus we consider that the collected data can be used as benign examples.

We explicitly collect data in two periods for different purposes: training and testing. We have collected training data from 6 machines and the collected data is denoted as **SET-1**. Later we have collected testing data on 8 machines (among them, 3 machines are also used for training data collection, but 5 machines are newly added). The data for testing is denoted as **SET-2**. Each module have collected process and network information for around 5 days.

There are 7,202 different domains and 77 different processes in **SET-1**. They are used for training. At this time, since we only have benign domain information, we have also collected malicious domain information from [16] to train our module. In **SET-2**, we have found 1,165 processes and we use their information to test false positive rates of our system.

2) Bot Malware Data Collection: To test the detection rate on real-world bots, we build a virtual environment to run several real-world bots. The environment consists of three virtual machines individually serving as an infected host, a controller and a monitor machine. All of them install `Windows XP SP3` operating system with basic software installed, such as IE browser and Microsoft Messenger.

ID	*Name*	*Protocol*	*Sample Functionalities*
B1	PhatBot	IRC	Steal Key, Spam Mail Send, Network Scan
B2	JarBot	IRC	Kill Process, Steal Key
B3	Storm/Peacomm	P2P $*$	Other
B4	Waledac	HTTP, P2P $*$	Other
B5	PhaBot.α5	IRC	Other
B6	Flux	Custom	Operate/Modify File, Kill Process, Capture Desktop/Screen,
B7	nuclearRat	Custom $*$	Download Update
B8	BiFrost	Custom	Operate File, Kill Process, Capture Screen, Steal Key
B9	Cone	Custom	Operate file
B10	Http-Pentest	HTTP	Operate File, Kill Process, Capture Screen
B11	Lizard	Custom	Capture Screen, DDoS
B12	PanBot	Custom	Flooding
B13	Penumbra	Custom	Operate File, Create Shell
B14	SeedTrojan	Custom	Download Update
B15	TBBot	Custom	Capture Screen, Create Shell

TABLE I: Evaluated Bots (*Custom* denotes a botnet using its own protocol. $*$ represents the protocol is encrypted. *Other* denotes other network/system malicious operations not categorized in the table.)

We have used a total of 15 different bots (including Peacomm/Storm, Waledac, PhatBot). Their names, C&C protocols, and sample functionalities are summarized in Table I.

B. Detection Results of Real-world Bots

1) Detection Results of Automatic Connections: To test the *human-process-network correlation analysis module*, we installed each bot in a host and leave it without any intervention. After a while, we find that all installed bots issue automatic connections to some remote servers (to be controlled). All of the automatic connections are captured by our module and the detected information is delivered to other upper-layer modules.

2) Detection Results of the Process Reputation Model: The *process reputation analysis module* analyzes the reputation of contacted domains of the process detected above. Since a bot contacts multiple domains, we analyze all contacted domains. If the module finds any malicious domain from the contacted domains, it considers the process malicious. The *process reputation analysis module* detects 12 bots but misses 3 bots (B2, B3, and B4).

We investigate why our module missed these three. In the case of B2 (Peacomm) and B3 (Waledac), both bots only contacted the remote servers which are either located in private IP addresses (192.168.X.X) or some inactive hosts using direct IP addresses. We could not get any useful information from the third parties for them. B4 (JarBot) contacts a regular IRC server and the server has been operated for several years, thus our module considers it as normal.

3) Detection Results of the System Resource Exposure Model: When we test the functionality of each malware in Table I, we find that the *system exposure analysis module* detects most of the malicious operations. The detection results are summarized in Table II (marked with “S”).

It only misses 2 malicious operations of “B6 (Flux)”; *operate file* and *capture screen*. When we analyze their resource exposure patterns, we find that their operations are very similar to normal programs (e.g., save files in local folder).

4) Detection Results of the Network Information Model: As listed in Table II (marked with “N”), the *network trading information analysis module* detects most malicious operations. It misses 7 malicious operations related to *download updates* and *file modification or operation*. In the case of the *download updates*, the process gains more data, thus our module does not issue an anomaly. Sometimes a botmaster sends commands frequently to an infected host, but does not require an answer. In this case, a bot also obtains more data. In terms of the aggregated view, our module detects all massive outgoing connection trials, such as DDoS and flooding.

5) Correlated Detection Results: If any of the above modules determines its decision, the decision result is delivered to the *correlation engine*. Based on all delivered results, the *correlation engine* makes a final decision for a process.

When we test malicious operations, the *correlation engine* can detect all malicious operations by bots. As we discussed before, even though some module misses an anomaly of an attack, other modules will complement it, thus our combined results can still detect all attacks and the combined results are shown in Table II.

C. False Positive Test

To test false positive cases, we have tested 1,165 benign processes in **SET-2** and we find that only 8 processes are detected as malicious (false positive) by EFFORT. We believe that only 8 false positives found in about a week of testing time on all 8 real-world heavily-used machines are quite low.

More details on the design and evaluation of EFFORT are available in an extended version of this paper [18].

Functionality	*B1*	*B2*	*B3*	*B4*	*B5*	*B6*	*B7*	*B8*	*B9*	*B10*	*B11*	*B12*	*B13*	*B14*	*B15*
Operate file						P,N		P,S	P,S,N	P,S,N			P,S,N		
Modify file						P,S									
Kill process		S,N				P,S		P,S		P,S,N					
Capture Desktop						P,S,N									
Capture screen						P,N		P,S,N		P,S,N	P,S,N				P,S
DDoS											P,S,N				
Flooding												P,S,N			
Create Shell													P,S,N		P,S,N
Download update							P,S							P,S	
Steal key	P,S,N	S,N						P,S,N							
Spam Mail Send	P,S,N														
Network Scan	P,S,N														
Other Operation			S,N	S,N	P,S										

TABLE II: Detection Results of All Modules (shaded cells represent functionalities provided by malware. Each "P", "S", and "N" denotes each *process reputation analysis*, *system resource exposure analysis*, and *network information trading analysis module* detect the functionalities, respectively.

IV. Related Work

There have been several approaches to detect bots at the network level [12] [22] [7], [6], [8]. Our work is different from the above approaches, because we design both of new network level sensors and host level sensors.

Detecting bots at the host level is also popular due to its effectiveness. They employ several interesting techniques to detect bots such as tainting memory [2], [19] and examining the system call sequences/graphs [13]. Our work designs several new host level sensors without analyzing *all* running processes *all* the time, but only investigating the process when necessary.

There are also several interesting studies related to detect bot malware [15] [14] [4]. Our work differs from them because we use different features at host level (e.g., socket creation) and detection models. Moreover, our work analyzes the process only when necessary. Zeng et al. [23] also proposed to detect bots combining information from both host and network levels. This work uses network sensors to trigger host analysis, thus it suffers from the same limitations of previous network-based detection approaches. EXPOSURE [1] is a system to detect malicious domains. Although our *Process reputation analysis module* shares some similar domain registration features with it, these are only small parts in our engine.

V. Conclusion

In this paper, we show a new, concrete host-network cooperated design to achieve both efficient and effective bot malware detection based on correlative and coordinated analysis. EFFORT is a first-of-its-kind real-world prototype system of such design and demonstrates great potential to defeat bots in this important malware battle.

References

[1] Leyla Bilge, Engin Kirda, Christopher Kruegel, and Marco Balduzzi. Exposure: Finding malicious domains using passive dns analysis. In *Proc of NDSS*, 2011.

[2] Juan Caballero, Pongsin Poosankam, Christian Kreibich, and Dawn Song. Dispatcher: Enabling Active Botnet Infiltration using Automatic Protocol Reverse-Engineering. In *Proc. of ACM CCS*, 2009.

[3] Corinna Cortes and V. Vapnik. Support-Vector Networks. In *Journals of the Machine Learning*, 1995.

[4] Weidong Cui, Randy H. Katz, and Wai tian Tan. BINDER: An Extrusion-based Break-In Detector for Personal Computers. In *Proc. of USENIX ATC*, 2005.

[5] T. Garfinkel and M. Rosenblum. A virtual machine introspection based architecture for intrusion detection. In *Proc. of USENIX Security*, 2003.

[6] Guofei Gu, Roberto Perdisci, Junjie Zhang, and Wenke Lee. BotMiner: Clustering Analysis of Network Traffic for Protocol- and Structure-Independent Botnet Detection. In *Proc. of USENIX Security*, 2008.

[7] Guofei Gu, Phillip Porras, Vinod Yegneswaran, Martin Fong, and Wenke Lee. BotHunter: Detecting Malware Infection Through IDS-Driven Dialog Correlation. In *Proc of USENIX Security*, 2007.

[8] Guofei Gu, Junjie Zhang, and Wenke Lee. BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic. In *Proc. of NDSS*, 2008.

[9] Ramakrishna Gummadi, Hari Balakrishnan, Petros Maniatis, and Sylvia Ratnasamy. Not-a-Bot (NAB): Improving Service Availability in the Face of Botnet Attacks. In *Proc. of NSDI*, 2009.

[10] Nicholas Ianelli and Aaron Hackworth. Botnets as a Vehicle for Online Crime. In *Proceedings of 18th Annual FIRST Conference*, June 2006.

[11] Xuxian Jiang, Xinyuan Wang, and Dongyan Xu. Stealthy malware detection through vmm-based out-of-the-box semantic view reconstruction. In *Proc. of ACM CCS*, 2007.

[12] A. Karasaridis, B. Rexroad, and D. Hoeflin. Wide-scale botnet detection and characterization. In *Proc. of USENIX HOTBOTS*, 2007.

[13] Clemens Kolbitsch, Paolo Milani Comparetti, Christopher Kruegel, Engin Kirda, Xiaoyong Zhou, and Xiaofeng Wang. Effective and efficient malware detection at the end host. In *Proceedings of 18th USENIX Security Symposium*, August 2009.

[14] Andrea Lanzi, Davide Balzarotti, Christopher Kruegel, Mihai Christodorescu, and Engin Kirda. AccessMiner: using system-centric models for malware protection. In *Proc. of ACM CCS*, 2010.

[15] Lei Liu, Songqing Chen, Guanhua Yan, and Zhao Zhang. BotTracer: Execution-Based Bot-Like Malware Detection. In *Proceedings of international conference on Information Security (ISC)*, 2008.

[16] MalwareDomains. Dns-bh malware domain blacklists. http://www.malwaredomains.com/wordpress/?p=1411.

[17] B. Scholkopf, J.C. Platt, J.Shawe-Taylor, A.J. Smola, and R.C. Williamson. Estimating the support of a high-dimensional distribution. In *Technical report, Microsoft Research, MSR-TR-99-87*, 1999.

[18] Seungwon Shin, Zhaoyan Xu, and Guofei Gu. EFFORT: Efficient and Effective Bot Malware Detection. Technical report, 2011.

[19] Elizabeth Stinson and John C. Mitchell. Characterizing the Remote Control Behavior of Bots. In *Proc. of DIMVA*, 2007.

[20] Elizabeth Stinson and John C. Mitchell. Towards systematic evaluation of the evadability of bot/botnet detection methods. In *Proc. of USENIX Workshop on offensive technologies (WOOT)*, 2008.

[21] I. Trestian, S. Ranjan, A. Kuzmanovic, and A. Nucci. Unconstrained Endpoint Profiling (Googling the Internet). In *Proceedings of ACM SIGCOMM 2008*, Mayt 200.

[22] T.-F. Yen and M. K. Reiter. Traffic aggregation for malware detection. In *Proc. of DIMVA*, 2008.

[23] Yuanyuan Zeng, Xin Hu, and Kang G. Shin. Detection of Botnets Using Combined Host- and Network-Level Information. In *Proc. of DSN*, 2010.