Error-Sensor: Mining Information from HTTP Error Traffic for Malware Intelligence

Jialong Zhang[1], Jiyong Jang[1], Guofei Gu[2], Marc Ph. Stoecklin[1], and Xin Hu[3]

[1] IBM Research
jialong.zhang@ibm.com, jjang@us.ibm.com, mpstoeck@us.ibm.com
[2] Texas A&M University
guofei@cse.tamu.edu
[3] Pinterest
huxinsmail@gmail.com

Abstract. Malware often encounters network failures when it launches malicious activities, such as connecting to compromised servers that have been already taken down, connecting to malicious servers that are blocked based on access control policies in enterprise networks, or scanning/exploiting vulnerable web pages. To overcome such failures and improve the resilience in light of such failures, malware authors have employed various strategies, e.g., connecting to multiple backup servers or connecting to benign servers for initial network connectivity checks. These network failures and recovery strategies lead to distinguishing traits, which are newly discovered and thoroughly studied in this paper. We note that network failures caused by malware are quite different from the failures caused by benign users/software in terms of their failure patterns and recovery behavior patterns.

In this paper, we present the results of the first large-scale measurement study investigating the different network behaviors of both benign user/software and malware in light of HTTP errors. By inspecting over 1 million HTTP logs generated by over 16,000 clients, we identify strong indicators of malicious activities derived from *error provenance patterns*, *error generation patterns*, and *error recovery patterns*. Based on the insights, we design a new system, Error-Sensor, to automatically detect traffic caused by malware from only HTTP errors and their surrounding successful requests. We evaluate Error-Sensor on a large scale of real-world web traces collected in an enterprise network. Error-Sensor achieves a detection rate of 99.79% at a false positive rate of 0.005% to identify HTTP errors generated by malware, and further, spots surreptitious malicious traffic (e.g., malware backup behavior) that was not caught by existing deployed intrusion detection systems.

1 Introduction

Malicious servers, such as command and control (C&C) servers, exploit servers and drop-zone servers, have become an essential part of recent cyber crimes. Most miscreants today rely on the malicious servers to control and monetize their

malicious software (malware). However, cyber criminal structures suffer from a single-point-of-failure problem when the malicious servers are discovered and blocked by defenders. To overcome such failures, cyber criminals have developed a variety of techniques to evade possible detection and launch more stealthy malicious activities. For example, a fast-flux service [30] allows cyber criminals to quickly change the IP addresses of malicious domains to avoid IP-based access controls. A domain generation algorithm (DGA) [1,33] allows cyber criminals to dynamically generate domain names to bypass domain-based blocking. Cyber criminals also compromise a large number of legitimate web servers as "stepping stones" or "redirectors" to keep their malicious activities surreptitious.

To defend against the sophisticated cybercrime systems, most, if not all organizations have already deployed a variety of security products to detect and block the malicious servers. Blacklists and intrusion detection systems (IDSes) are widely deployed in most companies. Several modern domain reputation systems [14,10] are also designed to search for the evidence of malicious activities observed at the domain names. The competition of such sophisticated evasion techniques deployed by cyber criminals and advanced detection systems deployed by companies results in two kinds of server connectivity failures in an enterprise network: DNS failures and HTTP errors. DNS failures occur when malware tries to connect to non-existing domains, and have been widely studied by researchers for malware detection [21,41], especially for DGA-based malware [12].

In this paper, we focus on *HTTP errors* which have been less investigated in previous work. We refer HTTP errors as HTTP connection failures whose response status codes are larger than `400`, as defined by the HTTP standard [2]. HTTP errors often occur when malware connects to compromised servers that have been cleared by administrators (e.g., resulting in HTTP `404` Not Found error), or to malicious servers that are blocked by an IDS or a web proxy (e.g., resulting in HTTP `403` Forbidden error) based on the policy violation.

During our investigation, we note that HTTP errors provide several new insights. First, malware often generates HTTP errors in the course of malicious activities. Most of the errors are caused by connecting to benign servers with bad parameters, connecting to compromised servers that have already been cleaned, or scanning vulnerable webpages that have been removed/protected. HTTP errors are also commonly generated because of the traffic blocks by the enterprise/ISP web proxy/gateway for policy violation or malware infection (e.g., `403` errors). Second, inspecting HTTP errors helps find out malware intelligence. When malware faces HTTP errors, it may start "recovery" behaviors to maintain the functionality of malicious infrastructures, and to ensure the reachability to the malicious infrastructure, such as connecting to some benign servers for network connectivity testing or connecting to other alternative malicious servers. Such recovery behaviors may not be easily characterized by existing IDSes, and malware bypasses security products to successfully connect to their backup malicious servers. In our experiments, we found that an IDS only detected limited parts of backup servers. Third, HTTP error-based detection is complementary to DNS failure-based detection. All the traffic related to HTTP errors have success-

ful DNS resolution, therefore DNS failure-based detection becomes less effective. Fourth, compared to existing work [40,27] which requires the entire enterprise network traffic as an input, inspecting HTTP errors dramatically reduces the amount of traffic to be analyzed (e.g., reducing 96.8% of traffic in the real enterprise network in our experiment). Fifth, different from other existing work that relies on malicious server reputation [14] and client side behavior patterns [18], focusing on the characteristics of HTTP errors detects malware-generated traffic, including both malicious and compromised servers, without requiring multiple infections, server reputation information, or infection/URL signatures.

It is not trivial to distinguish benign traffic and malicious traffic simply based on the HTTP errors because the act of generating HTTP errors itself is not a sign of inherently malware infection. However, since cyber criminals would prepare for network failures in their malware for resilient malicious operations, there exist different error generation patterns (e.g., frequencies, sequences, and statistics) between the errors generated by malware and the errors generated by benign users/software. In addition, to conquer such possible failures, malware often employs "recovery" mechanisms when facing network failures while benign users/software may have less or no pre-arranged recovery routines. Therefore, in this paper, through examining over 1 million HTTP errors from a large enterprise network, we derive new insights to detect malicious traffic, and design a lightweight yet effective detection system, Error-Sensor.

In summary, our work makes the following contributions:

- We conduct the first large-scale measurement study on 1 million HTTP errors collected in an enterprise network, and identify strong indicators of malicious activities derived from *error provenance patterns*, *error generation patterns*, and *error recovery patterns*.
- We design malware traffic detection from a new perspectives, i.e., HTTP error generating patterns and malware evasion intelligence in the face of HTTP errors, and develop Error-Sensor to automatically detect malware traffic.
- Error-Sensor is able to detect both compromised servers and malicious servers, even when benign servers are used by malware for Internet reachability testing, through analyzing HTTP error traffic. Furthermore, Error-Sensor does not rely on any infection/URL signatures, nor require multiple infections in a network unlike existing work.
- We evaluate Error-Sensor with large-scale, real-world enterprise network traces. Error-Sensor achieves a detection rate of 99.79% at a false positive rate of 0.005% to identify HTTP errors generated by malware. In addition, Error-Sensor finds surreptitious malware-generated traffic missed by an existing IDS, and uncovers evasion strategies employed by malware.

2 Background

2.1 HTTP Errors

A typical HTTP error is sent to the client web browser from a website when a problem is encountered while accessing a webpage. Based on the definition of

HTTP response status codes in RFC 7231 [2], the `4xx` class of status code is used to indicate the cases where the client caused an error.

In this paper, we focus on HTTP errors whose HTTP response status codes are in the `4xx` class because we note that most errors generated by malware activities belong to this category. For example, network scanning attacks may lead to `404` Not Found errors due to scanning non-existing vulnerable target webpages, or `403` Forbidden errors due to scanning webpages in protected paths. Policy violation requests also lead to `403` Forbidden errors. The complete list of all the error codes and their corresponding reasons can be found in [2] as the part of the HTTP/1.1 standard. For the simplicity reason, the term *error* denotes the HTTP error unless stated otherwise in the remaining of the paper.

We use a combination of a client, a server, a webpage, and an error code to represent a unique HTTP error, i.e., $\langle client_i, server_j, page_k, error\ code \rangle$. For example, an HTTP request from client i to URL `http://compromised.com/compromised_page.html` with 404 error is denoted as $\langle client_i, compromised.com, compromised\_page.html, 404 \rangle$.

2.2 Problem Statement

In this paper, we focus on the HTTP error traffic and their nearby successful request traffic (e.g., non-error traffic of a given client at about the same time as the error traffic), and conduct a large-scale systematic analysis of HTTP errors in the wild with a special focus on the error patterns and the corresponding recovery behaviors of error generators.

We first *uncover the differences* between the errors generated by users/software and the errors generated by malware. Based on the insights obtained from the analysis, then, we design a new method to *detect malware-generated errors and extract malware evasion intelligence*. In this context, intelligence means the evasion strategies used by malware in the face of HTTP errors, such as connecting to multiple alternative malicious servers, compromised servers and other benign servers for Internet connectivity testing.

We, however, do not aim to detect all the malicious traffic in an enterprise network, and may miss malware-generated traffic that never produces errors. Rather, our approach complements existing DNS failure based detection methods which address fast-fluxing and DGA [21,41,12] because all the errors here still have successful DNS resolutions. Furthermore, our approach complements existing detection systems [26] as we identify malware evasive traffic by extracting the malware evasion intelligence without having malware samples in hand.

3 Insights into HTTP Errors

In order to gain more in-depth understanding of HTTP errors generated by both malware and users/software, we first studied one day of real-world network traffic from a large enterprise network, from which we extracted all the error traffic[4].

[4] We excluded the cases where a client generated only a single error in the entire observation window, which might not provide sufficient insights for our study.

As a result, we collected more than 1 million HTTP error requests, and obtained 279,942 unique HTTP error requests (e.g., $\langle client_i, server_j, page_k, error\ code \rangle$), which only represents 3.2% of the entire one day of HTTP requests.

Table 1: $\mathcal{D}_{day}$ Data Sets

	# of clients	# of unique errors	# of errors
$\mathcal{B}$	16,205	71,998	925,277
$\mathcal{M}$	233	9,792	190,394
$\mathcal{M}_{IDS}$	233	965	35,239

Among the errors, 965 servers were detected as malicious servers by the deployed commercial intrusion detection system (IDS), and we labeled the errors as $\mathcal{M}_{IDS}$. Then we labeled the errors generated by the clients who sent requests to the servers in $\mathcal{M}_{IDS}$ as malicious errors $\mathcal{M}$. It is worth noting that the errors in $\mathcal{M}$ were generated by the malware infected clients; however, it *does not* mean that all of them are actually malicious errors. In this way, we collected 9,792 unique malicious errors. To collect benign errors, we first collected clients who never connected to malicious servers (servers in $\mathcal{M}$) nor generated policy violation requests. Then we labeled all the errors generated by these clients as benign errors $\mathcal{B}$, and collected 71,998 unique benign errors. Table 1 summarizes the data sets collected for the study, we label this one day of data as $\mathcal{D}_{day}$.

3.1 Key Observations

O1: Most benign errors are generated by an accident. When a client receives an error from a server, if the client makes at least one non-error connection to the same server during the observation time window[5], then we consider such an error is generated by a mistake and define it as an accidental error.

When a user faces an error, such as 404 Not Found error, the user may click other webpages on the same server either to figure out why the error is caused or to continue searching for other pages. In case of benign software, it may generate multiple different requests to the same server. Even if some of the requests are failed due to the misconfiguration, it still has some successful connections to the same server. For example, in our data set, Symantec liveupdate service always received 404 errors due to the misconfiguration of the proxy when it tried to request `http://liveupdate.symantec.com/liveupdate_3.3.0.107_english_livetri.zip`. However, the service also sent requests to `http://liveupdate.symantec.com` and other URIs on the domain `liveupdate.symantec.com`, which led to some successful connections. Consequently, we observed that a client encountering some errors on a server also had some successful contentions to the same server during the observation time window. However, malware may immediately try to connect to other backup servers in the face of errors. In fact, malware is typically programmed to request a certain number of pages on the same malicious servers or compromised servers. In addition, if an error is caused by an IDS or policy violation blocking, malware has no chance to establish any successful

[5] The observation time window was set to one day for $\mathcal{D}_{day}$.

connections to the malicious servers. Therefore, we would observe fewer number of accidental errors in malware-generated errors.

While examining the errors in $\mathcal{D}_{day}$, we found that 84.95% of benign errors belonged to accidental errors while only 3.94% of malicious errors belonged to accidental errors. Further study showed that most of malicious accidental errors were generated by web browsers requesting other web resources (e.g., JavaScript and image files) on the same servers, which led to successful connections.

O2: Malware infected clients generate more errors than benign clients. We define the malware infected clients as the clients who send at least one request to malicious servers. In our study, we define all the clients in $\mathcal{M}_{IDS}$ as malware infected clients. Similarly, benign clients are the clients in $\mathcal{B}$.

Intuitively, benign clients would generate fewer number of errors than malware infected clients would do because benign HTTP requests constructed by web browsers or benign software are typically well formatted, and the remote servers would respond those requests properly. On the contrary, there exist lots of uncertainties for malware generated HTTP requests, including the request format and the response from the remote servers. For example, an IDS blocks confirmed malicious servers, and malicious requests exploiting vulnerable web pages may lead to unacceptable request formats.

During our study on $\mathcal{D}_{day}$, we found that about 94% of benign clients generated less than 10 errors per day. However, only around 38% of malware infected clients generated less than 10 errors per day. The maximum numbers of errors were 2,767 and 60 for malware infected clients and benign clients, respectively. We used the number of errors per day rather than the error ratio to evaluate clients. It was because, compared to the volume of benign traffic, the volume of malware related traffic was very small so that the error ratio could be easily influenced by different volume of benign traffic. We also filtered out the clients with less than 100 requests per day to exclude the cases where fewer errors were simply due to fewer requests by benign clients.

O3: Most benign errors are generated by benign software. *User-Agent* is the field in an HTTP header to indicate who initiates the request. Typical values of the field are different browsers, spiders, or other end user tools. To understand who generate errors, we inspect the User-Agent for each error.

We define two kinds of User-Agents: browser User-Agents and custom User-Agents. *Browser User-Agents* are the User-Agents whose values reflect the version of different browsers. [3] lists the User-Agents of commonly used web browsers. Since it is difficult to collect all the User-Agents of different browsers for different versions, in this paper, we use keywords, such as Mozilla, and Opera, to label if a User-Agent is a browser User-Agent or not. Specifically, all the User-Agents started with these keywords are labeled as browser User-Agents. In practice, unless a user changes the User-Agent of the HTTP header, browser-generated errors usually include such keywords at the beginning of their User-Agents by default. *Custom User-Agents* are defined as User-Agents other than browser User-Agents. In practice, most software uses their customized User-Agents in order to be easily recognized by the servers.

During our study on $\mathcal{D}_{day}$, we collected 15,115 and 136 User-Agents for benign errors and malicious errors, respectively. We found that only 12.52% of benign errors were generated by browsers while 93.38% of malicious errors were generated by browsers. This showed that most benign errors were generated by custom software which kept requesting no longer available resources. Since different benign software usually had different customized User-Agents, we observed a large number of diverse custom User-Agents.

O4: The errors generated by malware have different generating patterns from the errors generated by benign software. To further analyze the network patterns of errors, we first clustered the errors based on the similarity of their pages and parameters. The detailed clustering algorithm is described in Section 4.3. In this way, we obtained 42 clusters for malware-generated errors ($\mathcal{C}_{\mathcal{M}}$) and 716 clusters for benign errors ($\mathcal{C}_{\mathcal{B}}$).

We examined their network patterns from three perspectives: (a) error sequences, (b) error patterns, and (c) error frequencies. From the error *sequence* perspective, 72.35% of benign errors in $\mathcal{C}_{\mathcal{B}}$ were generated in a sequence. That is, the client sent requests to servers in a specific order. The sequence typically followed the order of the servers loaded in a web page, or the order of the servers listed in a configuration file, which was steady over time. Only 33.33% of malware-generated errors were observed in a sequence. From the error *pattern* perspective, all the browser-generated errors were generated in a batch when a page was loaded. Most benign software also generated errors in a batch since it immediately tried to connect to multiple alternative servers in the face of errors. In fact, most, if not all of these alternative servers, still led to errors due to the same misconfiguration, and such errors kept appearing unless a user corrected the configuration. On the contrary, malware generated errors with a delay before trying alternative servers to avoid a traffic spike and to make its activities stealthy. From the error *frequency* perspective, we found that around 70% of errors in $\mathcal{C}_{\mathcal{B}}$ had less than 10 minimum frequency while around 70% of errors in $\mathcal{C}_{\mathcal{M}}$ repeatedly sent error requests more than 10 times per day. Surprisingly, the highest frequency for benign errors was 6,398 where Microsoft-Symbol-Server kept downloading non-existing `boinc_exe.pdb` from `berkeley.edu` and `microsoft.com`. However, the highest frequency for malware-generated errors was only 920 where a client kept accessing `file.php` from three malicious servers.

O5: Malware has more recovery behaviors than benign software. Due to the possible blocks by an IDS or ill-formatted exploit requests, malware usually employs recovery mechanisms to assure their malicious activities. We explore URI path correlation and temporal correlation to characterize such recovery intelligence. For example, the recovery behavior sending the same request to multiple alternative servers to avoid a single point of failure would lead to successful requests including the same URI (path and parameters) with error requests, and testing the network connectivity when facing errors would lead to successful requests having temporal correlation with error requests.

During our study on $\mathcal{D}_{day}$, 66.67% of malware-generated errors exhibited such recovery behaviors. On the contrary, benign software typically tried a few

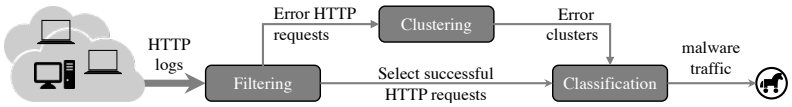

Fig. 1: Overview of Error-Sensor

alternative servers in the same domain or kept generating the same errors due to a lack of pre-arranged recovery methods. Only 8% of benign errors had temporal correlation with successful requests, and only 20% of them had similar pattern correlation with successful requests.

Lessons learned: Malware-infected clients usually generate more HTTP errors than benign clients, and most of the errors are related to malware activities. Based on our observation, malware-generated errors have significantly different error generating patterns and recovery behavior patterns from the errors generated by benign users/software. These different patterns exist because benign clients often lack recovery routines in the face of HTTP errors.

4 System Design

4.1 System Overview

We leverage the insights described in Section 3 to build a novel detection system, named Error-Sensor, which aims to detect malware behaviors by examining HTTP errors. Such malware behaviors may include HTTP attacks on benign servers (e.g., scanning vulnerable pages), communication with malicious server (e.g., C&C servers and compromised servers), and other benign behaviors (e.g., testing network connectivity). Since Error-Sensor relies on the network behaviors of malware in the face of errors, it detects malware traffic even when there are only a few (or just one) compromised clients in an enterprise network. In this paper, we focus on malware traffic related to HTTP errors, and malware traffic that is not correlated with errors is out of the scope of this work. We discuss the coverage of Error-Sensor in Section 6.

Fig. 1 shows the overview of Error-Sensor where it takes the entire HTTP traffic as an input. The filtering component first filters out noisy error traffic, and then forwards both the remaining error traffic and the select successful request traffic surrounding the remaining errors. Then, Error-Sensor groups the errors based on their HTTP URI pages and parameters, and all the errors sharing similar HTTP pages and parameter patterns are grouped together. During this process, Error-Sensor extracts various statistical features from three perspectives: error provenance, error generation and error recovery. The resulting feature vectors are then sent to the Error-Sensor classifier which is trained to distinguish malware-generated errors from benign errors.

4.2 Filtering

The goal of filtering is to reduce the amount of traffic to be processed by Error-Sensor and to filter the noisy errors. We define noisy errors as the errors gen-

erated by the clients who generate only a single error during the entire observation window[6] since it is difficult to acquire useful information from them. Since Error-Sensor relies on HTTP error patterns to identify malware-generated errors, in the filtering step, we filter out all the successful request traffic except for the successful requests within the time window T_w of error traffic. As a result, 96.8% of entire network traffic were filtered in our one day of enterprise traffic $\mathcal{D}_{day}$. As a result, 9.3% of errors were further filtered out. All the remaining HTTP requests are denoted in a form $\langle client_i, server_j, page_k, error\ code \rangle$. We acknowledge that we may miss some malware traffic by filtering out noise errors when malware probes C&C servers with a single request per day. However, it could be addressed by extending the observation time window.

4.3 Clustering

Given the filtered error traffic, the next step is to cluster them into groups. The rationale behind this step is that when facing errors, malware may start their recovery behaviors which would result in similar errors or similar successful connections. Since we rely on the recovery behaviors generated by the same client, we group the errors by each client rather than across different clients. In this way, compared to existing correlation-based detection method [18], Error-Sensor is capable of detecting malware traffic even when there is only a single infected client.

The key challenge for clustering is to determine which errors could be considered as the same. A straightforward way is to consider errors to be the same only when their URLs, including paths, pages, parameter names, are exactly matched. For example, during the vulnerable webpage scanning process, malware may send requests to multiple domains with the same target page files and the same exploit codes, or the clients may send requests to multiple compromised servers with the same compromised pages and parameters. However, certain malware campaigns may utilize obfuscated paths, such as Base64 or URL encoding for the page names. To address this problem, we set a threshold T_{len}[7] for the length of page names. If the length of the page name is shorter than T_{len}, we consider that it is unlikely an obfuscated page name, and group the errors based on `page names` and `parameters`. On the other hand, if the length of the page name is longer than T_{len}, we consider that the page is obfuscated, decode the page name with a URL decoder, and group the similar errors based on `len(page name)` and `parameters`. The clusters with a single error will be discarded because most of these errors are caused by misconfiguration where a client repeatedly sends the same requests to only one server.

4.4 Classification

In this step, Error-Sensor takes the clusters of errors and their surrounding successful HTTP requests as an input, and produces a verdict on whether the

[6] Our observation window was set to 1 day.
[7] T_{len} was empirically set to 25 based on [40].

clusters are malicious or not. Based on our key observations presented in Section 3, we develop a set of 18 features that describes the characteristics of an error cluster as summarized in Table 2.

Table 2: Feature Selection

Category	Features	Feature Domain	Novelty
Error Provenance	Client Reputation (f_1)	Integer	New
	Server Reputation (f_2)	Integer	[40]
	Software Error Ratio (f_3)	Real	New
	Accidental Error Ratio (f_4)	Real	New
	Referer Error Ratio (f_5)	Real	[31]
	Suspicious Server Ratio (f_6)	Real	New
Error Generation	Sequence (f_7)	Boolean	New
	$\text{Period}_{\text{min}}$ (f_8)	Integer	New
	$\text{Period}_{\text{median}}$ (f_9)	Integer	New
	$\text{Period}_{\text{max}}$ (f_{10})	Integer	New
	$\text{Frequency}_{\text{min}}$ (f_{11})	Integer	New
	$\text{Frequency}_{\text{median}}$ (f_{12})	Integer	New
	$\text{Frequency}_{\text{max}}$ (f_{13})	Integer	New
	$\text{Batch}_{\text{min}}$ (f_{14})	Integer	New
	$\text{Batch}_{\text{median}}$ (f_{15})	Integer	New
	$\text{Batch}_{\text{max}}$ (f_{16})	Integer	New
Error Recovery	Temporal Correlation (f_{17})	Boolean	[17]
	URI Path Correlation (f_{18})	Boolean	New

Error Provenance Pattern (EPP): This category consists of six features for evaluating the overall reputation of an error cluster.

Client Reputation (f_1) evaluates the client reputation of each cluster, which is measured by the number of errors generated by the clients in a cluster. It is worth noting that the number of errors generated by the clients includes the errors that were not initially clustered in the cluster, and the value of client reputation may be larger than the actual size of the cluster. Based on our observations discussed in Section 3, malware infected clients generate a lot more errors than benign clients does. In terms of reputation, the more errors a client generates, the lower reputation the client has.

Server Reputation (f_2) evaluates the reputation of servers in an error cluster, which is measured by the average number of clients connecting to the servers. The more popular (i.e., more clients communicating with) a server is, the less likely the server is malicious.

A *Software Error Ratio* (f_3) evaluates who generates errors, which is defined by the number of custom (non-browser) User-Agents over the total number of errors in a cluster. As noted in Section 3, majority of benign errors was generated by custom (non-browser) User-Agents while malware often used browser User-Agents to remain more stealthy.

An *Accidental Error Ratio* (f_4) evaluates how errors are generated, which is defined by the number of accidental errors over the total number of errors in a cluster. As noted in Section 3, malware often quickly gives up failed servers and moves on to other alternative servers, resulting in a high accidental error ratio.

A *Referrer Ratio* (f_5) evaluates where errors are generated. A referer provides information about the locations of the links from where a user reaches an error

page. Most malware[8] and benign software typically generate errors without referers (i.e., direct requests) while users/browsers typically generate errors with referers indicating the previous page of the error page. By default, a browser automatically add a referer field to each request [5]. We define the referrer ratio as the number of unique referers in a cluster divided by the number of errors in the cluster. Malware-generated errors would have zero or very low referer ratio.
A *Suspicious Server Ratio* (f_6) also measures the reputation of the servers in each error cluster. If a server generates only error traffic without any successful communication with its clients, ERROR-SENSOR flags the server as suspicious. These servers might be less popular servers which only few clients visit and generate errors, or malicious servers blocked by an IDS. The suspicious server ratio is defined as the number of suspicious servers divided by the total number of servers in the cluster. A higher suspicious server ratio in a cluster indicates that the cluster is more likely to be connected only by malware.

Error Generation Pattern (EGP): This category of features consists of four sub-groups of features extracted from error traffic.
A *Sequence Pattern* (f_7) characterizes whether the errors in a cluster are generated in a sequence. The rationale behind the feature is that the errors generated from browsers and benign software often follow a certain sequence while malware-generate errors are often observed in an arbitrary order. For example, a client may generate a series of 404 errors to outdated Ubuntu source repositories in the same sequence over time because the source list of update servers (e.g., `/etc/apt/sources.list`) is fixed. However, malware may randomly select C&C servers to send requests, which leads to an arbitrary order of requests.
A *Period Pattern* (f_8, f_9, and f_{10}) measures the minimum time interval for malware to generate the same errors (repeated errors). We observed that most user-generated errors did not yield repeated ones, and benign software generated errors often had short time interval of generating the same errors. However, malware typically employs some delay before reconnecting to the failed sever to avoid sudden traffic spikes. To characterize the timing pattern of repeated errors, we calculate the minimum, median, and maximum values of the minimum time interval between repeated errors.
A *Frequency Pattern* (f_{11}, f_{12}, and f_{13}) measures how many recurring errors are generated for each error per day. Most benign errors are typically generated once or per usage. For example, a set of recurring 404 errors caused by using an outdated Ubuntu source list is generated only when a user issues `apt-get` command. However, malware may periodically try to connect to malicious C&C servers to obtain new commands or updates. Considering not all of the errors in a cluster are repeated, we assess the maximum, median, and minimum of the error frequency for each cluster to characterize the error generating frequency.
A *Batch Pattern* (f_{14}, f_{15}, and f_{16}) measures the minimum time interval for malware to contact other alternative servers in a cluster. Most benign errors are often generated in a batch while malware may generate errors with some delays

[8] Although it is trivial for an attacker to manipulate the Referer field, it is easy to detect by checking if the current page is embedded in the referred page.

to avoid sudden spikes and to evade possible detection. For example, a set of 404 Not Found errors are usually generated at once when a page includes lots of missing/outdated links for scripts and resources. Typically, benign software quickly tries to reconnect to alternative servers in the face of errors. However, when malware faces errors, it may slowly complete its recovery behaviors (e.g., 1 minute to send multiple requests [36]), or delay sometime before contacting to other alternative servers to remain stealthy.

Error Recovery Pattern (ERP): This feature group consists of two features to characterize the error recovery patterns of malware in the face of errors. *Temporal Correlation* (f_{17}) characterizes the recovery behaviors of malware based on temporal correlation among errors and their nearby successful traffic. The rationale behind the feature is that when malware faces errors, it would start recovery mechanisms within a certain time. For example, malware may send requests to benign servers (e.g., `google.com/xyz` and `facebook.com/xyz` as shown in Table 7) to check network connectivity after several failed connections to malicious servers. Therefore, if a server frequently appears together with error requests, it is highly likely to be a part of malware recovery routines.

To characterize temporal patterns, we define a time window T_w to set the correlation scope, and all the requests surrounding the errors within T_w time window are extracted. To quantify the temporal correlation, we utilize association rule learning [9], which is widely used to discover significant relations between variables in a large database in information retrieval. We use the association rule learning to find out associated traffic with target errors. For each error traffic e, we extract surrounding traffic of e within T_w window, defining them as an error bucket. In this way, all the traffic in the same error bucket is considered as related traffic, and a recurring error generates a set of error buckets. Then, for each error bucket, we measure support $Supp(X)$ and confidence $Conf(X)$ in association rule mining to identify highly correlated traffic. $Supp(X)$ of traffic set X is defined as the number of error buckets containing traffic set X, which reflects how frequently traffic X appears together with the target error e. $Conf(X)$ is defined as $Supp(X)$ over the frequency of traffic set X appearing in the traffic, $Conf(X) = Supp(X)/Freq(X)$, where $Freq(X)$ is the frequency of traffic X in the surrounding traffic of target error e. Therefore, if traffic set X frequently appears together with the target error e (i.e., high $Supp(X)$) and only appears together with target error e (i.e., high $Conf(X)$), traffic set X is greatly correlated with error e and is highly likely to be the traffic of recovery mechanisms for error e. As a result, temporal correlation feature returns `True` if $Supp(X)$ is higher than threshold T_{Supp}[9], and $Conf(X)$ is higher than threshold T_{Conf}[10]; otherwise, it returns `False`. For the errors with the frequency less than 2, temporal correlation feature returns `False` since it is difficult to determine if they are truly correlated or not. The correlated traffic helps to identify backup

[9] We empirically set $T_{Supp} = 2$, which means that the traffic set X appears together with target error e at least twice.

[10] We empirically set $T_{Conf} = 0.8$, which means the majority of X appear together with error e. A lower T_{Conf} leads to low false negatives with high false positives.

malicious servers and to understand sophisticated evasion intelligence employed by malware.

URI Path Correlation (f_{18}) characterizes the recovery behaviors of malware based on URI pattern correlation among errors and their surrounding successful traffic. We note that malware may generate the same requests to multiple servers to avoid a single point of failure. In this case, some of malware traffic may lead to errors while others may be successful. For example, when malware connects to compromised servers, some of the compromised servers may have already been cleaned by administrators and lead to 404 errors while others may redirect clients to malicious servers. However, both error traffic and successful traffic would have similar content patterns (e.g., pages and parameters), and we measure the similarity between traffic using the method discussed in Section 4.3. If traffic set X is similar to target error e, path correlation feature returns `True`; otherwise, it returns `False`.

Building and Using a Classifier: Considering a set of diverse features, the classes of malicious error clusters and benign error clusters may not be linearly separable in their feature space, which makes Support Vector Machine (SVM) be less effective. In addition, tuning the parameters for diverse data is not trivial and parameter-free classifier would be desirable. Therefore, we leverage a random forest classifier (RFC) for classification, which does not require parameter tuning and is robust to handle outliers. The only two required parameters for RFC are the number of decision trees (N_t) and the number of features (N_f) per decision tree, and these parameters are independent of nuances of the data and have standard selection rules[11]. If the classifier determines that a cluster is malicious, then Error-Sensor also outputs its recovery servers based on the servers extracted through temporal and URI path correlation.

5 Evaluation

We collected 5 days of real-world web proxy logs from a large enterprise network, called $\mathcal{D}_{5days}$. The logs were gathered by Symantec ProxySG [4] infrastructure deployed at multiple locations inside the enterprise network. The proxy logs consist of connection information (e.g., source/destination IP addresses, ports, and timestamps) and HTTP header fields (e.g., Hosts, URLs, User-Agents, referers, and HTTP response codes). Overall, we collected and analyzed over 170 GB of raw proxy logs including about 1.5 billion web requests and responses. The ProxySG has a built-in intrusion detection system (IDS) with blacklists which flags known threats by matching signatures.

5.1 Effectiveness of Error-Sensor

10-fold Cross Validation. The ground truth data in $\mathcal{D}_{day}$ (shown in Table 1) consisted of 716 benign error clusters and 42 malicious error clusters. Training a

[11] N_t is typically data independent and was set to 100, and the value of N_f was log(total number of features)+1.

classifier on this unbalanced data set may bias the classification model towards the abundant class (benign errors in our case), and may be tailored for less important features, i.e., the features that may cause noises rather than contribute to the accurate classification. We addressed this problem by stratified sampling. For each client, we randomly selected a benign error cluster and keep all the malicious error clusters. As a result, our balanced training set $\mathcal{D}_{train}$ contained 136 benign error clusters and 42 malicious error clusters. We then trained a random forest model, and ran 10-fold cross validation on $\mathcal{D}_{train}$. ERROR-SENSOR achieved an average true positive rate of 98.3% at 2.2% false positive rate in terms of *error cluster* classification. The two false positive clusters included a total of 4 errors, and the single missed false negative cluster consisted of 2 errors. Therefore, the detection rate was 99.79% at 0.005% false positive rate in terms of *individual error* classification. We also applied our trained model on the remaining ground truth data ($\mathcal{D}_{day} - \mathcal{D}_{train}$), including 580 benign error clusters, and no false positive was reported.

We further performed in-depth investigation of the misclassified cases. We checked the values in their features and compared the difference between $\langle TP, FN \rangle$ and $\langle FP, TN \rangle$ to see which features led to misclassification. For the one false negative cluster, we found that there was no suspicious server (f_6=0), no accidental error (f_4=0), and no recovery behaviors (f_{17}=False, f_{18}=False), which resulted in being classified as a benign cluster. For the two false positive clusters, we found that both had a high suspicious server ratio (f_6=1), no accidental error (f_4=0), and high period time, which looked very similar to malicious errors.

Table 3: Performance with Different Features

Algorithms	TP rate	FP rate	F-score	ROC Area
ERP, EGP & EPP	0.983	0.022	0.983	0.994
ERP & EGP	0.944	0.165	0.942	0.972
Only ERP	0.815	0.502	0.79	0.701

To comprehend how different feature combinations would affect the performance of the classifier, we tested different feature groups on $\mathcal{D}_{train}$. Starting with only error recovery pattern (ERP) features from Table 2, we combined other feature categories one by one (error generation pattern (EGP) features, and error provenance pattern (EPP) features), and evaluated the performance of the classifier. As shown in Table 3, we observed that using only ERP features detected the majority of malicious errors, however led to a large number of false positives. The combination of ERP features and EGP features significantly improved the detection rate, and reduced the false positive rate. By combining all feature groups, the performance of the classifier further improved.

Real-World Application. To further evaluate the effectiveness of ERROR-SENSOR, we applied it to $\mathcal{D}_{5days}$. Table 4 reports the number of the error clusters detected by ERROR-SENSOR. Since Day-4 and Day-5 were the weekend, less amount of traffic was produced and ERROR-SENSOR therefore detected fewer malicious clusters. We verified reported clusters with the ground truth. If at least one error was flagged by an IDS, we denoted it as IDS in the table. If at least one error was labeled as policy violation by the proxy, we denoted it as

Table 4: Malicious Error Clusters Detected by Error-Sensor

	Day-1	Day-2	Day-3	Day-4	Day-5
Error-Sensor	239	216	164	45	26
IDS	32	34	17	10	6
Policy Violation	193	173	138	32	20
VirusTotal	3	0	2	0	0
Expired	1	3	0	2	0
False Positives	10	6	7	1	0

Policy Violation[12]. We also queried servers to VirusTotal [8] to see if the servers were blacklisted. If at least one server was labeled by VirusTotal as malicious, we denoted the cluster as VirusTotal. We checked Whois information of the servers. If at least the registration of one server was expired, we denoted it as Expired. We believe an expired server has a higher possibility of being exploited by cyber criminals since it has a short lifetime. For the remaining errors, we conservatively labeled them as false positives as no validated malicious evidence was available.

For example, on Day-1, Error-Sensor detected 239 malware-generated error clusters. Among them, 32 clusters were confirmed by an IDS, and 193 clusters belonged to the policy violation category. There were 3 clusters flagged by VirusTotal which were missed by the IDS and the proxy policy-based detection. There was 1 cluster containing only expired domains, indicating highly likely malicious servers. Although we had a relatively large number of false positive clusters, the sizes of the top 2 largest clusters were only 8 and 5 respectively, and all the other clusters had the size of 2 or 3. There were also several duplicate false positives recurring every day simply due to the software misconfiguration. For example, we found a client kept requesting `index.rdf` to multiple servers for RSS feeds. Some of the requests were successful while some led to `404` Not Found errors, which triggered temporal correlation and URI path correlation features.

5.2 Robustness of Error-Sensor

To evaluate the robustness of Error-Sensor, we measured the gain ratio with 10-fold cross validation to quantify the most discriminant features, which has been proven to be more robust than other alternative metrics, such as the information gain or the Gini index. Table 5 presents the top-5 features in a descending order of their gain ratios. The Avg. Rank is the average rank over 10 fold cross validation, and the Avg. Merit reflects how important a feature is (the higher, the more important) averaged over the cross validation. The numbers following $\pm$ denote the standard deviations.

Table 5: Error-Sensor Gain Ratios of the Top 5 Features

Features	Avg. Rank	Avg. Merit	Robustness
Suspicious Server Ratio (f_6)	1 ± 0	0.848 ± 0.023	High
Accident Error Ratio (f_4)	2 ± 0	0.566 ± 0.025	Low
$\text{Period}_{\text{median}}$ (f_9)	3.9 ± 0.94	0.475 ± 0.03	Medium
$\text{Period}_{\text{max}}$ (f_{10})	4 ± 0.77	0.483 ± 0.04	Medium
Client Reputation (f_1)	4.8 ± 0.87	0.435 ± 0.021	High

[12] Manual investigation confirmed all of the errors in the cluster were policy violation.

We also conservatively define the robustness of the features based on how difficult it is for cyber criminals to evade detection. If an attacker cannot manipulate or control a feature, we define the robustness of the feature as High. For example, an attacker cannot simply control how many errors are generated by a client; thus, we label the robustness of Client Reputation feature as High robustness. If an attacker is able to manipulate a feature, with some associated costs, we define the robustness of the feature as Medium robustness. For example, to influence $\mathsf{Period}_{\mathsf{median}}$ feature, an attacker is required to frequently make requests with a higher risk of being flagged as suspicious due to sudden connection spikes. In other words, an attacker might be able to evade the feature while increasing the probability of being detected. For the feature that does not require a high cost for an attacker, we label them as Low robustness. For example, an attacker may simply send requests to the valid page of the target server in order to establish successful connections and avoid accidental errors. We manipulated the value of Accident Error Ratio (f_4), $\mathsf{Period}_{\mathsf{median}}$ (f_9), and $\mathsf{Period}_{\mathsf{max}}$ (f_{10}) to zero to simulate possible evasion by an attacker on these less robust features. Then, we applied our previously trained model to the prepared evasive attack, and achieved 88.09% of the detection rate.

5.3 Case Study

In this section, we demonstrate the benefits of Error-Sensor with real cases detected by Error-Sensor. Due to the space limit, we only include a few of malicious servers in the tables.

Table 6: Conficker Botnet

	Server	Path	IDS Category
IDS	149.20.56.32	/search	Malicious_Outbound_Data/Botnets
	195.22.26.231	/search	Malicious_Outbound_Data/Botnets
	96.43.141.190	/search	Malicious_Outbound_Data/Botnets
Error-Sensor	205.164.24.45	/search	Placeholders
	149.20.56.33	/search	Computers/Internet
	216.172.154.35	/search	Placeholders

Case #1: Error-Sensor detected *more malicious servers* missed by an existing deployed IDS. Most IDSes use blacklists or signatures to detect malicious traffic, and they may miss recent sophisticated and evasive malware traffic. Table 6 shows the Conficker botnet [28] cluster. This cluster included 6 C&C servers of the Conficker botnet, which labeled by the deployed IDS as Malicious Outbound Data/Botnets. However, there were still 3 more malicious servers missed by the IDS, but detected by Error-Sensor. These severs were labeled as Placeholders and Computers/Internet categories by the IDS, and were not flagged or blocked by the IDS. On the contrary, Error-Sensor captured those 3 surreptitious malicious servers through temporal correlation and path correlation, which demonstrates the capability of Error-Sensor to identify stealthy attacks.

Case #2: Error-Sensor detected the *evasive recovery mechanisms* employed by malware. Regardless of the maliciousness, the recovery mechanisms of malware provide critical information to analyze and detect sophisticated mal-

Table 7: TDSS Botnet

	Server	Path	IDS Category
IDS	loftgun01.ru	/wet.php	Malicious_Sources
	postbox901.ru	/wet.php	Malicious_Sources
	teranian111.ru	/wet.php	Malicious_Sources
Error-Sensor	sbolt71.ru	/wet.php	Spam
	www.google.com	/efwgh/index.php	Search_Engines/Portals
	www.facebook.com	/dwrgh/index.php	Social_Networking

ware, which is often neglected by existing systems or IDSes. As shown in Table 7, the IDS detected 3 malicious servers used by the TDSS botnet [29]; however, the IDS mislabeled one malicious server as Spam category, and failed to block the malware traffic. Error-Sensor, on the contrary, precisely detected the missed malicious server through URI path correlation, and further identified 5 benign servers used in malware evasive recovery routines (e.g., testing network connectivity by connecting to benign popular servers not to raise suspicion) through temporal correlation. Further study confirmed that those benign servers were indeed reported to be used by malware [6].

Table 8: Cutwail Botnet

	Server	Path	IDS Category
IDS	diamondcpu.com	/	Malicious_Outbound_Data/Botnets
	emailmsn.com	/	Malicious_Outbound_Data/Botnets
	erzt.com	/	Malicious_Outbound_Data/Botnets
Error-Sensor	dangerous-minds.com	/	Newsgroups/Forums
	deloitte.com	/	Financial_Services
	linuxmail.org	/	Email

Case #3: Error-Sensor detected *more decoy and malicious servers* involved in malicious activities, but missed by the IDS. Table 8 shows the Cutwail botnet [7] cluster, which is notorious for sending spam emails. Based on the malware analysis report [7], the malware embeds the list of 176 hard-coded decoy servers, and it sends dummy HTTP requests to the randomly chosen server from the decoy server list before communicating with actual C&C servers. This is to minimize the exposure of actual malicious C&C communication traffic; however, it results in generating numerous errors to decoy servers. The IDS detected 13 of those decoy servers and one C&C server confirmed in [7]. Error-Sensor detected 168 new servers missed by the IDS by leveraging URI path correlation and temporal correlation. Since all the dummy HTTP requests to the decoy servers shared the same request pattern with the requests to C&C servers, it was not trivial to distinguish C&C communication from decoy communication.

6 Discussion

Limitation: Since Error-Sensor focuses on HTTP errors to detect malware traffic, it would not report malware that never generates HTTP errors. However, malware often uses HTTP [27] as either their server communication channels (e.g., C&C servers, redirectors, and payment servers) or attack channels (e.g., scanning vulnerable web pages/vulnerabilities, and attacking other web servers)

because HTTP is commonly allowed to cross enterprise network perimeters [20]. This gives Error-Sensor a great chance to detect malware traffic when such HTTP connections generate errors. In addition, malware may try to use HTTPS to evade detection, and this can be addressed by deploying web proxy servers that perform SSL-MITM in enterprise networks [25].

Evasion: An attacker who gains the knowledge about Error-Sensor might try to mislead our system by manipulating features.

Error Provenance Pattern: This group of features characterizes the properties of error sources, and it is not trivial for an attacker to influence some features. For example, an attacker may not precisely determine when and how many malware generates connection errors, especially when malicious/compromised servers get cleaned. Malware may monitor web traffic and try to manipulate a User-Agent field; however, it requires periodic monitoring and other key features help detect malicious errors as a User-Agent alone is not the most significant feature. An attacker may also easily change a Referer field; however, forged Referers can be detected by checking if the current page is embedded in the Referer page or sending the same requests to the Referer page.

Error Generation Pattern: Malware may try to change its communication patterns to yield different error generation patterns. However, it is not trivial for an attacker to achieve the goal without raising suspicion. For example, sending requests in a batch may cause connection spikes, which could be captured by existing detection systems [36]. Furthermore, an attacker may lose their reliable control if malware sends requests too slowly or randomly [20].

Error Recovery Pattern: Malware may evade temporal correlation by adding a large delay when facing errors. This can be addressed by tuning T_w threshold to handle the requests with a larger delay at an extra processing time cost. To evade URL path correlation, malware requires to target different pages and to generate different parameters. However, depending on the vulnerabilities and malicious activities, it is complicated for malware to change its attack patterns. For example, for scanning attacks on vulnerable pages, the specially crafted URI names and parameters cannot be changed, otherwise the attack does not work.

Although malware authors may be able to evade an individual feature, it is challenging to evade all of them. We believe Error-Sensor presents a new detection perspective, and a practical complement to existing malware traffic detection approaches in the battle against malware.

7 Related Work

Malicious Traffic Detection: Malicious traffic detection has been widely studied by identifying malicious domains from different angles. Many approaches detected malicious domains from the DNS point of view. Bilge et al. [14] utilized various features to evaluate the reputation of a domain. Kopis [11] monitored DNS traffic at the upper DNS hierarchy to detect malicious domains. Another line of research focused on network traffic analysis. Some approaches [27,25] detected malicious domains by extracting signatures from malware traffic. Gu

et al. [19,18] proposed anomaly-based botnet detection systems that looked for similar network behaviors across client hosts. Yen et al. [37] detected malware by aggregating traffic that shared the same external destinations or similar payload, and involved internal hosts with similar OS. Hu et al. [20] designed methods to detect regular callback patterns often generated by botnets in enterprise networks. Recently, Yen et al. [36] proposed a system to detect suspicious activities in enterprise networks by mining the features from the logs of a diverse security products. Zhang et al. [39] detected malicious servers by studying the redirection between visible servers and invisible servers. Kwon et al. [22] designed a system to detect lockstep behaviors, which captured a set of downloaders that were remotely controlled and the domains that they accessed.

Failure-based Detection: Zhu et al. [41] employed a supervised machine learning method to classify different attacks using a combination of DNS query failures and network traffic data collected for individual hosts. Yadav et al. [35] utilized the failures around successful DNS queries and the entropy of the domains belonging to those queries to detect botnet. Jiang et al. [21] characterized DNS query failures by analyzing DNS failure graphs to identify suspicious and malicious activities. Recently, Antonakakis et al. [12] extracted statistic features from DNS failures and built models for DGA botnets, which are then used for online detection. Thomas et al. [32] analyzed non-existent domain queries at several premier Top Level Domain (TLD) authoritative name servers to identify strongly connected cliques of malware related domains. Beside the DNS failure-based study, Beverly et al. [13] used network errors (e.g., TCP timeouts, retransmissions, reset) caused by bots for spam mitigation.

Malware Recovery Behavior Analysis: Some approaches [15,16] used the game theory to model interactions between an attacker and a honeypot operator to improve the information gained from honeypot. Nadji et al. [24] systematically designed a set of rules to proactively inject false network information in order to reveal the backup behaviors of malware. Dynamic binary analysis systems revealed malware behaviors by forcing execution of all possible branches, as addressed in [23,34]. Zhang et al. [38] analyzed the underlying triggering relations of a massive amount of network events, and explored such triggering relations to detect the stealthy malicious activities.

8 Conclusion

In this paper, we studied malware-generated web traffic from a new perspective, i.e., HTTP errors. We conducted the first large-scale measurement study on HTTP errors generated by both benign users/software and malware. We showed that malware-infected clients typically generated more HTTP errors than benign clients did, and there existed distinguishing patterns between the errors generated by malware and the errors caused by benign users/software. Leveraging our new findings, we designed a new system, Error-Sensor, to detect malware traffic. Our evaluation on real-world data sets demonstrated the effectiveness and robustness of Error-Sensor in detecting malware-generated traffic and com-

prehending the malware evasion intelligence. We believe that ERROR-SENSOR presents a new detection method and greatly complements existing works.

Acknowledgement.

This material is based upon work supported in part by the National Science Foundation (NSF) under Grant no. 1314823. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF.

References

1. Domain Generation Algorithms (DGA) in Stealthy Malware, https://www.damballa.com/domain-generation-algorithms-dga-in-stealthy-malware
2. Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content (RFC 7231), https://tools.ietf.org/html/rfc7231
3. List of all Browsers, http://www.useragentstring.com/pages/Browserlist/
4. ProxySG (Secure Web Gateway), https://www.symantec.com/products/secure-web-gateway-proxy-sg-and-asg
5. Referer Header, http://kb.mozillazine.org/Network.http.sendRefererHeader
6. Malware TROJ_KRYPTIK.LJC (2012), https://www.trendmicro.com/vinfo/us/threat-encyclopedia/malware/troj_kryptik.ljc
7. Story of the Cutwail/Pushdo hidden C&C server (2013), https://blog.avast.com/2013/06/25/15507/
8. VirusTotal (2017), https://www.virustotal.com
9. Agrawal, R., Imielinski, T., Swami, A.: Mining association rules between sets of items in large databases. In: SIGMOD (1993)
10. Antonakakis, M., Perdisci, R., Dagon, D., Lee, W., Feamster, N.: Building a Dynamic Reputation System for DNS. In: Usenix Security (2010)
11. Antonakakis, M., Perdisci, R., Lee, W., Vasiloglou, N., Dagon, D.: Detecting malware domains at the upper dns hierarchy. In: USENIX Security (2011)
12. Antonakakis, M., Perdisci, R., Nadji, Y., Vasiloglou, N., Abu-Nimeh, S., Lee, W., Dagon, D.: From throw-away traffic to bots: Detecting the rise of dga-based malware. In: USENIX Security (2012)
13. Beverly, R., Sollins, K.: Exploiting transport-level characteristics of spam. In: Proceedings of the Fifth Conference on Email and Anti-Spam (CEAS'8) (2008)
14. Bilge, L., Kirda, E., Kruegel, C., Balduzzi, M.: Exposure: Finding malicious domains using passive dns analysis. In: NDSS (2011)
15. Carroll, T., Grosu, D.: A game theoretic investigation of deception in network security. In: ICCCN (2009)
16. G. Wagener, R. State, A.D., Engel, T.: Self adaptive high interaction honeypots driven by game theory. In: Proceedings of the 11th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS '09) (2009)
17. Gao, H., Yegneswaran, V., Chen, Y., Porras, P., Ghosh, S., Jiang, J., Duan, H.: An empirical reexamination of global dns behavior. In: ACM SIGCOMM (2013)
18. Gu, G., Perdisci, R., Zhang, J., Lee, W.: Botminer:clustering analysis of network traffic for protocol- and structure-independent botnet detection. In: USENIX Security (2008)

19. Gu, G., Zhang, J., Lee, W.: Botsniffer: Detecting botnet command and control channels in network traffic. In: NDSS (2008)
20. Hu, X., Jang, J., Stoecklin, M.P., Wang, T., Schales, D.L., Kirat, D., Rao, J.R.: BAYWATCH: Robust Beaconing Detection to Identify Infected Hosts in Large-Scale Enterprise Networks. In: DSN (2016)
21. Jiang, N., Cao, J., Jin, Y., Li, L., Zhang, Z.: Identifying suspicious activities through dns failure graph analysis. In: IEEE ICNP (2010)
22. Kwon, B.J., Srinivas, V., Deshpande, A., Dumitras, T.: Catching Worms, Trojan Horses and PUPs: Unsupervised Detection of Silent Delivery Campaigns. In: NDSS (2017)
23. Moser, A., nd E. Kirda, C.K.: Exploring multiple execution paths for malware analysis. In: IEEE Security and Privacy (2007)
24. Nadji, Y., Antonakakis, M., Perdisci, R., Lee, W.: Understanding the prevalence and use of alternative plans in malware with network games. In: Proceedings of the Annual Computer Security Applications Conference (ACSAC'11) (2011)
25. Nelms, T., Perdisci, R., Ahamad, M.: Execscent: Mining for new c&c domains in live networks with adaptive control protocol templates. In: USENIX Security (2013)
26. Neugschwandtner, M., Comparetti, P.M., Platzer, C.: Detecting malware's failover c&c strategies with squeeze. In: ACSAC (2011)
27. Perdisci, R., Lee, W., Feamster, N.: Behavioral clustering of http-based malware and signature generation using malicious network traces. In: NSDI (2010)
28. Porras, P., Saidi, H., Yegneswaran, V.: A Foray into Conficker's Logic and Rendezvous Points. In: USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET'09) (2009)
29. Rusakov, V., Golovanov, S.: TDSS (2010), https://securelist.com/tdss/36314/
30. Salusky, W., Danford, R.: Know your enemy: Fast-flux service networks. In: The Honeynet Project (2008)
31. Stringhini, G., Kruegel, C., Vigna, G.: Shady paths: leveraging surfing crowds to detect malicious web pages. In: ACM CCS (2013)
32. Thomas, M., Mohaisen, A.: Kindred domains: Detecting and clustering botnet domains using dns traffic. In: WWW (2014)
33. Wang, T., Hu, X., Jang, J., Ji, S., Stoecklin, M., Taylor, T.: BotMeter: Charting DGA-Botnet Landscapes in Large Networks. In: ICDCS (2016)
34. Wilhelm, J., Chiueh, T.: A forced sampled execution approach to kernel rootkit identification. In: RAID (2007)
35. Yadav, S., Reddy, A.L.N.: Winning with dns failures: Strategies for faster botnet detection. In: SecureComm (2011)
36. Yen, T., Oprea, A., Onarlioglu, K., Leetham, T., Robertson, W., Juels, A., Kirda, E.: Beehive: Large-scale log analysis for detecting suspicious activity in enterprise network. In: ACSAC (2013)
37. Yen, T., Reiter, M.K.: Traffic aggregation for malware detection. In: DIMVA (2008)
38. Zhang, H., Yao, D., Ramakrishnan, N.: Detection of stealthy malware activities with traffic causality and scalable triggering relation discovery. In: ASIACCS (2014)
39. Zhang, J., Hu, X., Jang, J., Wang, T., Gu, G., Stoecklin, M.: Hunting for Invisibility: Characterizing and Detecting Malicious Web Infrastructures through Server Visibility Analysis . In: IEEE INFOCOM (2016)
40. Zhang, J., Saha, S., Gu, G., Lee, S., Mellia, M.: Systematic mining of associated server herds for malware campaign discovery. In: ICDCS (2015)
41. Zhu, Z., Yegneswaran, V., Chen, Y.: Using failure information analysis to identify enterprise zombies. In: SecureComm (2009)