Automated Generation of Models for Fast and Precise Detection of HTTP-Based Malware

Apostolis Zarras
Ruhr-University Bochum
apostolis.zarras@rub.de

Antonis Papadogiannakis
FORTH-ICS
papadog@ics.forth.gr

Robert Gawlik
Ruhr-University Bochum
robert.gawlik@rub.de

Thorsten Holz
Ruhr-University Bochum
thorsten.holz@rub.de

***Abstract*—Malicious software and especially botnets are among the most important security threats in the Internet. Thus, the accurate and timely detection of such threats is of great importance. Detecting machines infected with malware by identifying their malicious activities at the network level is an appealing approach, due to the ease of deployment. Nowadays, the most common communication channels used by attackers to control the infected machines are based on the HTTP protocol. To evade detection, HTTP-based malware adapt their behavior to the communication patterns of the benign HTTP clients, such as web browsers. This poses significant challenges to existing detection approaches like signature-based and behavioral-based detection systems.**

In this paper, we propose BotHound: a novel approach to precisely detect HTTP-based malware at the network level. The key idea is that implementations of the HTTP protocol by different entities have small but perceivable differences. Building on this observation, BotHound automatically generates models for malicious and benign requests and classifies at real time the HTTP traffic of a monitored network. Our evaluation results demonstrate that BotHound outperforms prior work on identifying HTTP-based botnets, being able to detect a large variety of real-world HTTP-based malware, including advanced persistent threats used in targeted attacks, with a very low percentage of classification errors.

I. Introduction

Over the last decade, we have witnessed a tremendous increase in the number of users connected to the Internet [17]. Simultaneously, the number of potential victims of malicious software (*malware*) has significantly increased, as many users run outdated or vulnerable software [20] and are susceptible to attacks that may lead to a compromise of their devices. These malware-infected devices, named *bots*, are typically organized into *botnets* (i.e., networks of compromised computers) that are remotely controlled by an entity, known as *botmaster* [1, 6]. Today, malware and botnets are the primary means for cybercriminals to carry out their nefarious tasks, such as sending spam emails [19, 26, 32], launching distributed denial-of-service (DDoS) attacks [10, 22], spreading new malware [12], generate revenue from online advertisements by performing click-frauds [29], or stealing personal data like email accounts and banking credentials [2]. To this end, bots establish a *Command and Control* (C&C) channel that allows botmaster to remotely control them and perpetrate criminal activities. There is a large variety of network protocols that have been used to implement the C&C channel ranging from IRC and HTTP [1, 6] to peer-to-peer (P2P) [7, 13] for a decentralized C&C structure.

Recently, numerous techniques have been proposed to detect botnets at the network level. Many detection systems focus on the transferred information during clients' infections. These systems can automatically generate [21, 23, 28] and match [24, 27] signatures on packet's payload. Others, like BotHunter [15], can recognize the information flow of the initial event sequence during a compromise. Although these approaches may detect infections at the first place, existing malware on compromised machines will probably not be detected. Other detection approaches correlate the behavior of hosts in the monitored network to unveil compromised clients and C&C servers [11, 14]. These approaches are typically protocol agnostic, but they are not effective on detecting a small number of infected machines within a monitored network, and may lead to a large number of false positive alerts. Furthermore, bots often change their behavior to be more effective and avoid detection, e.g., by emulating the behavior of benign programs. Also, most bots try to stay hidden and accomplish their tasks in a stealthy way, leaving the user unaware of an infection. These facts pose significant challenges to existing detection systems.

Nowadays, C&C channels and common types of malware, such as adware, trojans, and backdoors leverage HTTP protocol due to its large popularity [18, 19]. Consequently, the malicious traffic can blend with benign traffic and remain undetected. Existing signature-based detection systems cannot accurately distinguish between malicious and benign clients, as their resulting HTTP traffic may look alike. The main reason for this weakness is that the generic signature language used by these systems lacks higher-level information specific for HTTP, which could be used to identify the network traffic produced by HTTP-based malware. Additionally, existing behavioral-based detection systems may not be able to identify the set of infected hosts, as their network behavior can be adapted to the usual behavior of benign hosts when accessing the web.

To address the above issues, we propose BotHound: a framework for network-level detection of malware that leverage the HTTP protocol as the main channel to communicate with the C&C server or to perpetrate nefarious activities. In contrast to previous works, BotHound focuses on the HTTP requests used by both malware and benign web clients. The basic insight is the fact that two implementations of the HTTP protocol will slightly differ: empirical analysis results indicate that HTTP-relavent behavior differs due to the ambiguousness of the protocol and the respective quality of the implementation. To achieve accurate malware detection at the HTTP level, we define two new models: *header chains* and *HTTP*

templates, which can identify such implementation differences on the communication messages sent by malware and benign clients. Header chains analyze the sequence of HTTP headers sent in a request to detect a suspicious header order, while HTTP templates represent a generalized form of the HTTP headers name-value pairs generated by malware requests.

BOTHOUND can accurately classify the HTTP communications captured in a monitored network as *benign* or *malicious*, by comparing them against models of benign and malicious traffic. In particular, our framework automatically generates header chains for both benign and malicious HTTP requests, and HTTP templates for malicious requests sent by malware. Note that a captured HTTP request may match both benign and malicious header chains. Then, the HTTP templates matching define the classification. While header chains can classify rapidly the benign traffic, HTTP templates provide increased accuracy for the detection of malicious traffic, resulting in both high detection speed and accuracy. In case that no header chains or no HTTP templates match with a captured request, BOTHOUND will classify this request as *suspicious*. These suspicious requests are reported for further investigation. In general, BOTHOUND is less resource intensive, and thus faster, compared to existing Intrusion Detection Systems (IDS) that need to apply heavy pattern matching to all HTTP requests.

To demonstrate the practical feasibility of our approach, we present the design, prototype implementation, and experimental evaluation of BOTHOUND using real-world malware samples and a broad range of benign HTTP clients. We also deploy BOTHOUND in an operational network that counts tens of thousands different machines and generates several million HTTP requests on a daily basis and report our findings. Our results show that BOTHOUND detects practically all the HTTP requests sent by malware (99.97%) with a very low percentage of classification errors for the benign traffic (0.04%). In real-world deployment, BOTHOUND detected several malware instances from known malware families. Also, we demonstrate that our system is able to detect advanced persistent threats (APTs) such as *Duqu* or *Miniduke* used in targeted attacks.

In summary, we make the following main contributions:

- We propose a new approach to distinguish the network traffic generated by HTTP-based bots and malware from benign HTTP traffic. In contrast to previous detection techniques, our approach focuses on the different implementations of the HTTP protocol.
- We introduce two models that describe malicious and benign HTTP requests: header chains and HTTP templates. These models focus on particular characteristics, such as HTTP headers' sequence and structure, which reveal concrete discrepancies between benign and malicious requests.
- To assess the feasibility of our approach, we implemented and evaluated BOTHOUND. Our experimental results demonstrate that BOTHOUND is able to correctly identify the network traffic generated by HTTP-based malware in a real-world scenario with very low classification errors and high performance.

II. DETECTION APPROACH

A. Threat Model

In this work we want to detect HTTP-based malware at the network level. The detection system captures and analyzes HTTP traffic of a monitored network, aiming to detect HTTP connections originated by malware, and associate these connections with the respective malware-infected hosts. As HTTP-based malware we consider all the malware instances that use HTTP as C&C channel to communicate with their botmasters, as well as all the malware instances that perform their malicious activities over HTTP.

B. HTTP-level Detection

Our approach identifies groups of malware that interact with the web using common HTTP protocol implementations. Thus, we first learn the specific aspects of this particular implementation, and then use these insights to detect the presence of compromised machines in a monitored network. To achieve this goal, we analyze the sequence of the HTTP headers within each request and generate unique models (*header chains*) for every malware family. Since malware instances often attempt to disguise themselves as legitimate clients, we also construct distinctive templates that focus on specific patterns of the HTTP header values (*HTTP templates*). In the rest of this section, we describe the two strategies in detail.

C. Header Chains

To achieve reliable communication between a client and a server, both hosts need to "speak" the same protocol, (i.e., HTTP in our case). Note that the implementation of this protocol is not subject to strict rules, but a certain amount of freedom is granted and should be tolerated by both parties. Apart from the *Request-Line*, which includes the HTTP method, the identifier of the resource, and the protocol version, all the other headers can be omitted or provided in *any* sequence. As a matter of fact, legitimate and widely used web clients implement the HTTP protocol in slightly different ways. We expect that malicious requests will also slightly deviate from each other and from legitimate requests, as the malware authors have custom implementations of the HTTP protocol. This enables us to spot the small differences in the header order or in the individual headers, which results in a robust way to detect suspicious HTTP requests at the network level.

For this distinction, we model each client-specific implementation of the HTTP protocol. Thus, we need a set of HTTP requests generated by a web client when it performs a series of different tasks. Interestingly, the same client contains or omits HTTP headers depending on the assignment it accomplishes. For example, the header order can change after a user has performed a login at a website or new headers are added in case a web application sets a cookie. As a result, we need to model different header orders for a given web browser.

To represent the headers' order of an HTTP request, we define a *header chain* as a vector $\overrightarrow{H} = (h_1, h_2, ..., h_n)$ with header names as elements. For each HTTP request of a known client C, we create and store a pair $(\overrightarrow{H}, C)$ of the request's header chain $\overrightarrow{H}$ and the client's name C. To identify the unknown client that is responsible for an observed request,

we form its chain $U_{\overrightarrow{H}}$ and compare all our labeled header chains $\overrightarrow{H}$ with the unlabeled chain $U_{\overrightarrow{H}}$. If we find that a chain $\overrightarrow{H}$ is equal to the observed chain $U_{\overrightarrow{H}}$, we assume that the corresponding client C generated the new request.

It is possible that either no header chain matches or header chains from more than one clients match at the same time. Thus, we use header chains only as a first step of detection. If there is no match with a header chain, the request is classified as *suspicious*. If the request matches only with header chains of benign clients, it is classified as *benign*. Else, it is classified as *possibly malicious* and forwarded to the next phase.

D. HTTP Templates

In some cases, header chains may not be able to distinguish between two different clients, because they may produce identical vectors. This may happen if a malware sample attempts to use or spoof a legitimate web browser to send malicious requests, in exactly the same sequence as the legitimate browser. In this case, the header chains produced by malware and this browser will be identical.

To overcome this obstacle, we examine the name-value pairs of the HTTP headers (i.e., the whole requests) to detect similarities with malicious requests. During our experiments we found that the value of the *User-Agent* request header field can be used to identify some of these attempts. This field contains information about the user agent generating the request, and often contains subtle differences that enable us to detect suspicious requests. This header and the respective client's header chain can identify all the legitimate web clients. On the other hand, our experimental results show that malware that try to impersonate a legitimate web client usually change only the content of the *User-Agent* header, but they rarely change the headers' sequences. Thus, BOTHOUND is able to detect these malware instances. For example, we analyzed the binary of the Skynet malware (a Tor-powered trojan) and found that it contains 57 different hard-coded *User-Agent* header values, while the headers sequences always remain the same.

However, there are malware samples that produce requests where the *User-Agent* field corresponds to legitimate clients, and thus the header chains are insufficient. To address these issues, we introduce *HTTP templates*, in which we perform a full protocol parsing and clustering of the headers found in malicious requests. More precisely, we examine all the requests matching header chains of malicious traffic, or both malicious and benign, by extracting their HTTP templates and comparing them with the HTTP templates of malicious requests.

The template extraction mechanism works as follows. Initially, we distinguish between the headers' names and their values, and extract the latter for further analysis. Next, we analyze the extracted values and categorize them into bigger groups: IP addresses, port numbers, version numbers, and Base-64 encoding. Additionally, specific filetypes such as `exe`, `bin`, `jpg`, etc., are identified and generalized into clusters. For the remaining values we generate regular expression patterns. These regular expressions (we refer to them as *tokens*) can be (i) alphanumeric characters, (ii) hexadecimal digits, (iii) numerical digits, (iv) punctuation characters, or (v) whitespace characters. Finally, the generated tokens are combined with their header names to produce a new template.

When a possibly malicious request requires a deeper analysis, its template U_T is extracted and compared with the existing malicious HTTP templates T to determine if there is a match. Initially, we consider only a partial match between the request's template U_T and a template T. Given that many malware may dynamically change or generate the URL requested [3] a partial similarity is justified. Thus, we exclude the *Host* and *Request-Line* header values from the template comparison, as these values compose the URL.

If we find a partial match with at least one existing template T, we investigate the previously dismissed values. We use *URL signatures* to measure the similarity between two URLs using heuristics such as the length of the URL and the number of parameters it contains. In particular, we compare the observed URL R_U with each URL R_T contained in the partially matched templates and we measure: (i) the normalized Levenshtein distance between the first parts of the URLs that include path and page name, (ii) the Jaccard distance between the sets of parameter names, and (iii) the normalized Levenshtein distance between strings obtained by concatenating the parameter values. When the overall weighted average distance between the observed URL R_U and at least one of the URLs R_T from the partially matched HTTP templates of malicious requests is less than a specific threshold (see Section IV-C), the request is classified as malicious.

The HTTP templates are complementary to the header chains, and they do not replace them. While header chains offer a quick solution to filter benign traffic and focus only on possibly malicious requests, the templates increase the detection accuracy and raise alarms when it comes to real threats. For this reason, we use header chains as a pre-filtering stage for speed: when there is no match with malware's header chains, there will be no match with HTTP templates as well, so there is no need to continue with the computationally expensive template processing. Thus, the combination of header chains and HTTP templates create a fast and accurate detection model.

III. SYSTEM DESIGN

In this section, we describe the implementation of our approach. Our prototype implementation, called BOTHOUND and operates in two phases: a *training* phase where it learns the deviations in the implementation of the HTTP protocol by different web clients and malware and uses these details to automatically generate models, and a *detection* phase where it examines whether the captured HTTP traffic matches with the models of benign or malicious traffic. The core of the BOTHOUND architecture consists of three main components: (i) *HTTP Pool*, (ii) *Learner*, and (iii) *Decision Maker*. The HTTP Pool and Learner operate during the training phase, while the Decision Maker runs during the detection phase.

A. HTTP Pool

The HTTP Pool is a collection of Virtual Machines, each of which runs a different HTTP client. Clients are legitimate web browsers, crawlers, web libraries, and HTTP-based malware. We support four different operating systems (Windows, Mac OS X, Linux, and FreeBSD), and we use 21 distinct legitimate clients.The web clients are used with both vanilla and customized configurations: we have installed the

top ten most popular client extensions for the web browsers to also record differences based on such additions to a stock configuration. Furthermore, we capture traffic from mobile devices running iOS, Android, Symbian, and Windows 8 to also cover legitimate traffic generated by such mobile devices. To generate realistic HTTP traffic, each client visits the top $1,000$ most popular web sites based on alexa.com with a crawl depth of one. Additionally, to collect further benign data, we use a variety of software that leverage HTTP (web radio music player, HTTP-based video streaming, web stores like iTunes, Linux update processes, programs that update through HTTP, etc.). Finally, we use traffic generated from native operating system libraries, such as WinHTTP, WinInet, UrlMon, and libcurl.

Our dataset contains distinct malware samples as well. These malicious samples are collected in the wild through various malware collection and analysis platforms (such as Anubis [4], CWSandbox [31], and Cuckoo Sandbox [8]). All these binaries have been checked with virustotal.com to verify that they are indeed malicious. Previous studies have shown that malware, and especially botnets, switch controllers or download updates frequently (e.g., every two or three days [11]). In some cases, this timeframe is even smaller. Hence, all samples are repeatedly executed in our controlled environment to capture a diverse set of traffic traces.

To protect the outside world from malicious behaviors performed by malware binaries executed at the HTTP Pool, we use a firewall between the HTTP Pool and the Internet that restricts their malicious network traffic. We need to allow the network communication between malware and their C&C server (i.e., to receive updates and commands), but we want to limit their actual malicious activities. Thus, although the VMs have full network access, we throttle their bandwidth and block malicious connections through restrict firewall rules.

Although current botnet C&C communications tend to be unencrypted [16], some bots use HTTPS as their communication protocol to evade existing detection mechanisms. This problem can be solved by introducing an SSL Man-in-the-Middle (MiTM) proxy between the HTTP Pool and the actual network [5]. Nevertheless, such an approach is outside the scope of this paper.

B. Learner

The network traffic generated at HTTP Pool is forwarded and analyzed at Learner. Additionally, any other network traces captured by other sources can be fed into and analyzed by this component as well. Learner initially filters the irrelevant traffic to improve its efficiency, and extracts only the HTTP requests omitting any other types of traffic. Next, it analyzes all the HTTP requests and creates a header chain for each one. As we mentioned earlier, each client can contain more than one header chains. Moreover, for each chain it is possible that the Learner will correlate a set of different clients. In addition, Learner is responsible for the creation of the HTTP templates when the traffic is originated from malware samples. To create an HTTP template, the header names are extracted from the HTTP request and the HTTP header values are processed to create specific clusters or regular expression pattern lists. The names and pattern lists are combined to form a template for an HTTP request of a malware sample.

Learner stores each HTTP template together with the name of the responsible malware instance using a hashtable with linked lists to resolve collisions. As we keep track of the traffic generated by each malware instance, we are able to find the name of the malware that is responsible for each template. Due to the partial matching that is performed on templates, we apply a hash function only to the part of the template where an exact match is required. We use this hash value to insert and lookup templates in the hashtable very fast, and we store at each node the template together with malware names. As we have a part of the template, we may have more than one templates and malware names stored at a single node.

C. Decision Maker

The Decision Maker inspects the network traffic at real time and decides whether an HTTP connection is originated by malware or by legitimate web client. To this end, it extracts the HTTP requests, and for each request it creates the respective header chain. This chain is then compared with the entire header chains, generated at the training phase by the Learner. Instead of actually comparing the chain with every one of these header chains, it computes its hash value and performs a single lookup operation on the header chains hashtable, which reduces the searching time. If there is a match, the lookup will return a node with the set of clients that have a chain matching. In case of no match, the request is immediately classified as suspicious. Otherwise, if it matches only header chains of benign web clients, the request is classified as benign. Finally, if it matches only with header chains of malware, or with both benign and malicious clients, the request is labeled as possibly malicious and it is further inspected.

Regarding the HTTP requests that match with header chains from malware instances, we classify them as possibly malicious requests and we rely on the HTTP templates for the final decision. The classification works as follows. First, the Decision Maker computes the HTTP template of the request. Then, this template is compared with the HTTP templates produced during the training phase by malware-generated requests to obtain a partial match. As noted above, we compute the hash value only on the part of the template that an exact match is required (i.e., excluding the headers that should not completely match) to perform an efficient comparison for the partial match. A successful partial match increases the accuracy of the detection approach. In case we find at least one template with a partial match, we apply the *URL signatures* and attempt a full template match. The *Request-Line* and the *Host* values of the new request are compared against the respective values of the templates that were partially matched.

In case the Decision Maker finds a match with at least one template, the inspected request is classified as malicious. If there is no match, then it is classified as suspicious and further investigation with other tools is suggested. There is one case that a request will be classified as benign at the HTTP templates matching stage: if (i) the request matches with both benign and malicious clients, (ii) it does not match with any template, and (iii) the *User-Agent* header value accords with a legitimate client that is correlated with the request's chain. The last requirement ensures that the client reported at the *User-Agent* is the same with a client that corresponds with the request's chain, i.e., there is no mimicry attack.

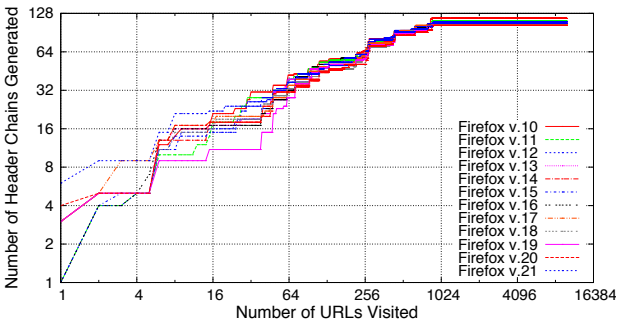

Figure 1. Number of the generated header chains for different versions of *Firefox* as a function of number of the visited URLs.

IV. Evaluation

In this section we evaluate the effectiveness of BotHound. First, we describe the ground truth we used during our experiments and then evaluate the accuracy of BotHound both with artificially created data and with real-world network traffic.

A. Establishing the Ground Truth

We created two datasets of HTTP traffic for which the ground truth has been validated, resulting in a *malicious* and a *benign dataset*. Initially, we executed at the HTTP Pool 813 malware samples, which belong to 24 malware families (e.g., Sality, ZeuS, Pushdo, SpyEye, etc.). From this process we collected more than $40,000$ HTTP requests, which form the malicious dataset. For the benign dataset, we used popular web clients and crawled the top $1,000$ domains according to alexa.com with a crawl depth equal to one. This process generated more than $7,000,000$ HTTP requests.

B. Model Generation in Various Web Clients

We noticed small variations in the header chains generated for different versions of the same client. To find the optimal number of URLs we should visit with a new client version to get all its possible header chains, we visited the top $10,000$ web sites listed by alexa.com with different versions of the *Firefox* web browser. Figure 1 shows the number of extracted header chains for 12 different versions of *Firefox* as a function of the number of visited URLs. We see that after a certain amount of URLs (less than 900) the number of generated header chains does not increase anymore. We observed similar results for the other web clients. Consequently, we conclude that after visiting the top $1,000$ URLs, all the possible header chains of a web client will have been generated.

Table I depicts examples of the number of header chains and HTTP templates generated by BotHound for different web clients. We observe that BotHound generates 527 header chains in total for all *Firefox* versions, while for each single version it produced approximately 120 chains. This implies that although a significant number of chains are identical among the different versions, different chains also exist. Additionally, we see that the number of header chains for malicious clients is significantly smaller compared to legitimate clients, presumably because their protocol implementation is not as complex as of benign clients. Moreover, we observe that the number of generated templates is slightly higher than the number of the header chains for a given malware family.

Table I. Examples for the number of header chains and HTTP templates generated for legitimate and malicious web clients.

	Web client	# Header Chains	# HTTP Templates
Benign	Mozilla Firefox	527	✗
	Google Chrome	249	✗
	Internet Explorer	470	✗
	Opera	171	✗
	Safari	143	✗
Malicious	Jorik	21	30
	Sality	4	14
	Sofilblock	13	17
	ZeroAccess	10	40
	ZeuS	17	26

Table II. Detection results for benign and malicious datasets as a function of different threshold values.

Threshold	False Positives	False Negatives
8	0.04%	0.06%
9	0.04%	0.05%
10	**0.04%**	**0.03%**
11	0.05%	0.03%
12	0.07%	0.03%

C. Detection Accuracy

To evaluate the detection accuracy of BotHound, we used a *ten*–fold cross validation for both benign and malicious datasets. Initially, we divided all the HTTP requests generated by each single benign web client into ten folds. Each disjoint fold contained approximately $700,000$ HTTP requests generated by all the benign clients. Regarding the malicious dataset, we first divided the malware instances belonging to a single malware family evenly into each fold. This way, each fold contained malware instances from all the 24 malware families. Then, we assigned the requests of each instance into the corresponding fold. As a result, each fold included around $4,000$ requests generated by malware instances. Finally, for both datasets we used 9 of these folds as training input for BotHound, and we used the remaining fold to evaluate the classification accuracy. We repeated the above procedure ten times, each time with a different fold as evaluation input.

In our first experiment, we varied the threshold used by BotHound for a full template matching to find its optimal value that minimizes both false positives and negatives. As false positives we consider the HTTP requests in the benign dataset that are classified as malicious or suspicious. Similarly, the false negative rate is the percentage of HTTP requests in the malicious dataset that are not classified as malicious. Table II shows the false positives and false negatives rates as a function of the threshold. We see that the optimal value for the threshold is 10. While smaller values increase the false negatives, larger values lead to a higher number of false alarms. Moreover, we see that BotHound with this threshold is able to detect 99.97% of the HTTP requests generated by malware with a very good accuracy, i.e., with a false positive rate of just 0.04%. Consequently, the rest of the experiments were performed using this threshold value.

To assess the classification accuracy of the three different algorithms used by BotHound, we applied them separately to the datasets and we measured their classification errors. First, we evaluated the URL signatures, then the header chains as a single detection approach, and finally the complete BotHound

Table III. CLASSIFICATION RESULTS OF DIFFERENT APPROACHES.

	URL signatures	Header Chains	BOTHOUND
False Positives	6.53%	0.49%	0.04%
False Negatives	0.04%	0.03%	0.03%

approach, which combines header chains, HTTP templates, and URL signatures. Table III shows the classification results of each approach. We see that BOTHOUND achieves significantly more accurate classification results compared to URL signatures approach, which exhibits a 6.53% false positive rate. Even BOTHOUND only with header chains outperforms URL signatures with a false positive rate of just 0.49%. We also observe that when HTTP templates are used in conjunction with header chains to improve detection, the percentage of false positives drops to 0.04%. In summary, BOTHOUND demonstrates high accuracy in the detection of malware and improves the state-of-the-art URL signature approaches.

D. Popular Malware Families

In this experiment we explore BOTHOUND capability to detect malicious traffic with and without prior knowledge of a malware family. To this end, we trained BOTHOUND with the whole benign dataset and with a portion of the malicious dataset that contains traffic only by six malware families. Then, we evaluated our approach by generating traffic from more malware families: from the six families that were included in our training datasets, and from five more families for which BOTHOUND was not trained. The results revealed that all the requests coming from malware families that were included in the training phase were correctly classified as *malicious*. On the other hand, the requests coming from the remaining families were classified as *suspicious*, i.e., BOTHOUND was able to successfully identify their respective requests as *not* originating from legitimate HTTP clients. We observed the same results for the other malware families as well. Hence, BOTHOUND is capable to accurately identify the traffic generated by malware families for which one or more samples are contained in the training datasets. Regarding the unknown malware instances, depending on BOTHOUND's configuration, it can also trigger alerts when the traffic is recognized as suspicious.

E. Real-World Deployment

To explore how BOTHOUND performs in a real-world scenario, we evaluated it using network traffic captured at the gateway of an operational network that counts tens of thousands different machines and generates millions of HTTP requests on daily basis. We trained BOTHOUND with the benign and malicious datasets we described in Section IV-A, and we used as test traffic three different datasets captured at this network during three different time periods. All the IP addresses in these datasets are stripped, but we can find the domain name of each web server from the *Host* header field of the respective HTTP request. Note that in some cases, malware may send an HTTP request to a different destination than the one displayed in the Host field, in order to evade detection. Since the actual IP addresses in our datasets are stripped, we validated the domains found in the Host field of HTTP requests classified as malicious against 30 known blacklists.

Table IV. CLASSIFICATION RESULTS OF BOTHOUND FOR THREE REAL-WORLD TRAFFIC DATASETS.

	1^{st} dataset	2^{nd} dataset	3^{rd} dataset
Benign	96.82%	99.31%	99.23%
Malicious	0.32%	0.22%	0.25%
Suspicious	2.86%	0.47%	0.52%

We evaluated BOTHOUND with the three datasets as follows. Initially, BOTHOUND inspected the 1^{st} dataset. After retrieving the classification results, we investigated the suspicious traffic looking for groups of similar header chains. Once we found clusters of header chains, we added the missing web clients in HTTP Pool. Then, we evaluated the two remaining datasets against the data-enriched BOTHOUND. Table IV presents the classification results for the three separate datasets. Regarding the malicious traffic, we see very close results in all datasets: the malicious HTTP connections detected by BOTHOUND range from 0.22% to 0.32% among the different datasets. We see a higher percentage of suspicious requests in the 1^{st} dataset (2.86%), which is reduced in the next two datasets to 0.47% and 0.52%, respectively, due to the addition of the benign clients found in 1^{st} dataset into the training set.

Obtaining ground truth for real-world data is difficult; especially when it contains tens of millions HTTP requests. To overcome this obstacle, we analyzed the requests classified by BOTHOUND as malicious and suspicious. To identify the domain name of the web server of each request, we examined the request's *Host* field. BOTHOUND classified 718 of the domains in our datasets as malicious. To analyze the extracted domains from malicious requests, we validated them against 30 blacklists. The blacklists classified 536 of these domains as malicious (74.7%). After one week we repeated the experiment and the blacklists classified 59 additional domains as malicious (82.9%). Then, we manually analyzed the 123 remaining domains. The 29 of them contained IP addresses in the Host field. When we tried to access them, all these IP addresses were unreachable. All the remaining 94 domains could be found in the alexa.com $100,000$ most visited web sites.

After reconstructing the full URL path, we noticed that in 17 of these domains the requested URLs did not exist. The traffic destined to the remaining 77 domains appeared legitimate with a first look. Interestingly, however, the header chains of the requests destined to these domains did not match with the header chains of the web client found in the respective *User-Agent* field; but they matched with header chains of existing malware families. A possible explanation of this finding is that many malware instances put a popular legitimate domain in the Host field, and send benign HTTP requests as noise traffic in order to confuse detection systems. Indeed, we found some malware instances in our dataset that used popular domain names in the Host field of the requests, while the requests were actually destined to different sites. We do not consider the classification of such noise requests generated by malware as false positives, because they can be used to identify malware and compromised machines, which send noise requests with the HTTP implementation that matches the header chains and HTTP templates of known malware. Thus, not only the noise is not useful for the malware, but also it turns out that it may help BOTHOUND to detect traffic generated by malware and pinpoint compromised machines.

Next, we examined the HTTP requests that were classified as suspicious. After a deeper insight on the data, we found that a large number of crawlers, which compose the 63.8% of the suspicious requests, were actively attempting to mimic the behavior of legitimate web browsers. Although they had changed their *User-Agent* headers using strings from real browsers, they had not changed their HTTP headers sequences. Thus, the header chains identified this discrepancy. The remaining 36.2% of the suspicious HTTP requests were originated either by malware that sent legitimate requests to blend malicious with benign traffic, or by versions of legitimate browsers we had not included in our training dataset.

F. Advanced Persistent Threats

In our next set of experiments, we tested BotHound's detection capabilities when it inspects the traffic generated by advanced malware samples used in targeted attacks (e.g., *Duqu* and *Miniduke*). We did not expect BotHound to classify the attack traffic of this evaluation dataset as malicious, because we had not previously trained it with the respective malware samples. However, it classified the HTTP requests generated by these samples as suspicious, because they did not match with any of the header chains of a legitimate web client. Thus, BotHound provides a clear indication of malicious traffic even for such unknown threats, while the large majority of the benign traffic can be identified.

Then, we trained BotHound with the HTTP requests of these malware samples to generate new header chains and HTTP templates. As expected, BotHound with the new models was able to successfully detect the traffic produced by these malware. To examine whether BotHound produces false positives with the new models, we evaluated it with the real-world traffic datasets we used in Section IV-E. We already knew that the requests from these malware were not contained in these datasets. Indeed, when BotHound was trained with the new malware samples, it did not report any malicious requests by them in the real-world datasets.

V. Discussion

A. Cloaking

A commonly used evasion attempt is performed when a malware injects noise into its traffic. This noise can be benign HTTP requests to cloak the real C&C communication channel, or malicious requests to benign domains to confuse a signature generation system. In Section IV-E we found that noise traffic generated by malware may actually help BotHound to detect the malware-infected machines. However, when BotHound uses such requests as a training dataset, it risks to also classify benign requests and domains as malicious or suspicious, resulting in classification errors. Indeed, in our experimental evaluation we observed that some malware try to blend malicious with benign traffic. These mislabeled data were responsible for a fraction of the suspicious requests we mentioned in Section IV-E. In case a malware uses its own implementation of the HTTP protocol, and it does not perfectly correspond with a legitimate client, the malware can be detected by BotHound even if it produces noise. However, if the malware uses a perfect implementation of a web browser's HTTP usage and sends noise over it, then we need more advanced behavioral-based algorithms for more accurate detection.

Unfortunately, the assurance that a dataset contains only benign or only malicious samples is an open problem to community, and most of the times requires manual efforts to correctly label the training data. BotHound can address this issue by introducing a pre-filtering mechanism in malware-generated traffic during the training phase, by filtering out the traffic that is not actually related to malicious activities [18].

B. Randomness

Attackers may also try to evade detection by having no constant patterns in their HTTP requests. For example, each new request could look completely, or partially, different from the previous requests. Such an attack could bypass our detection mechanisms, a limitation we share with other signature-based detection approaches. However, all the possible combinations follow a deterministic model. Thus, we believe that if we execute the malware samples in HTTP Pool for a long period, all the possible combinations will be revealed at the end.

VI. Related Work

Network-level intrusion detection systems utilize signatures to detect malicious traffic [24, 27]. A common practice is to generate signatures for detecting malware that use network protocols for their activities [28]. To encounter polymorphic malware, disjoint signatures are generated to find substrings that are unique even in polymorphic payloads [21, 23]. Such detection systems may not be able to accurately distinguish the traffic of benign clients from the traffic of malware that mimic the behavior of these clients, as these systems do not specialize in the implementation details of HTTP. In contrast, BotHound utilizes the header sequence and HTTP request structure to give more insights into the HTTP protocol implementation differences and thus, the false positive ratio remains low.

On the other hand, several botnet detection systems focus on network flow analysis [9, 14] or require deep packet inspection [15] to detect compromised machines within a local network. Other detection approaches aim to identify common spatio-temporal behaviors of bots when performing malicious activities [11, 14]. Such approaches are less effective when malware generates only a small fraction of the overall traffic and hides its malicious activities and C&C communication in benign-looking traffic (e.g., HTTP requests). In contrast, BotHound aims to detect slight differences in the HTTP protocol when implemented by malware and benign clients.

Perdisci et al. [25] propose a system that focuses on the network-level behavior of HTTP-based malware to find similarities and generate signatures for malware clusters. BotHound improves the accuracy of URL signatures proposed in this work by working on the HTTP request headers, instead of the complete HTTP content, which reduces the amount of data that need to be processed and results in increased processing throughput compared to content processing.

Template approaches have been applied in spam mitigation and such methods are closely related to our work. Botnet Judo [26] generates SMTP templates by monitoring spamming botnets in a controlled environment, and uses these templates to perform real-time spam filtering at the network-level. Similarly, we apply a generalized template-based detection to the HTTP protocol, using templates to model malicious requests.

Additionally, BOTHOUND leverages header chains as a first step to reduce the amount of requests that need to be analyzed with templates, increasing significantly the detection speed.

Stringhini et al. [30] introduce a spam detection system based on the insight that email clients and spamming bots implement the SMTP protocol in different ways. Their system automatically learns the different SMTP dialects of benign clients and bots, and uses them for identification. We apply a similar approach for HTTP-based malware detection, aiming to identify small but clear differences in the HTTP protocol implementation between benign clients and malware.

VII. CONCLUSIONS

In this paper we presented BOTHOUND, a network-level detection framework that focuses on HTTP-based malware and detects malicious HTTP traffic. The key idea of BOTHOUND is to identify slight deviations between different HTTP protocol implementations. BOTHOUND automatically generates models, which can accurately classify benign and malicious HTTP connections. We deployed BOTHOUND in an operational network verifying that it is able to detect a large variety of real-world malware, including advanced persistent threats used in targeted attacks, with very low false positive rate. We also found that BOTHOUND was able to early detect malicious domains before various popular blacklists publish them. Finally, we demonstrated the improved accuracy of our approach by comparing it with existing state-of-the-art approaches like URL signatures.

ACKNOWLEDGMENTS

This work has been supported by the German Federal Ministry of Education and Research (BMBF) under grant 16BP12302; EUREKA-Project SASER.

REFERENCES

[1] M. Abu Rajab, J. Zarfoss, F. Monrose, and A. Terzis. A Multifaceted Approach to Understanding the Botnet Phenomenon. In *ACM SIGCOMM Conference on Internet Measurement (IMC)*, 2006.

[2] D. S. Anderson, C. Fleizach, S. Savage, and G. M. Voelker. Spamscatter: Characterizing Internet Scam Hosting Infrastructure. In *USENIX Security Symposium*, 2007.

[3] M. Antonakakis, R. Perdisci, Y. Nadji, N. Vasiloglou, S. Abu-Nimeh, W. Lee, and D. Dagon. From Throw-Away Traffic to Bots: Detecting the Rise of DGA-based Malware. In *USENIX Security Symposium*, 2012.

[4] U. Bayer, C. Kruegel, and E. Kirda. TTAnalyze: A Tool for Analyzing Malware. In *Annual Conference of European Institute for Computer Antivirus Research (EICAR)*, 2006.

[5] F. Callegati, W. Cerroni, and M. Ramilli. Man-in-the-Middle Attack to the HTTPS Protocol. *Security & Privacy, IEEE*, 7(1):78–81, 2009.

[6] E. Cooke, F. Jahanian, and D. McPherson. The Zombie Roundup: Understanding, Detecting, and Disrupting Botnets. In *USENIX Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI)*, 2005.

[7] B. Coskun, S. Dietrich, and N. Memon. Friends of an Enemy: Identifying Local Members of Peer-to-Peer Botnets Using Mutual Contacts. In *Annual Computer Security Applications Conference (ACSAC)*, 2010.

[8] Cuckoo Sandbox. Automated Malware Analysis. http://www.cuckoosandbox.org/, Jul 2013.

[9] J. François, S. Wang, T. Engel, et al. BotTrack: Tracking Botnets Using NetFlow and PageRank. In *IFIP Networking Conference*, 2011.

[10] F. C. Freiling, T. Holz, and G. Wicherski. Botnet Tracking: Exploring a Root-Cause Methodology to Prevent Distributed Denial-of-Service Attacks. In *European Symposium on Research in Computer Security (ESORICS)*, 2005.

[11] J. Goebel and T. Holz. Rishi: Identify Bot Contaminated Hosts by IRC Nickname Evaluation. In *USENIX Workshop on Hot Topics in Understanding Botnet (HotBots)*, 2007.

[12] J. Goebel, T. Holz, and C. Willems. Measurement and Analysis of Autonomous Spreading Malware in a University Environment. In *Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*, 2007.

[13] J. B. Grizzard, V. Sharma, C. Nunnery, B. B. Kang, and D. Dagon. Peer-to-peer Botnets: Overview and Case Study. In *USENIX Workshop on Hot Topics in Understanding Botnet (HotBots)*, 2007.

[14] G. Gu, R. Perdisci, J. Zhang, and W. Lee. BotMiner: Clustering Analysis of Network Traffic for Protocol-and Structure-Independent Botnet Detection. In *USENIX Security Symposium*, 2008.

[15] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee. BotHunter: Detecting Malware Infection Through IDS-Driven Dialog Correlation. In *USENIX Security Symposium*, 2007.

[16] N. Ianelli and A. Hackworth. Botnets as a Vehicle for Online Crime. In *International Conference on Forensic Computer Science (ICoFCS)*, 2006.

[17] Internet World Stats. Internet Usage Statistics: World Internet Users and Population Stats. http://internetworldstats.com/stats.htm, Jun 2012.

[18] G. Jacob, R. Hund, C. Kruegel, and T. Holz. Jackstraws: Picking Command and Control Connections from Bot Traffic. In *USENIX Security Symposium*, 2011.

[19] J. P. John, A. Moshchuk, S. D. Gribble, and A. Krishnamurthy. Studying Spamming Botnets Using Botlab. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2009.

[20] Kaspersky Lab. Kaspersky Lab Report: 23% of Users Are Running Old or Outdated Web Browsers, Creating Huge Gaps in Online Security. http://www.kaspersky.com/about/news/virus/2012/kaspersky_lab_report_23_of_users_are_running_old_or_outdated_web_browsers_creating_huge_gaps_in_online_security, Nov 2012.

[21] Z. Li, M. Sanghi, Y. Chen, M.-Y. Kao, and B. Chavez. Hamsa: Fast Signature Generation for Zero-day PolymorphicWorms with Provable Attack Resilience. In *IEEE Symposium on Security and Privacy*, 2006.

[22] J. Mirkovic and P. Reiher. A Taxonomy of DDoS Attack and DDoS Defense Mechanisms. *ACM SIGCOMM Computer Communication Review*, 34(2):39–53, 2004.

[23] J. Newsome, B. Karp, and D. Song. Polygraph: Automatically Generating Signatures for Polymorphic Worms. In *IEEE Symposium on Security and Privacy*, 2005.

[24] V. Paxson. Bro: A System for Detecting Network Intruders in Real-Time. *Computer networks*, 31(23):2435–2463, 1999.

[25] R. Perdisci, W. Lee, and N. Feamster. Behavioral Clustering of HTTP-Based Malware and Signature Generation Using Malicious Network Traces. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2010.

[26] A. Pitsillidis, K. Levchenko, C. Kreibich, C. Kanich, G. M. Voelker, V. Paxson, N. Weaver, and S. Savage. Botnet Judo: Fighting Spam with Itself. In *ISOC Network and Distributed System Security Symposium (NDSS)*, 2010.

[27] M. Roesch. Snort: Lightweight Intrusion Detection for Networks. In *USENIX Large Installation System Administration Conference*, 1999.

[28] S. Singh, C. Estan, G. Varghese, and S. Savage. Automated Worm Fingerprinting. In *USENIX Symposium on Operating System Design and Implementation (OSDI)*, 2004.

[29] B. Stone-Gross, R. Stevens, A. Zarras, R. Kemmerer, C. Kruegel, and G. Vigna. Understanding Fraudulent Activities in Online Ad Exchanges. In *ACM SIGCOMM Conference on Internet Measurement (IMC)*, 2011.

[30] G. Stringhini, M. Egele, A. Zarras, T. Holz, C. Kruegel, and G. Vigna. B@bel: Leveraging Email Delivery for Spam Mitigation. In *USENIX Security Symposium*, 2012.

[31] C. Willems, T. Holz, and F. Freiling. Toward Automated Dynamic Malware Analysis Using CWSandbox. *Security & Privacy, IEEE*, 5(2):32–39, 2007.

[32] Y. Xie, F. Yu, K. Achan, R. Panigrahy, G. Hulten, and I. Osipkov. Spamming Botnets: Signatures and Characteristics. In *ACM SIGCOMM Conference on Data Communication*, 2008.