Automated Classification of C&C Connections through Malware URL Clustering

Nizar Kheir[1], Gregory Blanc[2], Hervé Debar[2], Joaquin Garcia-Alfaro[2], and Dingqi Yang[2]

[1] Orange Labs
92794 Issy-Les-Moulineaux
[2] Institut Mines-Telecom, Telecom SudParis
CNRS Samovar UMR 5157, 91011, Evry, France

Abstract. We present WebVisor, an automated tool to derive patterns from malware Command and Control (C&C) server connections. From collective network communications stored on a large-scale malware dataset, WebVisor establishes the underlying patterns among samples of the same malware families (e.g., families in terms of development tools). WebVisor focuses on C&C channels based on the Hypertext Transfer Protocol (HTTP). First, it builds clusters based on the statistical features of the HTTP-based Uniform Resource Locators (URLs) stored in the malware dataset. Then, it conducts a fine-grained, noise-agnostic clustering process, based on the structure and semantic features of the URLs. We present experimental results using a software prototype of WebVisor and real-world malware datasets.

1 Introduction

Malware constitutes a serious threat to the Internet. Once it infects a terminal, malware may perform a variety of actions, such as taking over the system, connecting to Command and Control (C&C) servers, leaking information to a dropzone, and recruiting the terminal to a botnet involved in activities such as spam and denial of service. Efforts in the literature aim to handle malware both at the system and network level. While traditional host-based malware detection systems suffer from low detection coverage [19], network-based detection offers a complementary approach to detect malware through its network activity [6, 7, 10, 12]. It usually adds network-level patterns, i.e., patterns referring to any network activity triggered by malware instances. For instance, they can leverage C&C activity, which is a key feature of malware operation. By comparing the network traffic of different malware samples, it is possible to identify similar patterns that can be further used for malware detection.

To keep pace with the large number of malware being collected daily, current solutions aim at automatically classifying malware and extracting appropriate detection signatures [20, 11, 24]. For example, the behavioral classification system in [20] correlates HTTP traffic from different malware samples and extracts network signatures for detection. It observes common HTTP artifacts in order to find trends that are shared among a large set of malware samples, and that may characterize a given malware family. Another approach provided earlier by [14] also observes network traffic for a large set of malware samples and identifies pattern signatures for detection. However, malware avoids being correctly classified by these systems as it uses several network obfuscation mechanisms such as encrypting its C&C traffic, injecting noise, using Domain Generation Algorithms (DGA) [1], or embedding efficient failover strategies [17].

In this paper, we present WebVisor, an automated tool to classify malware instances based on the features of their web-based C&C applications. These are the set of web server applications that are installed by an attacker (i.e. botmaster) in order to establish C&C communications with the remote infected bots. WebVisor implements a behavioral-based approach that observes the network activity of malware when executed in a sandbox. It classifies malware into families and generates family detection signatures. WebVisor targets malware C&C communication channels supported by standard network protocols, such as the Hypertext Transfer Protocol (HTTP). HTTP-based malware belonging to the same botnet family connects to a shared infrastructure that involves the same set of web C&C applications. Such malware C&C applications are uniquely identified and accessed using Uniform Resource Locators (URLs). The classification process conducted by WebVisor assumes that malware belonging to the same family shares similar C&C connection patterns, including similar sets of C&C attributes, both in terms of parameter names, semantics, and values.

The benefits of WebVisor are threefold. First, it provides a behavioral approach that classifies malware into families based on features of their network traffic, with no need to analyze the system-level activity of malware on the infected terminals. Second, it provides a malware detection system that operates semantical enrichment and density-based clustering in order to automatically reduce the impact of noise and common obfuscation mechanisms used by malware up to a certain level. Third, it identifies URL features common to a given malware family. Such features are further used to detect and classify other malware instances. To verify our claims, we present an experimental validation of a first prototype against live Internet traffic collected from a large ISP provider.

This paper is organized as follows. Section 2 provides the background and components underlying WebVisor. Section 3 describes our experimental results. Section 4 surveys related work. Section 5 concludes the paper.

2 Background and System Overview

Network-based malware detection solutions can be classified into two main categories. First, solutions addressing network activities attributed to synchronized botnet operations (e.g. [6, 15]). Second, solutions executing the malware and analyzing its associated traces, to learn new malware techniques (e.g. [7, 11]). The first category works only when multiple infected terminals are using the same botnet architecture and are controlled by a single entity. Modern botnets avoid this type of detection through the hiding of their C&C activity by, e.g., using fake connections and adding statistical inconsistencies in their network traffic [21]. The second category assumes that malware belonging to the same family (e.g., in terms of development tools) shares similar behavioral patterns that reflect its origin and purpose [24].

WebVisor belongs to the second category. It observes HTTP traffic triggered by malware during dynamic analysis, and identifies URL patterns that are shared among samples of the same family. Hereinafter, we interchangeably use the terms *C&C patterns* and *URL patterns* to refer to specific character strings in the URLs triggered by malware during its execution in a sandbox. When these patterns are shared among multiple variants of the same family, they characterize specific features of their C&C applications. In the end, the patterns are used to build appropriate detection signatures. Remaining malware URL instances that do not convey shared C&C activity, and so they do not characterize specific malware families, are automatically discarded by WebVisor and they are no longer used during detection. We detail next the main blocks underlying WebVisor.

Input Data – The malware C&C communication channels addressed by WebVisor are supported by standard network protocols, such as HTTP, which is the most common type of malware communication on the Internet today [4]. Malware using HTTP-based C&C channels may efficiently bypass firewall and proxy settings, by hiding the C&C exchanges within benign HTTP traffic. Furthermore, HTTP-based malware may also evade detection by leveraging infected or legitimate websites, which makes the detection very challenging [9]. The input HTTP data processed by WebVisor are the URL methods (e.g. Get, Post, Head), the absolute paths, and the parameters of the URLs associated to the C&C communication channels. They are captured during the dynamic analysis of malware instances on a sandbox. Domain names from the stored URLs are ignored, since they do not convey information about the structure and content of the C&C applications. As opposed to domain names, URL paths provide the precise applications at the C&C server to handle malware requests. This highlights common patterns that are likely to be shared among multiple variants of the same malware family. In addition to paths, WebVisor also uses the URL parameters, leveraging attribute names, their semantics and values. The stored URLs handled by WebVisor are grouped into an initial set of coarse-grained clusters, using the statistical clustering process that we outline next.

Statistical URL Clustering – This process partitions the input data into a collection of coarse-grained clusters based on common URL statistical features. URLs often include patterns (e.g., `/images/`, `/adi/`, `/generate_204/`) and keywords (e.g., `.php`, `.exe`, `.gif`) that refer to the nature and type of resources accessible on a remote server. WebVisor leverages the distribution of characters within URLs in order to group together malware URLs that include similar or redundant patterns. It builds a features vector that captures the distribution of characters within the URL. Paths and parameters are handled separately, since they hold different structural nature and semantics. In turn, the parameters are separated into keys and values. For instance, the following URL '`/doc/lat/widget?tp=2&nbr=1111&tag=11`', whose protocol identifier and domain name are already removed, is split up into the path '`/doc/lat/widget`'; keys '`tp`','`nbr`', and '`tag`'; and values '`2`','`1111`', and '`11`'.

Since high-level features such as the URL length, the number of paths and number of attributes do not capture relevant pattern signatures, WebVisor leverages string based features that capture shared patterns among different URL instances. The frequency of occurrence of each single character in a given URL is computed. For example, assuming the following path '`/doc/lat/widget`', the occurrence frequency of character '`o`' is 1, '`t`' is 2, '`/`' is 3, '`z`' is 0, and so on. Following such a rationale, each URL is transformed into an m-ary vector that captures the distribution of characters within the URL. Given that the HTTP standard sets to 128 the number of acceptable ASCII codes for a character in a given URL; and given that paths, keys and values in each URL are treated separately, the value of m is settled to 384 (i.e., 3×128). Based on the m-ary vector associated to each URL, the initial set is partitioned into coarse-grained clusters. A vector quantization clustering method is used to drive the process. For instance, the process can be conducted using incremental k-means [22], as reported in Section 3. Finally, each coarse-grained cluster is further processed by a second clustering process, to build the eventual fine-grained clusters whose structure and semantic shall characterize common C&C applications. Density-based clustering drives this second process that we outline next.

Density-based URL Clustering – After the statistical coarse-grained process, a fine-grained density-based clustering is conducted within each of the statistical coarse-grained clusters. The process starts by an enrichment procedure that adds meta-data to characterize the type and semantics of each URL value field, based on the types listed in Table 1. The first column in table 1 introduces a shortlist of the attribute types used by WebVisor, column 2 illustrates some examples, and column 3 provides a brief description. Such meta-data is further used to build fine clusters where URLs are associated to semantically equivalent instances. This way, and instead of comparing values as strings via, e.g., string distance functions, the density-based clustering process considers that two values are similar when they share the same semantics (e.g., both are timestamps). The rationale behind this configuration is that botnets usually add encryption and use URL encodings to evade network detection signatures, since it alters the entropy of characters distribution in a URL. The proposed enrichment process aims to handle such evasion techniques and remove the encoded values when it compares two different URLs. Other non-encoded parameters in the URL, such as IP or MAC addresses, country code and timestamps, are also compared semantically. A '`No_Type`' entry is introduced in order to handle unknown types. After the semantic enrichment process, fine-grained clusters are built up using a density-based classifier. For instance, assuming a density-based classifier based on DBScan [5], WebVisor builds up a similarity matrix containing the distance between each couple of URLs. Inputs to the URL distance function include the URL method (e.g., Get, Post, Head), the URL path, and the URL parameters. The similarity between two URLs is computed by using the Jaro-Winkler distance [8] to compare URL paths as string chains, and by comparing the pa-

Type	**Example**	**Description**
URL redirection	`http://example.com`	Phishing attacks or obfuscation using URLs similar to legitimate websites
File path	C:\test.txt	File location on the victim terminal
SHA1	97d07314f735998585bb-8e2d6b5acb5ac7956690	Cryptographic hash function including 40 hexadecimal characters
Base64	dG90bw==	Encoding schemes that represent binary data using ASCII or UTF-8 formats
MD5	4f863423326e85d44aae-147d2d86e1c0	Cryptographic hash function consisting of 32 hexadecimal figures.
MAC address	0a:00:27:00:00:01	MAC address of the infected terminal
IP address	192.168.0.10	IP address of the infected terminal
Serial number	06AE-B34D	The volume serial number on the infected terminal
Timestamp	Mar 30 2014 00:30:08	The local time on the victim terminal
No_type	utv42	Any value not matching a previous type

Table 1. Non-exhaustive list of types used during the semantic enrichment process

rameters and values semantically, i.e., two parameters in two different URLs are similar in case they have the same key and the same semantic type.

Generation of Signatures – Detection signatures are created by extracting the longest common substrings for all URLs in a given dense cluster, using the Generalized Suffix Trees algorithm [3]. It builds a token-subsequence starting with the longest token in the ordered list of longest common substrings. Selected tokens are further used as input to the Best Alignment Algorithm [18] in order to build a pattern-based signature that characterizes all URLs that belong to the same dense cluster.

WebVisor Prototype – A software prototype of WebVisor is available for testing purposes at `http://j.mp/WVProto`. It implements all the processes introduced in this section, i.e., statistical clustering, semantic enrichment, density-based clustering, and signature generation processes. The prototype has been used in order to process some real-world malware datasets that we describe in section 3, with the objective of generating detection signatures. The set of signatures has also been tested against live Internet traffic from a large ISP provider. Results and discussions about our findings are provided in the following section.

3 Experimental Results

Experiments were conducted using the WebVisor prototype, on an Intel 8-core 2.67Ghz server, with 16Gb RAM. The statistical URL clustering process was conducted using incremental k-means [22] and Euclidean distance to compare the feature vectors. The density-based URL clustering process was conducted using DBScan [5]. Finally, a python script is used to transform the fine-grained clusters into regular expressions as detection signatures.

The malware URL C&C communication dataset used to generate the detection signatures was collected from multiple public and private sources, including commercial feeds, public repositories (e.g., `http://malware.lu/` and Malicia [16]), and HTTP traces triggered by malware from the Anubis database [2] (during their execution in a dynamic analysis environment). Almost a quarter million malware samples were considered, and more than two million HTTP traces were collected, from which duplicates and empty URLs were excluded. The MD5 hashes of malware binaries associated to each URL were also used to label our dataset. To ease the analysis, the dataset was divided into three separate categories, according to the year of collection of the URLs (24 months, from June 2011 until July 2013). Using WebVisor, we separately processed each category, and generated the corresponding family clusters and detection signatures.

Year	**Samples**	**Families**	**URLs**	**Get**	**Post**	**Head**	**Coarse Clusters**	**Signa-tures**	**Process Time**
2011	75,398	127	886,077	68%	20%	12%	27	120	1h15min
2012	87,648	129	592,104	65%	24%	11%	27	182	2h01min
2013	85,597	84	848,998	76%	17%	7%	29	315	2h50min

Table 2. Dataset summary

To properly validate the experiments, ground truth labels indicating malware families were generated. More than two-hundred distinct families, including each more than a dozen malware samples, were settled by using AntiVirus (AV) signatures from services such as VirusTotal (cf. `http://virustotal.com/`). Notice that AV editors usually assign conflicting signatures for the same malware sample. For example, the `SpyEye` malware has a *kaspersky* signature of `Trojan-Spy.Win32.SpyEyes` and a *McAfee* signature of `PWS-Zbot.gen.br`. To avoid errors, the AV labels were associated to multiple keywords, and common prefixes such as `W32`, `Mal` and `Trojan` were discarded. Generic malware identifiers, such as `Heur`, `Worm`, `Gen`, and `malware`, were also discarded. The site `http://spywareremove.com/` was used to group together all aliases of a given family. For example, the signatures `win32.spammy` by *kaspersky* and `W32/Sality` by *McAfee* were identified as aliases for the same `sality` malware, and considered as part of the `Sality` family. Multiple malware families may cover two or more years, including examples such as `Zeus`, `ZeroAccess`, and `Sality`. This overlap in families between years is explained by the fact that samples of the same malware family can be distributed through multiple infection campaigns.

Table 2 summarizes some of the above information, as well as the time required for WebVisor to generate the detection signatures. We provide in the following sections a more elaborated analysis of the experimental results, such as evaluation of the obtained clusters and signatures, and evaluation of the signatures against live Internet traffic.

3.1 Cluster Validation

The malware clustering problem is assumed to be a classification subproblem. We use two distinct quality metrics, precision and recall, to evaluate the quality of each individual behavioral cluster. A cluster family is defined as being the ground truth label associated with a maximal number of samples in a cluster. Note that malware belongs to a cluster when it has at least one URL that is classified by WebVisor into this same cluster. Moreover, malware may belong to multiple clusters in case it interacts with multiple C&C applications during analysis, and whose associated URLs are classified by our system into different clusters. The cluster precision captures the level of *mis-classifications* within the cluster, which is the rate of samples in the cluster that are not associated with the cluster family. Let η_c be the number of malware samples in cluster c, and $|Sig_c| \leq \eta_c$ the maximal number of samples in c that have the same ground truth label. Then, the precision index of c is computed as $\mathcal{P}_c = \frac{|Sig_c|}{\eta_c}$. The cluster recall captures the proportion of samples that should belong to a cluster, but misclassified into other clusters. Let $|\overline{Sig}_c|$ be the number of samples in the ground truth dataset that should be classified in c, but that were misplaced into other clusters. Then, the cluster recall index of c is computed as $\mathcal{R}_c = \frac{|\overline{Sig}_c|}{|Sig_c|+|\overline{Sig}_c|}$. Tables 3 and 4 contain the distributions of the cluster precision ($\mathcal{P}_c$) and recall ($\mathcal{R}_c$) coefficients for the 617 fine-grained clusters provided by WebVisor during our experiments (cf. Table 2, fine-grained clusters used to generate the detection signatures of the 2011, 2012, and 2013 subsets). Left columns provide the index ranges. Right columns provide the percentage of clusters having similar indexes. With regard to

Index Range	**Percentage**
$0.98 \leq \mathcal{P}_c < 1.0$	65%
$0.96 \leq \mathcal{P}_c < 0.98$	15%
$0.94 \leq \mathcal{P}_c < 0.96$	8%
$0.92 \leq \mathcal{P}_c < 0.94$	7%
$0.00 \leq \mathcal{P}_c < 0.92$	6%

Table 3. Cluster precision index results

Index Range	**Percentage**
$0.6 \leq \mathcal{R}_c < 1.0$	4%
$0.4 \leq \mathcal{R}_c < 0.6$	6%
$0.2 \leq \mathcal{R}_c < 0.4$	21%
$0.04 \leq \mathcal{R}_c < 0.2$	14%
$0.00 \leq \mathcal{R}_c < 0.04$	55%

Table 4. Cluster recall index results

the cluster precision metric, we can observe that almost 94% of the fine-grained clusters held more than a 92% precision index. In other words, 92% of malware in each fine-grained cluster is properly classified in the correct malware family. This validates the accuracy of the clustering process of WebVisor. Concerning the cluster recall metric, 67% of the fine-grained clusters hold a recall index lower than 10%. This means that only 10% of malware were misclassified by WebVisor into other clusters. Most of the remaining 33% of the fine-grained clusters held a recall index ranging between 20% and 40%. A manual analysis of these clusters revealed that certain clusters included different versions of the same malware family (e.g., different versions of the `conficker` malware family). Some other clusters were also associated with generic ground truth labels such as `Heur` and `Agent`. These anomalies rather depend on the quality of the ground truth labels. Therefore, they should not be considered a weakness of the system.

3.2 Evaluation of the Detection Signatures

To verify the quality of the detection signatures generated by WebVisor, we evaluate them in terms of false positives and false negatives. We recall that in density-based clustering, a cluster represents an area that has a relatively higher density in the dataset, whereas the noise concept is represented as isolated objects that belong to sparse areas in the same dataset. DBScan [5], used by WebVisor to conduct the fine-grained clustering process, is a density-based algorithm that implements the aforementioned concept. It takes as input the minimum number of objects in a cluster and the maximum neighborhood radius between two objects. Although these parameters affect the total number of clusters, the latter is a result of the clustering process and is not required as input of the process. DBScan creates multiple clusters. Each cluster represents a dense area in the initial dataset. URLs belonging to sparse areas are grouped into a single noise cluster which is further discarded by WebVisor. Each dense cluster includes similar URLs that are shared among multiple variants of a malware family, and that characterize a specific C&C application. Therefore, and since WebVisor automatically discards noise into separated clusters, we also evaluate the corresponding noise clusters.

The coverage of the signatures is evaluated by analyzing their ability to detect malware communication not included in the experimental dataset. We separately processed the malware samples at our disposal by the year of collection. We tested each set of signatures against the input dataset, and the datasets collected in the following years. Due to space limitations, we discuss only the evaluation in terms of the 2011 traffic collection. Similar results were obtained with the other collections. Table 5 illustrates the distribution of URLs across the

URL Range	Percentage
0 to 2000	14%
2000 to 4000	37%
4000 to 8000	11%
more than 8000	38%

Table 5. Distribution of statistical clusters

Noise Rate	Percentage
less than 0.02	33%
0.02 to 0.1	30%
0.1 to 0.2	22%
more than 0.2	15%

Table 6. Noise rate distribution

27 statistical clusters. The left column represents the number of distinct URLs in each statistical cluster. The right column represents the percentage of clusters having similar number of URLs. We recall that the 2011 dataset (cf. Table 2) contains $75,398$ distinct malware samples. According to the ground truth labels, these samples were classified into 127 distinct families. We use the incremental k-means algorithm for statistical clustering. It starts from one centroid and incrementally adds new centroids when the distance between a new entry and all existing centroids exceeds an input threshold τ_h. Incremental k-means iterates over the input dataset until k - *which is the number of output clusters* - reaches its optimal value based on the value of τ_h.

The output of the statistical clustering, using an experimental threshold $\tau_h = 0.15$, includes 27 clusters. Almost 14% of clusters in table 5 contained less than two-thousand URLs, mostly including very short URLs such as '*/a/*', '*/2/*', and '*/?src=*`integer`'. These are irrelevant URLs that were later discarded as noise by the density-based clustering. In terms of outliers, 6 clusters were also discarded. These outliers contained more than twenty thousand URLs. Because of their small number, these clusters were manually analyzed. Almost all URLs in these clusters were associated with generic web operations, including URLs like '*/json?c=*`resolution`', and '*/addserver/www/*`...`'.

After the statistical clustering, URL enrichment and density-based clustering was applied to each statistical cluster. An overall number of 120 distinct fine-grained clusters (represented as detection signatures in Table 2) were generated. The processing of each statistical cluster during the density-based clustering led to a stable average of noise rate. We recall that noise here represents those URLs that were further classified during the density-based clustering, by DBScan, as noise clusters. Table 6 summarizes these results. The left column provides the rate of noise in each statistical cluster, and that were further discared by WebVisor. The right column provides the percentage of statistical clusters leading to similar noise rates. Few outlier clusters, mostly consisting of very small clusters or including a large number of URLs, contained noise rates exceeding 20%. The resulting 120 signatures were identified by WebVisor as being associated with URLs that carry true C&C activity. They only describe fine-grained clusters including URLs that have almost identical structure and semantics.

Detection Rate – We cross-validated the set of signatures against the initial malware dataset. A signature is considered to detect malware when it matches at least one URL during its dynamic analysis. Note that we would not expect 100% detection rate as WebVisor is grouping irrelevant URLs that belong to sparse areas in the dataset into noise clusters during the fine-grained clustering, and so it may mistakenly discard relevant C&C URLs during this process. Table 7 illustrates the detection rates that we obtained during the experiments. According to Table 2, the whole dataset was divided into three subsets (from year 2011 to 2013), according to the year at which malware samples were collected. A 10-fold experiment was applied, where a 10% of the samples from the initial dataset were repeatedly removed before building the detection signatures. Each set of signatures was matched against the samples collected during the corresponding year, as well as for the remainder years. The goal is to evaluate the ability of WebVisor for detecting malware that belongs to the families defined in the initial training dataset. The ability of the signatures to detect samples that were unknown at the time of generating such signatures was also evaluated. As a result, it was obtained that WebVisor achieves near 84% average detection rates when tested against the same year dataset. This means that the remaining 16% of undetected malware had their C&C activity mistakenly classified into noise clusters (discarded by WebVisor). The detection rate drops to near 60% for malware collected in the next year following the signatures generation, and to almost 20% in the third year. The decrease in detection rates is explained by new emerging malware families that use new C&C applications. To overcome such weakness, WebVisor can be continuously fed with streams of new malware

Signa-tures	Malware Dataset		
	2011	2012	2013
2011	87%	64%	21%
2012	*NA*	86%	57%
2013	*NA*	*NA*	81%

Table 7. Detection rates

Signature	Family
`POST /includes/inc/helps/[.*].php`	Zeus
`GET /logos[.]*.gif?[0-0,a-z]6=[0-9]*`	Sality
`GET /streamrotator/thumbs/[a-z]2/[0-9]*.jpg`	Srizbi
`GET /generate/software/?[A-Z]3RND=[0-9]*`	Zango

Table 8. Samples signatures generated by WebVisor

HTTP traffic. New malware families that appear in the wild would have their HTTP traffic processed by WebVisor to update its signatures database.

The experiment proves the ability of WebVisor to capture URL patterns that are shared among samples of the same malware family, and that characterize common features of their C&C applications. In fact, binary polymorphism modifies the malware signature but it does not affect the network behavior of malware, including the web toolkit that is shared among samples of the same family. While network obfuscation, including fake connections, attacks and connectivity checks, makes generic network signatures less efficient, WebVisor eliminates noise using density-based clustering. It discards noise into separate sparse clusters and builds detection signatures only for the main C&C connections which are shared among samples of the same malware family.

False Positives – To evaluate false positives, we collected one day of network activity, at March 2014, from a well protected corporate network. Terminals connected to this network are all equipped with updated antivirus software. Access to this network is possible only through firewall gateways and monitored using web proxies. Although we cannot rule out the possibility of few terminals being infected, these would be limited with respect to the large set of terminals being connected. Hence, we may still reasonably consider this traffic to include only benign web activity. We further developed a python script that extracts URLs and matches them against our signatures. We collected near 1.8 million distinct URLs, including both regular web activity and scheduled software updates, and that we tested against our entire set of 617 signatures. The collected dataset includes all distinct URLs that are triggered by up to $3,500$ active network terminals. It includes URLs towards thousands of distinct remote domains. Almost 9% of URLs were dedicated to Google search queries, while remaining URLs included regular browsing activity such as webmails, advertising, media websites, social networks and content downloads. Although our malware dataset is relatively old compared to the benign traffic at our disposal, this does not affect our experimental setup as we are only considering false positives. We consider all matching signatures to be false positives. In the end, 72 alerts out of the initial 1.8 million URLs were matched with only 21 distinct signatures. Hence, WebVisor achieved 0.004% false positives, and we only identified 3.4% *weak* signatures (i.e., signatures that triggered false positives during evaluation). We recall that a main property of our system is that WebVisor does not need to implement a pruning step in order to eliminate rogue signatures. This is a tedious step as it would require a large set of benign traffic, as well as an automated process to generate valid ground truth from the collected benign traffic.

3.3 Evaluation Against Live Internet Traffic

We tested WebVisor against two days of real live Internet traffic from a large ISP network. More than 150 GB of anonymized traffic, collected during September 2013, and including the entire network communications for near ten thousand distinct IP addresses, was analyzed. We extracted URLs using the same Python scripts that we used for the previous experiment, and we tested against our entire set of 617 signatures. Our system triggered 173 alerts, associated with 19 distinct signatures that were matching with 93 distinct IP addresses. Since the traffic at our disposal was few months old, we checked the domain names reputation using the domain search functionality on services like virusTotal, and searched for evidence on the Internet about the matching URLs. Unfortunately we could not check the status on the infected terminals since all traffic at our disposal was anonymized, and the ISP did not offer to contact infected clients in order to validate our findings.

We could not verify 95 alerts that were triggered by 15 detection signatures, including 9 weak signatures that triggered false positives in our previous experiment. Domains contacted through these URLs seem to be benign domains. We could not find signs of infection on the remote websites. Possibly these websites have been used temporarily or as stepstones through web or system vulnerabilities. Since we could not validate the exactness of these alerts, we considered them as false positives. In addition, the 4 signatures in Table 8 triggered almost 78 alerts, associated with 11 IP addresses. The first signature was matching with 3 IP addresses. The detected URLs were all associated with the domain name *marytraders.in*, which is identified by Zeus Tracker (cf. https://zeustracker.abuse.ch/) as a C&C domain. The second signature was triggered by two IP addresses (and associated with the `Sality-A` label in the dataset). The corresponding domain included pornographic content and is associated with botnet activity according to Google safe browsing. We also detected six other IP addresses that matched with the two remaining signatures. They were confirmed malware infections, validating the reliability of WebVisor.

3.4 Resilience to Malware Evasion

To evaluate the resilience of our system against noise, we trained WebVisor using a dataset that we obtained by adding random benign URLs to the malware dataset. We used for this purpose the traffic that we collected from the corporate network. We computed the average cluster precision and recall indexes for different values of Signal to Noise Ratio (SNR). Our main assumption is that malware can use fake benign URLs in order to evade our detection system. Therefore, we evaluate the ability of our system to discard fake URLs and keep only common C&C patterns as input to build detection signatures. As described in [21], malware may also trigger specific noise patterns in order to evade our system. However, there still need to be multiple samples of such malware in our dataset in order to interleave with our dense clusters. Although it is interesting to evaluate the resilience of our system against such specific evasion techniques, we focus in this paragraph only on random noise patterns. We plan in future work to conduct a deeper evaluation of WebVisor against targeted evasion techniques, taking into account the fact that malware herders may be aware of the process implemented by our system. The results of our experiments are illustrated with the ROC curve in Figure 1. WebVisor has an overall good resilience against noise. While its performance decreases at slow rate for decreasing values of SNR, WebVisor achieves almost 80% cluster precision for SNR values around 40%. Yet, we noticed a significant degradation of the quality of our malware families for noise rates exceeding 50%. The degradation of the precision index is mainly due to the threshold τ_h that we use for statistical clustering. Since we experimentally set the value of this threshold using our malware dataset, adding 50% benign URLs to this dataset alters its statistical consistency. WebVisor would thus classify URLs that are associated with similar C&C activities into different statistical clusters. While the degradation of malware clusters comes as a reasonable consequence to the increasing noise ratio, our system still achieves stable clustering results for up to 40% noise in our initial malware dataset. This is a main contribution of our system compared to other state of the art solutions where a pruning process is usually required in order to eliminate rogue signatures.

Another property of WebVisor is that it processes URL parameters using regular expressions that characterize all attributes shared between malware and its remote C&C applications. Although WebVisor captures specific obfuscation mechanisms such as encodings (e.g., base64) and hash functions (MD5 or SHA-1), it would be unable to correctly build expressions for URLs that have their entire set of parameters encrypted within a single chain of characters. It associates these parameters with the 'No_type' label, and handles them as string values. Note that the density-based clustering process may still identify shared patterns in case they appear in all encrypted URLs for a given malware family, as previously shown in [14]. However, malware that fully encrypts its URLs, with no shared patterns between URLs of the same family, would be unlikely to be detected using our system, and so it is more likely to be dropped into noise clusters. This is a common limitation to all network-based malware detection systems, as long as they are unable to access the content of malware communications with its remote C&C applications.

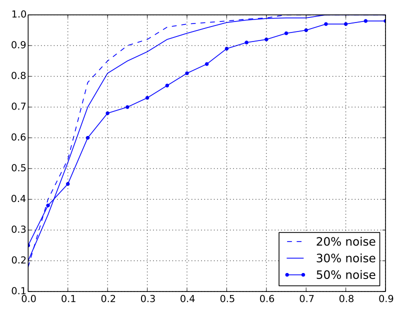

Fig. 1. ROC curve and SNR ratios

4 Related Work

Multiple contributions in the literature propose the use of supervised machine processing to classify malware activities and build behavioral models for detection. Solutions in this category include tools such as Firma [23], PhishDef [13], and JackStraws [7]. As opposed to them, WebVisor uses an unsupervised clustering approach. It does not require an initial set of benign network traffic to train the classifier prior to generating detection signatures. This is an important issue, since obtaining valid ground truth labels for benign network activity is a tedious task that cannot be easily automated.

Similarly to WebVisor, Perdisci et al. propose in [20] the use of unsupervised clustering processing to analyze malware HTTP connections and build detection signatures. As opposed to our work, their approach classifies malware families using all kinds of HTTP requests triggered by malware executed in a sandbox. This includes not only C&C traffic, but also any other kinds of malware activity such as benign connectivity checks. Therefore, the approach is not robust against malware obfuscation, since it may reduce the accuracy of the resulting detection signatures. To handle the issue, i.e., to avoid a high rate of false positives, Perdisci et al.'s approach requires to carefully verify all those generated signatures against benign web activity. This is a tedious and error prone task. First, obtaining a large-scale representative ground truth dataset of benign network traffic to prune out unnecessary signatures is very challenging. Second, it makes infeasible to automatically build and deploy effective detection signatures. WebVisor offers an alternative approach to classify malware using only relevant C&C traffic. Although it is difficult to detect C&C connections during a single malware analysis, the C&C activity becomes more apparent when observing a larger set of malware. Furthermore, WebVisor automatically discards noise and identifies common C&C requests used by variants of the same malware family.

5 Conclusion

We have presented WebVisor, an automated tool for the generation of malware detection signatures. WebVisor targets HTTP malware belonging to the same family, e.g., malware that uses the same C&C applications and equivalent sets of URLs. We have outlined the main design properties underlying WebVisor and evaluated a software prototype against real-world malware datasets. Our experiments verify the capability of WebVisor at identifying the main and invariant features of malware C&C activity.

Acknowledgments — The authors thank A. Gupta, S. Martinez-Bea and A. Verez for all their work on the implementation of the WebVisor prototype.

References

1. 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*, pages 24–40, 2012.
2. U. Bayer, C. Kruegel, and E. Kirda. TTAnalyze: A Tool for Analyzing Malware. In *15th EICAR Conference*, 2006.

3. P. Bieganski, J. Riedl, J. Cartis, and E. Retzel. Generalized suffix trees for biological sequence data: applications and implementation. In *Proc. of International Conference on System Sciences*, volume 5, pages 35–44, 1994.
4. Z. Bu, P. Bueno, R. Kashyap, and A. Wosotowsky. The new era of botnets. In *White paper from McAfee*, 2010.
5. M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. of KDD*, 1996.
6. G. Gu, R. Perdisci, J. Zhang, and W. Lee. Botminer: Clustering analysis of network traffic for protocol and structure independent botnet detection. In *Proc. of IEEE SSP*, 2008.
7. G. Jacob, R. Hund, C. Kruegel, and T. Holz. JackStraws: Picking Command and Control Connections from Bot Traffic. In *USENIX Security*, 2011.
8. M. A. Jaro. Advances in record-linkage methodology as applied to matching the 1985 census of tampa, florida. In *Journal of the American Statistical Association*, volume 4, 1989.
9. E. J. Kartaltepe, J. A. Morales, S. Xu, and R. Sandhu. Social network-based botnet command-and-control: emerging threats and countermeasures. In *Applied Cryptography and Network Security*, pages 511–528, 2010.
10. N. Kheir. Behavioral classification and detection of malware through http user agent anomalies. *Journal of Information Security and Applications*, 2013.
11. N. Kheir and X. Han. PeerViewer: Behavioral Tracking and Classification of P2P Malware. In *Proc. of CSS*, pages 282–298, 2013.
12. N. Kheir and C. Wolley. BotSuer: Suing Stealthy P2P Bots in Network Traffic Through Netflow Analysis. In *Proc. of CANS*, 2013.
13. A. Le, A. Markopoulou, and M. Faloutsos. Phishdef: URL names say it all. In *IEEE INFOCOM*, 2011.
14. Z. Li, M. Sanghi, Y. Chen, M.-Y. Kao, and B. Chavez. Hamsa: fast signature generation for zero-day polymorphic worms with provable attack resilience. In *Proc. of IEEE SSP*, 2006.
15. S. Nagaraja, P. Mittal, C.-Y. Hong, M. Caesar, and N. Borisov. BotGrep: Finding P2P Bots with Structured Graph Analysis. In *USENIX Security*, 2010.
16. A. Nappa, M. Z. Rafique, and J. Caballero. Driving in the Cloud: An Analysis of Drive-by Download Operations and Abuse Reporting. Proc. of DIMVA, 2013.
17. M. Neugschwandtner, P. M. Comparetti, and C. Platzer. Detecting malware's failover C&C strategies with squeeze. In *Proc. of ACSAC*, 2011.
18. J. Newsome, B. Karp, and D. Song. Polygraph: Automatically generating signatures for polymorphic worms. In *Proc. of IEEE SSP*, pages 226–241. IEEE, 2005.
19. J. Oberheide, E. Cooke, and F. Jahanian. CloudAV: N-Version antivirus in the network cloud. In *USENIX Security*, 2008.
20. R. Perdisci, D. Ariu, and G. Giacinto. Scalable Fine-Grained Behavioral Clustering of HTTP-Based Malware. In *Special Issue on Botnet Activity: Analysis, Detection and Shutdown*, volume 57, pages 487–500, 2013.
21. R. Perdisci, D. Dagon, W. Lee, P. Fogla, and M. Sharif. Misleading Worm Signature Generators Using Deliberate Noise Injection. In *Proc. of IEEE SSP*, 2006.
22. D. T. Pham, S. S. Dimov, and C. D. Nguyen. An incremental K-means algorithm. In *Journal of Mechanical Engineering Science*, volume 218, pages 783–795, 2004.
23. Z. Rafique and J. Caballero. FIRMA: Malware Clustering and Network Signature Generation with Mixed Network Behaviors. In *Proc. of RAID*, 2013.
24. K. Rieck, T. Holz, C. Willems, P. Dssel, and P. Laskov. Learning and classification of malware behavior. In *Proc. of DIMVA*, 2008.