Countering Malicious Processes with Process-DNS Association

Suphannee Sivakorn*, Kangkook Jee†, Yixin Sun‡, Lauri Korts-Pärn§, Zhichun Li†, Cristian Lumezanu†, Zhenyu Wu†, Lu-An Tang†, Ding Li†

*Columbia University, ‡Princeton University, †§NEC Laboratories America, Inc.

*suphannee@cs.columbia.edu, ‡yixins@cs.princeton.edu, §lauri@cyberdefense.jp,
†{kjee, zhichun, lume, adamwu, ltang, dingli}@nec-labs.com

***Abstract*—Modern malware and cyber attacks depend heavily on DNS services to make their campaigns reliable and difficult to track. Monitoring network DNS activities and blocking suspicious domains have been proven an effective technique in countering such attacks. However, recent successful campaigns reveal that attackers adapt by using seemingly benign domains and public web storage services to hide malicious activity. Also, the recent support for encrypted DNS queries provides attacker easier means to hide malicious traffic from network-based DNS monitoring.**

We propose PDNS, an end-point DNS monitoring system based on DNS sensor deployed at each host in a network, along with a centralized backend analysis server. To detect such attacks, PDNS expands the monitored DNS activity context and examines process context which triggered that activity. Specifically, each deployed PDNS sensor matches domain name and the IP address related to the DNS query with process ID, binary signature, loaded DLLs, and code signing information of the program that initiated it. We evaluate PDNS on a DNS activity dataset collected from 126 enterprise hosts and with data from multiple malware sources. Using ML Classifiers including DNN, our results outperform most previous works with high detection accuracy: a true positive rate at 98.55% and a low false positive rate at 0.03%.

I. Introduction

The Domain Name System (DNS) is one of the fundamental services of the Internet. DNS translates domain names into IP addresses, thereby isolating the naming scheme from the effects of underlying network changes. This makes DNS a critical attack vector, as it allows malicious users to hide their network footprint behind short-lived domain names (*e.g.,* command and control, fast-flux [56], [3]). How to distinguish legitimate domains from those involved in malicious activities (*e.g.,* fast-flux networks, bot networks, DGA domains, spam networks) has been a constant focus in the security research community [5], [6], [7], [13], [46], [22].

Most research on DNS-based malicious activity detection focuses on finding victim hosts by analyzing and modeling various properties of the DNS traffic, such as the diversity of resolved IPs [5], [6], [7], [13], [46], geographic information [7], [46], name string structure [7], [13] and DNS Time-to-live (TTL) values [13], [46]. However, recent examples of successful attacks ([24], [23]) reveal that attackers obfuscate the communication channel indistinguishable at the network level by incorporating legitimate web and cloud storage services to hide their command and control (C&C) connections. For example, the C&C channels and drop sites of the recent malware campaigns such as HammerToss [24], and Poshspy [23] occur only on legitimate services (*e.g.,* Twitter, GitHub, Dropbox, Photobucket). These seemingly imitating normal user behavior makes these malware hard to be identified by traditional DNS traffic-based detection systems. Also, due to the recent rise of deploying encryption over DNS queries (*e.g.,* DNS-over-TLS [1], DNS-over-HTTPS [2]), the attackers now have another way to establish a channel to their C&C bypassing the existing network-level detection systems. To detect such attacks, one would need to expand the monitored context around the DNS activity of the *host* and examine the *processes* that initiate such activity.

In this paper, we propose PDNS, the first large-scale process-level DNS monitoring system. PDNS consists of several DNS sensors, deployed on end-hosts, and a back-end analytic server. The DNS sensors monitor and capture information about the DNS activity of hosts along with associated programs and processes. The back-end server collects data from sensors and builds machine learning models to detect malicious DNS behavior and its associated process.

To build a model to detect malicious behavior, PDNS collects and analyzes two types of DNS activity features. On one hand, it collects established and previously proposed *network-based* DNS features, such as IP and domain location diversity [7], [46], [37], [6], domain string entropy and length [7], [13], resolve failure percentage [7], [13], and domain registered and renewed periods [22], [37]. On the other hand, it introduces new *process-based* features that characterize the relationship between processes and DNS activities, such as the number of requested domains per process, the number of resolved IPs per process, code signing information and domain registrants.

An end-host approach to DNS monitoring has the following benefits: *(1)* PDNS supports a more precise analysis and detection by narrowing the scope from host-level to process-level. Because DNS sensors can access a system's internal features, such as loaded DLLs or binary signature information, they can help model activities that are indiscernible from network-only monitoring. *(2)* Immediately determines the malfeasant process associated with suspicious behavior. While some information (*e.g.,* domains, resolved IPs) can be collected over the network-level monitoring, our DNS sensor collects in a process-centric manner which allows us

Network and Distributed Systems Security (NDSS) Symposium 2019
24-27 February 2019, San Diego, CA, USA
ISBN 1-891562-55-X
https://dx.doi.org/10.14722/ndss.2019.23012
www.ndss-symposium.org

to identify and associate to a particular malicious process and in turn enhance our detection in the long run. *(3)* Improves visibility over encrypted DNS. DNS-over-TLS/HTTPS [1], [2] are designed to prevent man-in-the-middle attacks on DNS traffics. However, it also hides DNS traffics from traditional network-based monitoring solutions. Our end-host approach still monitors malware that queries domain names using DNS-over-TLS/HTTPS.

To demonstrate the effectiveness of PDNS, we deployed end-point DNS sensors to 126 Windows hosts on an enterprise network. We collected over 131M DNS records and their responsible process information between February and August 2017. In addition, we gathered malware samples from different sources at different times. Total 19k collected malware samples were executed in a sandboxed environment [17] to capture their DNS and process activities. Our PDNS back-end system runs several classifiers, such as Random Forrest, Logistic Regression, K-Nearest Neighbors, Support Vector Classification, and Deep Neural Networks, to build a detection model for benign and malicious process behaviors. Our models can accurately distinguish benign from malware processes with a maximum true positive rate of 98.55% and false positive rate of 0.03%. Furthermore, PDNS demonstrates its capability in identifying stealthy attacks which are undetected by previous detection methods, as their malicious activities (including C&C and drop sites) only occur on legitimate domains.

We bring the following contributions.

Firstly, we propose and design PDNS, a novel end-point DNS monitoring system that immediately detects the malicious process inside the compromised host by analyzing DNS along with their associated program information. To the best of our knowledge, our work is the first of its kind that automatically associates an individual process and its DNS queries on an enterprise scale. PDNS also explores an extensive set of network- and host-based features. PDNS re-interprets the existing features by narrowing down its analysis scope to *process-level*.

Secondly, PDNS proposes new features by integrating existing DNS and host-based features that can reveal hidden relationships between (seemingly legitimate) DNS requests and the processes that trigger them. While newly introduced integrated features reduce the false positive rate by 46.7% (Section V-G2), in overall, PDNS approach accurately classifies benign programs from malware programs with a true positive rate of 98.55% and false negative rate of 0.03% (Section V-D).

Finally, PDNS improves the visibility of security monitoring to support detection of stealth attacks equipped with counter forensic techniques. As the first step, PDNS approach successfully classified HammerToss attack (Section VI-A). We further explore the design possibilities of building robust features leveraging PDNS data collection.

II. Motivation

To motivate the need for a combined network- and process-based DNS activity analysis, we present exemplary program-DNS profiles for a well-known program (`Skype.exe`) and a malware example (`Mal.exe`). To understand them, we first define two concepts used throughout the paper.

- **Program** represents a software or an application. A program binary or a group of binaries can represent a program. For instance, the `Skype.exe` program can have different binary versions that fulfill the same functionality.
- **Process** defines a runtime instance of a program. A program can have multiple processes with different program identifiers (PID).

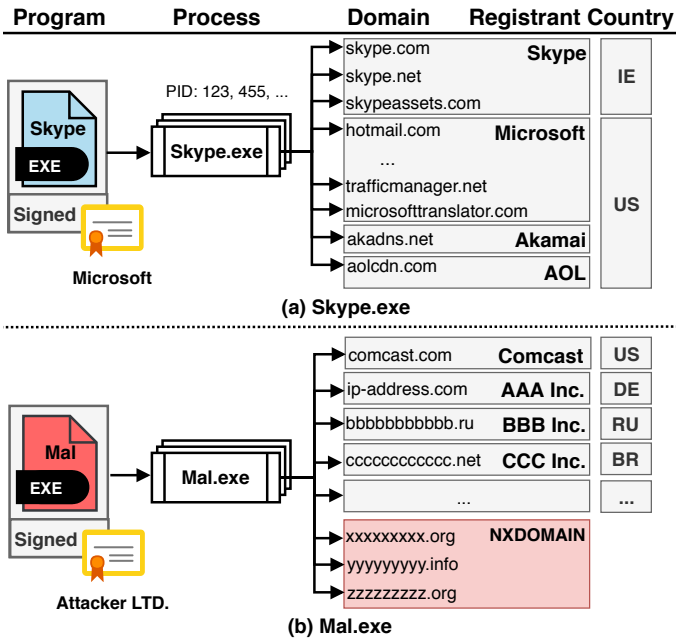


Fig. 1: Program-DNS profiles of benign program (`Skype.exe`) and malware (`Mal.exe`) captured by PDNS.

Program-DNS Profile for Skype.exe. Figure 1(a) presents the program-DNS profile for `Skype.exe` captured by PDNS. `Skype.exe` *program* instantiates itself into multiple *processes* by loading the executable binary. In order to use the service, each client process connects to the skype's domains (and related domains e.g., Microsoft's for signing in), which in turn requires making DNS queries to obtain IP addresses. PDNS collects data includes system internal sources for process-related information and external sources for network-based information. By correlating the network- and process-based information, we can build more accurate program profiles.

Throughout the monitoring period, we observed a high correlation between program and their DNS queries. For example, in the case of `Skype.exe`, all collected 29K DNS queries from `Skype.exe` processes are corresponding to only 19 distinct domains, where 17 of which are registered by Microsoft or Skype. More interestingly, they are registered under only four organizations and only registered to United States (US), except Skype's domains registered to Ireland (IE). Program-DNS profile also indicates the correlation between the software publisher of the program and the registrant of the domains accessed by the program. The code signing information for `Skype.exe` indicates the software publisher as "Skype Software Sarl". Correspondingly, the DNS WHOIS records confirm that majority portions of DNS queries have either "Skype" or "Microsoft" as registrants which are the same company in the end. Finally, we also observed that these processes load GUI and user-interaction related DLLs. This is reasonable as the program operates with a graphical interface and allows user interactions.

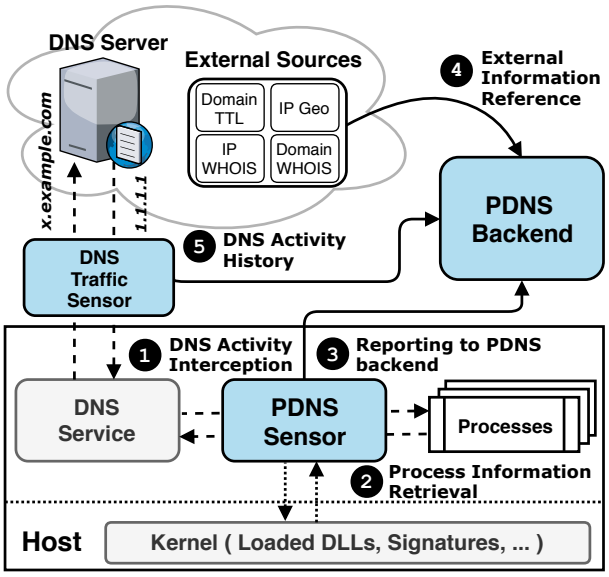


Fig. 2: PDNS data collection. The figure depicts delegated DNS service model in Windows and the latest version of Linux systems.

Program-DNS Profile for Mal.exe. Figure 1(b) presents the program-DNS profile for a malware (`Mal.exe`) captured by PDNS. The program visited around 20 distinct domains. Given that the malware is executed only once from our sandbox environment, domains that the program visited are very diverse. All domains are registered under different organizations from diverse geo-locations. Some domains are Non-existence (NXDOMAIN). Lastly, although the malware is properly signed, the code signer hardly matches the domains' registrants. The malware mostly runs as a background process without loading GUI and user interaction DLLs.

While there are approaches that detect malicious activities exclusively from process-based information [43], [44] or network-based information [45], [59], [20], [19], [48], we propose to use and correlate both types of information altogether to improve the malicious behavior detection accuracy.

III. System Overview

This section sketches the architecture of PDNS. We first describe our data collection from end-hosts to the backend that aggregates and analyzes program-DNS inputs. We then describe our operational workflow of PDNS that builds models from real-world DNS inputs to detect malicious activities.

A. PDNS Data Collection

PDNS collects DNS activity data at both process and network level. Figure 2 depicts the overall flow of PDNS data collection. The primary data collection module, PDNS sensor, is installed inside a host and monitors DNS queries for all processes. The sensor intercepts DNS activities for each process (Figure 2 ❶). It also obtains each process information from the kernel, such as loaded dynamic-link libraries and binary signatures (Figure 2 ❷), before reporting to our DNS backend (Figure 2 ❸). The PDNS backend aggregates process-level DNS activity reports from all PDNS sensors installed on each host and further extends the network and DNS related information by referring to other information sources for DNS WHOIS, IP WHOIS, IP Geo-location, *etc.* (Figure 2 ❹). PDNS also collects DNS activity history from the local DNS server and cross-checks with the DNS records reported to PDNS backend (Figure 2 ❺). This final component acts as a fail-safe in case the attacker takes over the end-host and launches an attack against the PDNS sensor.

There are two key technical challenges of DNS sensor implementation: *(1)* how to capture all DNS activities from the system, and *(2)* how to associate each DNS activity with its originating process. We address the above challenges using an ensemble of data collection mechanisms. Programs usually perform name resolutions via one of the two channels: *(1)* direct UDP communication with the name server, or *(2)* DNS task delegation to a system service. To cover the direct channel, we leverage network tracing facilities – specifically, the HONE [55] open source project, which provides kernel modules that not only capture system-wide network traffic, but also associate every packet with corresponding process ID. However, network tracing cannot provide useful information when programs utilize the delegated channel, because all captured traffic will be associated with the DNS service daemon, regardless of the original requesting process. To cover the delegated service model, we either resort to intercepting the inter-process communication between the name resolution service and requesting processes, or, if available, digest detailed diagnostic logs produced by the service.

We implement and deploy prototype PDNS sensors for both Linux and Windows operating systems. For Linux systems, we found that most hosts in our target enterprise environment do not utilize the system internal service for DNS caching, every process directly queries to outside DNS servers for name resolutions. Hence our Linux sensor uses network tracing (HONE) as its sole event source. For Windows systems, the situation is the inverse – the OS provides a dedicated *DNS Client Service* [39] which most programs take advantage of, and rarely any program performs direct name resolution. On Windows 8 and below, we intercept and log LRPC communications between querying program and *DNS Client Service*; On Windows 8.1 and above, similar information can be obtained more easily from the *DNS Client Service Event Tracing for Windows* [40].

Recently, we observe two notable changes in how systems configure DNS service and how programs make DNS queries. The latest Linux distributions switch to dedicate system services for DNS caching as their default configuration. Programs have options to configure themselves make direct DNS query for the privacy [1], [2]. To achieve complete monitoring coverage, PDNS sensor should include both channels.

B. PDNS Training and Detection

This section illustrates the PDNS training and detection workflow. Firstly, we deploy PDNS sensors to an environment that we want to protect. PDNS sensor from each end-host monitors and records everyday DNS activities of all programs. The inputs accumulate to establish benign dataset in the backend. In case it is uncertain that all captured activities are indeed benign, we take additional steps to sanitize data by antivirus scanning, malware database cross-checking, or by manually inspecting the suspicious processes. For comparative analysis, we set up a constrained environment using the Cuckoo sandbox [17], and capture malicious program and DNS activities. We follow a number of common guidelines [30], [25], [18], [8]

to avoid triggering *anti-VM* techniques employed in various malware. For this analysis, we gathered total 131k malware samples from different sources at different periods of time. After the careful process of filtering, we retained 19k samples for the training and testing. We detail the collection and filtering process of malware dataset in Section V-B.

The resulting benign and malicious datasets are labeled and fed into our *training module*, which performs feature extraction and supervised learning using several classifiers (*e.g.,* Random Forest, Logistic Regression, Deep Neural Networks) (Section V-C). We deploy the model with the best training results for real-time detection. We implement the learning module with an open-source machine learning library, Scikit-learn [50]. The library supports various classifiers and other machine learning algorithms. We also leverage their rich functionalities: cross-validation, hyper-parameters tuning, model exporting, and classification metrics in our model training.

Finally, while the detection module can also be deployed at the host-level, we opt for the backend deployment (centralized-based). In addition to the convenience in regularly updating, and retraining the model, this design also prevents the detection module from being tampered when the local machine is compromised as well as allowing the host's DNS activity to be cross-checked with the local DNS server (Section III-C). The detection is designed to provide real-time classification results based on various classifiers including the deep-learning based classification model. We describe the detail of our training in Section V-C.

C. Threat Model

As for the attacker's capabilities, we assume that the attacker can compromise the individual host. But the organization's IT infrastructure, including PDNS backend and network services, is well guarded and remains in a trust domain. As mentioned, our execution model does not rule out the possibility of the attackers tampering PDNS sensors from one or more hosts. To counter this, PDNS cross-checks DNS records from PDNS backend and the DNS query history obtained from local DNS servers for data consistency (Figure 2 ❺). The attacker may also attempt to redirect DNS queries to the outside by using external DNS servers, and this can still be hindered by monitoring TCP/UDP 53 traffic going outside the network.

IV. Feature Selection

In this section, we identify key features and describe the intuition behind their selection. In total, PDNS uses 61 features to build program DNS behavior models. Table I presents an overview of features and Appendix A includes more details. We classify the features into three categories, based on their sources. *(1) Network-based features* come from external sources of data, such as WHOIS records and IP locations from DNS requests. *(2) Process-based features* characterize the relationship between processes' attributes and their DNS behaviors. Although most network features have been proposed in previous works [13], [7], [5], [6], [37], [63], [22], [46], our network-features are associated with its responsible processes to interpret DNS behavior from the individual processes' perspective (Section IV-A). Finally, *(3) Integrated features* combine information from multiple sources to define meaningful correlations between processes and their network activities.

Main Source	Data Source	Feature Category	# Feats	Related Works
Network-based (External sources)	WHOIS	Domain Duration	12	[22][37][63]
		Domain Registrant	5	[37]
		Location	2	[37][46]
	WHOIS, IP Location	AS Number	4	[5][6][46]
		Location	2	[6][46][37]
	Authoritative Nameserver	Domain TTL	3	[13][46]
Process-based (Internal sources)	DNS Activity, Code Signing, OS Kernel	Domain and Hostname	22	[5][6][7][46][13]
		Domain Resolve FR	1	[13]
		Code Signing	1	[44][43]
		Loaded DLLs	3	[17]
Integrated	WHOIS, IP Location, Code Signing	Location	2	
		Publisher and Registrant	4	

Feats: The number of features derived from each feature category.

TABLE I: Overview of PDNS features (in total 61 features). The entire list of features is presented in Table IV (Appendix A).

We first describe terminologies to interpret features which we evaluate them in a process-centric manner. Next, we explain how we transform these features (feature transformations) to fit our ML classification.

A. PDNS Feature Presentation

1) Terminologies: We define new concepts that capture and present DNS activities in the context of individual process. Given a process p running on a host, we define:

- **RHNs (Requested Hostnames)** is the set of all hostnames requested by p and captured by PDNS sensor. A hostname is a name string in a DNS query. Hostnames can be names of internal hosts (*e.g., print*, *wpad*) or fully qualified domain names (FQDN) that include all labels (subdomains) (*e.g., www.a.example.com*).
- **RDNs (Requested Domains)** is the set of all domain names requested by p and captured from PDNS sensor. The domain name represents a label adjacent to the top-level domain (TLD) of FQDN *e.g., example.com*, *example.co.uk*. To handle TLDs with two labels (*e.g., co.uk*) and to classify them correctly as TLD, we cross-check with Mozilla's public suffix list [42] which maintains the current list of public suffix domains. The definition of **RDNs** is similar to the *2TLD* feature propose by Antonakakis *et al.* [5].
- **RIPs (Resolved IP Addresses)** is the set of all IPs resolved from hostnames in DNS queries sent by p.

2) Feature Transformation: We apply various transformations to the values of certain features to better express them as inputs for the ML classifiers.

- **Numerical.** We extract statistical measures of average, median, and standard deviation (SD) for the numerical data we collect (*e.g., Registered Duration*, *Domain TTL*). For instance, when we process the *Registered Duration* feature of a given process p, we calculate statistical measures for all *RDNs* requested by p.
- **Frequency.** We monitor the occurrences of various events (*e.g., number of domains*, *DNS requests*) and express the results with three frequency metrics: the number of all events, the number of *distinct* events, and the ratio between the numbers of distinct events and all events (*distinct ratio*). Because processes have different lifetimes, our frequency-based features cannot always be compared to each other

directly. For example, long-lived processes might have many but sparse DNS requests, while shorter-lived processes might have the same number of requests, but occurring more frequently. We represent each frequency-based feature in two ways: the frequency for the entire lifetime of the process and the average frequency over one-minute intervals.

- **String.** We translate string-based data (*e.g.,* domain name, registrant strings) into a numerical representation, before extracting statistical measures. For example, to represent our hostname structure, we compute the length and entropy of characters in the hostname string. Then, we calculate the average, median, and SD of the length and entropy of RHNs. Some features (*e.g.,* domain registrant and software publisher) require string matching; we compute the string matching score as described in Section IV-D.
- **Category.** Part of our data collection belongs to certain classes or binary *e.g.,* whether the library is loaded or whether the code is signed. We encode each class using one hot encoding vector. To illustrate, in the *Loaded DLLs* features, we use the value '1' to represent the library category that is loaded by the process and '0' to represent not loaded. Similarly, the *Code Signing* feature also use '1' or '0' to represent code signed and verification status.

B. Network-based Features

Given DNS activities, we collect network-based features from one of three external sources: domain WHOIS, IP WHOIS, and domain authoritative nameserver list. We interpret these features in a process-centric manner in order to achieve a better understanding of the association between processes and their DNS activities.

1) The Domain WHOIS Record: maintains information about registered domain names: registrant, registered country, nameservers, registered date, updated date, expiration date[1]. Previous research explored the WHOIS information and its role in classifying malicious domains [22], URL [37], phishing emails and sites [63]. In the case where domain name's WHOIS record is not found, PDNS assigns -1 value (which indicates no data) to all related features. PDNS includes the following WHOIS items to its feature set:

Domain Duration features are derived from domain registered and renewed durations.

Registered Duration is the time difference between the domain's registered time and the current time. This feature reflects the freshness of a domain. Malicious domains tend to be fresher due to frequent blocking and re-generation cycles [22], [37]. We observe this pattern across our datasets, where overall, processes in our benign dataset have on an average registered duration of 6,266 days (17.2 years), while the processes in our malicious dataset have 3,181 days (8.7 years).

Renewed Duration is the time difference between the domain's updated time and its expiration time (initially introduced by Ma *et al.* [37]). While benign services register their domains for a long period, malicious domains prefer to register for a shorter duration due to the lower cost. We also observe this behavior in our datasets, specifically, the majority of malicious processes have shortened domain renew duration (Figure 3(a)).

Domain Registrant is the organization who registered the domain. We observed that the domains requested by a benign process often generally belong to a smaller number of registrant organizations than those requested by malicious processes. Figure 1 presents DNS requests captured from `Skype.exe` processes on our enterprise deployment. Notice that *only* four external domain registrants are associated with 29k DNS requests from 19 unique domains. In our datasets, malicious processes have on average 2.5 times more registrants than benign processes.

Domain Location is the location information of the organization registering the domain. We obtain location information for a domain by analyzing the country code (CC) for each WHOIS registrant. If the CC does not exist, then we refer to TLD's CC. This information helps PDNS in constructing the geolocational profile of a process. Overall, processes in our benign dataset have on average distinct country ratio at 0.19, lower than the malicious processes at 0.53. This confirms our assumption on the variation of domain country can indicate the suspiciousness of malicious processes.

2) IP WHOIS Record and IP Location: records maintain registration and location information about each IP. We generate the following features:

AS Number is the number of the autonomous system to which the IP belongs. Legitimate processes tend to have fewer distinct AS numbers than malicious process due to their more deterministic patterns for IP connections [5], [6], [46]. In particular, we found that our malicious dataset has on average roughly twice higher distinct AS ratio at 0.41 compared to our benign dataset at 0.28.

IP Location represents the country of the organization registering the IP. Given a process p, we obtain geolocation information by referring to IP WHOIS records of IP in the process's RIPs When the information is not available, we refer to the country field in IP geolocation record from IP geolocation services [26], [31].

3) The Authoritative Nameserver: holds the DNS records of a name. We generate the domain Time To Live (TTL) features as follows.

Domain TTL specifies how long each domain should be cached by a local DNS server. As Bilge *et al.* [13] pointed out, low TTL values allow malicious domains to better resist DNS blacklisting and takedowns. A Fast-flux network is an example of an attack where the set of resolved IPs changed rapidly [56], [4].

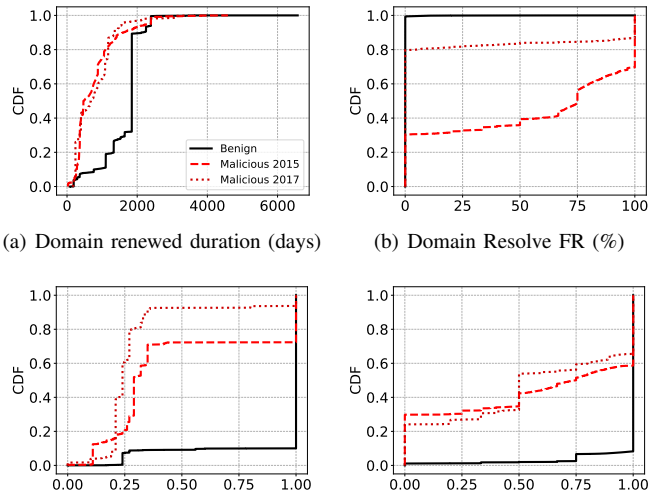
C. Process-based Features

To monitor DNS activities with improved visibility, PDNS first captures DNS activities and the responsible processes and then probes internal system resources to collect more process-related information.

1) DNS Activity: features characterize each process's external DNS behavior.

Domain and Hostname features characterize the domains and hostnames requested by each process. We first generate

[1]Our WHOIS features only rely on public (privacy-safe) information. We exclude sensitive information (*e.g.,* name, and email) to comply with the new ICANN GDPR Specification (https://www.icann.org/dataprotectionprivacy)



(a) Domain renewed duration (days)

(b) Domain Resolve FR (%)

(c) Software publisher and domain registrant similarity score

(d) IP and Domain geo-correlation score

Fig. 3: Examples of CDF of features derived from different activities and properties from benign and malicious processes.

length and entropy values for each domain (RDNs) and hostname (RHNs) and then compute statistical measures, generating overall 22 derived features. The domain entropy features are useful in detecting DGA-like DNS activities as shown by Antonakakis *et al.* [7].

Domain Resolve Failure Rate (RF) has been used in network-level detection [37]. Here, we use it in the context of process-level analysis. Given a process p, PDNS computes failed DNS queries over all DNS queries made by p. Figure 3(b) plots the CDF graph of resolve failure rate from our datasets. The feature clearly differentiates the benign and malicious datasets. Among malware datasets, unfiltered (2015-) malware dataset shows a higher failure rate than unfiltered one (2017-). Only 0.78% of processes in our enterprise (benign) dataset has a non-zero resolve failure rate higher than 0%. On the other hand, over 91% of processes from malware datasets of 2015 and 2017 show non-zero failure rate.

2) Process Information: features characterize the system behavior of a process. The process-DNS relation enables PDNS to probe the internal system sources and gather useful information in restoring context around DNS activity. PDNS includes features about Windows code signing scheme[2] and libraries loaded at runtime.

Code Signing features include *(1) the process (software) signed status*, and *(2) the process (software) publisher*. This information is located in the subject name or common name of the software's certificate. While code signing status on Windows itself cannot guarantee that the binary is safe to run, given that users can ignore the warnings or that malicious software can fake the certificate [34], our study shows that most unwanted software collected is not signed. Essentially, we trace executable file of a process and retrieve its signed status. We then assign a category to each status: *unsigned*, *signed and verified*, and *signed but unverified*. To this end, we found that 91% of processes in our benign dataset are signed and correctly verified. Only 0.006% are signed but unverified. On the other hand, 57-63% of processes in our malicious datasets are signed and 0.12-0.16% of processes are signed but unverified.

Loaded DLLs features characterize the nature of the process and its expected DNS query patterns. We first attempt to categorize processes into three groups based on the dynamic-link libraries (DLL) loaded at runtime: *user interactive*, *command-line*, and *non-interaction* processes.

User-interactive process: We observed that a process with a high volume of distinct domain queries often includes the graphic user interface (GUI) and web communication (WebComm) DLLs as they allow users to have control over destination hostnames. Examples of such processes are interactive GUI applications (*e.g.,* firefox.exe, chrome.exe, outlook.exe). Malicious processes often request a high distinct number of domains queries without having any user interactions.

Command-line process: A process with only User Interaction (UI) DLLs (not including GUI and WebComm) often results to be a command-line process *e.g.,* `svchost.exe`, `python.exe`, `lucoms~1.exe` (Symantec antivirus updater), `jusched.exe` (java updater). Processes in this category are likely to have a lower number of distinct domain requested compared to UI processes.

Non-interactive (service) process: Processes that do not belong to either of the above two categories often visit a specific set of domains. For instance, `taskhost.exe` goes only to Microsoft or internal domains. If a process queries a high number of distinct domains but loads neither DLLs for user interactions nor command line control, it is likely to be a malicious process.

D. Integrated Features

All features discussed so far are collected from a single source. We now combine these features to find a new correlation and obtain more powerful, multi-source features.

Software Publisher and Domain Registrant. As malware can abuse the trust process when obtaining certificates [34], relying solely on code signing information might mislead the ML classifier. We introduce a new feature by combining domain registrants from Domain WHOIS record and software publisher information from the process code signing information. The majority of processes in benign dataset requests DNS queries whose domain registrants have connections to software publishers, say owner domains. Figure 1 illustrates an example of the strong association between publisher (signed by "Skype Software Sarl") and the majority of domains (skype.com, skype.net, and skypeassets.com) extracted from DNS queries.

While the software controlled by users (*e.g.,* web browsers) is less likely to preserve the publisher-to-registrant correlation, we observed a stronger association during the software start period. Figure 4 illustrates an example case for Firefox, which compares the number of cumulative DNS queries captured during the entire runtime and during the start period (first 120 seconds, depicted in the center). The portion of DNS queries to "Mozilla Corporation" is higher during the start period.

As the software publisher and domains registrant names may not of exact matches, (*e.g.,* Microsoft Inc, and Microsoft Corporation), we apply Fuzzy String Matching [51] based on

[2]Code signing feature is specific to Windows systems and is not a common security practice for Linux systems.

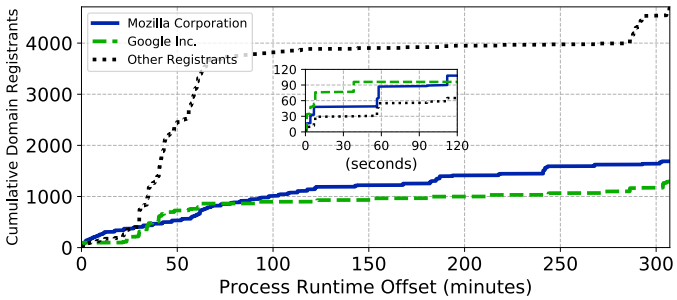


Fig. 4: Majority of DNS queries of Firefox.exe depend on user input. However, during the start time (first 120 seconds, in the center), the Firefox processes on average have a higher queries to domains registered under Mozilla, which is also their software publisher.

Levenshtein distance and string tokenization comparison when comparing two strings. We propose four new features which are derived by computing two string matching scores for both *(1)* the start period and *(2)* entire process runtime. Figure 3(c) presents the software publisher and domain registrant name matching score during the start period, where a higher score (closer to 1.0) indicates higher similarity between those names. As shown in the CDF, the majority of malicious processes have lower similarity score when compared to benign processes (over 90% of benign processes have the score of 1.0).

Relation of IP and Domain Location. Previous works showed that location-based features are effective in detecting malicious DNS behavior [5], [37], [46]. We re-visit these features by combining geolocation information for IP and Domain for a given process. PDNS constructs the set of CCs from RDNs and RIPs, then computes statistical features such as the number of distinct countries from domains (in RDNs) and IPs (in RIPs), the number of distinct continents from domains and IPs, and the ratio between distinct countries and all countries. We also compare the country sets retrieved from RDNs and RIPs, where we define the *correlation score* (C_{DN}) as follows.

Let $C_{DN} = \{A_1, A_2, ..., A_n\}$ be the set of country codes retrieved from domains in RDNs, and $C_{IP} = \{B_1, B_2, ..., B_n\}$ be the set of country codes retrieved from IPs in RIPs. We compute the intersection between each pair of the two sets $I_{i,i} = A_i \cap B_i$, for $i = 1, ..., n$. Note that when the A_i and B_i are physical neighbors, we allow the result of the intersection operation to be the domain country code, instead of the null element (*e.g.,* A_j = "CZ" and B_j = "SK", then I_j = "CZ"). Finally, we compute the *correlation score* as the ratio between the length of the intersection set I and the length of the RDNs set (or RIPs set). Legitimate services are likely to maintain their domains and resolved IP address in the nearby region (*e.g.,* subnet, country). Malware's IP addresses, however, are altered over time and placed in a number of different location to avoid being permanently blocked (*e.g.,* dynamic DNS). Therefore a lower correlation score (closed to 0.0) is likely to indicate suspicious domain, IP pairs (Figure 3(d)).

CDN (Content Delivery Network) hosted services are served from geographically distributed locations which may be far off from the domain register location. When legitimate processes visit CDN hosted services, the geo-correlation between the IP and domain locations may be low, similar to malicious processes. However, in this case, other IP based features (*e.g.,* the distinct IP's country, distinct AS number) offset the penalty, as resolved IPs are located in the nearby regions. On the other hand, malicious processes have a diversified profile for these features, which results in a lower score in overall.

V. Evaluation

In this section, we evaluate PDNS seeking answers on the following research questions:

Q1: PDNS overhead: How much computational overhead does PDNS incur to end-hosts and its backend (Section V-A)?
Q2: Detection capability: How effective is PDNS's model in detecting malicious activities (Sections V-D and V-E)?
Q3: Feature importance: What roles do the host and network features play in PDNS detection? and How do newly proposed integrated features contribute to detection accuracy (Section V-G)?
Q4: False positives: How many false alarms PDNS generate and for what reason (Section V-H)?

A. PDNS System Overhead

Our end-host PDNS Windows sensor uses 2% CPU time and 55 MB of memory; our Linux sensor uses on average 0.2% CPU time and 4 MB of memory. We use Postgres for our backend database, which accommodates about 1,200K records per day (approximately 2GB storage space). The backend database uses on average 20% CPU time and 200 MB of memory. Our training and detection are implemented in Java and Python and designed to report alerts within a one-minute threshold. While the computational cost for our periodic offline model updates is high, the cost for online detection is quite small and practical – requiring only less than 1% CPU time and 50 MB of memory on average.

Result 1: PDNS system imposes a small or reasonable amount of computational overhead to its end-hosts and backend.

B. Datasets

1) Benign Dataset: We deployed PDNS sensors (described in Section III-A) to our enterprise environment comprising over 126 Windows workstations, in which we collected their network and process DNS activity data during February - August 2017. Our enterprise workstations are equipped with basic applications *e.g.,* browser, office, email, chat applications, developer tools, as well as antivirus software. Users are allowed to install their own software. We performed our evaluation only on Windows hosts, due to the limited number of Linux malware samples. To ensure that the collected data contains only benign processes, we retrieved hashes of executable binaries (MD5, SHA-1, or SHA-256) from all processes and cross-checked them against the public malware database [59]. Overall, our benign dataset contains 130,579,550 DNS requests, from 455,470 processes (corresponding to 643 unique programs and 1,543 unique hashes).

2) Malicious Dataset: To understand the behavior of malicious processes, we collected malware samples from several sources and executed them in a sandboxed environment equipped with PDNS sensor. We describe *(1)* the collection and validation of malware samples, *(2)* our post-processing for

Dataset	Source	#Samples	Reported Year	#DNS Queries	#Processes	%Processes Conn. Successful
Train-1	VirusSign	12,657	2015	1,177,116	4,676	72.93
Train-2	VirusShare, VXVault	4,070	2017	613,399	2,309	79.08
Train-Total	**–**	**16,727**	**–**	**1,790,515**	**6,985**	**74.96**
Test	VirusShare	2,687	2018	96,601	1,557	59.83
Total	**–**	**19,414**	**–**	**1,887,116**	**13,970**	**72.22**

TABLE II: Malware dataset statistics. PDNS uses *Train-1*, and *Train-2* datasets for training model, and *Test* dataset for evaluating our model on unseen malware.

malicious DNS traffics we captured from sandbox executions, and *(3)* malware process connectivity check as follows.

We collected malware data from publicly available sources at different times. Table II summarizes the sources. We use the 2015 and 2017 malware datasets for assessing the effectiveness of our feature engineering, training and verification of malware detection models (Section V-D), and the 2018 dataset to validate the detection capability of our model specifically in detecting new and unseen malware (Section V-E).

As we could not confirm that downloaded samples are active and malicious, to refine malicious dataset, we applied two filtering conditions. First, we filtered them based on their submission time to guarantee the freshness of malware; samples that have their submission time older than one month from our execution time are discarded. We then excluded irrelevant random files by referring to public Malware Database [59]. After our filtering process, we retained total 19,414 unique malware samples. We set aside 2,687 samples from the latest dataset (2018) to test the capability of our model in detecting unseen malware samples.

When running the malware in the sandboxed environment, PDNS sensor captures DNS traffic from both malware processes and existing background system processes (*e.g.,* `svchost.exe`). While we can safely regard DNS queries from newly introduced binaries as malicious, traffic from pre-existing system processes can represent either legitimate or malicious requests. PDNS labels a process p as malicious if one of the following conditions is met: *(1)* p is instantiated from a newly introduced program binary, *(2)* p makes queries to blacklisted domain names [47], [28], *(3)* p is a pre-existing program (*e.g.,* `svchost.exe`) but its DNS query behavior deviates from its baseline profile established from our benign dataset.

To create the baseline profile for each pre-existing program, we first extract the RDNs (Section IV) for each program in our benign dataset. This establishes a relationship between program and list of requested domains. We then manually select the profiles that have ≤ 10 requested distinct domains and create our *program-based profile* sets, which contain the expected domains. We then use these sets to check against all processes in our malicious dataset and regard the process as malicious when their RDNs are not the expected domains. After our post-processing, our final malicious dataset contains over 1.89M DNS activities from a total of 13,970 processes.

Lastly, we check the connectivity of malware processes and retain one that has at least one successful connection to an external IP address. The rightmost column in Table II shows that, among all malware processes, 72.22% has at least one successful TCP connection. Section V-F details our classifiers trained using malware processes with successful connections.

C. Training Methodology

We train our models using the benign dataset and the 2015 and 2017 malware datasets.

Dataset Normalization. The benign dataset contains almost two orders of magnitude more DNS queries than the malicious dataset and may affect models towards classifying malicious activities as benign. To counter the problem, we balance the two datasets using the SMOTE technique [15] – a combination of over-sampling the minority class and under-sampling the majority class. To avoid overfitting due to re-sampling, we perform ten-fold cross-validation before over-sampling, a standard practice recommended by prior work [36].

Classifiers. We experiment with five classifiers: Random Forest (RF), Logistic Regression (LR), K-Nearest Neighbors (KNN), Linear Support Vector Classification (LinearSVC)[3] and Deep Neural Network (DNN) built with a Neural Network Multi-layer Perceptron classifier.

Parameter Tuning. To achieve the best results, we use *Hyperparameter tuning*, specifically GridSearchCV, to find the approximate good parameter ranges, we then manually fine-tune each parameter within its range. For RF, we set Gini Impurity as the criterion for splitting, the number of trees$= 80$, and $max\_depth = 25$. For LR, we set *liblinear* as our solver with the inverse of regularization strength $C = 0.01$. For KNN, we found the optimal number of neighbors to be 3. For LinearSVC, our optimal penalty parameter $C = 0.1$. Finally, our optimal number of hidden-layers for the DNN is 10.

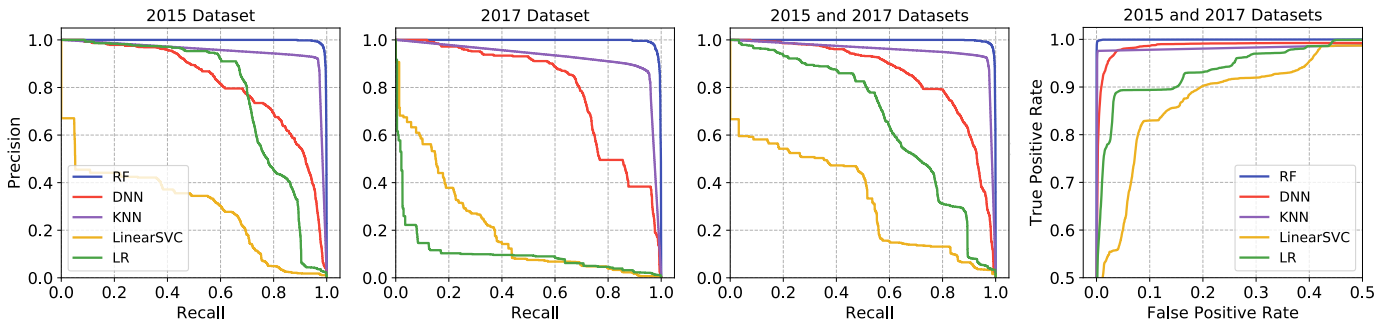
Metrics. To capture the accuracy of the classifiers, we evaluate true positive rate (TP_{rate}), false positive rate (FP_{rate}), precision-recall curve, receiver operating characteristic (ROC), area under the ROC curve (AUC). The TP_{rate} measures the percentage of malicious processes that are correctly identified as "malicious" by our model, while the FP_{rate} measures the percentage of benign processes that are incorrectly identified as "malicious" by our model (false alarm rate). The ROC and AUC visualize the relationship of TP_{rate}, FP_{rate}, and all possible classification thresholds.

To understand the importance of each feature, we use the *mean decrease in impurity (MDI)* [14], [35], designed for tree-based ensemble classifiers. The MDI is a score between 0.0 to 1.0 assigned to each feature, where a higher score indicates a greater importance of the feature from a trained classifier.

D. Detection Accuracy

Figure 5(a) shows the detection results from each classifier using the precision-recall curves. With 2015 malware dataset, the RF classifier reports the best detection results. Specifically, from ten-fold cross-validation (detection threshold equal to 0.5), the RF classifier achieves an average, minimum and maximum of TP_{rate} at 98.03%, 97.22% and 98.71%, respectively, and an average, minimum and maximum FP_{rate} at 0.02%, 0.01% and 0.03%, respectively.

[3] An implementation of Support Vector Machine classifier using liblinear, designed to scale to a large number of samples



(a) Precision-recall (b) ROC

Fig. 5: Precision-recall and ROCs interpolated from ten-fold cross-validation of malicious-2015, 2017, and 2015 and 2017 (combined) datasets. Random Forest (RF) gives the best detection results with $accuracy >= 0.999$, $precision >= 0.979$, $recall >= 0.968$, $f1score >= 0.975$ and mean $AUC >= 0.995$ in all datasets.

Next, we show that our models trained on the 2015 dataset also work well against 2017 malware dataset. We use the same classifiers with the same tuned parameters. The RF classifier achieves the best result again: the average, minimum, and maximum TP_{rate} at 96.80%, 94.37% and 98.70%, respectively, and the average, minimum and maximum FP_{rate} at 0.01%, 0.003% and 0.015%, respectively. We also evaluate the accuracy with a combination of both 2015 and 2017 malware datasets and the result also achieves a high accuracy, on average $TP_{rate} = 98.55\%$ and $FP_{rate} = 0.03\%$, presented on the rightmost of Figure 5(a) and ROC curve in Figure 5(b).

Result 2-1: PDNS achieves high accuracy in detecting malicious processes with an average of 98.55% true positives and 0.03% false positives in the training phase, assuring the effectiveness of our feature engineering.

E. Unseen Malware Detection Accuracy

While PDNS achieves high detection accuracy on the 2015 and 2017 malware datasets, we also attempt to evaluate its effectiveness with *unseen* malware – *newly observed and excluded from the training*. We use the latest malware dataset collected during 2018 (marked as "Test" in Table II) and select the RF detection model trained from ten-fold cross-validation of 2015 and 2017 datasets (Section V-D) which achieve the highest accuracy compared to other classifiers. To this end, our model still maintains a very high accuracy, on average $TP_{rate} = 98.03\%$. This result confirms the predictive power of our approach which also works against malware that is unseen and relatively new.

Result 2-2: PDNS detects malicious processes from *new and unseen* malware with high accuracy ($TP_{rate} = 98.03\%$)

F. Malware Freshness

Dynamic malware analysis often presents challenges on the freshness and activation of the sample used. For example, as C&C servers are typically short-lived, and running a malware sample that is unable to contact its C&C server might result in the malware not being activated. To see if this might affect the accuracy of our prediction model, in addition to the DNS traffic, we also capture all network traffics of the malware processes after the DNS resolution. Next, we select malware processes that have at least one successful connection from the 2015 and 2017 dataset (Table II) and retrain our classifiers. Using the same features, our RF classifier still achieves a high accuracy, on average at $TP_{rate} = 97.84\%$ and $FP_{rate} = 0.03\%$ ($\Delta TP_{rate} =\downarrow 0.71\%$ and $\Delta FP_{rate} = 0.0\%$).

Rank	2015 dataset			2017 dataset		
	Feature Category	ID	MDI	Feature Category	ID	MDI
1	Domain Resolve FR	53	0.098	Location	47	0.117
2	Publisher and Registrant	58	0.077	Location	50	0.073
3	Publisher and Registrant	57	0.066	Domain Duration	33	0.072
4	Publisher and Registrant	55	0.066	Domain Duration	39	0.066
5	Domain and Hostname	22	0.050	Domain Duration	40	0.052
6	Publisher and Registrant	56	0.045	Publisher and Registrant	55	0.052
7	Domain and Hostname	21	0.042	Publisher and Registrant	56	0.046
8	Code Signing	54	0.030	Publisher and Registrant	58	0.038
9	Domain Duration	41	0.025	Domain Registrant	43	0.035
10	Location	48	0.024	Domain Registrant	34	0.029

ID: Feature ID, MDI: Mean Decrease Impurity

TABLE III: Overview of top-ten feature importance ranking from RF models trained with 2015 and 2017 malware datasets. The feature category and ID refer to the feature category, ID in Table IV in Appendix A. Note that our newly proposed features are ranked in top-two in 2015 and top-six in 2017 dataset.

G. Feature Importance Study

In this section, we measure the contribution of each feature to the detection accuracy and how it changes over time. We also examine the effectiveness of PDNS newly proposed features.

1) Features with High Importance: We compute the MDI for all 61 features described in Section IV and present the top ten features for the 2015 and 2017 malware datasets in Table III. Appendix B contains the MDI scores for all features. Next, we discuss and highlight the most significant features.

Publisher and Registrant. The feature category *Publisher and Domain Registrant* which measures the similarity between software signer information and the domain registrant of the program's primary domain names, shows a very high impact on both 2015 and 2017 malware dataset. As shown in Table III, it achieves high MDI scores and ranks second for 2015 dataset and sixth for 2017 dataset.

Domain Duration (*e.g., Domain Registered Duration* and *Renewed Duration*) features, are key features in the 2017 dataset. This indicates that attackers still prefer newly regis-

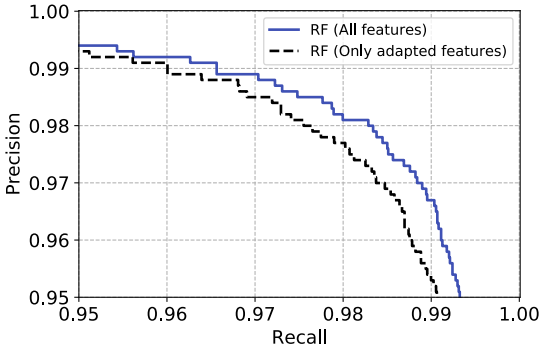


Fig. 6: Comparison of precision-recall curves from *(1)* model trained with all features including our newly proposed features (*All features*) and *(2)* model trained only with adapted features from prior works (*Only adapted features*).

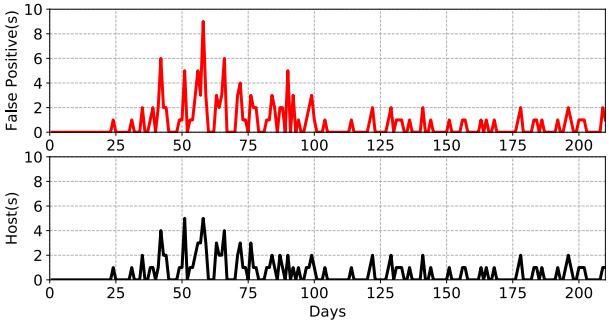


Fig. 7: The daily distribution of false positives.

tered domains as their C&C *e.g.,* communication, drop sites as Perdisci*et al.* [46] also discovered. However, since a higher number of domains in the 2015 dataset have become outdated, less active, or missing; these features, therefore, are less significant in the 2015 dataset.

Domain Resolve FR is highly ranked in the 2015 dataset but becomes less important in the newer 2017 malware dataset. Since the 2017 malware samples are less likely to be blocked or assigned to DNS sinkholes (still active and undetected), this feature rank low for the model trained against 2017-dataset. However, this feature may bias our trained model towards inactive malware (already detected and their C&C sites were taken down) and penalize detection accuracy for the new and fresh malware. To accurately assess the potential influence of this side effect, we re-train our classifiers without all features that are affected by domain resolved failure (*i.e.,* DNS resolve failure rate, resolved AS number, resolved IP's country). Our RF model can still achieve high accuracy ($TP_{rate} = 97.74\%$), albeit with slightly more false alarms ($FP_{rate} = 0.54\%$).

2) Newly Proposed Features: PDNS introduces new features by integrating and combining pre-existing features (Table I). Among all features, the features that correlate between host and network information turns out to be most powerful. For example, the feature category *Publisher and Domain Registrant* is highly important to our detection model as shown. To better assess the effectiveness of these newly proposed features, we run the experiment with and without them and measure the detection accuracy. Figure 6 shows the precision-recall curve resulted from models with and without these features. We focus on the RF model, as it performs best. Overall, the re-trained RF model without our newly introduced features still achieves an accuracy of $TP_{rate} = 98\%$ on average while incurring more false alarms. Specifically, our model with new features reduces false positives at least 46.7%.

Result 3: Our newly proposed feature category that correlates software publisher and domain registrant information is highly ranked in the feature importance study and improves detection accuracy by reducing 46.7% of false positives.

H. False Positives

During our evaluation, a total of 146 unique processes were reported as false alarms from 45 distinct hosts of 126 deployments. As the benign dataset collected over a period of seven months, the distribution of false positives across our enterprise is 0.7 false alarm per day on average. The total 1,328 DNS requests are misclassified which amount to 0.001% of all DNS requests. Figure 7 presents the distribution of false positives produced from our classifier during a period of seven months on our enterprise.

We now discuss interesting cases worth paying attention to: *(1)* Although our detection model has a low FP rate, we identify a recurring pattern across the FP processes. Specifically, over 40.52% of the FP results are from *command-line driven* processes and *browsers* processes where users can control their domain inputs. *(2)* We present the case where benign processes are classified as malicious because of their erratic behaviors.

Command-line Driven Process. The runtime logic of command-line driven processes (*e.g.,* pythonw.exe, java.exe, javaw.exe, PowerShell.exe) are determined by the command-line arguments given at program execution. For instance, both Eclipse and Minecraft run with java.exe, but their runtime behaviors are completely different. Current PDNS data collection includes the command-line arguments for each process. Parsing those argument to identify and extracting the core argument entries is a challenging task. Nonetheless, building models that include those arguments, in addition to the program name would help improving detection accuracy.

Browser Process. Some browser processes (*e.g.,* firefox.exe, chrome.exe, iexplore.exe) are misclassified as false positives, especially when those send a highly large volume of DNS requests than usual processes. As the browser's DNS access pattern highly depends on the user's behavior, it is hard to build a stable model. The models thus are deemed to have *false positives* as well as *true negatives* – malicious domains accessed through browsers. We, therefore, believe that browsers are in their best interest to have their own security protection that is tailored to their specific behaviors.

Benign Process with Erratic Behavior. We also observed various programs that produced false positives. Among those, the *videodl.exe* is a representative case that falls in borderline between benign and malicious. PDNS classified videodl.exe as an alert for the following reasons, *(1)* the process issued high-frequency DNS queries in a short period of time, *(2)* geo-locations of destinations are quite diverse, and *(3)* the program itself is not signed. Our further investigation disclosed that the program frequently makes requests to video sites *e.g.,* googlevideo.com, youtube.com, openload.co. However, interestingly it also makes a large number of DNS requests to

number of unrelated-video sites *e.g.,* spadesplus.com (a game site), pages.ebay.com (product pages of ebay), flyporter.com (a flight booking site) and chinawomendating.asia (a dating site). Upon a reporting of videodl.exe, our blue team identified that the program was legitimately installed from an employee in order to download media files for analytic purposes. Nonetheless, our finding highlights suspicious behaviors of the program which is harder to be distinguish at network-level.

Result 4: PDNS produced only 146 false alarms during our evaluation, translated into FP_{rate} of 0.03%. On daily basis, PDNS would report 0.7 false alarm.

VI. Stealthy Attack Detection

The goal of our study aims at enhancing detection by improving context and visibility. We are also interested in detecting stealthy attack cases which present challenges to existing solutions but can be mitigated with our proposal.

A. HammerToss Malware

The HammerToss malware [24] is first reported in the early of 2015, one of its first kind that actively employs counter forensic techniques to hide itself by mimicking user legitimate traffics. To understand its behavior, we inspect and analyze the malware closely. The behavior of the malware can be summarized as follows:

1) Visit different Twitter pages and retrieve the attacker's twitted messages which include URLs. These URLs point to the attacker-controlled GitHub's pages.
2) Go to these GitHub URLs and fetch their contents (*e.g.,* images) at a specific time or during the victim's active time. For this step, HammerToss invokes Internet Explorer (IE) process via DCOM communication and delegates the connection.
3) Retrieve contents from the IE's cache in order to get commands hidden in those contents (steganography).
4) Execute the commands to obtain user's information and upload it to cloud storage services (*e.g.,* Dropbox).

To evaluate the effectiveness of our model in detecting HammerToss, we extended publicly available prototype [49] and executed the malware in our sandbox environment. We evaluate the attack in zero-day fashion, where our training data does not contain the HammerToss samples. We measure if PDNS can detect the HammerToss process itself and/or IE processes that make connections on its behalf.

HammerToss Detection. Our model successfully detects the HammerToss process in all cases. The multiple features contributed to the detection. Most of all, as reported [24], HammerToss binary is not signed and the process makes multiples DNS queries without having GUI and UI DLLs loaded. Such profiles are rarely observed from the benign dataset, whereas they are common behaviors among malicious processes.

Only about a half (53.8%) of IE processes are classified as malicious, primarily because its DNS queries violate *Publisher and Domain Registrant* relation. However, such behavior, delegating of Internet connection to IE, is occasionally observed from our benign dataset, as the functionality is officially supported by Microsoft. This explains why PDNS did not label all HammerToss's IE processes as malicious.

B. Robust Detection of Counter-forensic Malware

While PDNS and its features are proven to be effective in detecting a stealthy attack such as HammerToss malware, attackers can still upgrade their malware logic to remain undetected. As discussed, the attacker group behind HammerToss (later authored POSHSPY [23]) uses legitimate and public domains to disguise their connections. The malware further hides its process trails with trusted system functionalities such as WMI and PowerShell. These kind of techniques are getting sophisticated and become harder to be detected. Since there has not been a definite solution to deal with the stealthy attacks that actively adapt various ways to counter forensic detections, previous detections require manual effort and heuristics.

Building a program-DNS profile establishes a promising way to enhance a mitigate on such attack classes. As PDNS enhances monitoring with improved context and visibility, PDNS can produce and consume threat intelligence information that captures both network- and process-level behaviors. A program-DNS profile is also applicable to distinguish a normal program's behavior from irregular behaviors when malicious logic is injected. Our previous work [54] showed that malware-infected programs have more erratic querying behavior and tend to query domains that, albeit legitimate, are not associated with the original program.

VII. Related Work

In this section, we overview previous detection systems and highlight the absence of appropriate measures for the process-to-DNS relationship.

A. Network-based DNS Detection Systems

Since DNS is a critical attack vector, numerous works [60], [5], [13], [6], [7], [46] have been proposed to analyze DNS activities on DNS servers outside the host – at different levels of DNS hierarchies. A survey [10] provided an overview of the state-of-the-art network-based detection systems (including other types of network traffic, *e.g.,* packet, flow size). These approaches collect passive DNS records mainly to map the IP to DNS name from which it is translated and to aid forensic analysis and network monitoring. The key difference between these approaches and ours is that none of these focused on detecting malicious activities in a process-centric manner. They rather focused on detecting malicious domains, URLs, and phishing sites. In the following, we summarize these works and categorize them based on their detection goals.

Malicious Domains. Bilge *et al.* [13] presented *EXPOSURE*, a system that can detect domain names involved in malicious activities by performing passive DNS analysis on a large amount of DNS query data. They proposed a number of behavioral network-based features (based on DNS answer, domain TTL, and domain name) that their system uses to build malicious domain classifier and achieved high accuracy and low false positive rate. Similar to previous works, Antonakakis *et al.* [5] also proposed a system *Nato*, which computes a reputation score for a *new and unseen* domain indicating the probability of being malicious. This work introduces a number of new features, *e.g.,* Domain-IP relationship, and diversity of IP locations. A follow-up work [6], Kopis, which uses a similar set of features, is designed to operate on upper-level DNS servers (*i.e.,* top-level domain and AuthNS). Deploying at

the upper-level hierarchy enables Kopis to enhance its visibility and detect malicious activities faster than previous works.

Malicious URLs. Recent works from Bartos *et al.* [10] developed a malware detection technique by constructing an invariant representation of network-flow features, particularly the information in HTTP URL strings (*e.g.,* query value, parameters, length, path). While their approach is able to achieve high accuracy and identify new patterns of URLs for each attack presented, it has the main limitation for malicious activities occurring on HTTPS (encrypted traffic) aiming to hide from detection, which are recently proved to be 37% of all malware and have dramatical growth [38].

B. Host-based Detection Systems

Host-based detections analyze malicious processes, codes and their runtime behaviors *e.g.,* network and host activities.

Static and Dynamic Malware Analysis. Though numerous works proposed to detect malware through *static* analysis [16], [53] through methods of disassembly or reverse engineering, malware authors likely create malware instances that avoid being detected by these techniques, as pointed out by [41], [21], as well as employ obfuscation techniques [62] to make the malware even harder to be identified.

The detection systems use the *dynamic* analysis to observe the malware behaviors while the malware is being executed (*e.g.,* function call, information flow tracking [29]). Willems *et al.* [61] proposed *CWSandbox*, a malware analysis tool for Win32 OSes that tracks and monitors system calls, modified files, Windows registry, loaded DLLs and network connections. Malware behaviors are collected through execution in a restricted environment and reports are automatically produced. Besides CWSandbox, studying malware behaviors under a restricted environment is also proposed in [11], [58], [52]. However, unlike PDNS, none of these works has studied the relationship between DNS activities and processes in detail.

Code Signing Information. Kotzias *et al.* [34] studied the Microsoft's Authenticode and evaluated the effectiveness of the mechanism in practice. They identified a number of weaknesses, for example, Window's applications (executable files) are still correctly validated even when the signing certificates have been revoked. Their study also revealed that out of 350k malware samples, most malware was largely unsigned (only 5-12% signed). While the overall rate is similar to our malware dataset, our recent malware samples from 2017 dataset have a higher signed and validated rate compared to 2015 dataset. Kim *et al.* [32] also studied the code signing. However, unlike other prior works that studied code signing on unwanted program ecosystem [33], [34], [57], they focused on signed malware and showed a number of weaknesses in the mechanism that allows attackers to abuse the trust of the signing process. These attacks confirm that detecting malware cannot solely rely on the code signing and verification process.

VIII. Discussion

In this section, we discuss the evasion possibilities of our newly proposed features as well as propose the direction to improve the robustness of our detection and features.

A. Evasion Attacks

A determined attacker who knows our detection logic might try to evade PDNS detection. Here we elaborate on possible evasions and discuss some limitations of our system.

Forged Loaded DLL. As presented in Section IV-C, we utilize the relations between a specific set of the process's loaded DLLs (GUI, UI) and their DNS usage pattern as one of our anomaly indicators. Despite proven to be effective (as shown in HammerToss detection VI-A), once such logic is disclosed; the attacker can bypass the detection by simply loading those DLLs without using them. Nonetheless, our detection can be improved by extending in-host data collection. Specifically, PDNS sensor can leverage OSes' accessibility features and support APIs to capture per-process user interactions *e.g.,* the number of keystrokes and mouse actions. This enhances our detection granularity and makes it more difficult to bypass.

Forged DNS Activity. The attacker could actively work to bypass DNS activity-based features. For example, the malware could falsely perform DNS queries to the binary's owner domains during the start period to ensure the relationship between the binary signer and domain registrar is established. However since our detection model relies on over 61 features, these evasions come with more cost and complexity for the attacker to avoid more features. Finally, while the current feature proposals might not be robust enough to penalize the attacker who actively attempts to evade PDNS detection, PDNS can easily extend its system data collection to strengthen the detection thanks to the host-based sensor design. We leave concrete proposals and their implementations as future work.

Encrypted DNS query. DNS-over-TLS [1] is a new proposal to send DNS queries over an encrypted connection. As mentioned, while the scheme is designed to enhance user privacy on the Internet, essentially prevent man-in-the-middle attacks on DNS traffics; it also provides a useful mean for the attackers to hide their malicious connections from being alerted by traditional network-based detections. These attacks become even stealthier with DNS-over-HTTPS [2], since it operates over HTTPS, allowing seemingly hidden under HTTPS sessions.

The PDNS approach can be a viable option to combat against the malware with encrypted DNS traffic as follows.

1) In a case where a program delegates their DNS queries to the dedicated DNS service (DNS-over-TLS/HTTPS adopted by DNS system service at the OS-level). PDNS sensor can capture all unencrypted DNS requests between the program and the DNS system service.
2) When a process directly queries using DNS-over-TLS/HTTPS, our current implementation cannot obtain the DNS queries, instead we can only infer from its destination host (DNS providers *e.g., Google*, *Cloudflare*), port number (*e.g.,* DNS-over-HTTPS (443) and DNS-over-TLS (853)) and message size. However, since we found that most processes in our dataset that make connections using port 443 or 853 are loaded with the system cryptographic libraries, we can extend our sensor (already set to run at highest privilege) to intercept cryptographic API calls in order to retrieve the encryption key for decrypting the DNS queries.
3) The programs themselves can statically include all necessary cryptographic functions and directly send DNS

requests using DNS-over-TLS/HTTPS without referring to the system's cryptographic APIs. Nonetheless, as such behavior is rarely seen from benign processes, PDNS can safely label them as suspicious.

Malware with Idle DNS Activity. One of the main limitations is the fact that PDNS is unable to detect malicious processes with no network or DNS activities, as PDNS sensor only records and reports processes that make DNS requests. Therefore, if a (malicious) process has no DNS request, network activity, or direct IP connection, its activities are hidden from our detection. Nonetheless, as mentioned earlier, the majority of attacks require an Internet connection as C&C or drop sites, where employing domain names is a critical attack vector and more stable compared to IPs. Recent studies also confirmed that malware relies more on DNS to support their agile C&C infrastructure [27], [3].

Cross-Process DNS Delegation. For data collection, PDNS sensors capture and report the association between DNS query and its immediate requester process. However, in reality, the modern OSes allow processes to delegate their DNS and network activities to another process. This may mislead PDNS data collection to have incorrect labeling of requester process for given DNS queries. MS Windows supports *WebClient* API call that creates Internet Explorer (IE) process to make a network call on its behalf. It then returns its result via DCOM channel to the original process (Section VI-A). In such cases, the lifetime of IE processes are very short only lasting for a few seconds, and the IE process makes only a small number of DNS queries. This differ from PDNS's model for the program. Therefore, PDNS labeled IE process created by HammerToss as suspicious one. Rather fundamentally, PDNS can stably track the caller and callee relationship by instrumenting and intercepting *WebClient* API.

A more difficult case is when DNS queries are delegated via illegitimate channels – using one of the cross-process injection techniques [9]. This attack leaves very little footprints while malware injects their logic into the benign programs. Detection systems can hardly guess the relationship. While the security community [61], [12] put efforts to mitigate such attacks, no reliable solution has been proposed yet that goes beyond the statistical approximations. As we discussed from Sec VI-B, building a program-DNS profile case be viable solution. Abnormal behaviors of the well-known programs will violate PDNS's model and therefore will be reported as malicious activities. Upon detection, PDNS can use timing information to infer the original requester process for the malicious DNS queries.

B. Detection Enhancement

Feedback Loop. The malware trends and behaviors evolute and change over time. It is important for any ML-based solution to also stay in the game. Given that our system is designed to update and re-train (centralized-based) model regularly (Section III-B), we can feed malicious samples that are known to bypass our detection into our training set in an attempt to "patch" these evasions. Also, as our training and detection reside on the backend, we can observe any new statistical differences between new malware and benign processes and adjust our features and models accordingly.

Comprehensive System Event Monitoring. Currently, PDNS sensor collects a limited set of process information from host internal sources. More system information would improve the precision of PDNS model and detection capability. System events, such as process creation, file access and IPC communication, may not be strong signals by themselves. However, these events can play a role in increasing detection power when they are combined together and associated with DNS activities.

Threat Intelligence Support. The network and host security solutions [44], [19], [48] maintain each their own knowledge bases for detections. However, the updates for the knowledge bases cannot catch up with the speed of growth rate of threats and malware, resulting in limited coverage of security solutions. One way to address this issue is by diversifying the source of threat intelligence and integrate inputs to make a decision. Being connected to host and network security domains at once, PDNS system can bridge host- and network security domains for threat information exchange. As an example, the system can label a program executable as malicious when its process queries to a known malicious domain and vice versa.

IX. Conclusion

In this paper, we presented PDNS, a novel end-point DNS monitoring system consisting of end-point DNS sensors and a centralized backend. PDNS utilizes extensive monitoring DNS activities and the set of host-based features which narrow down our analysis scope to the *process-level*. This enhances PDNS visibility and context of monitoring, thus provides capability necessary in coping with a new class of stealthy attacks equipped with various counter forensic techniques.

We trained and evaluated PDNS using real-world and large-scale data. We deployed PDNS sensors to our enterprise environment comprise 126 Windows hosts to collect DNS activities over a seven-month period, and we also collected over 19K malware execution instances from sandboxed environments. To this end, PDNS detection reports a high true positive rate at 98.55% and a low false positive rate at 0.03% for its training datasets, and 98.03% for the *new and unseen* malware dataset. While PDNS demonstrated its capability in detecting a stealth attack instance, we also discussed available design options that PDNS provided in building robust detection features.

Acknowledgments

We would like to thank anonymous reviewers and our shepherd, Prof. Zhou Li, for their feedback in finalizing this paper. We would also like to thank Marios Pomonis and Giannis Karamanolakis for informative discussions. Author Suphannee Sivakorn is partially supported by the Ministry of Science and Technology of the Royal Thai Government. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of their employers or the sponsors.

References

[1] RFC 7858: Specification for DNS over Transport Layer Security (TLS).
[2] RFC 8484: DNS Queries over HTTPS (DoH).
[3] Akamai. (2017, October) Fast Flux Botnets Still Wreaking Havoc On The Internet According To Akamai Research. http://bit.ly/2yWJ5Eg.
[4] J. Albors. (2017, January) Fast Flux networks: What are they and how do they work? https://www.welivesecurity.com/2017/01/12/fast-flux.

[5] M. Antonakakis, R. Perdisci, D. Dagon, W. Lee, and N. Feamster, "Building a Dynamic Reputation System for DNS," in *Proceedings of the USENIX Security Symposium*, 2010.

[6] M. Antonakakis, R. Perdisci, W. Lee, N. Vasiloglou, II, and D. Dagon, "Detecting Malware Domains at the Upper DNS Hierarchy," in *Proceedings of the USENIX Security Symposium*, 2011.

[7] 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 *Proceedings of the USENIX Security Symposium*, 2012.

[8] Y. Assor and A. Slotky. Anti-VM and Anti-Sandbox Explained. https://www.cyberbit.com/blog/endpoint-security/anti-vm.

[9] ATT&CK. Process Injection. https://attack.mitre.org/techniques/T1055/.

[10] K. Bartos, M. Sofka, and V. Franc, "Optimized Invariant Representation of Network Traffic for Detecting Unseen Malware Variants," in *Proceedings of the USENIX Security Symposium*, 2016.

[11] U. Bayer, C. Kruegel, and E. Kirda, *TTAnalyze: A Tool for Analyzing Malware*. Wien, Techn. Univ., Dipl.-Arb., 2005.

[12] S. B. Bhatkar, S. Nanda, and J. S. Wilhelm, "Techniques for Behavior-based Malware Analysis," October 2013, US Patent 8,555,385.

[13] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi, "EXPOSURE: Finding Malicious Domains Using Passive DNS Analysis," in *Proceedings of the Network and Distributed System Security Symposium*, 2011.

[14] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, *Classification and Regression Trees*. Routledge, 1984.

[15] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic Minority Over-sampling Technique," *Journal of artificial intelligence research*, vol. 16, 2002.

[16] M. Christodorescu and S. Jha, "Static Analysis of Executables to Detect Malicious Patterns," WISCONSIN UNIV-MADISON DEPT OF COMPUTER SCIENCES, Tech. Rep., 2006.

[17] Cuckoo Sandbox. https://cuckoosandbox.org/.

[18] D. Desai. (2016) Malicious Documents leveraging new Anti-VM & Anti-Sandbox techniques – ZScaler. http://bit.ly/2GRNWvA.

[19] C. J. Dietrich and C. Rossow, "Empirical Research of IP Blacklists," in *Information Security Solutions Europe Conference*, 2008.

[20] DNSDB. https://www.dnsdb.info.

[21] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A Survey on Automated Dynamic Malware-analysis Techniques and Tools," *ACM Computing Surveys (CSUR)*, vol. 44, no. 2, 2012.

[22] M. Felegyhazi, C. Kreibich, and V. Paxson, "On the Potential of Proactive Domain Blacklisting." in *Proceedings of the USENIX Conference on Large-scale Exploits and Emergent Threats: Botnets, Spyware, Worms, and More*, 2010.

[23] FireEye Inc. (2015, July) Dissecting One of APT29's Fileless WMI and PowerShell Backdoors (POSHSPY). https://bit.ly/2zGem1m.

[24] ——, "HammerToss: Stealthy Tactics Define a Russian Cyber Threat Group," FireEye Inc., Tech. Rep., 2015.

[25] R. H. Frederic Besler, Carsten Willems. (2017) Countering Innovative Sandbox Evasion Techniques Used by Malware. http://bit.ly/2GRO2TY.

[26] freegeoip.net. https://freegeoip.net.

[27] ICANN. (2017, August) Statistical Analysis of DNS Abuse in gTLDs. https://www.icann.org/news/announcement-2017-08-09-en.

[28] Intellex. Blacklist Domain or IP Address Check. https://git.io/fpXWn.

[29] K. Jee, G. Portokalidis, V. P. Kemerlis, S. Ghosh, D. I. August, and A. D. Keromytis, "A general approach for efficiently accelerating software-based dynamic data flow tracking on commodity hardware," in *In Proc. of the 19 th NDSS*, 2012.

[30] D. Keragala. (2016) Detecting Malware and Sandbox Evasion Techniques. http://bit.ly/2uSHTkk.

[31] KeyCDN. IP Location Finder. https://tools.keycdn.com/geo.

[32] D. Kim, B. J. Kwon, and T. Dumitraş, "Certified Malware: Measuring Breaches of Trust in the Windows Code-Signing PKI," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, 2017.

[33] P. Kotzias, L. Bilge, and J. Caballero, "Measuring PUP Prevalence and PUP Distribution through Pay-Per-Install Services," in *Proceedings of the USENIX Security Symposium*, 2016.

[34] P. Kotzias, S. Matic, R. Rivera, and J. Caballero, "Certified PUP: Abuse in Authenticode Code Signing," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, 2015.

[35] G. Louppe, L. Wehenkel, A. Sutera, and P. Geurts, "Understanding Variable Importances in Forests of Randomized Trees," in *Advances in Neural Information Processing Systems*, 2013.

[36] L. Lusa, "Joint Use of Over- and Under-sampling Techniques and Cross-validation for the Development and Assessment of Prediction Models," *BMC Bioinformatics*, vol. 16, no. 1, 2015.

[37] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker, "Beyond Blacklists: Learning to Detect Malicious Web Sites from Suspicious URLs," in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2009.

[38] A. Magnúsardóttir. (2017, June) Cyren Blog - Malware is Moving Heavily to HTTPS. http://bit.ly/2nk7ijP.

[39] Microsoft. Microsoft-Windows-DNS-Client. https://bit.ly/2SteAQ2.

[40] ——. What's New in DNS Client. https://bit.ly/2SrZD0J.

[41] A. Moser, C. Kruegel, and E. Kirda, "Limits of Static Analysis for Malware Detection," in *Proceedings of the Annual Computer Security Applications Conference*, 2007.

[42] Mozilla. Public Suffix List. https://publicsuffix.org/.

[43] Norton Antivirus. Virus Removal and Virus Protection. http://us.norton.com/antivirus.

[44] J. Oberheide, E. Cooke, and F. Jahanian, "CloudAV: N-Version Antivirus in the Network Cloud," in *Proceedings of the USENIX Security Symposium*, 2008.

[45] Passive Total. https://www.passivetotal.org.

[46] R. Perdisci, I. Corona, D. Dagon, and W. Lee, "Detecting Malicious Flux Service Networks Through Passive Analysis of Recursive DNS Traces," in *Proceeding of the Annual Computer Security Applications Conference*, 2009.

[47] D. Plohmann, K. Yakdan, M. Klatt, J. Bader, and E. Gerhards-Padilla, "A Comprehensive Measurement Study of Domain Generating Malware," in *Proceedings of the USENIX Security Symposium*, 2016.

[48] A. Ramachandran and N. Feamster, "Understanding the Network-level Behavior of Spammers," *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 4, 2006.

[49] S. Saeli. HammerToss: A Proof of Concept. https://git.io/fpXZd.

[50] Scikit-learn. Machine Learning in Python. http://scikit-learn.org/.

[51] SeatGeek. FuzzyWuzzy: Fuzzy String Matching in Python. http://bit.ly/1hfXsIB.

[52] M. Sharif, A. Lanzi, J. Giffin, and W. Lee, "Automatic Reverse Engineering of Malware Emulators," in *Proceedings of the IEEE Symposium on Security and Privacy*, 2009.

[53] D. Song, D. Brumley, H. Yin, J. Caballero, I. Jager, M. G. Kang, Z. Liang, J. Newsome, P. Poosankam, and P. Saxena, "BitBlaze: A New Approach to Computer Security via Binary Analysis," in *International Conference on Information Systems Security*, 2008.

[54] Y. Sun, K. Jee, S. Sivakorn, Z. Li, L. Korts-Pärn, C. Lumezanu, Z. Wu, J. Rhee, C. H. Kim, and P. Mittal, "You are What You Query: Program DNS Behavior under Malware Injection," in *NEC Technical Report 2018-TR020*, 2018.

[55] The Hone Project. http://hone.cs.princeton.edu.

[56] The Honeynet Project. (2008, August) How Fast-flux Service Networks Work. http://www.honeynet.org/node/132.

[57] K. Thomas, J. A. E. Crespo, R. Rasti, J. M. Picod, C. Phillips, M.-A. Decoste, C. Sharp, F. Tirelo, A. Tofigh, M.-A. Courteau *et al.*, "Investigating Commercial Pay-Per-Install and the Distribution of Unwanted Software," in *Proceedings of the USENIX Security Symposium*, 2016.

[58] A. Vasudevan and R. Yerraballi, "Cobra: Fine-grained Malware Analysis Using Stealth Localized-executions," in *Proceedings of the IEEE Symposium on Security and Privacy*, 2006.

[59] VirusTotal. https://www.virustotal.com.

[60] F. Weimer, "Passive DNS Replication," *FIRST Conference on Computer Security Incident*, 2005.

[61] C. Willems, T. Holz, and F. Freiling, "Toward Automated Dynamic Malware Analysis Using CWSandbox," *IEEE Security & Privacy*, vol. 5, no. 2, 2007.

[62] I. You and K. Yim, "Malware Obfuscation Techniques: A Brief Survey," in *Proceedings of International Conference on Broadband, Wireless Computing, Communication and Applications*, 2010.

[63] Y. Zhang, J. I. Hong, and L. F. Cranor, "Cantina: A Content-based Approach to Detecting Phishing Web Sites," in *Proceedings of the International Conference on World Wide Web*, 2007.

APPENDIX A
FEATURE LIST

Feature Category	ID	Name
AS Number	1	Number of distinct ASs
	2	Distinct AS ratio
	3	Number of ASs (normalized t)
	4	Number of distinct ASs (normalized t)
Domain and Hostname		**Domain**
	5	Number of domains
	6	Number of distinct domains
	7	Number of domains (normalized t)
	8	Number of distinct domains (normalized t)
	9	Distinct domain ratio
	10	Average domain entropy
	11	Median of domain entropy
	12	SD of domain entropy
	13	Average domain length
	14	Median of domain length
	15	SD of domain length
		Hostname
	16	Number of hostnames
	17	Number of distinct hostnames
	18	Number of hostnames (normalized t)
	19	Number of distinct hostnames (normalized t)
	20	Distinct hostname ratio
	21	Average hostname entropy
	22	Median of hostname entropy
	23	SD of hostname entropy
	24	Average hostname length
	25	Median of hostname length
	26	SD of hostname length
Domain TTL	27	Average domain TTL
	28	Median of domain TTL
	29	SD of domain TTL
Domain Duration		**Registered duration (days)**
	30	Average registered duration
	31	Median of registered duration
	32	SD of registered duration
	33	Average registered duration of distinct domains
	34	Median of registered duration of distinct domains
	35	SD of registered duration of distinct domains
		Renewed duration (days)
	36	Average renewed duration
	37	Median of renewed duration
	38	SD of renewed duration
	39	Average renewed duration of distinct domain
	40	Median of renewed duration of distinct domain
	41	SD of renewed duration of distinct domain
Domain Registrant	42	Number of registrants
	43	Number of distinct registrants
	44	Number of registrants (normalized t)
	45	Number of distinct registrants (normalized t)
	46	Distinct registrant ratio
Location	47	Distinct domain's country ratio
	48	Distinct IP's country ratio
	49	Domain's and IP's country correlation score
	50	Distinct domain's continent ratio
	51	Distinct IP's continent ratio
	52	Domain's and IP's continent correlation score
Domain Resolve FR	53	Domain resolve failure rate (%)
Code Signing	54	Sign and verify
Publisher and Registrant		**Software publisher and domain registrant**
	55	Levenshtein dist. (setup t)
	56	Levenshtein dist. on partial string (setup t)
	57	Levenshtein dist. (entire t)
	58	Levenshtein dist. on partial string (entire t)
Loaded DLLs	59	Loaded Graphic User Interface (GUI) DLLs
	60	Loaded User Interaction (UI) DLL
	61	Loaded Web Communication (Web Comm) DLLs

t: process run time

TABLE IV: All features (61 features in total).

APPENDIX B
FEATURE IMPORTANCE RANKING

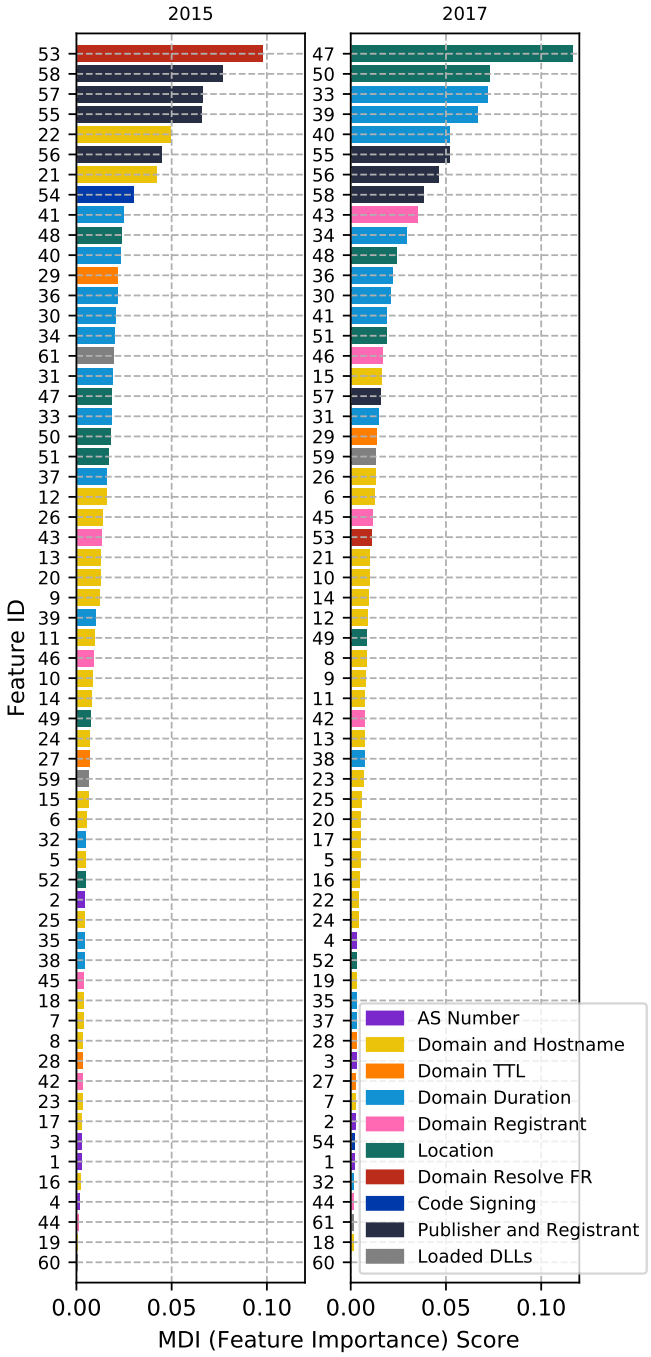


Fig. 8: Feature importance ranking

Figure 8 shows feature importance ranked by MDI (feature importance) score (Section V-G), where each feature ID (y-axis) represents a feature mapped from the ID in Table IV. Both 2015- and 2017-dataset have an average feature importance score at 0.016 (SD 0.02). The maximum scores of 2015 and 2017 are 0.10 and 0.12, and minimum scores are 0.0005 and 0.0002, respectively.