Detecting Malware Injection with Program-DNS Behavior

Yixin Sun[1], Kangkook Jee[2], Suphannee Sivakorn[3], Zhichun Li[4], Cristian Lumezanu[5], Lauri Korts-Parn[6], Zhenyu Wu[7], Junghwan Rhee[5], Chung Hwan Kim[5], Mung Chiang[8], Prateek Mittal[9]
[1]*University of Virginia* [2]*University of Texas at Dallas* [3]*Rajamangala University of Technology Tawan-ok*
[4]*Stellar Cyber* [5]*NEC Labs America* [6]*Cyber Defense Institute* [7]*Google* [8]*Purdue University* [9]*Princeton University*

***Abstract*—Analyzing the DNS traffic of Internet hosts has been a successful technique to counter cyberattacks and identify connections to malicious domains. However, recent stealthy attacks hide malicious activities within seemingly legitimate connections to popular web services made by benign programs. Traditional DNS monitoring and signature-based detection techniques are ineffective against such attacks.**

To tackle this challenge, we present a new program-level approach that can effectively detect such stealthy attacks. Our method builds a fine-grained Program-DNS profile for each benign program that characterizes what should be the "expected" DNS behavior. We find that malware-injected processes have DNS activities which significantly deviate from the Program-DNS profile of the benign program. We then develop six novel features based on the Program-DNS profile, and evaluate the features on a dataset of over 130 million DNS requests collected from a real-world enterprise and 8 million requests from malware-samples executed in a sandbox environment. We compare our detection results with that of previously-proposed features and demonstrate that our new features successfully detect 190 malware-injected processes which fail to be detected by previously-proposed features. Overall, our study demonstrates that fine-grained Program-DNS profiles can provide meaningful and effective features in building detectors for attack campaigns that bypass existing detection systems.

1. Introduction

Recent successful attacks reveal the ability of attackers to obfuscate the DNS activity of the victim by hiding requests to malicious domains in plain sight inside legitimate traffic [50], [31]. Moreover, they delegate malicious logic to long-lived and trusted benign programs, who then establish command and control (C&C) channels through connections to legitimate sites (e.g., Twitter, Dropbox, Google) [27], [2], [26]. Such attacks, which we call *stealthy attacks* throughout the paper, leave no or little footprint on the hosts, thereby evading host defense. Their seemingly innocuous DNS requests are also immune to network DNS-based detection.

Such stealthy attacks are becoming more and more common practice across attackers who want to bypass the existing host and perimeter defenses (e.g., AV scanners and firewalls), which are universally deployed in a modern computing environment. Recent studies report the rapid adoption of such stealthy attack vectors – cyberattacks that impersonate benign programs surged by 432% and 265% in 2017 and 2018 respectively, constituting 35% of the cyberattacks, and are ten times more likely to succeed than traditional attacks [52], [28].

Attackers now actively employ and combine host and network evasion techniques to bypass the traditional defenses [5], [6], [7], [13], [15], [48]. Previous defenses fall short in detecting such stealthy attacks as they only consider network-based features that aim to detect malicious domains and IPs, whereas the attackers can take advantage of legitimate domains to host malicious content. A recent approach integrates network-based features with process-level information [54], however, it is still not effective at detecting such stealthy attacks given that the process-level information are indistinguishable when benign programs are injected by malware. All these approaches lack insight into *how a benign program is expected to behave*.

In this paper, we propose the first *program-level* fingerprinting approach to build fine-grained, individualized profile for each benign program that characterizes the nature of their DNS activities. Such a program-DNS profile can be effectively used to detect *stealthy attacks*, where (1) legitimate web services are leveraged to channel C&C communications, whose DNS queries appear benign, and (2) trusted benign programs are controlled by malware logic to carry out all the DNS activities, whose program- and process-level information appear benign.

We first define three important terms which we will use throughout the paper.

- ***Program:*** a binary executable that represents a software or an application. For instance, "Skype.exe" is a program.
- ***Process:*** a distinct runtime instance or execution of a program, with its own process ID, start time, etc. For instance, when a user launches the Skype application, it creates a process of the "skype.exe" program; when the user exits Skype, the process is terminated. Note than a program can have many processes.
- ***Malware-injected Process:*** a process which is created from a benign program, however, has been injected by malware logic (*cross-process injection* [2], [42]). For instance, if a malware injects the "skype.exe" program and initiates malicious logic, this process is a *malware-injected process*.

To build the program-DNS profiles for benign programs, we collected program data and DNS data from 126 real-world enterprise host machines over the period of seven months where we collected over 130 million DNS requests along with corresponding processes from 636 programs. In addition, to evaluate the effectiveness of our program-DNS profiles in detection, we collected malware samples from different sources with

DOI 10.1109/EuroSP48549.2020.00042

dates ranging from 2015 to 2019 where we collected over eight million DNS requests along with corresponding processes. Among them, we identified 5003 malware-injected processes which were created from 32 distinct benign programs. With the data, we design new metrics to build the program-DNS profile for benign programs, and then develop them into new features to train machine learning classifiers to detect malware-injected processes. We demonstrate the effectiveness of our new features by comparing the detection results with that of previously-proposed features on our datasets. In particular, we make the following contributions:

Building Program-DNS profile. We *jointly* analyze all benign processes of each benign program to characterize what should be the "expected" behavior of each program. We develop two novel sets of metrics, *consistency ratios* which measure the similarity of a process' DNS requests with other processes of the same program, and *domain types* which capture the types of domains queried by the processes. Our key findings are:

- Benign processes have a very coherent behavior with high *consistency ratios*, with *consistency ratios* > 0.96 for more than 85% of the processes. On the contrary, malware-injected processes created from benign programs have a more erratic behavior with low *consistency ratios*, with *consistency ratios* < 0.2 for 90% of the processes.
- Interactive processes (browsers, mail clients) have less coherent behavior (i.e., lower consistency ratios) compared to rest of processes (non-interactive), as their behaviors are driven by users. However, there is still a clear distinction in consistency ratios between benign processes and malware-injected processes, especially when we further narrow down the scope of analysis to each individual user. For instance, benign interactive processes have *consistency ratios* > 0.6 for 57% of the processes, while malware-injected interactive processes have *consistency ratios* < 0.1 for 90% of the processes.
- Benign processes query significantly less "irrelevant" domain types (i.e., domains that are not related to the functionality of the program, defined in Section 4.3) compared to malicious processes. 65.4% of benign programs do not query any "irrelevant" domains at all, while 93.5% of malware programs have processes that query "irrelevant" domains.

Detecting malware-injected processes. Motivated by the above findings, we develop six novel features based on the *consistency ratios* and *domain types*. The key insight behind our features is that a benign process' DNS behavior is consistent with its program-DNS profile, while malware-injected processes deviate from such "expected" behavior. We demonstrate the effectiveness of these new features by using them to train classifiers to detect malware-injected processes. We also compare the detection accuracy between our new features and previously-proposed features [7], [54], [24], [38], [13]. Our key results are:

- By performing ten-fold cross-validation using a Random Forest classifier to detect malware-injected processes with only our new features, we achieve > 0.99 ROC AUC (area under the Receiver Operating Characteristics curve) across all datasets, ranging from 2015 to 2019. The false positive rates are $< 0.1\%$ with true positive rates $> 98.5\%$.
- We use 2015 and 2017 datasets as training data for the Random Forest classifier, and 2018 and 2019 datasets as testing data to perform detection. We demonstrate that our new features are significantly more effective than previously-proposed features, where we achieve 94.3% true positive rate (successfully detect 94.3% of the malware-injected processes in testing data) with 0% false positive rate (does not mistakenly flag any benign process as malicious), compared to 25.9% true positive rate with previously-proposed features. Furthermore, we achieve 100% true positive rate with only 0.15% false positive rate. On the contrary, previously-proposed features can only achieve up to 94.1% true positive rate with 0.33% false positive rate (lowest rate, and true positive rate does not keep increasing until false positive rate reaches 100%), which completely fail to detect 190 malware-injected processes (5.9%).

In summary, we propose the first program-level approach that builds a Program-DNS profile for each benign program. We demonstrate the effectiveness of such profiles in detecting malware-injected processes by evaluating it using real-world datasets and show that our approach is significantly more effective compared to previous approaches.

2. Motivation and Threat Model

We first describe stealthy attacks and how existing malware detection systems fail to detect them. Then, we motivate our work on building fine-grained *Program-DNS Profile* to detect such attacks by using the Skype program as an example. Finally, we specify the adversary's capabilities for our threat model.

2.1. Evading Host Detection

To overcome universally deployed host security solutions, the attackers hide their identities by impersonating well-trusted benign programs and carrying out their missions through the benign programs. A well-known example is the Fileless malware [41], which hides its malicious codes and footprints, and executes inside legitimate processes. Fileless attacks are hard to be detected by conventional defense methods [2], [43] and are estimated to have grown by 265% in the first half of 2019 and constituted 35% of all attacks [52], [28].

Cross-process injection. The attack injects arbitrary code in the address space of a live process. This technique provides attackers more visibility into benign processes as running code in another process may allow attackers to access system resources and memory (e.g., record keystrokes) [43]. Attackers can also migrate their malicious code to a long-lived background process (e.g., system processes) [42], which makes the attack even more persistent. The code injection attack is hard to detect as it is implemented using legitimate system APIs officially supported by Windows. Furthermore, attackers have many viable options for the injection mechanisms [21], making the detection even more challenging.

Script-based attack. Attackers also abuse preinstalled scripting engines to carry out their missions. The scripting engines allow attackers to directly access core system components and dynamics that provide variety of primitives to obfuscate payload. Attackers can embed scripts inside archive files or benign files like Microsoft Office documents. The scripts are interpreted by whitelisted applications such as PowerShell and Windows Management Instrumentation (WMI) and directly run in the memory.

2.2. Evading DNS Monitoring

DNS has been an important primitive for the perimeter defense. To evade DNS-based monitoring and achieve the goals, the attackers leverage public well-trusted web services to hide their traffic among benign network connections. HammerToss [27] malware is one of its kind that embeds its Command and Control (C&C) location using cloud services (e.g., twitter handle, an image on Github). Recently, we see more malware using similar techniques [44], [20], [45]. For instance, the attacker group Turla has used Google Apps Script as its C&C server [22], [23]. MITRE ATT&CKś web service page enumerates numerous attack instances of this kind [1].

2.3. Stealthy Attacks

As the attackers now have various options to minimize their footprints and evade host and network detectors, now we're seeing attacks that combine the above techniques to launch stealthy campaigns.

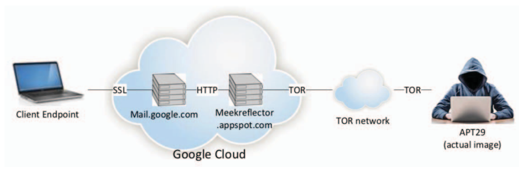


Figure 1: POSHSPY attack injects activities through Windows Management Interface event and leverages Google cloud service to reach its C&C server.

Figure 1 depicts POSHSPY malware [26] by APT29, an exemplary attack that is engineered to evade traditional host and network detectors. After its initial penetration, the attack attains the persistence on the victim's system by embedding its backdoor logic into one of Windows Management Interface (WMI) event. Then, it connects to a proxy server hosted on Google cloud using domain fronting technique, which appears as a connection to legitimate Google service and hides its eventual destination inside the encrypted traffic. Finally, the proxy server will connect to the attacker's C&C server via the Tor network.

Another well-known example is Kazuar [44], a multi-platform backdoor Trojan that injects into "explorer.exe" and uses legitimate WordPress blogs as its C&C servers. In addition, Empire [20] is an open source post-exploitation framework that provides multiple modules for cross-process injections, such as Invoke-PSInject. It can also use Dropbox and GitHub as its C&C servers.

While some techniques have been proposed to mitigate these attacks that inject into benign programs to perform malicious activities [62], [12], detections still largely rely on manual investigation and heuristics. In Section 5.4, we run a POSHSPY attack sample within an isolated environment and show that our new detection approach successfully detects the attack.

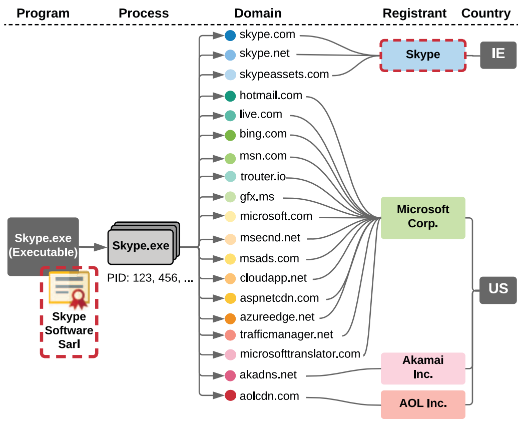


Figure 2: Program-DNS profile of Skype.exe.

2.4. Program-DNS Profile

Numerous works [61], [5], [13], [6], [7], [15], [48], [9] have been proposed to analyze DNS activities on DNS servers outside the host at different levels of DNS hierarchies. However, if the malware leverages *benign web services* to contact its C&C servers, then its DNS activities can look completely benign (e.g., querying mail.google.com), and the existing network-based detection solutions may fail to detect it. On the other hand, numerous host-based detection systems have also been proposed to detect malware through static analysis [16], [55] as well as dynamic analysis [63], [62]. However, if the malware injects its activities through benign programs, then the attacks would become much harder to detect.

The challenge of detecting stealthy attacks with malware injection that leverage benign web services to communicate with C&C servers have motivated us to explore new fine-grained detection techniques.

Our intuition towards detecting stealthy attacks is that for each benign program, there should be a *Program-DNS profile* that characterizes how processes of the program would behave. We expect the DNS behavior of benign processes of the program to be more "stable" as it is bound to the original program logic. In contrast, when the program is injected by malware, then the behavior of the resulting process would be more erratic and deviate from its benign profile.

Figure 2 illustrates how a *Program-DNS profile* for a given program (e.g., Skype.exe) may look by combining network-based DNS information with process-level information from the kernel. Skype.exe program instantiates itself into multiple processes by loading the executable binary. Each process sends DNS queries to get IP addresses and connects to other Skype hosts. The DNS queries can then be further linked to domain registrants and registration countries. Here, we can see from the Program-DNS profile that Skype processes mostly query domains registered by Skype Inc. and Microsoft Corporation. In

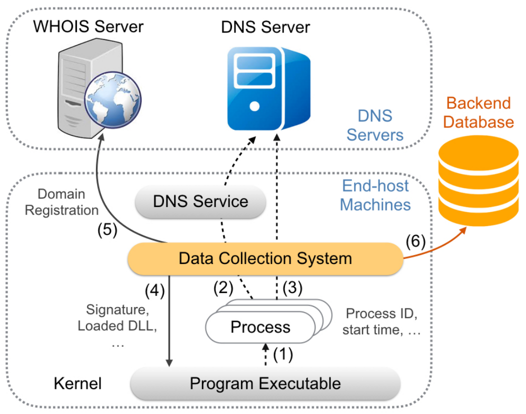


Figure 3: Data collection system in the back-end on end-host machines.

Section 4, we will develop new metrics to quantify such Program-DNS profile for each benign program.

2.5. Threat Model

The attacker is capable of compromising end-host machines with malware, which may inject malicious logic to long-lived and trusted benign programs to obfuscate DNS activities. The attacker is also capable of establishing the C&C servers via various ways, leveraging popular web services as relays to C&C servers. We assume the process data collected from kernel space are not tampered and thus the kernel is trusted, which follows the threat model of previous works on system monitoring [33], [34], [35], [36], [39], [40], [10], [30], [37], [54]. Kernel-level attacks that compromise security monitoring systems are beyond the scope of this study. However, we do consider the possibility of the attacker compromising our data collection system and tampering the data sent to the backend database. To ensure the integrity of our data, we cross-check the collected data from our collection system with kernel logs and local DNS resolvers for data consistency. We assume that the enterprise's IT infrastructure, including the local DNS resolvers, is trusted and not compromised by the attack.

3. Data Collection and Statistics

In this section, we describe our data collection approach and our datasets. We also discuss our data processing methodology and provide an overview of the data.

3.1. Data Collection System

We design our data collection system to be run on end-host machines, as depicted in Figure 3. The high-level goal is to collect DNS data (e.g., DNS query activities) *associated* with the process- and program-level data (e.g., executable path, signatures) of the process that initiates the DNS query.

In step (1) in the figure, a process is initiated from a program executable. The process then starts sending DNS queries, either through step (3) where the process directly sends the query to the recursive DNS server, or through step (2) where the process delegates the query to the DNS system service that handles queries for all processes. Step (2) is more commonly seen in Windows operating systems, while step (3) is more common in Linux-based systems. Due to the potential delegation of DNS queries, in order to accurately associate DNS queries with the correct process that initiates the queries, our data collection system needs to intercept the inter-process communication between the DNS service and the process to capture such information.

From step (1), (2), and (3), the data collection system has already *passively* collected DNS activities (DNS query and answer) *associated* with the process-level data (e.g., process name, start time, etc.). The system then complements the data by *actively* retrieving program-level data (e.g., binary signatures, etc.) from the kernel (step (4)), as well as obtaining domain registration information for the query from external WHOIS server (step (5)).

Finally, the data collection system on each end-host machine aggregates all the data collected through steps (1)-(5) for the process, and sends to the centralized back-end database for storage and further analysis.

3.2. Benign and Malware Data Collection

We build and deploy the data collection system on 126 Windows machines within a corporate environment. The system records everyday DNS activities of all processes on each machine, used by real users. We obtained approval from the corporate's legal office on the research experiments. Our data collection approach is compliant with the corporate privacy policy and the collected data is only accessible by authorized collaborators via a secure channel. The data collection period spanned from February 2017 to August 2017, during which we collected over 130 million DNS queries from more than 455 thousand processes, as shown in Table 1. This constitutes our benign dataset.

Data integrity. We need to ensure that all the captured data are indeed benign. Even though all the end-host machines are behind the corporate's firewall and under constant monitoring, we take additional steps to ensure data integrity by cross-checking the executable binary signatures (MD5, SHA-1, or SHA-256) of all collected processes against VirusTotal Malware Database [59] and performing additional manual inspection. As we deploy data collection system to each end-host, there is a possibility that the attacker takes over the host and sends forged reports. To counter this threat, our system cross-checks its collected DNS activities with the log from local DNS resolver to which the end-host reports.

Malware dataset. We collected malware samples with unique signatures from three public sources: VirusSign [58], VirusShare [57] and VXVault [60]. The time span of the malware samples range from 2015 to 2019. We set up a constrained environment using the Cuckoo sandbox [17] and run each malware sample individually. We follow common guidelines [32], [29], [18], [8] to avoid triggering *anti-VM* techniques employed in various malware. We deploy our data collection system inside the sandbox to collect the malware data, the same way the system collects the benign data from user machines. In total, we ran over 20k malware samples and collected 8.3

Dataset	Collected/Reported Year	# DNS Queries	# Processes
Benign	2017	130,579,550	455,468
Malware	2015	3,264,235	5,736
Malware	2017	2,218,678	6,382
Malware	2018	533,477	1,550
Malware	2019	2,284,197	248,524

TABLE 1: Summary statistics of benign and malware datasets

million DNS queries from over 260k processes from the sandbox, as shown in Table 1. Note that Table 1 shows the statistics of each malware dataset by year.

Malware-injected processes. We first capture the activities of benign processes that already exist in the sandbox operations without running any malware. Then, we run the malware in the sandbox, which might carry out its malicious activities by injecting the malicious logic into existing benign live processes [2]. Next, our goal is to build a labeled dataset consisted of malware-injected processes which are created from benign programs but whose behaviors are controlled by injected malicious logic. Thus, we filter out processes whose activities are identical to the activities they already have before running the malware. We also filter out malware processes which perform all the activities themselves without injecting into any benign processes, since it is not the focus of our study (these processes are very easily identifiable from their binary signatures). The resulting dataset includes 7840 malware-injected processes whose binaries have the same signatures as their benign non-injected counterparts.

3.3. Data Preprocessing and Statistics

Our eventual goal is to create a *Program-DNS profile* for each benign program based on the data from the benign dataset, and use it to differentiate a benign process from a malware-injected process which is initiated by the same benign program. To this end, we perform two preprocessing steps on our data:

- *Group processes into programs.* One program (unique executable binary) can initiate many processes. We group all processes of the same program together and analyze their behaviors jointly. This enables us to build a robust and comprehensive profile for programs on different machines of different users. After this grouping, we have 641 unique programs in our benign dataset.
- *Analyze domain name as opposed to full DNS query.* Due to the large number of distinct DNS queries, we instead focus on the *domain name*, i.e., the second-level domain. For instance, the domain name for the query "mail.google.com" is "google.com". For special classes of queries, such as queries for the local network, we assign three domain names to represent them in our analysis: "INTERNAL" for all internal DNS lookups, "ARPA" for all reverse DNS lookups, and "OCSP" for all queries requesting or checking digital certificates.

After the above preprocessing steps, we show the average number of distinct domains (in log scale) queried by processes of each benign program in Figure 4. While 86% of the programs only query at most 5 distinct domains

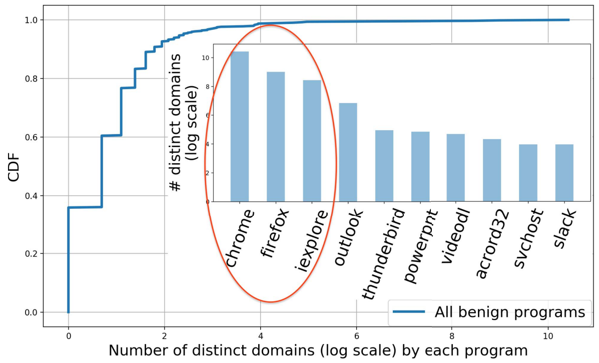


Figure 4: Average number of distinct domains queried by processes of each program (log scale).

(and 36% of programs only query 1 distinct domain), the CDF has a very long tail where very few programs query a large number of distinct domains.

We show the top 10 programs in terms of number of distinct domains on the bar chart, plotted on a log scale in Figure 4. The top 3 programs – Chrome, Firefox, and IExplore – clearly stand out. This highlights the difference in the nature of programs: user-interactive programs such as browsers and mail clients (e.g., , Outlook and Thunderbird) can have much higher heterogeneity in their DNS behaviors. On the contrary, non user-interactive programs could have relatively predictable behaviors. We will further measure and quantify this difference in Section 4.

4. Profiling Program-DNS Behavior

Stealthy attacks may inject malware logic into benign programs and request DNS queries on their behalf [2], and/or establish C&C channels using only legitimate domain names [27], [26], [1]. Existing malware detection techniques, such as checking program executable against malware databases or analyzing DNS requests, may mistakenly label such malicious activities as benign.

In order to distinguish such stealthy attacks from benign activities, we aim to develop a Program-DNS profile for each benign program. Our intuition is that the *pattern* of domains queried by benign processes of a program is significantly different from the *pattern* of domains queried by malware-injected processes of the same program, *even if all the queried domains are legitimate and benign*.

To this end, we develop two groups of metrics to build the Program-DNS profiles and distinguish between benign processes and malware-injected processes: (1) Frequency Ratios and Consistency Ratios (Section 4.1), and (2) Domain Types (Section 4.3).

4.1. Frequency Ratio and Consistency Ratio

Each benign program tends to frequently query certain domains. For instance, as illustrated in Figure 2, processes initiated by the Skype program usually query domains like "skype.com" and "live.com", whose registrants are "Skype" and "Microsoft Corp", located in countries "IE" (Ireland) and "US" (United States). When we observe a process initiated by the Skype program, we would expect such behavior pattern. On the other hand, processes from some benign programs may exhibit seemingly suspicious activities such as querying certain advertisement servers. Such behaviors should be captured and "expected" from processes of those benign programs as well.

In order to quantify such "expected" behavior pattern for each program, we analyze all the processes initiated by each benign program in our benign dataset, and develop *frequency_ratio* to represent how "common" a certain behavior is (e.g., querying a certain domain). Then, we develop *consistency_ratio* to measure how much a process' behavior is "consistent" with its expected behavior (e.g., querying domains that the majority of benign processes from the same program have also queried). We will delve into the details in the following sections.

4.1.1. Domain Consistency Ratio. We first define *frequency_ratio* for a domain as the fraction of processes of a given program that query the domain, as following:

$$Frequency\_Ratio_{pg,d_i} = \frac{\text{\# benign processes of } pg \text{ querying } d_i}{\text{total \# benign processes of } pg}$$

where pg is a given benign program, and d_i is a domain that is queried by benign processes of program pg.

For instance, if 9 out of the 10 legitimate Skype processes query "skype.com", then the $Frequency\_Ratio_{Skype,skype.com}$ is 0.9. If no benign process of pg has queried d_i, then the ratio is 0. The intuition behind this metric is to measure how "common" a domain is to be queried by a benign process.

Next, we define *domain_consistency_ratio* for a given process (either benign or malware-injected) as the average of $Frequency\_Ratio_{pg,d_i}$ for all its queried domain d_i, as following:

$$Domain\_Consistency\_Ratio_{pc} = \frac{\sum_{d_i \in D} Frequency\_Ratio_{pg,d_i}}{|D|}$$

where pc is the given process, pg is the benign program from which pc is initiated, and D is the set of domains queried by pc.

For instance, if $Frequency\_Ratio_{Skype,skype.com} = 0.9$, and a Skype process only queries "skype.com", then its *Domain_Consistency_Ratio* is 0.9. On the contrary, if a Skype process only queries "example.com", where $Frequency\_Ratio_{Skype,example.com} = 0$, then its *Domain_Consistency_Ratio* becomes 0, indicating that this Skype process could be maliciously initiated by malware-injected Skype program. The intuition behind this metric is to measure how "consistent" the behavior of a process is with the behavior of known benign processes of the same program.

4.1.2. Registrant Consistency Ratio. We now extend the analysis further to reflect the domain registration information, namely, the domain registrant. This is designed to accommodate potential variety or churn in the domain queries. For instance, if half of the Skype processes only query "skype.com" and the other half only query "skype.net", then the $Frequency\_Ratio_{Skype,skype.com}$ and $Frequency\_Ratio_{Skype,skype.net}$ will both become 0.5, which is relatively low. However, if we take a look at the domain registrant rather than the actual domain name, then we will see that both "skype.com" and "skype.net" have the same registrant, which is "Skype". Thus, including the registrant information can further complement the $Frequency\_Ratio$ and $Consistency\_Ratio$ analysis.

We define *frequency_ratio* for a domain registrant as the fraction of processes of a given program that query at least one domain registered by the registrant, as following:

$$Frequency\_Ratio_{pg,reg_i} = \frac{\begin{array}{r}\text{\# benign processes of } pg \text{ querying} \\ \text{domains registered by } reg_i\end{array}}{\text{total \# benign processes of } pg}$$

where pg is a given benign program, and reg_i is the domain registrant of at least one domain queried by benign processes of program pg.

For instance, in the above case where half of the Skype processes only query "skype.com" and the other half only query "skype.net", the $Frequency\_Ratio_{Skype,Skype}$ will be 1. This captures how "common" a domain registrant is for domains queried by a benign process.

Next, we define *registrant_consistency_ratio* for a given process (either benign or malware-injected). Similar to the *domain_consistency_ratio*, it's the average of $Frequency\_Ratio_{pg,reg(d_i)}$:

$$Registrant\_Consistency\_Ratio_{pc} = \frac{\sum_{d_i \in D} Frequency\_Ratio_{pg,reg(d_i)}}{|D|}$$

Note that we use $reg(d_i)$ here to retrieve the registrant for *every* domain queried by the process. If the registrant hasn't been seen from any other benign process, then similar to the domain frequency case, the *frequency_ratio* for this registrant would be 0. However, in some cases, we may not be able to retrieve *any* registrant at all, e.g., the domain registration information cannot be found, or the domain is an "INTERNAL" query or reverse DNS lookup ("ARPA"). In these cases, we assign $Frequency\_Ratio = 1$ if the domain d_i is a *reserved* domain (as defined in Section 3.3), which does not have any registrant by its nature; otherwise, we assign 0 if we are unable to retrieve the domain registrant from the WHOIS server.

4.1.3. Country Consistency Ratio. In addition to leveraging domain registrant information, we also take into consideration the country where the domain registrant is located. Both the $Frequency\_Ratio$ and $Consistency\_Ratio$ definitions for domain country are very similar to the domain registrant case:

$$Frequency\_Ratio_{pg,country_i} = \frac{\begin{array}{r}\text{\# benign processes of } pg \text{ querying} \\ \text{domains registered in } country_i\end{array}}{\text{total \# benign processes of } pg}$$

$$Country\_Consistency\_Ratio_{pc} = \frac{\sum_{d_i \in D} Frequency\_Ratio_{pg,country(d_i)}}{|D|}$$

We handle $country(d_i)$ the same way as $reg(d_i)$ in case the country information of a domain cannot be found.

Note that we provide a set of possible definitions for $Frequency\_Ratio$ and $Consistency\_Ratio$ above to profile the Program-DNS behaviors, but there could be other variations of the ratios that may potentially work.

4.2. Evaluation for Consistency Ratios

We now evaluate the three consistency ratios on our data. To better demonstrate the difference between benign and malware-injected processes, we first describe the five groups of processes that we will evaluate.

- **Non-interactive Benign**: benign processes of non-interactive programs in the benign dataset.

	# Programs	# Processes	Dataset
Non-interactive Benign	636	446,871	Benign
Non-interactive Common	32	50,320	Benign
Non-interactive Malware	32	5,003	Malware
Interactive Benign	5	8,597	Benign
Interactive Malware	5	2,837	Malware

TABLE 2: Number of programs and processes in each group.

- **Non-interactive Common** (subset of *Non-interactive Benign*): benign processes of non-interactive programs in the benign dataset, *where the same programs are injected by malware in the malware dataset*.
- **Non-interactive Malware**: malicious processes of non-interactive benign programs injected by malware in the malware dataset.
- **Interactive Benign**: benign processes of interactive programs in the benign dataset.
- **Interactive Malware**: malicious processes of interactive programs injected by malware in the malware dataset.

Note that interactive programs include $chrome.exe$, $firefox.exe$, $iexplore.exe$, $outlook.exe$, $thunderbird.exe$, while non-interactive programs include all the other programs. Table 2 summarizes the number of programs and processes in each group.

Non-interactive programs. Figure 5 shows the results for consistency ratios of *Non-interactive Benign*, *Non-interactive Common*, and *Non-interactive Malware* processes. Note that the frequency ratios of non-interactive programs are established across all users.

Non-interactive Benign and *Non-interactive Common* processes have very high domain consistency ratios, where more than 85% of the processes have values > 0.96. However, *Non-interactive Malware*, which are the malware-injected processes of the *same 32 programs* in *Non-interactive Common*, show significantly lower domain consistency ratios, where 90% of the processes have values < 0.2.

The registrant consistency ratio has very similar pattern as the domain consistency ratio, where *Non-interactive Benign* and *Non-interactive Common* processes have very high registrant consistency ratios while *Non-interactive Malware* processes have very low ratios. For the country consistency ratio, the *Non-interactive Malware* processes have relatively higher ratios compared to the previous two ratios due to the much smaller set of countries. However, they are still significantly lower than the ratios from the benign processes.

Interactive programs. The DNS behaviors of Interactive programs, including browsers, are hard to model due to their user-interactive nature. Previous work [54] tried to mitigate this issue by only considering DNS activities during the first 120 seconds of the process to avoid being affected by user-initiated domains. However, the malware can simply wait for 120 seconds until it starts its activities to avoid being detected.

We propose a new way to profile DNS behaviors from interactive programs utilizing the frequency ratios and consistency ratios. Instead of building the ratios based on all processes from *all users* for each program, we fine-grain the ratios further to the *user-level* by considering all processes from *each user* for each program. The intuition is that each user generally has its own frequently-visited domains, which can be very different from the frequently-visited domains of another user. We can profile such DNS behaviors on the *program-level and user-level* to capture the pattern and use it to differentiate from the activities by malware. Note that we consider *all activities* from a process, instead of just the starting period. Figure 6 shows the results for the *Interactive Benign* and *Interactive Malware* processes.

We can see that, although the domain consistency ratio of *Interactive Benign* is lower than that of the non-interactive processes, it still has a very clear difference from the domain consistency ratio of *Interactive Malware*. The malware-injected interactive processes have much more erratic behaviors and vastly different queries.

For registrant consistency ratios and country consistency ratios, both *Interactive Benign* and *Interactive Malware* have higher values compared to domain consistency ratios due to the much smaller set of registrants and countries. However, the clear difference between them remains. Note that *Interactive Benign* have very high country consistency ratios due to the geographic location of our data collection environment, which is in the US. Thus, while users may visit vastly different domains by different registrants, the majority of the domains are still registered in the US. On the contrary, *Interactive Malware* are not affected by such user pattern, and thus still have much lower country consistency ratios.

In Section 6.2, we will discuss more details on factors that may hinder the attacker from injecting into interactive programs.

Key Takeaway: Consistency ratios characterize the "expected" DNS behavior of a benign program and capture significant difference between benign and malware-injected processes initiated from the same benign program.

4.3. Domain Type Analysis

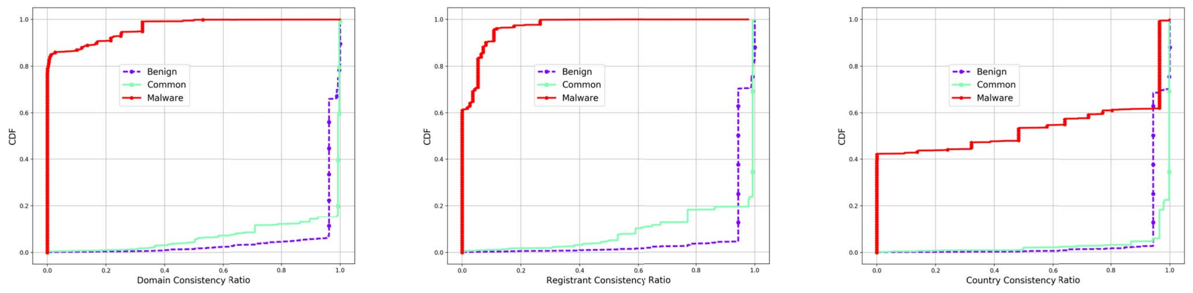
Consistency ratios in Section 4.1 are based on domain information that *has already been observed* from past benign processes. However, if a new process queries a domain which has not been observed before, we need further information to analyze this behavior. To this end, we leverage program-level information to build *domain type analysis* that serves as an indicator for whether a domain should be "expected" – even if the domain *has not been observed before* from past benign processes.

We classify the domains into three types:

- *Reserved*: this includes all domains that are "INTERNAL", "ARPA", "OCSP", as defined in Section 3.
- *Owner*: this includes domains that are considered as "owned" by the same corporation/entity/organization that "owns" the program. We will provide the detailed approaches to determine and compare "ownership" in Section 4.3.1.
- *Other*: all other domains that do not belong to the above two types.

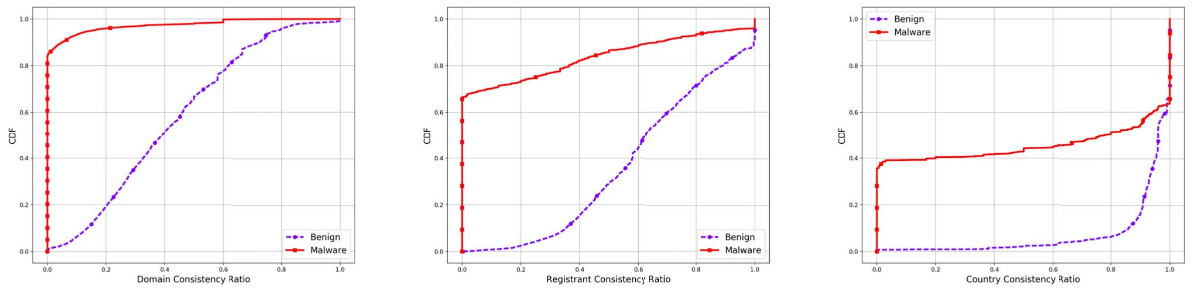
4.3.1. Identify "Owner" domains for a program.

The *Owner* type identifies domains registered by the organization who *owns* a program. For instance, "java.com"



(a) Domain consistency ratio (b) Registrant consistency ratio (c) Country consistency ratio

Figure 5: Consistency Ratios for Non-interactive Processes.



(a) Domain consistency ratio (b) Registrant consistency ratio (c) Country consistency ratio

Figure 6: Consistency Ratios for Interactive Processes.

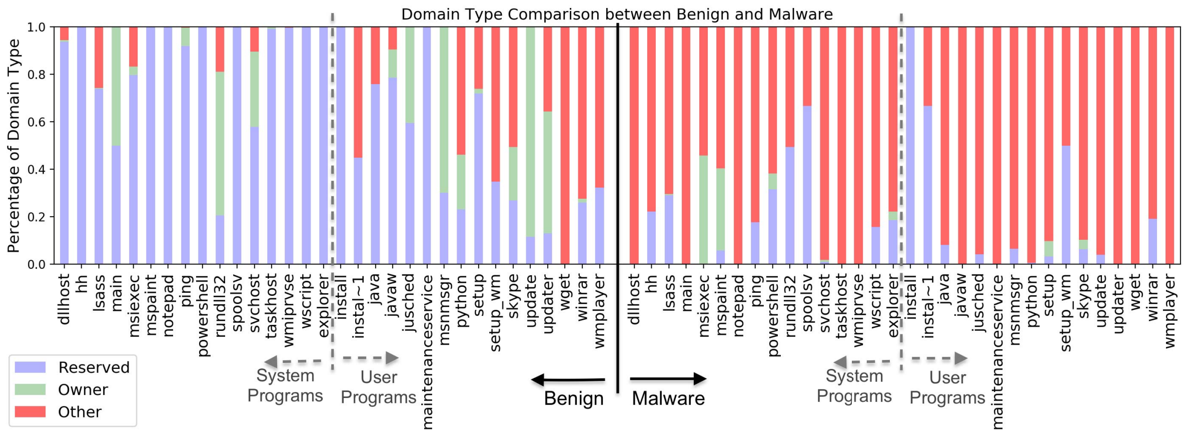


Figure 7: Domain type distribution for 32 common programs in benign and malware datasets.

and "oracle.com" are both *Owner*-type domains for "jp2launcher.exe" program, which is part of Java framework and therefore developed and signed by Oracle, which is also the registrant of both domains. To infer this relationship, we need to extract "owner" information for both the program and the domain.

"Owner" for program. We rely on three sources: (1) program name, e.g., "nvidia.exe" is an indicator for NVIDIA; (2) program signer extracted from the code signature of the program binary, if the program is signed; (3) program path, e.g., the path "C:/Program Files/NVIDIA Corporation/Update Core/NvProfileUpdater64.exe" indicates that the program "NvProfileUpdater64.exe" belongs to NVIDIA.

"Owner" for domain. We rely on two sources: (1) domain name, e.g., "nvidia.com" indicates NVIDIA; (2) domain registrant from WHOIS record.

Note that when extracting "owner" information, we also strip away common strings seen in an organization's name, such as "Corporation", "LLC", etc., to ensure accurate comparison results.

4.3.2. Domain type distribution for common programs. We analyze the domain type distributions for *Non-interactive Common* and *Non-interactive Malware*

Dataset	# Programs *Reserved*	# Programs *Owner*	# Programs *Other*
Benign	213 (33.5%)	203 (31.9%)	220 (34.6%)
Malware	359 (6.4%)	6 (0.1%)	5264 (93.5%)

TABLE 3: Domain Type Distributions for All Programs

processes from the 32 common programs, as defined in Section 4.2. For each process, we compute the fraction of each domain type (*Reserved*, *Owner*, *Other*) from all the domains it queried. Then, we group processes of the same program together, and compute the *average fraction* of each domain type for all processes of a given program. For comparison, we separate benign processes and malware-injected processes of the same program.

Results. Figure 7 shows the domain type distributions for both benign and malware-injected processes of the 32 common programs for direct comparison. The left half shows the benign processes from the benign dataset, and the right half shows the malware-injected processes initiated by the same programs in the same order from the malware dataset. We also separate programs into two groups based on their origins: (1) system programs, which typically come pre-installed with the Windows OS, and (2) user programs, which are typically installed by the users.

At a first glance, we can clearly see that malware-injected processes query a lot more *Other* domains (red bars) than their benign counter-parts. Furthermore, benign processes from system programs query less non-*Reserved* domains (*Owner* + *Other*) than benign processes from user-installed programs.

On the other hand, malware-injected processes from system programs and user-install programs do not show a significant difference from each other – they both query a large number of *Other* domains.

One program, “wget.exe”, queries many *Other* domains and does not exhibit a clear distinction between benign and malware-injected processes. This is due to the user-interactive nature of the program, where most of its queries depend on user input.

4.3.3. Domain type distribution for all programs.

We also extend the analysis to all non-interactive benign and malware (or malware-injected) programs. Table 3 shows the number of programs corresponding to each domain type distribution for both benign and malware programs. We also show the detailed domain type distributions for all non-interactive benign programs in Appendix A.

We can see that more than 65% of the benign programs whose processes have only queried *Reserved* domains and/or *Owner* domains. Such stable pattern is very desirable, as it facilitates the “prediction” regarding what types of domains that a process from these programs may query. On the contrary, only 6.5% of the malware programs have the same pattern.

We also show the CDF for fractions of *Other* domain type queried by all processes of non-interactive benign and malware programs in Figure 8. We can see that there is a clear difference between benign and malicious processes in the fraction of *Other* domains, where the malicious processes query significantly higher fraction of *Other* domains.

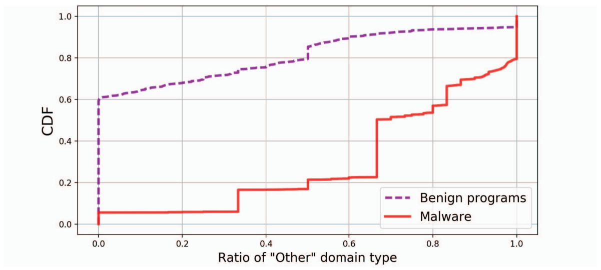


Figure 8: Fraction of *Other* domain type for all processes of non-interactive programs.

Key Takeaway: Domain type analysis incorporates program-level information to classify domains and measures domain type distribution, which effectively distinguishes benign processes and malware-injected processes from the same program.

5. Detection Using Program-DNS Profile

We have shown in Section 4 that we can build Program-DNS profile using *consistency ratios* and *domain types* that capture significant differences between benign and malware-injected processes, even when they are initiated from the same benign program where program- and process-level information are indistinguishable. In this section, we further demonstrate the effectiveness of the Program-DNS profile by developing them into six new features to train machine learning classifiers to detect malware-injected processes on real-world datasets. We also compare the detection results of our new features with that of previously-proposed features.

5.1. Dataset and Features

To evaluate detection accuracy on malware-injected processes, we focus on processes from the 32 common programs that are present in both the benign dataset and at least one of the malware datasets. Table 4 summaries the data we will use for detection.

For each malware dataset, we identify the malware-injected processes initiated by *programs that are present* in the benign dataset; then, we identify the benign processes initiated by the *same programs* from the benign dataset. Column “# Process Samples” shows the number of such benign and malware-injected processes we identify for each dataset.

We leverage six new features to detect malware-injected processes based on the Program-DNS profile from Section 4, as following:

1) Domain consistency ratio (Section 4.1.1)
2) Registrant consistency ratio (Section 4.1.2)
3) Country consistency ratio (Section 4.1.3)
4) Percentage of *Reserved* domain type (Section 4.3)
5) Percentage of *Owner* domain type (Section 4.3)
6) Percentage of *Other* domain type (Section 4.3)

We evaluate our new features from two aspects:

- *Cross-validation on each dataset.* To evaluate the effectiveness of our features, we perform ten-fold cross-validation on each of the 2015 – 2019 datasets. Due to the imbalance in the number of benign

Year	# Programs	Source	# Process Samples	Total	# Process Samples After SMOTE	Total
2015	23	Benign	49,905	50,444	49,905	99,810
		Malware	539		49,905	
2017	16	Benign	6,013	7,260	6,013	12,026
		Malware	1,247		6,013	
2018	6	Benign	4,279	4,295	4,279	8,558
		Malware	16		4,279	
2019	3	Benign	4,231	7,432	4,231	8,462
		Malware	3,201		4,231	

TABLE 4: Detection dataset includes processes initiated by programs that are present across benign and malware datasets.

and malware-injected processes in some of the datasets (e.g., 2015 dataset), we apply the SMOTE technique [14] to oversample the minority class (malware-injected) to the same number as the majority class (benign), shown in column "# Process Samples After SMOTE". We also compare the evaluation results using data with and without SMOTE.

- *Comparison with previously-proposed features.* To mimic real-world scenarios where training is done using existing data and the trained model is used to detect future attacks, we use the 2015 and 2017 datasets as training data, and the 2018 and 2019 datasets as testing data. Note that we do *not* apply SMOTE oversampling for either the training or testing phase. We also compare the detection results with previously-proposed features.

5.2. Cross-Validation using New Features

We perform ten-fold cross-validation for each of the 2015 – 2019 datasets using Random Forest classifier. We use Random Forest classifier as an example to demonstrate the effectiveness of our new features, but other classifiers can potentially work as well. We will compare results between different classifiers in Section 5.3.

We evaluate true positive rate (TPR) and false positive rate (FPR) for the classification, where TPR measures the percentage of malicious processes being correctly detected, and FPR measures the percentage of benign processes being incorrectly marked as malicious. We use the receiver operating characteristic (ROC) curve to show the relationship between true positive and false positive, and the precision-recall curve to show the relationship between false positive and false negative.

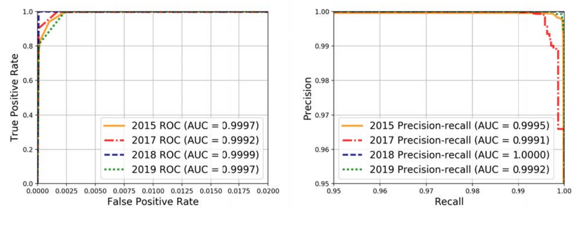


Figure 9: ROC curve and Precision-Recall curve with ten-fold cross validation using new features.

Results. Figure 9 shows the ROC curve and precision-recall curve from ten-fold cross-validation using Random Forest for each dataset after using SMOTE technique to balance the benign and malicious samples. We can see that both ROC AUC and precision-recall AUC are > 0.99 for all four datasets. We also compare the results using data with and without SMOTE oversampling in Table 5. We include the true positive rate (TPR) and false positive rate (FPR) at an "optimal" point (where the difference between TPR and FPR is the largest). We can see that there is very negligible difference between using data with and without SMOTE oversampling. In addition, high TPR ($> 99\%$) can be achieved with very low FPR ($< 0.1\%$) for most of the datasets.

Year	SMOTE	TPR	FPR	ROC AUC
2015	No	98.5%	0.1%	0.9952
	Yes	99.9%	0.08%	0.9997
2017	No	99.4%	0.05%	0.9938
	Yes	99.7%	0.08%	0.9992
2018	No	100%	0.02%	0.9998
	Yes	100%%	0.01%	0.9999
2019	No	99.97%	0.05%	0.9996
	Yes	99.95%	0.07%	0.9997

TABLE 5: TPR and FPR for each dataset with ten-fold cross validation, with and without SMOTE oversampling.

5.3. Comparison with Previous Works

In this section, we perform detection by using the 2015 & 2017 datasets as training data and the 2018 & 2019 datasets as testing data, *without* SMOTE oversampling, to reflect real-world scenario. We also compare detection results with previously-proposed features.

Previous works have proposed various process-based and network-based features for malware and malicious DNS detection [7], [24], [13], [51], [38], [47], [6], [5], [25]. Recently, Sivakorn et al. proposed new integrated features that combine process-level and network-level information [54]. However, none of these previous works target malware-injected processes, where the process-level information can be indistinguishable (i.e., benign programs are invoked and perform all the DNS activities) and benign domains are involved (e.g., benign web services can be exploited to carry out attacks). The previously-proposed features mostly consider process- and network-level information separately, and the recently-proposed integrated features [54] only consider each process independently without capturing the Program-DNS behavior across processes. In this section, we will perform detection on malware-injected processes and compare the results using two sets of features: (1) *only previous-proposed features* (43 features in total), and (2) *only our new features* (6 features in total).

5.3.1. Previously-proposed features. We extract 43 features in total based on previous works [7], [54], [24], [38], [13] from various feature categories. Recent work has also combined these features to perform detection [54]. The feature categories are as following:

1) *Query Name and Domain Name:* this type of features directly measures the characteristics of DNS query names, including the number of queries/domains, number of distinct queries/domains, and query name patterns. For instance, benign query names are likely to have shorter and less random names compared to malicious query names. The features have been shown to be effective in detecting DGA-based malware [7] and have also been used in recent work [54]. We include 22 features from this category.
2) *Timing information:* this type of features includes domain registration duration (time between current time and the domain's initial registration time), domain renewal duration (time between the domain's updated registration time and its expiration time), and domain TTL ("Time To Live", which specifies how long each domain should be cached by a local DNS server). These features have been shown to be very effective in previous works [24], [38], [54] due to the fact that malicious domains tend to be fresher and newer because of frequent blocking, registered for a short duration to lower the cost, and have a shorter TTL to enable frequent change of IP addresses. We include 9 features from this category.
3) *Registrant information:* malware may query domains registered by a large number of distinct registrants due to its need to frequently change registrants to avoid blocking. These features have been successfully used in previous works [5], [54]. We include 5 features from this category.
4) *Location information:* this mainly includes country location where the domain is registered. Malicious domains tend to be registered in certain countries and may also frequently change locations. These features have been effectively used in [13], [6], [54]. We include 2 features from this category.
5) *Process-based and Integrated Features:* while the above features focus on DNS queries, process-based features such as the program signer (if any) have also been found useful. Most recently, integrated features that combine DNS-based and process-based information have been proposed, where the program signer and domain registrant of the domains queried by the processes are compared for similarity [54]. We include 5 features from this category.

The full list of these 43 features can be found in Appendix A. Note that we did not include features such as the query resolve failure rate or features that depend on the answer of the query. This is due to the potentially high failure rate of DNS queries from older malware samples. For instance, malicious domains built into malware may be active for a very short period of time (e.g., couple of weeks), and by the time we obtain and run the malware sample, its malicious servers have been taken down and its subsequent DNS queries would result in failure. Although we have done our best to search for fresher malware samples and run them as soon as possible, including such features can heavily bias our detection result and does not accurately represent the real scenario when the malware is still active. Thus, we exclude such DNS answer-based features from our detection. On the other hand, we do include features that can be derived from the DNS query itself and/or WHOIS server, e.g., domain registration information, which is relatively more stable and long lasting.

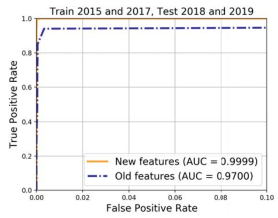
5.3.2. Evaluation Results. To mimic real-world scenarios, we use the 2015 and 2017 datasets as training data for the classifier, and then use the 2018 and 2019 datasets as testing data. We do not perform any oversampling on the data for either the training or testing phase. Our testing data contains 8510 benign processes (negative) and 3217 malicious processes (positive). We compare the detection performance between two sets of features: (1) only our new features (6 features in total), and (2) only previously-proposed features (43 features in total). We train the model using Random Forest and K-Nearest Neighbors classifiers, respectively, which were found to be the most effective classifiers in previous works [38], [13], [6], [54].

Figure 10 shows the ROC curves and precision-recall curves, where Figure 10a and 10b correspond to the Random Forest classifier and Figure 10c and 10d correspond to the KNN classifier. Note that the ROC curves show false positive rates in the range of $[0, 0.1]$ and the precision-recall curves display both precision and recall values within $[0.9, 1]$. We can see that our new features have high AUC values > 0.99 and outperform the previously-proposed features for both classifiers. The previously-proposed features, while performing well with the Random Forest classifier, show a drop in performance with the KNN classifier. We also tried the SVM with linear kernel classifier and the Logical Regression classifier, for which our new features achieve ROC AUC of 0.99 and 0.96, respectively; on the contrary, previously-proposed features only achieve AUC of 0.80 and 0.54, respectively.

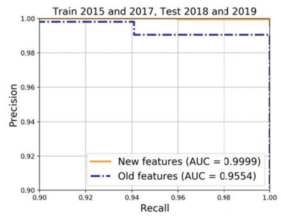
In summary, our new features outperform the previously-proposed features and maintain high accuracy rate regardless of the classifier in use.

5.3.3. False Positive and False Negative analyses. Next, we take a further look into the tradeoff between true positive rate (TPR) and false positive rate (FPR), as well as detailed false positive (FP) and false negative (FN) cases. We will focus on detection results using the Random Forest classifier, given that it is the best-performing classifier for both new and previously-proposed features.

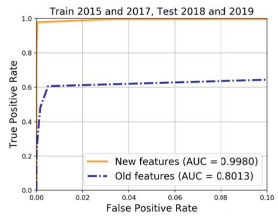
Results. Table 6 shows the results. The table does not include all TPR/FPR points, but rather three representative points for each features set: (1) 0% FPR point with the corresponding TPR (highest value); (2) lowest non-zero FPR point with the corresponding TPR (highest value); (3) highest TPR point, where FPR < 1 (otherwise if FPR $= 1$, then it's trivial that TPR will also be 1). We can see that at 0% FPR, our new features achieve 94.3% TPR compared to 25.9% TPR with previously-proposed features – this is 12X reduction on false negative rates. Similarly, at 0.012% FPR, our new features achieve 99.5% TPR compared to 37.4% TPR with previously-proposed features. Finally, our new features can achieve 100% TPR with only 0.15% FPR; on the contrary, previously-proposed features can only achieve up to maximum 94.1%



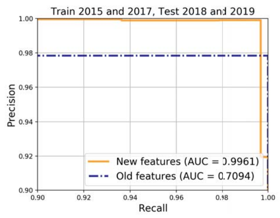
(a) ROC curve with Random Forest Classifier.



(b) Precision-recall curve with Random Forest Classifier.



(c) ROC curve with KNN Classifier.



(d) Precision-recall curve with KNN Classifier.

Figure 10: Comparison between our new features and previously-proposed features using Random Forest and K-Nearest Neighbors classifiers, with 2015 & 2017 datasets as training data and 2018 & 2019 datasets as testing data.

Features	TPR	FPR	FP Programs	FN Programs
New	**94.3%**	0%	-	svchost(182)
	99.5%	0.012%	ping(1)	svchost(16)
	100%	0.15%	ping(1), svchost(12)	-
Previously Proposed	**25.9%**	0%	-	wscript(1), notepad(3), ping(6), svchost(2375)
	37.4%	0.012%	main(1)	notepad(2), ping(6), svchost(2006)
	94.1%	0.33%	ping(1), notepad(2), explorer(2), main(3), svchost(20)	svchost(190)

TABLE 6: Comparison of TPR and FPR between new and previously-proposed features. FP programs are programs whose processes are misclassified as malicious, where the number inside the parentheses indicates the number of misclassified processes of the program. Similarly, FN programs are programs whose processes are misclassified as benign.

TPR, indicating that the remaining 5.9% of the malware-injected processes completely fail to be detected using only previously-proposed features.

Deeper look into the processes. We can see from Table 6 that 5.9% of the malicious processes (190 in total, all svchost processes) are indistinguishable from benign processes using only previously-proposed features, while all of them can be successfully detected by our new features. We take a deeper look into the queries from these processes. We found that all the processes query seemingly-benign domains. For instance, 64 processes query various blog posting websites, and the following is an example of the queries sent by an svchost process:
1.bp.blogspot.com, 2.bp.blogspot.com, blogblog.com, img1.blogblog.com, pagead2.googlesyndication.com, www.blogblog.com, www.blogger.com, www.facebook.com
While the blog posting sites are legitimate themselves, they are known to be vulnerable to malware infection, resulting in malicious content hosted on the site. However, from the DNS-based point of view and the process-based point of view, the activities are indistinguishable from benign activities, which is why these stealthy malware-injected processes fail to be detected by using previously-proposed features. However, with our new Program-DNS profile-based features, we can detect when the process' behavior deviates from its "expected" behavior – in this case, svchost processes are *not* expected to query blog positing domains and thus these abnormal activities are successfully flagged by our new features.

Key Takeaway: Our new features achieve 94.3% True Positive Rate with 0% False Positive Rate, compared to 25.9% True Positive Rate with previously-proposed features. Our new features can also successfully detect all malware-injected processes with only 0.15% False Positives Rate, while previously-proposed features completely fail to detect 5.9% of malware-injected processes.

5.4. Case Study on POSHSPY

In this section, we experiment with a stealthy strain of malware – POSHSPY. As described in Sec 2, POSHSPY employs a Fileless scheme to hide its backdoor logic as one of WMIC events which contains PowerShell payload. To evade network monitoring, the malware uses Meek Tor plug-in that connects to a Tor entry relay which is designated as one of Google Cloud Services and would eventually lead to C&C.

To perform the experiment, we downloaded the payload of POSHSPY [26]. We launched the malware from an isolated environment and captured the DNS activities from the environment. We configured the malware to connect to domains including google.com and dropbox.com, which were used in previous real-world attacks.

From the captured data, we observed that a powershell process was injected by the malware and sending queries on the malware's behalf, querying dropbox.com four times and google.com two times. We then used the trained model in Section 5.3 with 2015 and 2017 datasets to perform detection. Our new features successfully detected the malware-injected powershell process, while previously-proposed features failed due to the seemingly-benign process information and DNS activities.

6. Related Work and Discussion

In this section, we review existing DNS and malware detection systems, including network-based solutions, host-based solutions, and a recently-proposed process-level detection system that combines both network- and process-level information. Then, we describe two strains of malware that inject malicious logic to benign programs and establish C&C channels through connections to legitimate domains. Finally, we discuss challenges on injecting malware logic to interactive programs, such as browsers.

6.1. Existing Detection Systems

Network-based DNS detection systems. Monitoring and analyzing the DNS traffic of Internet hosts is the first line of defense against fast-flux networks, bots, DGA (Domain Generation Algorithms) domains and spammers. Numerous works [61], [5], [13], [6], [7], [15], [48], [49] have been proposed to analyze DNS activities on DNS servers at different levels of DNS hierarchies, aiming to detect malicious domain names that are associated with various attacks. A survey [9] provided an overview of the state-of-the-art network-based detection systems (including other types of network traffic, e.g., packet, flow size). However, the key difference between these approaches and ours is that they all focus on detecting malicious domains. This will not be effective against stealthy attacks where *legitimate and benign* web services and domains are being leveraged by attackers to carry out attacks.

Host-based detection systems. Host-based detection systems aim to detect malicious programs and processes, as opposed to malicious domain names. Numerous works have proposed to detect malware through static analysis [16], [55] via disassembly or reverse engineering. However, malware may very likely create instances that can avoid being detected by these techniques [46], [19], [64]. Due to the limitation of static analysis, dynamic analysis techniques have been proposed, where the detection systems are built from observing the malware behaviors while the malware is being executed (e.g., function call, information flow tracking [63]). Studying malware behaviors under a restricted environment have also been proposed in [11], [56], [53], [62].
While advanced host-based detection techniques can also be effective, they are orthogonal to our approach and generally involve extensive system logging. We aim to provide a more light-weight, but effective approach that collects fewer data fields and focuses on Program-DNS behaviors. Our approach and host-based detection techniques can be complementary to each other.

Integrated network-based and host-based detection systems. Recently, Sivakorn et al. [54] proposed PDNS, a process-level detection system that combines network-based features (e.g., DNS query name and answer) with process-based features (e.g., code signing) to perform detection at the process-level. The system enriches the context of monitoring data, and proposes new integrated features by combining domain information and program information. However, this system analyzes each process *independently*, without developing a fine-grained Program-DNS profile at the program level by analyzing processes of the same program *jointly*. As a result, the system is not effective at detecting stealthy attacks where DNS queries are delegated via legitimate programs using cross-process injection techniques [2], [43].

6.2. Interactive Program as a Target for Injection

In Section 4, we built Program-DNS profile for interactive programs, including browsers and mail clients. Although our metrics can still distinguish malware-injected processes from benign processes for interactive programs, the DNS behaviors of these programs are intrinsically harder to model due to their user-interactive nature. Thus, interactive programs may seem to be perfect targets for malware injection to increase stealthiness of the attack.

However, the attacker also needs to take risks in choosing highly interactive programs as its injection target. The attacker cannot have a stable expectation for the existence or lifetime for its injection target and any irregular behavior caused by malicious logic is more noticeable to the user. For stealthiness and persistence, the attacker may prefer to choose long-running background processes as its target [42].

Yet, there exist injection attacks that specifically targets browser programs [2], aiming to steal sensitive user information (e.g., passwords, credit card numbers) stored in browser's memory space. To respond, Browser vendors (Google Chrome and Mozilla Firefox) have come up with in-browser defense measure that block injection attempts by other processes [3], [4].

6.3. Generalizability and Limitation

Our Program-DNS profiling approach is generally applicable to different types of environments. However, the resulting profiles could be different. Thus, we should rebuild the profile and retrain the model, instead of directly applying the trained model from our corporate environment to another environment, such as home users or data centers. Alternatively, we may employ domain adaptation techniques to transfer our trained model to another environment.

While our system is not specifically tailored against an adaptive attacker, it does raise the costs for adaptive attackers targeting our system. For example, if the attackers want to evade detection on injection into programs with highly consistent behaviors, e.g., never querying "nonowner" domains, they have to either move the C&C server to the "expected" domains, which are usually a limited set of domains and often high-profile domains such as microsoft.com, or choose other programs to inject.

7. Conclusion

In this paper, we proposed novel metrics to build fine-grained Program-DNS profile for benign programs. We deployed a data collection system on 126 enterprise end-host machines and collected over 130 million DNS requests along with the program- and process- level information. We showed that malware-injected processes and benign processes from the same program have significant difference in their DNS behavior, even when program- and process-level information are indistinguishable and the queried domains are legitimate and benign. We demonstrated the effectiveness of such Program-DNS profile in detecting malware-injected processes by proposing six new features that are based on Program-DNS profiles. By evaluating across malware datasets ranging from 2015 to 2019, we showed that our new features are more effective at detecting malware-injected processes compared to previously-proposed features. Overall, our work sheds light on leveraging program-level behavior to detect malware injection, and inspires future ideas for defending against increasingly complex malware.

References

[1] Attacks Leveraging Web Services. https://attack.mitre.org/techniques/T1102/.

[2] Attacks with Process Injection. https://attack.mitre.org/wiki/Technique/T1055.

[3] (2019) Firefox will block DLL Injections. https://www.ghacks.net/2019/01/21/firefox-will-block-dll-injections/.

[4] (2019) Google Chrome 72's Code Injection Blocking Detailed. https://news.softpedia.com/news/google-chrome-72-s-code-injection-blocking-detailed-524759.shtml.

[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, pp. 273–290.

[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, pp. 1–16.

[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, pp. 24–24.

[8] Y. Assor and A. Slotky. (2016) Anti-VM and Anti-Sandbox Explained – CYBERBIT. https://www.cyberbit.com/anti-vm-and-anti-sandbox-explained/.

[9] 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, pp. 807–822.

[10] A. Bates, D. J. Tian, K. R. Butler, and T. Moyer, "Trustworthy whole-system provenance for the linux kernel," in *USENIX Security Symposium)*, 2015.

[11] U. Bayer, C. Kruegel, and E. Kirda, *TTAnalyze: A Tool for Analyzing Malware*, 2006.

[12] S. B. Bhatkar, S. Nanda, and J. S. Wilhelm, "Techniques for behavior based malware analysis," Oct. 8 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, pp. 14:1–14:28.

[14] 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, pp. 321–357, 2002.

[15] H. Choi, H. Lee, H. Lee, and H. Kim, "Botnet Detection by Monitoring Group Activities in DNS Traffic," in *Computer and Information Technology, 2007. CIT 2007. 7th IEEE International Conference on*, 2007, pp. 715–720.

[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] 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, p. 6, 2012.

[20] Empire. Empire powershell post-exploitation agent. http://www.powershellempire.com.

[21] Endgame. ten process injection techniques. https://bit.ly/32R7cD7.

[22] ESET. (2017) Gazing at Gazer. https://www.welivesecurity.com/wp-content/uploads/2017/08/eset-gazer.pdf.

[23] ESET. (2018) Diplomats in Eastern Europe bitten by a Turla mosquito. https://www.welivesecurity.com/wp-content/uploads/2018/01/ESET_Turla_Mosquito.pdf.

[24] 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, pp. 6–6.

[25] I. Fette, N. Sadeh, and A. Tomasic, "Learning to Detect Phishing Emails," in *Proceedings of the International Conference on World Wide Web*, 2007, pp. 649–656.

[26] FireEye. (2015, July) Dissecting One of APT29's Fileless WMI and PowerShell Backdoors (POSHSPY). https://www.fireeye.com/blog/threat-research/2017/03/dissecting_one_ofap.html.

[27] FireEye. (2015, July) HammerToss: Stealthy Tactics Define a Russian Cyber Threat Group. https://www2.fireeye.com/rs/848-DID-242/images/rpt-apt29-hammertoss.pdf.

[28] Forbes. (2019) Flying Under The Radar: The Biggest Malware Threats Hiding In Plain Sight. https://www.forbes.com/sites/forbestechcouncil/2019/05/06/flying-under-the-radar-the-biggest-malware-threats-hiding-in-plain-sight/#12c6f0eb19bd.

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

[30] P. Gao, X. Xiao, D. Li, Z. Li, K. Jee, Z. Wu, C. H. Kim, S. R. Kulkarni, and P. Mittal, "SAQL: A Stream-based Query System for Real-Time Abnormal System Behavior Detection," in *USENIX Security Symposium)*, 2018.

[31] B. Hawkins, "Case study: The home depot data breach," SANS Institute, Tech. Rep., 2015.

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

[33] S. T. King and P. M. Chen, "Backtracking intrusions," in *ACM Symposium on Operating Systems Principles (SOSP)*, 2003.

[34] S. T. King, Z. M. Mao, D. G. Lucchetti, and P. M. Chen, "Enriching Intrusion Alerts Through Multi-Host Causality." in *NDSS*, 2005.

[35] K. H. Lee, X. Zhang, and D. Xu, "High Accuracy Attack Provenance via Binary-based Execution Partition." in *NDSS*, 2013.

[36] K. H. Lee, X. Zhang, and D. Xu, "LogGC: garbage collecting audit log," in *ACM SIGSAC conference on Computer & communications security (CCS)*, 2013.

[37] Y. Liu, M. Zhang, D. Li, K. Jee, Z. Li, Z. Wu, J. Rhee, and P. Mittal, "Towards a Timely Causality Analysis for Enterprise Security." in *NDSS*, 2018.

[38] 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, pp. 1245–1254.

[39] S. Ma, K. H. Lee, C. H. Kim, J. Rhee, X. Zhang, and D. Xu, "Accurate, low cost and instrumentation-free security audit logging for windows," in *Annual Computer Security Applications Conference (ACSAC)*, 2015.

[40] S. Ma, X. Zhang, and D. Xu, "Protracer: Towards Practical Provenance Tracing by Alternating Between Logging and Tainting." in *NDSS*, 2016.

[41] McAfee. What is Fileless Malware? https://www.mcafee.com/enterprise/en-us/security-awareness/ransomware/what-is-fileless-malware.html.

[42] Microsoft. Detecting stealthier cross-process injection techniques with Windows Defender ATP: Process hollowing and atom bombing. https://bit.ly/2NQL9Is.

[43] Microsoft. Uncovering cross-process injection with Windows Defender ATP. https://www.microsoft.com/security/blog/2017/03/08/uncovering-cross-process-injection-with-windows-defender-atp/.

[44] MITRE. Kazuar. https://attack.mitre.org/software/S0265/.

[45] MITRE. Turla. https://attack.mitre.org/groups/G0010/.

[46] A. Moser, C. Kruegel, and E. Kirda, "Limits of Static Analysis for Malware Detection," in *Computer security applications conference, 2007. ACSAC 2007. Twenty-third annual*, 2007, pp. 421–430.

[47] T. Nelms, R. Perdisci, M. Antonakakis, and M. Ahamad, "WebWitness: Investigating, Categorizing, and Mitigating Malware Download Paths," in *Proceedings of the USENIX Security Symposium*, 2015, pp. 1025–1040.

[48] 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, pp. 311–320.

[49] 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, pp. 263–278.

[50] T. Radichel, "Case study: Critical controls that could have prevented target breach," SANS Institute, Tech. Rep., 2013.

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

[52] Security Boulevard. (2019) Fileless Malware on the Rise. https://securityboulevard.com/2019/10/fileless-malware-on-the-rise/.

[53] 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, pp. 94–109.

[54] S. Sivakorn, K. Jee, Y. Sun, L. Kort-Parn, Z. Li, C. Lumezanu, Z. Wu, L.-A. Tang, and D. Li, "Countering malicious processes with process-dns association." in *NDSS*, 2019.

[55] 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, pp. 1–25.

[56] 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, pp. 15–pp.

[57] VirusShare. https://virusshare.com/.

[58] VirusSign. http://www.virussign.com/.

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

[60] VXVault. http://vxvault.net/.

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

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

[63] H. Yin, D. Song, M. Egele, C. Kruegel, and E. Kirda, "Panorama: Capturing System-wide Information Flow for Malware Detection and Analysis," in *Proceedings of the 14th ACM conference on Computer and communications security*, 2007, pp. 116–127.

[64] I. You and K. Yim, "Malware Obfuscation Techniques: A Brief Survey," in *Proceedings of International Conference on Broadband, Wireless Computing, Communication and Applications*, 2010, pp. 297–300.

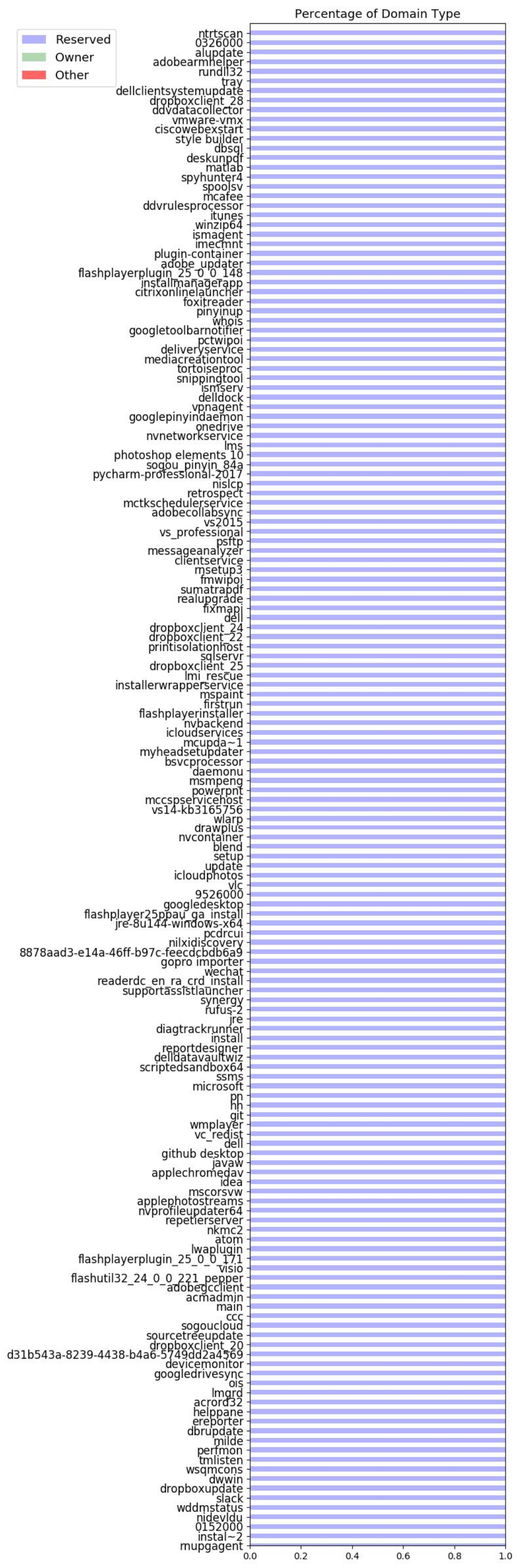
Appendix

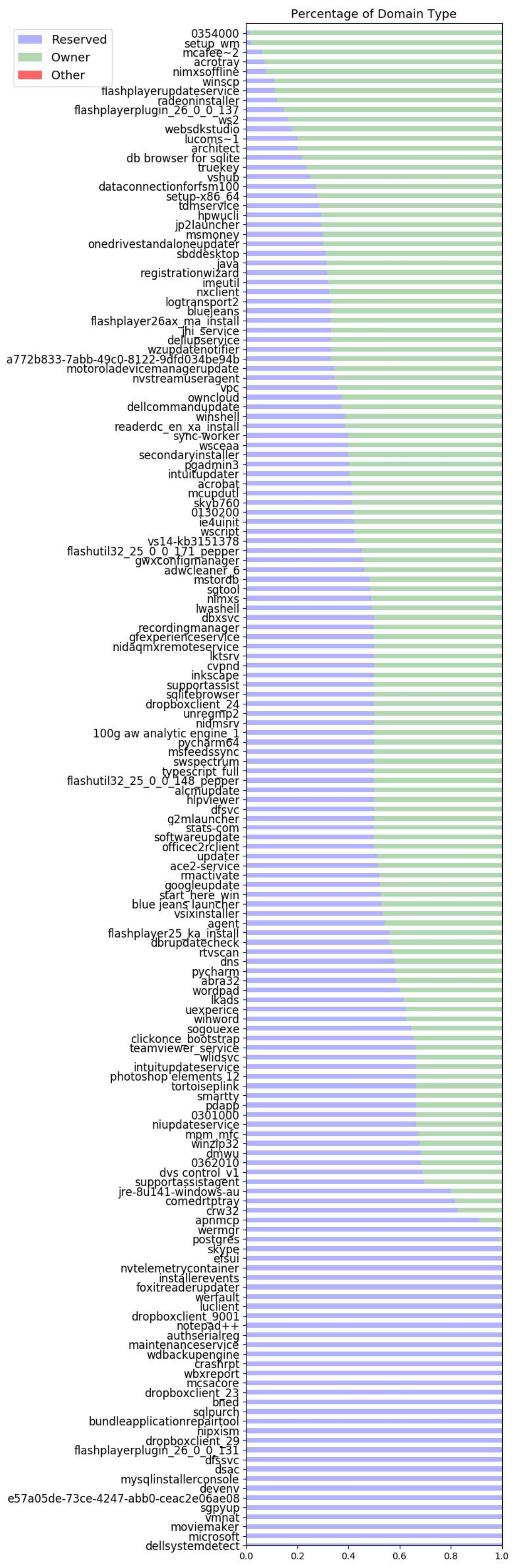
Table 7 lists the 43 features that we extract from previous works.

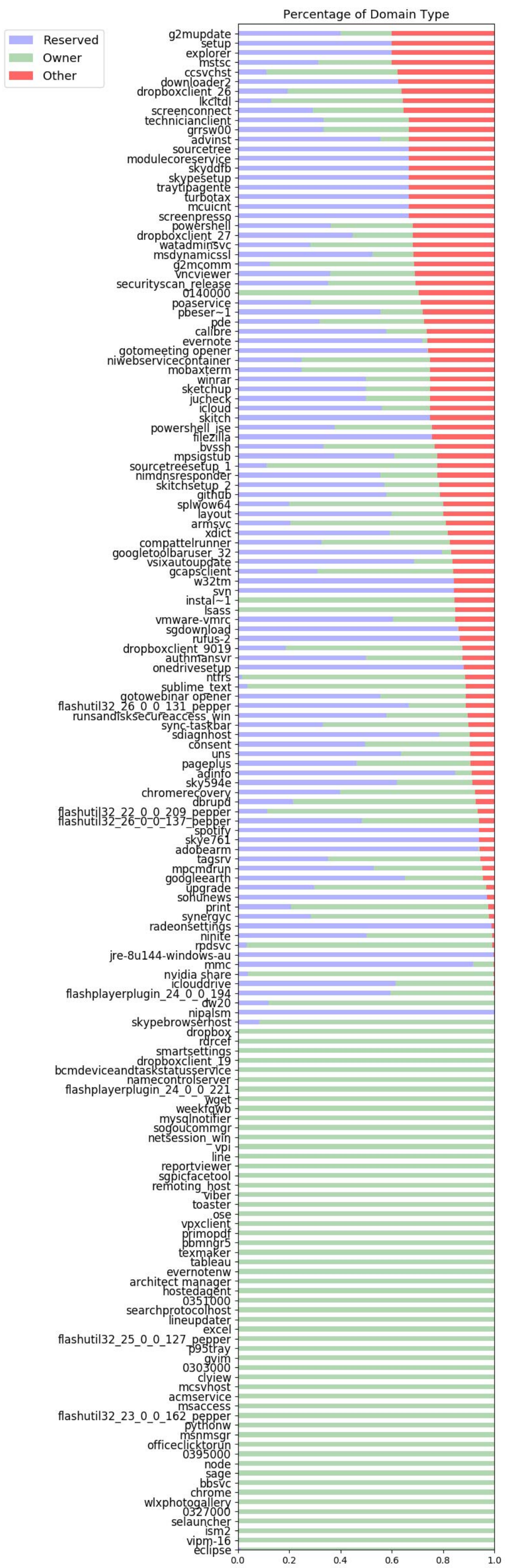
We also show the complete domain type distribution for all 636 non-browser benign programs in our dataset in the following.

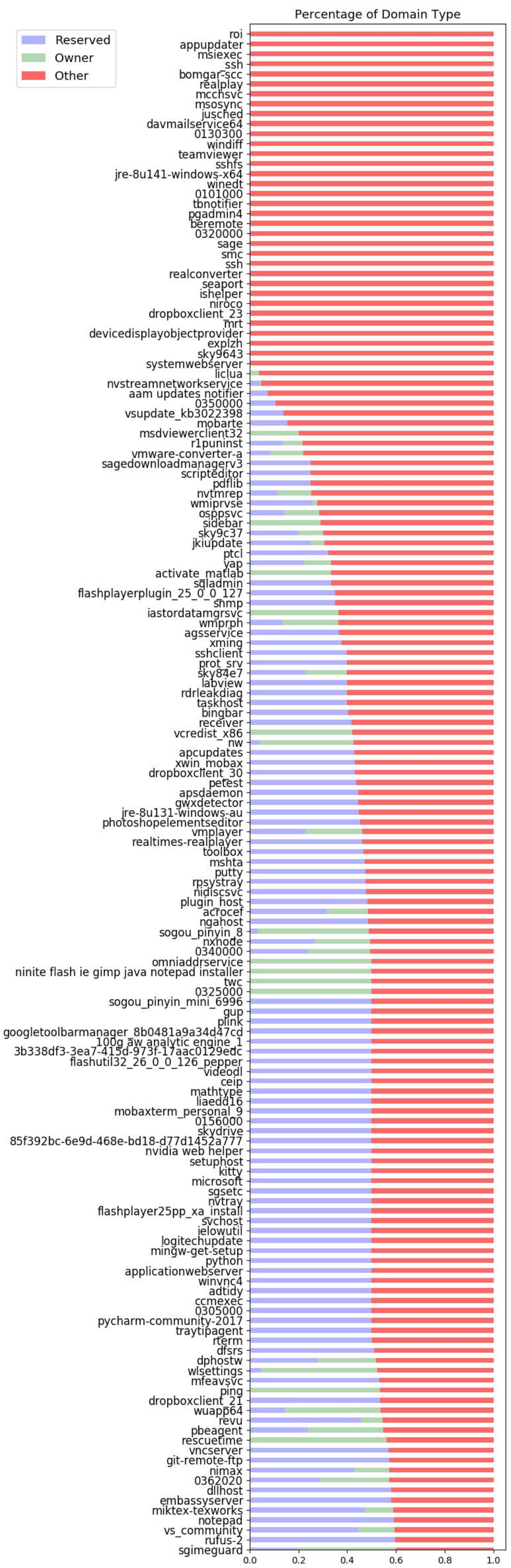
Feature Type	Feature Name
Query Name and Domain Name	Number of domains
	Number of distinct domains
	Number of domains (normalized)
	Number of distinct domains (normalized)
	Ratio of distinct domains
	Average domain name entropy
	Median domain name entropy
	SD of domain name entropy
	Average domain length
	Median domain length
	SD of domain length
	Number of queries
	Number of distinct queries
	Number of queries (normalized)
	Number of distinct queries (normalized)
	Ratio of distinct queries
	Average query name entropy
	Median query name entropy
	SD of query name entropy
	Average query length
	Median query length
	SD of query length
Timing Info	Average domain TTL
	Median domain TTL
	SD of domain TTL
	Average registered duration
	Median registered duration
	SD of registered duration
	Average renewed duration
	Median renewed duration
	SD of renewed duration
Registrant Info	Number of registrants
	Number of distinct registrants
	Number of registrants (normalized)
	Number of distinct registrants (normalized)
	Ratio of distinct registrants
Location Info	Number of distinct countries
	Ratio of distinct countries
Process-based and Integrated Features	Whether program is signed
	Similarity score, full string (signer v.s. registrant, setup time)
	Similarity score, partial string (signer v.s. registrant, setup time)
	Similarity score, full string (signer v.s. registrant, all time)
	Similarity score, partial string (signer v.s. registrant, all time)

TABLE 7: List of 43 previously-proposed features of various types.


Percentage of Domain Type
Reserved
Owner
Other
ntrtscan
0326000
alupdate
adobearmhelper
rundll32
tray
dellclientsystemupdate
dropboxclient_28
ddvdatacollector
vmware-vmx
ciscowebexstart
style builder
dbsql
deskunpdf
matlab
spyhunter4
spoolsv
mcafee
ddvrulesprocessor
itunes
winzip64
ismagent
imecmnt
plugin-container
adobe_updater
flashplayerplugin_25_0_0_148
installmanagerapp
citrixonlinelauncher
foxitreader
pinyinup
whois
googletoolbarnotifier
pctwipoi
deliveryservice
mediacreationtool
tortoiseproc
snippingtool
ismserv
delldock
vpnagent
googlepinyindaemon
onedrive
nvnetworkservice
lms
photoshop elements 10
sogou_pinyin_84a
pycharm-professional-2017
nislcp
retrospect
mctkschedulerservice
adobecollabsync
vs2015
vs_professional
psftp
messageanalyzer
clientservice
msetup3
fmwipoi
sumatrapdf
realupgrade
fixmapi
dell
dropboxclient_24
dropboxclient_22
printisolationhost
sqlservr
dropboxclient_25
lmi_rescue
installerwrapperservice
mspaint
firstrun
flashplayerinstaller
nvbackend
icloudservices
mcupda~1
myheadsetupdater
bsvcprocessor
daemonu
msmpeng
powerpnt
mccspservicehost
vs14-kb3165756
wlarp
drawplus
nvcontainer
blend
setup
update
icloudphotos
vlc
9526000
googledesktop
flashplayer25ppau_ga_install
jre-8u144-windows-x64
pcdrcui
nilxidiscovery
8878aad3-e14a-46ff-b97c-feecdcbdb6a9
gopro importer
wechat
readerdc_en_ra_crd_install
supportassistlauncher
synergy
rufus-2
jre
diagtrackrunner
install
reportdesigner
delldatavaultwiz
scriptedsandbox64
ssms
microsoft
pn
hh
git
wmplayer
vc_redist
dell
github desktop
javaw
applechromedav
idea
mscorsvw
applephotostreams
nvprofileupdater64
repetierserver
nkmc2
atom
lwaplugin
flashplayerplugin_25_0_0_171
visio
flashutil32_24_0_0_221_pepper
adobegcclient
acmadmin
main
ccc
sogoucloud
sourcetreeupdate
dropboxclient_20
d31b543a-8239-4438-b4a6-5749dd2a4569
devicemonitor
googledrivesync
ois
lmgrd
acrord32
helppane
ereporter
dbrupdate
milde
perfmon
tmlisten
wsqmcons
dwwin
dropboxupdate
slack
wddmstatus
nidevldu
0152000
instal~2
rmupgagent
0.0
0.2
0.4
0.6
0.8
1.0


Percentage of Domain Type
Reserved
Owner
Other
0354000
setup_wm
mcafee~2
acrotray
nimxsoffline
winscp
flashplayerupdateservice
radeoninstaller
flashplayerplugin_26_0_0_137
ws2
websdkstudio
lucoms~1
architect
db browser for sqlite
truekey
vshub
dataconnectionforfsm100
setup-x86_64
tdmservice
hpwucli
jp2launcher
msmoney
onedrivestandaloneupdater
sbddesktop
java
registrationwizard
imeutil
nxclient
logtransport2
bluejeans
flashplayer26ax_ma_install
jhi_service
dellupservice
wzupdatenotifier
a772b833-7abb-49c0-8122-9dfd034be94b
motoroladevicemanagerupdate
nvstreamuseragent
vpc
owncloud
dellcommandupdate
winshell
readerdc_en_xa_install
sync-worker
wsceaa
secondaryinstaller
pgadmin3
intuitupdater
acrobat
mcupdutl
skyb760
0130200
ie4uinit
wscript
vs14-kb3151378
flashutil32_25_0_0_171_pepper
gwxconfigmanager
adwcleaner_6
mstordb
sgtool
nimxs
lwashell
dbxsvc
recordingmanager
gfexperienceservice
nidaqmxremoteservice
lktsrv
cvpnd
inkscape
supportassist
sqlitebrowser
dropboxclient_24
unregmp2
nidmsrv
100g aw analytic engine_1
pycharm64
msfeedssync
swspectrum
typescript_full
flashutil32_25_0_0_148_pepper
alcmupdate
hlpviewer
dfsvc
g2mlauncher
stats-com
softwareupdate
officec2rclient
updater
ace2-service
rmactivate
googleupdate
start_here_win
blue jeans launcher
vsixinstaller
agent
flashplayer25_ka_install
dbrupdatecheck
rtvscan
dns
pycharm
abra32
wordpad
lkads
uexperice
winword
sogouexe
clickonce_bootstrap
teamviewer_service
wlidsvc
intuitupdateservice
photoshop elements 12
tortoiseplink
smartty
pdapp
0301000
niupdateservice
mpm_mfc
winzip32
dmwu
0362010
dvs control_v1
supportassistagent
jre-8u141-windows-au
comedrtptray
crw32
apnmcp
wermgr
postgres
skype
efsui
nvtelemetrycontainer
installerevents
foxitreaderupdater
werfault
luclient
dropboxclient_9001
notepad++
authserialreg
maintenanceservice
wdbackupengine
crashrpt
wbxreport
mcsacore
dropboxclient_23
bned
sqlpurch
bundleapplicationrepairtool
hipxism
dropboxclient_29
flashplayerplugin_26_0_0_131
dfssvc
dsac
mysqlinstallerconsole
devenv
e57a05de-73ce-4247-abb0-ceac2e06ae08
sgpyup
vmnat
moviemaker
microsoft
dellsystemdetect
0.0
0.2
0.4
0.6
0.8
1.0


Percentage of Domain Type
Reserved
Owner
Other
g2mupdate
setup
explorer
mstsc
ccsvchst
downloader2
dropboxclient_26
lkcltdl
screenconnect
technicianclient
grrsw00
advinst
sourcetree
modulecoreservice
skyddfb
skypesetup
traytipagente
turbotax
mcuicnt
screenpresso
powershell
dropboxclient_27
watadminsvc
msdynamicssl
g2mcomm
vncviewer
securityscan_release
0140000
poaservice
pbeser~1
pde
calibre
evernote
gotomeeting opener
niwebservicecontainer
mobaxterm
winrar
sketchup
jucheck
icloud
skitch
powershell_ise
filezilla
bvssh
mpsigstub
sourcetreesetup_1
nimdnsresponder
skitchsetup_2
github
splwow64
layout
armsvc
xdict
compattelrunner
googletoolbaruser_32
vsixautoupdate
gcapsclient
w32tm
svn
instal~1
lsass
vmware-vmrc
sgdownload
rufus-2
dropboxclient_9019
authmansvr
onedrivesetup
ntfrs
sublime_text
gotowebinar opener
flashutil32_26_0_0_131_pepper
runsandisksecureaccess_win
sync-taskbar
sdiagnhost
consent
uns
pageplus
adinfo
sky594e
chromerecovery
dbrupd
flashutil32_22_0_0_209_pepper
flashutil32_26_0_0_137_pepper
spotify
skye761
adobearm
tagsrv
mpcmdrun
googleearth
upgrade
sohunews
print
synergyc
radeonsettings
ninite
rpdsvc
jre-8u144-windows-au
mmc
nvidia share
iclouddrive
flashplayerplugin_24_0_0_194
dw20
nipalsm
skypebrowserhost
dropbox
rdrcef
smartsettings
dropboxclient_19
bcmdeviceandtaskstatusservice
namecontrolserver
flashplayerplugin_24_0_0_221
wget
weekfqwb
mysqlnotifier
sogoucommgr
netsession_win
vpi
line
reportviewer
sgpicfacetool
remoting_host
viber
toaster
ose
vpxclient
primopdf
pbmngr5
texmaker
tableau
evernotenw
architect manager
hostedagent
0351000
searchprotocolhost
lineupdater
excel
flashutil32_25_0_0_127_pepper
p95tray
gvim
0303000
clview
mcsvhost
acmservice
msaccess
flashutil32_23_0_0_162_pepper
pythonw
msnmsgr
officeclicktorun
0395000
node
sage
bbsvc
chrome
wlxphotogallery
0327000
selauncher
ism2
vipm-16
eclipse
0.0
0.2
0.4
0.6
0.8
1.0


Percentage of Domain Type
Reserved
Owner
Other
roi
appupdater
msiexec
ssh
bomgar-scc
realplay
mcchsvc
msosync
jusched
davmailservice64
0130300
windiff
teamviewer
sshfs
jre-8u141-windows-x64
winedt
0101000
tbnotifier
pgadmin4
beremote
0320000
sage
smc
realconverter
seaport
ishelper
niroco
dropboxclient_23
mrt
devicedisplayobjectprovider
explzh
sky9643
systemwebserver
liclua
nvstreamnetworkservice
aam updates notifier
0350000
vsupdate_kb3022398
mobarte
msdviewerclient32
r1puninst
vmware-converter-a
sagedownloadmanagerv3
scripteditor
pdflib
nvtmrep
wmiprvse
osppsvc
sidebar
sky9c37
jkiupdate
ptcl
yap
activate_matlab
sqladmin
flashplayerplugin_25_0_0_127
shmp
iastordatamgrsvc
wmprph
agsservice
xming
sshclient
prot_srv
sky84e7
labview
rdrleakdiag
taskhost
bingbar
receiver
vcredist_x86
nw
apcupdates
xwin_mobax
dropboxclient_30
petest
apsdaemon
gwxdetector
jre-8u131-windows-au
photoshopelementseditor
vmplayer
realtimes-realplayer
toolbox
mshta
putty
rpsystray
nidiscsvc
plugin_host
acrocef
ngahost
sogou_pinyin_8
nxnode
0340000
omniaddrservice
ninite flash ie gimp java notepad installer
twc
0325000
sogou_pinyin_mini_6996
gup
plink
googletoolbarmanager_8b0481a9a34d47cd
100g aw analytic engine_1
3b338df3-3ea7-415d-973f-17aac0129edc
flashutil32_26_0_0_126_pepper
videodl
ceip
mathtype
liaedd16
mobaxterm_personal_9
0156000
skydrive
85f392bc-6e9d-468e-bd18-d77d1452a777
nvidia web helper
setuphost
kitty
microsoft
sgsetc
nvtray
flashplayer25pp_xa_install
svchost
jelowutil
logitechupdate
mingw-get-setup
python
applicationwebserver
winvnc4
adtidy
ccmexec
0305000
pycharm-community-2017
traytipagent
rterm
dfsrs
dphostw
wlsettings
mfeavsvc
ping
dropboxclient_21
wuapp64
revu
pbeagent
rescuetime
vncserver
git-remote-ftp
nimax
0362020
dllhost
embassyserver
miktex-texworks
notepad
vs_community
rufus-2
sgimeguard
0.0
0.2
0.4
0.6
0.8
1.0