IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 4 December 2023, accepted 29 December 2023, date of publication 4 January 2024,
date of current version 19 January 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3349943

RESEARCH ARTICLE

Anteater: Advanced Persistent Threat Detection With Program Network Traffic Behavior

YANGZONG ZHANG[1], WENJIAN LIU[1], KAIIAN KUOK[1], AND NGAI CHEONG[2], (Senior Member, IEEE)
[1]Faculty of Data Science, City University of Macau, Macau, China
[2]Faculty of Applied Sciences, Macao Polytechnic University, Macau, China
Corresponding author: Wenjian Liu (andylau@cityu.edu.mo)

ABSTRACT Recent stealth attacks cleverly disguise malicious activities, masquerading as ordinary connections to popular online services through seemingly innocuous applications. These methods often evade detection by traditional network monitoring or signature-based techniques, as attackers frequently hide Command and Control (C&C) servers within well-known cloud service providers, making the traffic anomalies appear normal. In this paper, we introduce an application-level monitoring system, *Anteater*. *Anteater* constructs a detailed profile for each legitimate software's network traffic behavior, outlining the *expected* traffic patterns. By scrutinizing a program's network traffic configuration, *Anteater* efficiently pinpoints and intercepts the IP addresses associated with abnormal program access. Implemented in a real-world enterprise environment, *Anteater* was tested on a dataset containing over 400 million real-world network traffic sessions. The evaluation results demonstrate that *Anteater* achieves a high detection rate for malware injections, boasting a true positive rate of 94.5% and a false positive rate of less than 0.1%.

INDEX TERMS Malware injection detection, advanced persistent threat, program traffic behavior, network security, Anteater.

I. INTRODUCTION

Malware has emerged as a significant threat to global cybersecurity. The U.S. Cybersecurity and Infrastructure Security Agency (CISA) has reported a notable 62% increase in malware incidents year-over-year from 2021, correlating with a subsequent 20% rise in financial losses [1]. Furthermore, the 2022 1H Global Threat Landscape Report highlights an alarming surge in malware applications, recording an increase of 1,070% from July 2021 to June 2022 [2].

In the realm of cyber attacks, the complexity of threat models is escalating as attackers ingeniously exploit trusted, seemingly harmless applications within client environments. They embed malware into otherwise benign programs, which then initiate legitimate-looking connections to communicate with Command and Control (C&C) servers. To the operating system, these malicious activities are disguised as normal user operations. Despite Microsoft Windows having a robust user access control system, it faces inherent challenges.[1] In the initial versions of Windows Vista, Microsoft implemented stringent security checks to verify the legitimacy of user actions. However, these measures often led to user frustration due to frequent prompts with the *Are you sure?* dialogue box, resulting in users hastily disabling the feature or habitually clicking *OK* without due consideration. This behavior, coupled with the elusive nature of fileless access attacks that leave minimal traces on the host system, leads to defensive shortcomings and a degraded user experience. Consequently, traditional host-based antivirus and network intrusion detection methods face difficulties in pinpointing anomalies associated with applications [3].

Fileless attacks, often categorized under Low Observable Characteristic (LOC) assaults, represent a sophisticated form of cyber threat. Unlike conventional viruses, these stealthy attacks operate directly in the system's memory, eluding most traditional security measures. In a fileless attack, the

The associate editor coordinating the review of this manuscript and approving it for publication was Alessandra De Benedictis.

[1]Malware exploits the user's implicit trust in the Microsoft Windows operating system to operate effectively.

malicious payload never resides on the hard disk; instead, it directly infiltrates the system's memory. This approach allows the malware to evade detection since it does not alter files on the disk. Fileless malware typically leverages trusted, whitelisted applications to initiate its malicious processes, exploiting the inherent trust that security applications place in these whitelisted entities. By doing so, these attacks cleverly bypass the monitoring mechanisms that usually ignore trusted programs [4].

To establish a communication profile for legitimate applications, we gathered over 1 million sets of communication data and associated processes from the *svchost* application in enterprise environments. This data collection was instrumental in developing the *Anteater* network traffic behavior monitoring system. The inspiration for the name 'Anteater' comes from the South American mammal known for extracting ants from tree trunks using its elongated tongue, symbolizing our system's capability to unearth viruses concealed deep within computer systems. To assess *Anteater*'s proficiency in detecting anomalies in communication behavior, we employed *Cobalt Strike* [5] to simulate malicious injections into processes initiated by 11 distinct benign programs. This experimental setup was crucial for training our machine learning algorithms to distinguish between normal and malware-compromised communication patterns of processes.

In this paper, we clarify three fundamental concepts essential for understanding our discussion:

- **Program:** This term refers to a binary executable file that constitutes an application. For example, *Chrome.exe* is a representation of a program.
- **Process:** This is a unique operational instance of a program, distinguished by its own process ID, initiation time, and other specific attributes. When the Chrome application is executed, it generates a process for the *Chrome.exe* program. This process persists until the user exits Chrome. Notably, a single software can spawn multiple processes.
- **Malware-injected process:** This occurs when malicious code is inserted into a process originating from a legitimate software. For instance, a process is deemed malware-injected if malware targets the *Chrome.exe* program and activates harmful operations within it.

In developing the *Anteater* application for monitoring communication behavior, we conducted a comprehensive analysis of the communication patterns of each benign application to establish their *expected* communication characteristics. This led to the creation of a metric system known as conformance ratios.

- We observed that the communication behavior of benign programs is generally stable, with more than 84% of these programs exhibiting conformance ratios exceeding 0.96. In contrast, processes compromised by malware, even when originating from harmless applications, tend to display erratic behavior and significantly lower conformance ratios, with approximately 89.8% of such processes having ratios below 0.2.
- Interactive processes, such as web browsers and email clients, exhibit more variable behavior (i.e., lower conformance ratios) compared to less interactive processes. The latter's behavior is predominantly user-driven.

Processes infected with malware are identified and isolated. The fundamental principle of our approach is that while the network communication behavior of a benign process aligns with its established traffic profile, a process compromised by malware deviates from this *normal* pattern. These deviations form additional features that are utilized to train our classifier, enhancing its ability to detect malware-compromised processes.

A. OUR MAIN CONTRIBUTIONS

For our experimental analysis, we employed the Cobalt Strike dataset as our testing ground, while utilizing the 2020 and 2021 datasets for training our random forest classifier. The results demonstrate that our newly developed features significantly outperform traditional signature-based detection methods. We achieved a 94.5% true positive rate (successfully identifying 93.5% of malware-injected processes in the test dataset) and maintained a 0% false positive rate (accurately ensuring no benign process was misclassified as malicious). This performance is markedly superior to the 25.8% true positive rate achieved using previously established features.

- Acknowledging the limitations of Domain Name System (DNS) detection in identifying malicious process injections, we designed an application-level monitoring system named *Anteater*. This system is adept at detecting lateral malware injection attacks.
- Given that attackers frequently use reputable cloud services to disguise their C&C servers, distinguishing between normal and abnormal traffic becomes challenging. *Anteater* addresses this by efficiently identifying and intercepting the IP addresses linked to abnormal program access through detailed analysis of network traffic configurations.
- *Anteater* was deployed in a real-world enterprise environment, analyzing over 400 million network traffic sessions. The evaluation showcases *Anteater*'s exceptional capability in detecting malware injections, with a true positive rate (TPR) of 94.5% and a false positive rate (FPR) of less than 0.1%.

II. RELATED WORK

Attackers are increasingly adopting distributed architectures for deploying Command and Control (C&C) hosts. To evade detection and blocking of C&C addresses, they are leveraging both host-based and network-based evasion tactics, along with the use of steganography for updating the IP addresses of C&C servers used by malware [6]. Traditional defense mechanisms primarily focus on network-based and signature-based virus monitoring [7]. However, these

methods fall short in effectively intercepting and defending against such threats. The unpredictable nature of malware behavior poses a significant challenge for monitoring systems, often leading to their inability to accurately detect when benign programs are compromised by malware injections.

Sun et al. Created a fine-grained Application-DNS profile that described any innocuous program's *normal* DNS behavior. They discovered that the DNS behavior of malware-infected processes differed significantly from the Application-DNS profile of benign apps. Then they used a dataset that contained over 130 million DNS queries from real-world businesses and 8 million requests from a sample of malware operating in a sandbox setting to create six distinct characteristics based on the Application-DNS profile. compared the new characteristics' detection performance to those of previously suggested features and found that they were capable of recognizing 190 malware-injected processes. Overall, The findings indicate that fine-grained program-DNS properties may be used to develop detectors for attack actions that escape present detection systems [8].

Jin et al. Developed an application registry and DNS request monitoring based on the They concentrate on these peculiarities. They offer a software-defined networking (SDN) and DNS Response Policy Zone (DNS RPZ) based user terminal anomaly detection system. Outbound traffic initiated by unknown programs or intended for IP addresses not obtained through DNS name resolution will be recognized and banned at the user terminal in the proposed system [9].

In our study, we introduce *Anteater*, an application-level monitoring system designed to meticulously profile each benign program. *Anteater* achieves this by gathering detailed data on the program's network traffic, thereby characterizing the specific nature of its network activities. Such an application-level monitoring tool as *Anteater* is adept at identifying and mitigating stealthy cyber attacks, a capability that we explore in the following section.

III. THREAT MODEL AND BEHAVIORAL MOTIVATION

Initially, we focus on illustrating the nature of stealth attacks and the limitations of current malware detection systems in identifying these covert threats.

A. EVADE HOST INSPECTION

Attackers often disguise their malicious activities by emulating the behavior of trusted, benign applications, executing actions through these programs to circumvent standard host security protocols. A notable instance of this tactic is fileless malware [10], which embeds its malicious code within legitimate processes, thereby executing it stealthily [11], [12], [13]. Such fileless attacks pose a significant challenge to conventional defensive strategies due to their elusive nature. In the following section, we delve into the three prevalent types of fileless attacks.

1) PORTABLE EXECUTABLE INJECTION (PE INJECTION)

Malware can stealthily integrate its malicious code into an already running process, bypassing the need to specify a LoadLibrary location. This integration can be achieved through simple shellcode or by invoking *CreateRemoteThread*. One significant advantage of PE (Portable Executable) injection is that it eliminates the need for the malware to write the malicious DLL onto the disk, a step required in Load Library methods. Instead of creating a *DLL path*, the malware embeds its code directly into the host process using *WriteProcessMemory* (e.g., through VirtualAllocEx). However, this technique faces the challenge of adjusting the base address of the injected image. When the malware injects its PE into a different process, it is assigned a new base address, necessitating dynamic recalibration of the PE's fixed address. To overcome this, the malware must locate the relocation table address of the host process and correctly resolve the absolute address. This method leaves no files on the hard drive, aligning it with other fileless techniques like reflective DLL injection and memory modules [13]. Conversely, methods like memory modules and reflective DLL injection are notably more challenging to detect. These techniques operate without the need for additional Windows APIs, as they execute entirely in memory, bypassing functions like CreateRemoteThread or LoadLibrary. Reflective DLL injection involves a DLL that self-maps into memory during execution, independent of the Windows loader. The in-memory module approach bears similarities to reflective DLL injection. However, in this case, it is the injector or loader that takes on the responsibility of mapping the target DLL into memory, as opposed to the DLL mapping itself, as illustrated in Fig. 1.

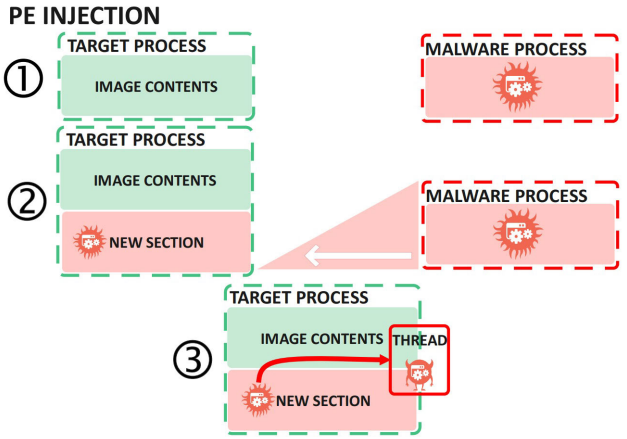

FIGURE 1. PE injection.

2) PROCESS HOLLOWING

Instead of employing traditional code injection methods like DLL injection, malware can opt for a technique known as *process hollowing*. This method involves the malware first removing (or 'hollowing out') legitimate code from a target process's memory. Subsequently, it replaces this

vacated memory space with a malicious executable, a tactic often employed with common processes (e.g., svchost.exe). The malware initiates a new process to accommodate its

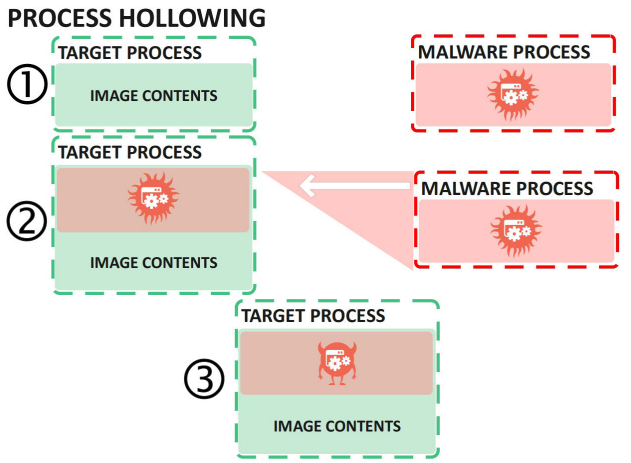

FIGURE 2. Process hollowing.

malicious code in a suspended state. As depicted in Fig. 2, this is achieved by employing *CreateProcess* with the process creation flag set to CREATE SUSPENDED (0×00000004). The main thread of this newly created process remains in suspension until the ResumeThread function is activated. The malware's next step involves replacing the original file's content with its harmful payload. To unmap the memory of the target process, it executes *ZwUnmapViewOfSection* or *NtUnmapViewOfSection*, both of which effectively free the memory associated with the section. Following the memory unmap, the loader utilizes *VirtualAllocEx* to allocate new memory for the malware and *WriteProcessMemory* to transfer each segment of the malware into the target process's space. Finally, *SetThreadContext* is used by the malware to redirect the entry point to the newly inserted code section.

3) THREAD EXECUTION HIJACKING

This approach bears resemblance to the previously discussed process hollowing technique. Thread execution hijacking is a type of malware attack that focuses on existing threads within a process, deliberately avoiding the creation of new, conspicuous processes or threads. In such scenarios, one might observe the invocation of *CreateToolhelp32Snapshot* and *Thread32First*, followed by *OpenThread*, to analyze the target process, as illustrated in Fig. 3. Upon acquiring a handle to the target thread, the malware employs *SuspendThread* to pause the thread, setting the stage for injection. It then uses *VirtualAllocEx* and *WriteProcessMemory* for memory allocation and code injection. The injected code often includes elements like shellcode, a pathway to the malicious DLL, and the address of LoadLibrary.

B. EVADING IPS MONITORING

Intrusion Prevention Systems (IPS) have been fundamental in bolstering perimeter security. Attackers, in their quest to bypass IPS monitoring, are increasingly leveraging public

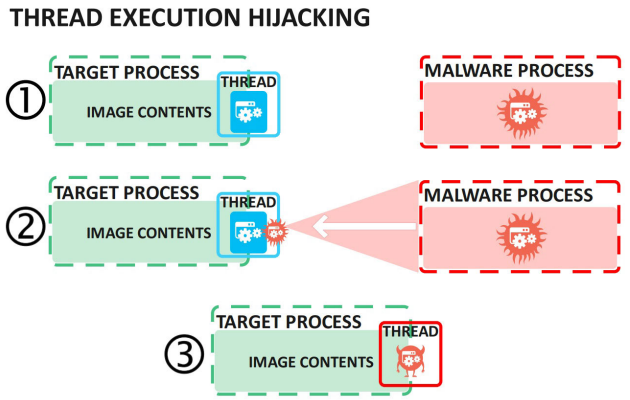

FIGURE 3. Thread execution hijacking.

TABLE 1. Function description.

Function Keyword	Description
VirtualAllocEx	Reserves, commits, or changes the state of a region of memory within the virtual address space of a specified process. The function initializes the memory it allocates to zero.
LoadLibrary	Loads the specified module into the address space of the calling process. The specified module may cause other modules to be loaded.
WriteProcessMemory	Writes data to an area of memory in a specified process. The entire area to be written to must be accessible or the operation fails.
CreateProcess	Creates a new process and its primary thread.
ResumeThread	Decrements a thread's suspend count. When the suspend count is decremented to zero, the execution of the thread is resumed.
ZwUnmapViewOfSection	The ZwUnmapViewOfSection routine unmaps a view of a section from the virtual address space of a subject process.
SetThreadContext	Sets the context for the specified thread.
CreateToolhelp32Snapshot	Takes a snapshot of the specified processes, as well as the heaps, modules, and threads used by these processes.
Thread32First	Retrieves information about the first thread of any process encountered in a system snapshot.
SuspendThread	Suspends the specified thread.

and widely trusted web services to blend their malicious traffic with legitimate network activities. A notable instance of such tactics is the HammerToss malware [14], which ingeniously employs cloud services, like a Twitter account or an image hosted on Github [15], to obscure its Command and Control (C&C) communications. This method is gaining traction among cyber threats, as evidenced by the Turla malware [16].

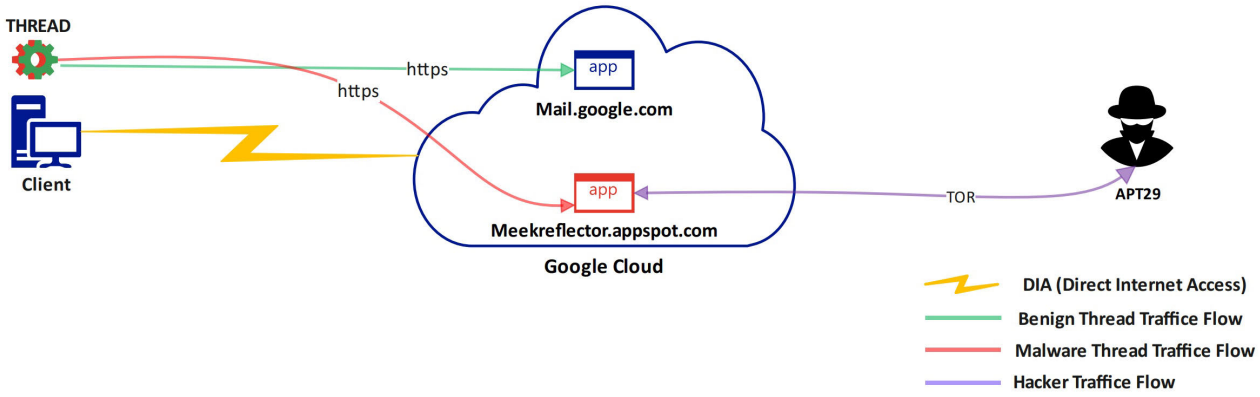

FIGURE 4. The POSHSPY attack uses the Google cloud service to reach its C&C server by injecting activities through the Windows Management Interface event.

C. STEALTHY ATTACKS

Given the increased opportunities to reduce their detectable presence both on the host and within the network, we observe that attacks integrating the aforementioned methods tend to be more clandestine.

Fig. 4 depicts the APT29 POSHSPY malware, a paradigm of an attack designed to elude conventional host and network detection mechanisms. Following its initial breach, the attack secures its presence on the compromised system by integrating a backdoor mechanism within a Windows Management Interface (WMI) event. It then employs domain fronting to establish a connection to a Google Cloud-based proxy server. This connection, while appearing to be a legitimate interaction with a Google service, actually conceals its true endpoint within encrypted traffic. Ultimately, this proxy server facilitates a connection to the attacker's Command and Control (C&C) server via the Tor network.

Kazuar [17] stands as another notable instance, a versatile backdoor Trojan that targets *explorer.exe* and utilizes authentic WordPress websites for its Command and Control (CC) operations. Additionally, the Empire framework, an open-source tool for post-exploitation, includes modules for cross-process injection such as Invoke-PSInject [18]. Platforms like Dropbox and GitHub are also increasingly being repurposed as CC servers in these scenarios.

Although various strategies have been developed to address the issue of malicious code being injected into benign applications, the majority of detection methods still predominantly depend on manual analysis and heuristic approaches [19], [20].

D. THREAT MODEL

obscure their network traffic, attackers may compromise end-host systems by injecting malicious code into established, trustworthy, and seemingly harmless applications. Additionally, they can configure Command and Control (C&C) servers in various manners, including the use of well-known online services as relay points for their C&C communications. In our research, we operate under the assumption that the process data sourced from the kernel space remains unaltered, thus deeming the kernel as a reliable entity. This assumption aligns with the threat models utilized in prior studies focusing on system monitoring. Our study's purview does not extend to kernel-level attacks that could undermine security monitoring mechanisms [21], [22], [23]. Nevertheless, we have contemplated the scenario where an attacker might compromise our data collection systems and alter the data transmitted to our backend database. To safeguard the integrity of our data, we corroborate the information obtained from our collection systems with kernel logs and local network traffic records.

IV. ANTEATER: PROGRAM NETWORK BEHAVIOR SYSTEM MODEL

In this section, we focus on the **svchost** program as a case study to demonstrate the application of our developed *Anteater* system for monitoring application traffic behavior. We elaborate on the methodology employed for data collection and provide a comprehensive description of the dataset utilized. Additionally, this section includes a summary of the data along with a detailed explanation of our data processing techniques.

A. ANTEATER: DESIGN SCHEME AND PREMISE

Numerous studies have proposed examining DNS activities on servers located at different levels of the DNS hierarchy beyond the host [24], [25]. Notably, when malware communicates with its Command and Control (C&C) servers via legitimate web services, DNS queries (like those for mail.google.com) might appear benign. Such activities can easily evade traditional network-based detection methods, underscoring the need for vigilance. Conversely, various host-based detection strategies have been developed, utilizing both static and dynamic analysis to identify malware [19], [26]. However, the complexity of detecting an attack increases when malware is injected through an ostensibly innocent program.

Detecting malware injection attacks that utilize legitimate services for Command and Control (CC) server interactions presents a significant challenge. This situation

necessitates the exploration of advanced, fine-grained detection methodologies.

To effectively pinpoint covert attacks, we posit that each benign application should possess a distinct network traffic profile, delineating the expected behavior of its processes. Since the network traffic patterns of a program's benign processes are typically consistent with the program's inherent logic, we anticipate these patterns to exhibit a high degree of *stability*. Conversely, when malware infiltrates a program, it alters the process's behavior, leading to deviations from the program's established traffic norms.

The network traffic profile for a specific application, such as svchost.exe, is a critical component in our analysis. This profile is derived by integrating network-based traffic statistics with process-level information from the kernel. The svchost.exe application, upon executing its executable binaries, initiates multiple processes. These processes are involved in data transmission and form connections predominantly with Microsoft's servers. The analysis of target ports and IP addresses reveals their associations with specific countries and organizations. Notably, the network traffic profile of the svchost process frequently indicates connections to Akamai and Microsoft registries. To enhance our understanding, new metrics will be developed to assess the network traffic profiles of each benign program, as depicted in Figure 5.

B. ANTEATER: DATA COLLECTION SYSTEM

Our *Anteater* system, specifically developed to function on end-host computers, is dedicated to the comprehensive collection of network data. This includes detailed network traffic activities and associated process and program-level information, such as destination ports and IP addresses, which are crucial for identifying the origins of network traffic flows. The design and operational framework of *Anteater* are depicted in Figure 6.

The operational sequence of our *Anteater* system is depicted in several steps within the figure. Initially, in step (1), the journey begins with a process being initiated by an executable program, which subsequently commences transmitting data externally. Following this, in step (2), the traffic generated by this process is captured and routed to the *Anteater* data collection framework. The next phase, step (3), involves the forwarding of this process data to the Internet, facilitated by the Anteater agent. A critical aspect of our methodology is the interception and analysis of network flows and inter-process communications by our data collection tools, ensuring accurate association of network traffic with its originating process.

In the initial stages, specifically steps (1), (2), and (3), the *Anteater* system passively captures network traffic activities, including destination ports and IP addresses. This collection is intricately linked with process-level data, encompassing elements such as process ID and start time.

At each endpoint, the *Anteater* system aggregates the data accumulated from steps (1) to (3). This consolidated data, encompassing both network and process information, is then systematically transmitted to a centralized back-end data warehouse as outlined in step (4).

The final stage, denoted as step (5), involves the Anteater machine learning module. This module conducts a real-time analysis of the data stored in the warehouse. Additionally, it retrieves registration details of the requested IP addresses from an external WHOIS server, thereby enriching the analysis with crucial contextual information.

TABLE 2. Summary statistics of benign datasets.

Dataset	Collected/Repoted Year	Network session	Processes
Benign	2020	158,182,240	6,556
Benign	2021	124,672,685	2,550
Benign	2022	125,241,720	4,382

C. BENIGN AND MALWARE DATA COLLECTION

In a corporate environment, we deployed the *Anteater* data collection system across 50 Windows-based workstations. This system meticulously records the daily network traffic activities associated with every process on each workstation, utilized by actual users. The implementation of this data collection was conducted with the full approval and oversight of the company's legal department, ensuring compliance with all relevant legal and ethical standards.

Our data collection methodology aligns with the organization's privacy policies, guaranteeing that the gathered data is strictly accessed through secure channels and only by authorized personnel. As detailed in table 2, the data collection spanned from 2020 to 2022. During this period, we successfully gathered in excess of 400 million network session logs, originating from more than 6556 distinct processes.

Data Integrity: Guaranteeing the security and integrity of our collected data is paramount. While all end-host systems are safeguarded by corporate-level firewalls and undergo continuous surveillance, we adopt additional measures for enhanced security. This involves the integration of executable binary signatures (utilizing MD5, SHA-1, or SHA-256 algorithms) for all gathered processes. These signatures are then meticulously cross-referenced with the VirusTotal malware databases. Furthermore, to reinforce the reliability of our data, we conduct thorough manual verifications as an additional layer of scrutiny.

Malware Dataset: In our study, we employed malware instances sourced from CobaltStrike [5] to replicate authentic attack scenarios. These samples were accumulated over the period from 2020 to 2022. Utilizing VMware Vsphere [27], we established a controlled experimental setup where each malware instance was executed in isolation. This approach was designed to circumvent the activation of the diverse anti-virus mechanisms inherent in the malware. In this controlled environment, we integrated an *Anteater* data collection framework. This framework was

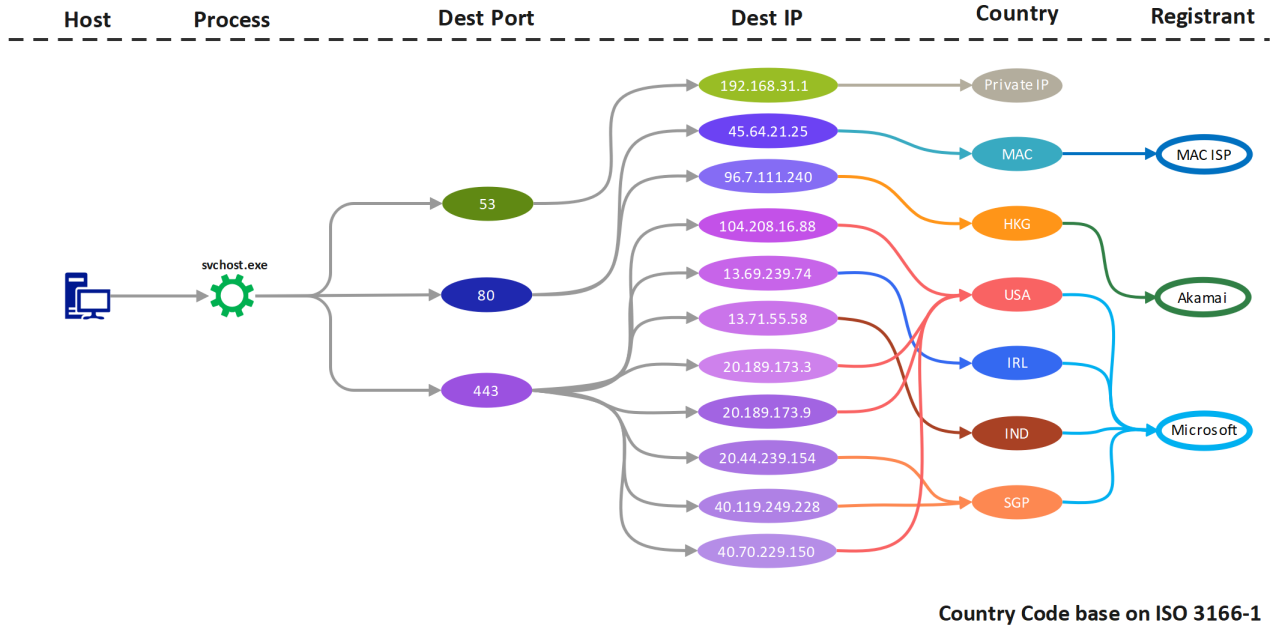

FIGURE 5. Program network profile of svchost.exe.

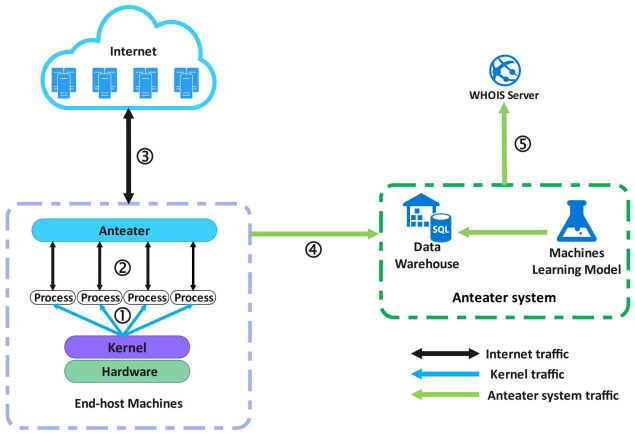

FIGURE 6. Data collection system anteater.

tasked with the collection of malware data, mirroring the process used for gathering benign data in actual user workstations.

Malware-Injected Processes: In our experimental setup, we first documented the standard operational patterns of benign processes within the sandbox environment. Following this, malware was introduced into the sandbox, potentially leading to the corruption of these benign processes through the injection of malicious code [11]. Our objective was to compile a comprehensive database of these malware-altered processes. These processes, while outwardly appearing innocuous, are in fact manipulated by malevolent logic. Our focus was on identifying and segregating processes that deviated from their typical behavior post-malware exposure. Processes that independently executed malicious activities were excluded from our study, as their distinct binary signatures make them relatively straightforward to identify.

D. DATA PREPROCESSING AND STATISTICS

The ultimate aim of our study is to construct distinct network traffic profiles for each legitimate program using data derived from our benign dataset. This will enable us to distinguish between genuine programs and those compromised by malware, despite originating from the same benign source. To achieve this, we engage in a two-step data preparation process.

- Consolidation of processes under their respective programs. Each unique executable binary, or program, can initiate multiple processes. We aggregate these processes under their originating program, analyzing their collective behavior. This approach enables us to formulate a comprehensive behavioral profile for each application, considering its deployment across various workstations and users. In our benign dataset, this method led to the identification of 453 distinct programs.
- Investigating the connection between program activities and network sessions. Programs typically generate a multitude of network sessions, each with unique characteristics. We focus on the correlation between the program's network activity and the specific destination addresses and ports it accesses. For instance, in examining the network sessions of *Skype.exe*, we identified two primary public addresses and a destination port that characterize its network footprint. The port *443* is identified as the primary destination port for Skype services, while the addresses *13.107.42.16* and *52.174.193.75* are recognized as key destination addresses for these services.

After completing the preliminary processing steps mentioned earlier, Fig. 7 illustrates the cumulative distribution function (CDF) of the average count of unique IP addresses accessed by the processes of each benign application,

presented on a logarithmic scale. The data reveals that a vast majority (98%) of applications initiate queries to no more than 20 distinct IP addresses, with a significant portion (64%) limiting their queries to 10 or fewer unique IPs. However, the CDF demonstrates a pronounced long tail, indicating that a minority of applications engage with a substantially higher number of different IP addresses.

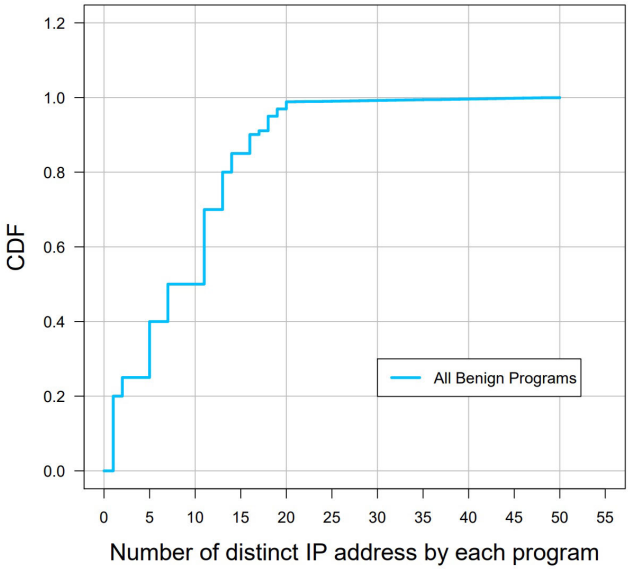

FIGURE 7. The average number of different IP addresses searched by each program's processes (log scale).

In Fig. 8, a heatmap is presented, utilizing a logarithmic scale to detail the diversity in IP address connections among various programs. The data showcases a list of the top ten applications based on the count of unique IP addresses they access. Notably, browsers such as Chrome, Firefox, and Edge are prominent at the top of this list. This observation underscores the inherent differences in the types of applications: User-interactive applications, like web browsers and email clients (e.g., Outlook), tend to exhibit a broader range of IP address interactions. In contrast, applications with non-interactive functionalities generally display more consistent and predictable network behavior.

V. PROFILING PROGRAM NETWORK TRAFFIC BEHAVIOR

Stealth attacks frequently employ the tactic of embedding malicious code into otherwise harmless software. This code then mimics legitimate network traffic, effectively camouflaging its presence [11]. Such strategies ingeniously establish Command and Control (CC) channels, often using authentic domain names for concealment [14], [28], [29]. Conventional malware detection techniques, focusing on matching executable files to known malware signatures or analyzing DNS requests, may fail to detect these sophisticated threats, erroneously classifying them as benign.

Our research aims to formulate a unique network traffic signature for each authorized software application, thus enabling accurate identification of normal activities versus

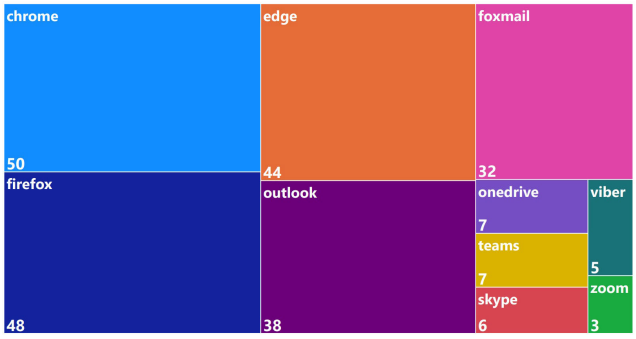

FIGURE 8. Top 10 programs in terms of the number of different IP addresses.

stealth attacks. We hypothesize that even when a program leverages legitimate network connections and domain queries, the pattern of its network interactions under normal circumstances will be distinctly different from when its processes are corrupted by malware.

To realize this goal, we propose the implementation of two novel metrics to construct detailed network traffic profiles of programs and differentiate between their regular and malware-infected operations: (1) Varieties of IP Address Types and (2) Analyses of Frequency Ratios and Consistency Ratios.

A. FREQUENCY RATIO AND CONSISTENCY RATIO

Most legitimate software applications characteristically reach out to certain IP addresses with regularity. As demonstrated in Fig. 5, the *svchost* software, for instance, predominantly communicates with IP addresses linked to Microsoft servers. These servers, affiliated with *Microsoft Corp*, are distributed globally, with notable locations including the *USA* (United States of America), *IRL* (Ireland), *IND* (India), and *SGP* (Singapore). Such patterns are typical for *svchost* processes. Nevertheless, some processes from nominally benign applications may engage in less typical activities, like connecting to specific ad servers, which also warrants monitoring within these applications' usual operations.

To accurately map these *expected* patterns for each software, our methodology encompasses an analysis of all activities initiated by benign programs in our database. We introduce a *frequency ratio* metric, which gauges the *regularity* of specific behaviors, like the frequency of IP address queries. Following this, we calculate a *consistency ratio* to evaluate how *aligned* a process's activities are with the normative behaviors of that software. This involves, for example, checking whether the process queries IP addresses that are typically accessed by the majority of benign processes of the same software. The forthcoming sections will elaborate on these methods in greater detail.

B. IP ADDRESS CONSISTENCY RATIO

The *frequency ratio* for an IP address is defined initially as the proportion of processes in a specific program that queries

the IP address as follows:

$$Frequency\_Ratio_{ant,ip_i} = \frac{\#\text{ benign processes of } ant \text{ querying } ip_i}{\text{total \# benign program of } ant} \quad (1)$$

In our framework, *ant* symbolizes the name of a selected benign program, and *ip* represents a collection of IP addresses queried by the benign processes associated with *ant*.

The *Frequency Ratio* is computed as follows: Suppose, for instance, that 90% of valid Svchost processes are observed to initiate requests to *104.208.16.88*. In such a case, the *Frequency Ratio* for the pair *Svchost*, 104.208.16.88 is assigned a value of 0.9. On the other hand, if no benign process of *ant* has ever inquired about ip_i, the ratio is set to one. Similarly, this ratio is maintained at one if there is at least one query to ip_i by any benign process. This metric aims to quantify the *regularity* with which an IP address is solicited in benign operations, offering a precise and effective means of measurement.

In the case of a single process (whether benign or malware-injected), we define *IP address consistency ratio* as the average of *Frequency Ratio ant*, ip_i for all of the IP addresses ip_i that have been queried by the process, as follows:

$$IP\_Address\_Consistency\_Ratio_{pid} = \frac{\sum_{ip_i \in A} \text{Frequency\_Ratio}_{ant,ip_i}}{|A|} \quad (2)$$

In this context, *pid* denotes the specific process under consideration, *ant* refers to the benign program initiating *pid*, and A symbolizes the set of IP addresses queried by *pid*.

For instance, if $Frequency\_Ratio_{Svchost,104.208.16.88} = 0.9$, and a Svchost process only queries *104.208.16.88*, then its *IP_address_Consistency_Ratio* is 0.9. Quite the opposite, in fact, if a Svchost process only queries *40.119.249.228*, where $Frequency\_Ratio_{Svchost,40.119.249.228} = 0$, then its *IP_Address_Consistency_Ratio* It is possible that this Svchost process was maliciously begun by a malware-injected Svchost application, as shown by the value of 0. The concept behind this metric is to assess how *consistent* the behavior of a process is with the behavior of known benign processes within the same program, which is based on previous experience.

C. REGISTRANT CONSISTENCY RATIO

We now expand the research to include information about IP address registration, namely the IP address registrant. This is to account for potential churn or variation in IP address inquiries.

For example, if half of the Svchost is dedicated to processing just requests of type *104.208.16.88* and the other half only queries of *40.119.249.228*, then the $Frequency\_Ratio_{Svchost,104.208.16.88}$ and $Frequency\_Ratio_{Svchost,40.119.249.228}$ will both become 0.5, which is relatively low. However, we may look at the IP address registrant instead of looking at the actual IP address. Then we will see that both *104.208.16.88* and *40.119.249.228* have the same registrant, which is *Microsoft Corporation* As a result, integrating registrant data may help to round out the *Frequency_Ratio* and *Consistency_Ratio* analyses.

According to the definition, the fraction of processes in a single application that query at least one IP address registered by the registrant is *frequency_ratio* for an IP address registrant, as follows:

$$Frequency\_Ratio_{ant,reg_i} = \frac{\#\text{ benign processes of } ant \text{ querying ip address registered by } reg_i}{\text{total \# benign processes of } ant} \quad (3)$$

where *ant* is the name of a particular benign program, and reg_i is the IP address registrant of at least one IP address queried by benign processes inside the program.

In the above example, the Frequency Ratio of Svchost, Svchost will be one since half of the Svchost processes only query *104.208.16.88* and the other half of processes only query *40.119.249.228* and the Frequency Ratio of Svchost, Svchost will be one, the $Frequency\_Ratio_{Svchost,Svchost}$ will be 1. This metric measures how *common* an IP address registrant is when a benign process queries an IP address.

Following that, we define the procedure for a certain process *Registrant_Consistency_Ratio* (either benign or malware-injected).Similar to the *IP_Address_Consistency_Ratio*, it's the average of $Frequency\_Ratio_{ant,reg(ip_i)}$:

$$Registrant\_Consistency\_Ratio_{pid} = \frac{\sum_{ip_i \in A} \text{Frequency\_Ratio}_{ant,reg(ip_i)}}{|A|} \quad (4)$$

It's worth noting that we're getting the registrant for each IP address sought by the process using $reg(ip_i)$. If no other benign approach has been used to identify the registrant, then, identical to the case of IP address frequency, the registrant's *Frequency_ Ratio* will be 0. However, in exceptional circumstances, we may be unable to discover any registrant at all, such as when IP address registration information is unavailable, or the IP address is a *Private IP* enquiry. If the IP address ip_i is a reserved IP address that by definition does not have a registrant, we appoint $Frequency\_Ratio = 1$; otherwise, we assign 0 if the IP address registrant cannot be identified on the WHOIS server.

D. COUNTRY CONSISTENCY RATIO

We take into account the country where the IP address registrant is situated in addition to using IP address registrant information. The criteria for IP address country *Frequency_Ratio* and *Consistency_Ratio* are fairly similar to those for IP address registrant:

$$Frequency\_Ratio_{ant,cty_i} = \frac{\#\text{ benign processes of } ant \text{ querying ip address registered by } cty_i}{\text{total \# benign processes of } ant} \quad (5)$$

$$Country\_Consistency\_Ratio_{pid} = \frac{\sum_{ip_i \in A} \text{Frequency\_Ratio}_{ant,cty(ip_i)}}{|A|} \quad (6)$$

If the country information for an IP address cannot be found, *country* ip_i is treated identically to *reg* ip_i.

TABLE 3. Number of programs and processes in each group.

	# Programs	# Processes	Dataset
Non-interactive Benign	313	223,436	Benign
Non-interactive Common	32	25,160	Benign
Non-interactive Malware	32	2,502	Malware
Interactive Benign	5	4,298	Benign
Interactive Malware	5	1,418	Malware

It's worth noting that we've provided a number of different options *Frequency_Ratio* and *Consistency_Ratio* to profile the program's IP address habits, use the criteria above, although other ratios may work as well.

VI. EVALUATION FOR CONSISTENCY RATIOS

From our data, we now analyze the three consistency ratios. To help you see the distinction between benign and malware-injected processes, we'll go through the five types of processes we'll be looking at.

- **Non-interactive Benign:** In the benign dataset, there are benign processes of non-interactive applications.
- **Non-interactive Common** (subset of Non-interactive Benign): In the benign dataset, there are benign processes of non-interactive applications; *however, in the malware dataset, the same applications are injected by malware.*
- **Non-interactive Malware:** In the malware dataset, malware injected malicious processes into non-interactive benign Apps.
- **Interactive Benign:** In the benign dataset, there are benign processes of interactive programs.

Please keep in mind that interactive programs include *chrome.exe* / *firefox.exe* / *edge.exe* all other programs are classified as non-interactive programs. The number of programs and processes in each category is shown in Table 3.

Non-Interactive Programs: Fig. 9 shows the IP address consistency ratios of Non-interactive Benign and Non-interactive Common processes are pretty high, with more than 85% of the processes having values greater > 0.96. Non-interactive Malware, on the other hand, which is the malware-injected processes of the same 32 applications in Non-interactive Common, has substantially lower IP address consistency ratios, with 90% of the processes having values of < 0.2.

The registrant consistency ratio follows a similar trend to the IP address consistency ratio, with extremely high registrant consistency ratios for *Non-interactive* Benign and *Non-interactive* Common processes and very low rates for *Non-interactive Malware* processes. Due to the substantially smaller range of nations, the *Non-interactive Malware* procedures have relatively greater country consistency ratios than the preceding two ratios. They are, however, much smaller than the ratios resulting from benign processes.

Interactive Programs: The dynamic nature of user-interactive applications, like web browsers, presents challenges in modeling their network traffic behaviors. Prior studies have limited their analysis to network activities within the initial 120 seconds to mitigate the impact of user-driven network interactions [30]. However, this approach might overlook malware that delays its network activity beyond this window.

In our approach, we introduce an innovative method to characterize network traffic from interactive applications using frequency ratios and consistency ratios, tailored at a more granular *user-level*. This involves analyzing all processes for each user and each program, instead of aggregating data across all users for each program. We operate under the assumption that individual users have distinct sets of frequently accessed IP addresses. By profiling these unique network traffic patterns at both program and *user-levels*, we can effectively identify and differentiate malware activities. Notably, our analysis encompasses the entirety of a process's actions, not just its initial phase, as illustrated in Fig. 10.

Observations reveal that while the IP address consistency ratio for Interactive Benign is lower compared to non-interactive processes, it still markedly differs from that of *Interactive Malware*. Malware-infected interactive processes tend to exhibit more erratic behavior with a diverse range of queries.

Additionally, both *Interactive Benign* and *Interactive Malware* show higher values in registrant and country consistency ratios compared to IP address consistency ratios, due to a more limited variety of registrants and countries involved. However, a clear distinction exists between the two. Given our data collection setting in Macau, Interactive Benign processes display a significantly high country consistency ratio, suggesting that despite accessing IP addresses from various registrants, most IPs are registered within Asia. In contrast, Interactive Malware processes, not influenced by such user patterns, demonstrate considerably lower country consistency ratios.

A. IP ADDRESS TYPE ANALYSIS

Consistency ratios are traditionally derived from IP address data collected from historical benign processes. However, this method encounters limitations when a process queries a new IP address, previously unrecorded in our datasets. To address this challenge, we employ the *Anteater* dataset to implement an IP address type analysis. This innovative approach is designed to predict the likelihood of an IP address being *anticipated* in a benign context, even when it has not been encountered in prior benign activities. This methodology provides a more robust framework for evaluating the nature of newly observed IP addresses, enhancing our ability to accurately assess unfamiliar network activities.

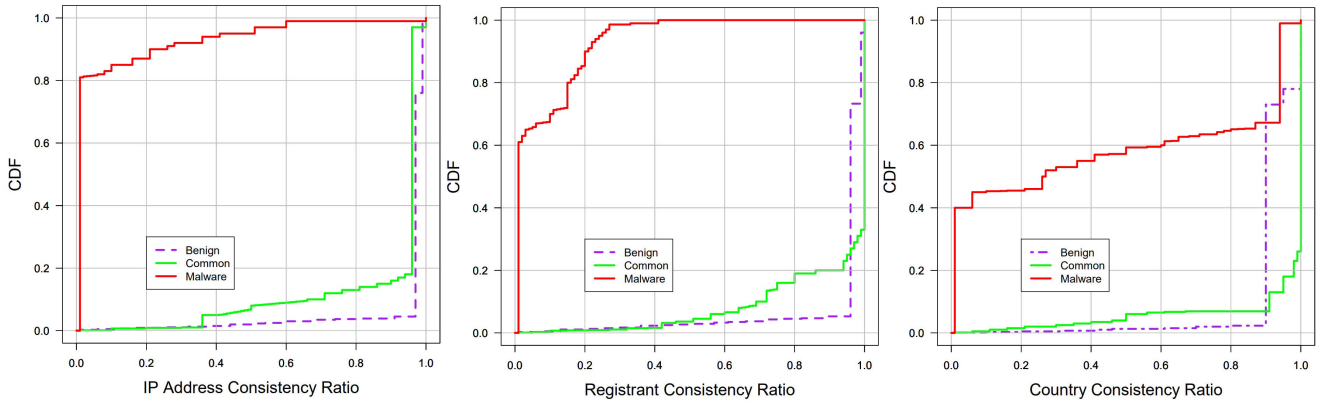

FIGURE 9. Consistency ratios for non-interactive processes.

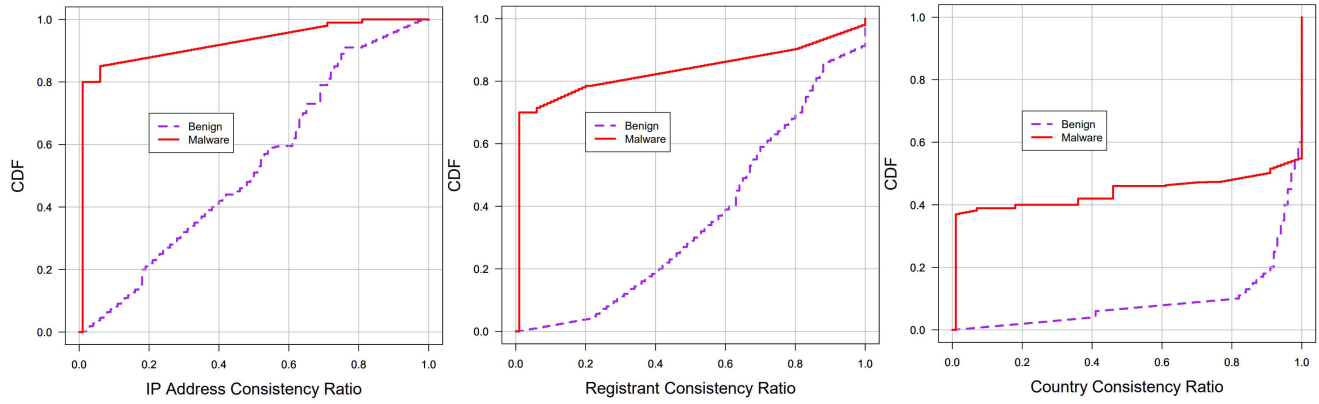

FIGURE 10. Consistency ratios for interactive processes.

We categorize IP addresses into three types:

- *Reserved:* All IP addresses that are specified as *Private IP* are included in this category.
- *Owner:* Ownership refers to IP addresses that are *owned* by the same corporation/entity/organization that*owns* the application. We will present specific methods for determining and comparing *ownership*.
- *Other:* All additional IP address information does not fall into one of the two categories listed above.

B. IDENTIFY OWNER IP ADDRESS FOR A PROGRAM

The classification of an 'Owner' type IP address is applied when it is registered by the same entity that owns the software. For instance, IP addresses *52.114.40.58* and *52.113.194.132* are categorized as 'Owner' type for *Teams.exe*, a component of the Teams framework, developed and authenticated by Microsoft, who is also the registrant of these IP addresses. Establishing this link requires access to 'owner' details for both the software and the IP address.

Owner for program. We collate our data from three key sources: (1) The program's name itself, like *Update.exe*, which suggests a Microsoft association. (2) The code signature of the program binary, which identifies the program signer. (3) The application's file path, such as *C:/Users/Windows 10/AppData/Local/Microsoft/Teams/Update.exe*, indicates Microsoft's ownership of *Update.exe*.

Owner for IP address. Our primary source here is the registrant information from the WHOIS record of the IP address.

For our analysis, while compiling 'owner' information, we standardize organizational names by omitting common designations like *Corporation*, *LLC*, etc.

C. IP ADDRESS TYPE DISTRIBUTION FOR COMMON PROGRAMS

The IP address type distributions We examine IP address type distributions for Non-interactive Common and Non-interactive Malware processes across 32 standard applications. The process involves calculating the proportion of each IP address type (Reserved, Owner, Other) in the total IP addresses accessed by each process. Then, we aggregate these proportions across all processes of a given application to determine the average distribution of each IP address type. This allows us to make comparative assessments between benign and malware-infected processes within the same software.

Furthermore, we categorize programs based on their origin into (1) system programs, typically pre-installed with the Windows OS, and (2) user-installed applications.

Our analysis reveals a distinct pattern: malware-infected processes tend to query a significantly higher proportion of 'Other' IP addresses compared to their benign

TABLE 4. IP address type distributions for all programs.

Dataset	Programs Reserved	Programs Owner	Programs Other
Benign	106	101	110
Malware	179	3	2632

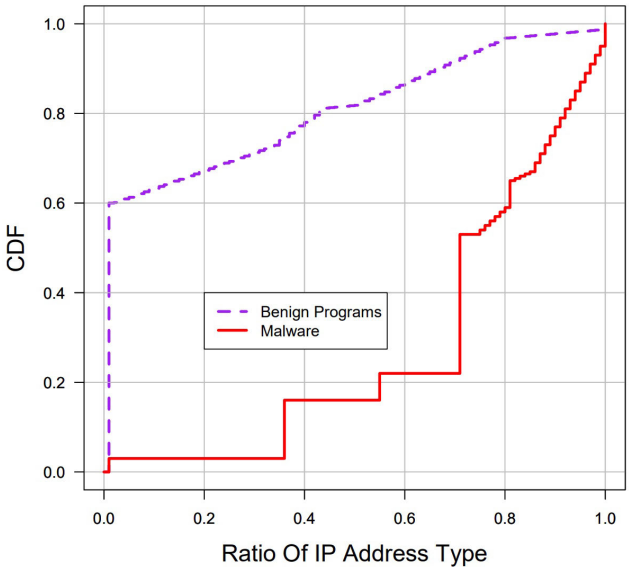

FIGURE 11. For all processes of non-interactive applications, a fraction of Other IP address type is used.

counterparts. Additionally, benign processes from system programs are observed to access fewer non-Reserved IP addresses *(Owner + Other)* compared to those originating from user-installed applications.

Contrastingly, malware-injected processes, whether from system software or user-installed applications, show a similar tendency in querying a large number of 'Other' IP addresses.

D. IP ADDRESS TYPE DISTRIBUTION FOR ALL PROGRAMS

Our study's scope has been broadened to encompass an extensive analysis of both non-interactive benign and malicious (or malware-infected) applications. This comprehensive approach is reflected in Table 4, which presents a detailed breakdown of the number of applications linked to each type of IP address distribution, categorizing them into benign and malicious groups.

Our analysis reveals that a significant portion, exceeding 65%, of benign applications predominantly engage with either *Reserved* or *Owner* IP addresses. This distinct pattern is notably advantageous, as it facilitates the *prediction* of potential IP addresses that might be queried by processes within these applications. In stark contrast, a mere 6.5% of malware-infected programs exhibit a similar trend in their IP address requests.

In Fig. 11, the Cumulative Distribution Function (CDF) illustrates the variation in the percentage of 'Other' IP address types accessed by processes within non-interactive benign and malicious applications. This depiction highlights a significant disparity in the frequency of 'Other' IP address usage between benign and malicious processes. Notably, the malicious processes exhibit a markedly higher reliance on this category of IP addresses.

VII. DETECTION USING NETWORK TRAFFIC PROFILE

In this research, we have developed a network traffic profile, incorporating consistency ratios and IP address types, that effectively discriminates between benign and malware-infected processes. These processes originate from the same benign application and exhibit similar program- and process-level characteristics. This section expands on how we enhance the utility of the network traffic profile by converting it into six novel features. These features are instrumental for training machine learning classifiers to accurately identify malware-infected processes in real-world scenarios. Furthermore, we benchmark the detection efficacy of our newly proposed features against those previously established in the field [31].

A. DATASET AND FEATURES

Our analysis focuses on processes from 32 commonly used applications present in both benign and malware datasets to determine the accuracy of detecting malware-infected processes. The specifics of the data employed for detection are outlined in Table 5.

For each malware dataset, we pinpoint the malware-infected processes initiated by programs also found in the benign dataset. Concurrently, we identify the corresponding benign processes initiated by these same programs from the benign dataset. The column *Process Samples* in each dataset enumerates the number of identified benign and malware-infected processes.

The following six new features are used to detect malware-injected processes based on network traffic profiles:

1) IP address consistency ratio
2) Registrant consistency ratio
3) Country consistency ratio
4) Percentage of Reserved IP address type
5) Percentage of Owner IP address type
6) Percentage of Other IP address type

We consider two factors while evaluating our new features:

- *Cross-validation on each dataset.* To validate the effectiveness of our proposed features, we conduct ten-fold cross-validation for each dataset spanning the years 2020-2022. Additionally, we analyze the results both with and without the application of SMOTE [32] to balance the data.
- *Comparison with previously-proposed features.* Our methodology entails using the 2020 datasets for training purposes and the 2021 and 2022 datasets as test sets. This approach mirrors real-world scenarios, where models trained on existing data are subsequently applied to predict future cyber threats. We also juxtapose the detection outcomes of our features with those that have been previously proposed in the domain.

B. CROSS-VALIDATION USING ANTEATER

We employed the Random Forest classifier to perform ten-fold cross-validation across datasets from 2020 to 2022. While Random Forest serves as a primary example to demonstrate the efficacy of our novel features, it's worth noting that other classifiers might also yield similar results. The effectiveness of various classifiers in comparison will be discussed in the subsequent section.

In our classification process, we focus on evaluating the true positive rate (TPR) and false positive rate (FPR). TPR represents the percentage of correctly identified malicious processes, whereas FPR is the proportion of benign processes incorrectly classified as malicious. We visualize this relationship using the receiver operating characteristic (ROC) curve and the precision-recall curve Fig12.

Post-application of the SMOTE technique for balancing benign and malicious samples, we observe the ROC curve and precision-recall curve outcomes from the ten-fold cross-validation using Random Forest for each dataset. Both the ROC AUC and precision-recall AUC scores exceed 0.99 for all datasets. Table 6 details the results derived from datasets with and without SMOTE oversampling. At the *optimal* point–where the difference between TPR and FPR is maximal–we note that the discrepancy between using datasets with or without SMOTE oversampling is minimal. Additionally, in most datasets, a high TPR over 99% is achievable alongside a very low FPR 0.1%.

C. CASE STUDY ON POSHSPY

In this part of our study, we investigate POSHSPY, a sophisticated malware variant. As outlined in Sec III-C, POSHSPY employs a Fileless strategy, camouflaging its backdoor functionality within WMIC events that carry a PowerShell payload. To elude detection, it utilizes the Meek Tor plug-in, establishing a connection first to a Tor entry relay masquerading as a Google Cloud Service, and then proceeding to its Command and Control (C&C) network.

For experimental purposes, we replicated the POSHSPY payload using Cobalt Strike and executed the malware in a controlled environment, closely monitoring the resulting network traffic. To mimic the attack setting, we allowed the malware to establish connections to IP addresses associated with both Azure and Google Cloud.

Analysis of the data revealed that the malware commandeered a PowerShell process to execute its queries. This included twelve separate requests to Azure cloud at one-hour intervals and four requests to Google cloud spaced three hours apart. We then applied our trained model, based on the Section and 2020 datasets, for detection. Our system, Anteater, successfully identified the malware-compromised PowerShell process. However, traditional features failed to detect the malware due to its seemingly benign process information and network traffic patterns.

The collected data indicated that the malware had injected commands into a PowerShell process, which made a series of systematic queries including twelve to the Azure cloud and four to the Google cloud, with one-hour and three-hour intervals between the requests, respectively. Detection was then performed using the model trained on the 2020 datasets. Anteater detected the malware-infected PowerShell process, but conventional baseline features were ineffective, as they were misled by the process's ostensibly legitimate information and network traffic behavior.

VIII. FUTURE RESEARCH DIRECTIONS AND DISCUSSION

In this section, we examine contemporary malware detection methodologies, including traditional network-based and host-based solutions, alongside an innovative process-level detection approach that amalgamates network and process-level data for enhanced accuracy. Additionally, we delve into two specific types of malware that embed their malicious code into legitimate applications and establish Command and Control (C&C) channels via legitimate IP addresses. Lastly, we address the complexities associated with detecting malware that embeds itself into interactive platforms, such as web browsers.

A. EXISTING DETECTION SYSTEMS

Network-Based Detection Systems: Monitoring and analyzing DNS traffic of Internet servers forms the cornerstone for combating threats like fast-flux networks, bots, DGA domains, and spammers. Various studies [33], [34], [35], [36], [37], [38], [39], [40] have emphasized the importance of scrutinizing DNS activities at different DNS hierarchy levels to identify malicious domains associated with diverse attacks. A comprehensive survey [41] has outlined existing network-based detection techniques, covering various network traffic aspects like packets and flow size. However, the key difference between these approaches and ours lies in their focus on identifying malicious domains, a strategy that falls short against stealth attacks, where attackers utilize legitimate online services and domains.

Host-Based Detection Systems: Focusing on malicious applications and processes, host-based detection methods employ strategies such as static analysis [42], [43], disassembly, and reverse engineering. Nevertheless, malware can often evade these techniques [44], [45], [46]. Owing to the limitations of static analysis, dynamic analysis methods have been developed, analyzing malware behavior during execution, including function calls and information flow tracking [19], [47], [48], [49]. These recommend examining malware behavior in a controlled environment.

While advanced host-based detection methods are effective, they often necessitate extensive system monitoring, which differs from our approach. We aim to introduce a streamlined yet potent solution that gathers minimal data fields, focusing on program network traffic behaviors. Our method could complement host-based detection approaches effectively.

Integrated Network-Based and Host-Based Detection System: A recent proposal by Sivakorn et al. [30] introduced

TABLE 5. Detection dataset includes processes initiated by programs that are present across benign and malware datasets.

Year	Programs	Source	Process Samples	Total	Process Samples After SMOTE	Total
2020	12	Benign	2733	3275	2733	5466
		Malware	542		2733	
2021	16	Benign	5916	6972	5916	11832
		Malware	1056		5916	
2022	6	Benign	4168	4271	4168	8336
		Malware	103		4168	

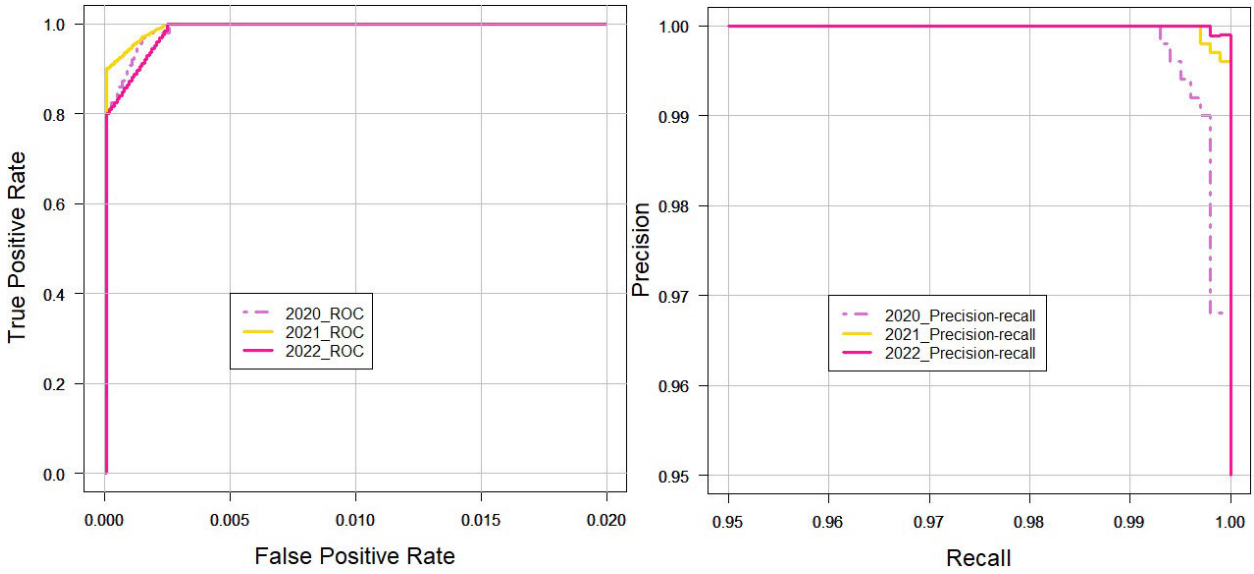

FIGURE 12. ROC curve and precision-recall curve with ten-fold cross validation using anteater.

TABLE 6. TPR and FPR for each dataset with ten-fold cross validation, with and without SMOTE oversampling.

Year	SMOTE	TPR	FPR
2020	No	98.6%	0.1%
	Yes	99.3%	0.07%
2021	No	99.4%	0.06%
	Yes	99.8%	0.08%
2022	No	99.96%	0.05%
	Yes	99.93%	0.07%

Program-DNS, a process-level detection system combining network-based features (like DNS query name and answer) with process-based features (such as code signing). This system provides enriched data context and new integrated features by merging domain and program information. However, it analyzes each process separately, lacking a detailed Program network traffic profile at the program level through joint process analysis. Consequently, it struggles to detect stealthy attacks that utilize legitimate programs for network connections via cross-process injection techniques [11], [50].

B. INTERACTIVE PROGRAM AS A TARGET FOR INJECTION

We have developed a unique program network traffic profile tailored for interactive applications like browsers and email clients. While our approach remains effective in distinguishing between benign processes and those injected with malware in such interactive applications, the prediction of their network traffic patterns is inherently challenging due to the dynamic nature of user interactions. Consequently, interactive applications may inadvertently become prime targets for malware injections, enhancing the stealthiness of an attack.

However, attackers face significant risks when targeting highly interactive applications for injection. The uncertainty surrounding the presence or persistence of the target application means that any anomalous behavior caused by the injected malware is more likely to be noticed by users. To maintain stealth and ensure persistence, attackers might lean towards targeting long-running background processes instead [12].

While there are instances of injection attacks on browser programs [11], aimed at extracting sensitive user data like passwords and credit card information from browser memory, browser developers like Google Chrome and Mozilla Firefox have actively countered such threats. They have implemented in-browser security mechanisms to thwart injection efforts by external processes [51], [52], [53].

C. GENERALIZABILITY AND LIMITATION

Anteater, our innovative detection system, is versatile enough for a wide array of environments. However, it is crucial to recognize that the network traffic profiles it generates

can differ significantly across these environments. Therefore, directly applying a model trained in a corporate setting to a different environment, like home networks or data centers, is not advisable. Instead, it is more effective to construct a new profile and retrain the model for the specific environment. An alternative approach could involve employing domain adaptation techniques to adjust our trained model to new settings.

Although Anteater is not specifically engineered to thwart adaptive attackers, it inherently raises the bar for such attackers targeting our system. For instance, if attackers aim to remain undetected while injecting malware into programs with highly predictable behavior patterns, like those never querying textslnon-owner IP addresses, they face a strategic dilemma. They must either relocate their Command and Control (C&C) servers to textslexpected IP addresses, which are typically limited in number and often high-profile (such as Azure cloud IPs), or choose different programs as their injection targets.

IX. CONCLUSION

Our research innovatively employed detailed measures to create comprehensive profiles of benign program network traffic. Implementing Anteater in a corporate environment, we analyzed network sessions from 50 end-host PCs, totaling over 400 million sessions, coupled with detailed program and process-level data. Our findings highlight that, within the same application, the network behaviors of malware-injected and benign processes are distinct, despite having identical program- and process-level information and being associated with legitimate IP addresses. By scrutinizing malware datasets spanning from 2020 to 2022, we established that our novel metrics significantly outperform previously established features, achieving a true positive rate of 94.5%. These new features successfully detected malware-injected processes in 93.5% of test cases, with a remarkable 0% false positive rate, a stark contrast to the 25.8% rate observed with previously suggested features.

In summary, our study sheds light on the nuances of program-level behavior to identify malware injections and lays the groundwork for developing more sophisticated security approaches against advanced malware threats.

REFERENCES

[1] CISA. (2022). *Ransomware Awareness for Holidays and Weekends*. [Online]. Available: https://www.cisa.gov/uscert/ncas/alerts/aa21-243a

[2] *Global Threat Landscape Report*, Fortinet, Sunnyvale, CA, USA, 2021, p. 16.

[3] Paloalto. (Nov. 2017). *Threat Brief: Why Ransomware Hurts So Much and is so Hard to Stop*. [Online]. Available: https://unit42.paloaltonetworks.com/threat-brief-ransomware-hurts-much-hard-stop/

[4] McAfee. (2019). *What is Fileless Malware*. [Online]. Available: https://www.mcafee.com/enterprise/en-us/security-awareness/ransomware/what-is-fileless-malware.html

[5] (2022). *Cobalt Strike | Adversary Simulation and Red Team Operations*. [Online]. Available: https://www.cobaltstrike.com/

[6] ESET. (Oct. 2019). *Operation Ghost: The Dukes Aren't Back They Never Left*. Section: Malware. [Online]. Available: https://www.welivesecurity.com/2019/10/17/operation-ghost-dukes-never-left/

[7] T. Jirsik and P. Velan, "Host behavior in computer network: One-year study," *IEEE Trans. Netw. Service Manage.*, vol. 18, no. 1, pp. 822–838, Mar. 2021.

[8] Y. Sun, K. Jee, S. Sivakorn, Z. Li, C. Lumezanu, L. Korts-Parn, Z. Wu, J. Rhee, C. H. Kim, M. Chiang, and P. Mittal, "Detecting malware injection with program-DNS behavior," in *Proc. IEEE Eur. Symp. Secur. Privacy (EuroS&P)*, Sep. 2020, pp. 552–568.

[9] Y. Jin, M. Tomoishi, and N. Yamai, "Anomaly detection on user terminals based on outbound traffic filtering by DNS query monitoring and application program identification," in *Proc. Int. Conf. Hum.-Mach. Interact.* New York, NY, USA: Association for Computing Machinery, May 2021, pp. 47–56, doi: 10.1145/3478472.3478481.

[10] Norton. (2019). *What is Fileless Malware and How Does it Work*. [Online]. Available: https://us.norton.com/internetsecurity-malware-what-is-fileless-malware..html

[11] MITRE. (2017). *Process Injection, Technique T1055 Enterprise*. [Online]. Available: https://attack.mitre.org/techniques/T1055/

[12] Microsoft. (Jul. 2017). *Detecting Stealthier Cross-Process Injection Techniques With Windows Defender ATP: Process Hollowing and Atom Bombing*. [Online]. Available: https://www.microsoft.com/security/blog/2017/07/12/detecting-techniques-with-windows-defender-atp-process-hollowing-and-atom-bombing/

[13] Elastic. (Jul. 2017). *Ten Process Injection Techniques: A Technical Survey of Common and Trending Process Injection Techniques*. [Online]. Available: https://www.elastic.co/blog/ten-process-injection-technical-survey-common-and-trending-process

[14] FireEye. (2019). *HAMMERTOSS: Stealthy Tactics Define a Russian Cyber Threat Group*. [Online]. Available: https://www.fireeye.com/current-threats/apt-groups/rpt-apt29.html

[15] Kaspersky. (Jun. 2020). *Steganography in Attacks on Industrial Enterprises (Updated) | Kaspersky ICS CERT*. [Online]. Available: https://ics-cert.kaspersky.com/publications/steganography-in-attacks-on-industrial-enterprises/

[16] MITRE. (2017). *Kazuar, Software S0265 | MITRE ATT&CK*. [Online]. Available: https://attack.mitre.org/software/S0265/

[17] Paloalto. (May 2017). *Kazuar: Multiplatform Espionage Backdoor With API Access*. [Online]. Available: https://unit42.paloaltonetworks.com/unit42-kazuar-multiplatform-backdoor-api-access/

[18] MITRE. (2017). *Empire, Software S0363 | MITRE ATT&CK*. [Online]. Available: https://attack.mitre.org/software/S0363/

[19] C. Willems, T. Holz, and F. Freiling, "Toward automated dynamic malware analysis using CWSandbox," *IEEE Secur. Privacy Mag.*, vol. 5, no. 2, pp. 32–39, Mar. 2007.

[20] S. B. Bhatkar, S. Nanda, and J. S. Wilhelm, "(54) Techniques for behavior based," Tech. Rep., 2011, p. 16.

[21] S. Sitaraman and S. Venkatesan, "Forensic analysis of file system intrusions using improved backtracking," in *Proc. 3rd IEEE Int. Workshop Inf. Assurance (IWIA)*, Mar. 2005, pp. 154–163.

[22] Y. Kwon, F. Wang, W. Wang, K. H. Lee, W.-C. Lee, S. Ma, X. Zhang, D. Xu, S. Jha, G. Ciocarlie, A. Gehani, and V. Yegneswaran, "MCI: Modeling-based causality inference in audit logging for attack investigation," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2018, pp. 1–29. [Online]. Available: https://www.ndss-symposium.org/wp-content/uploads/2018/02/ndss2018_07B-2_Kwon_paper.pdf

[23] Z. Xu, Z. Wu, Z. Li, K. Jee, J. Rhee, X. Xiao, F. Xu, H. Wang, and G. Jiang, "High fidelity data reduction for big data security dependency analyses," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.* New York, NY, USA: Association for Computing Machinery, Oct. 2016, pp. 504–516, doi: 10.1145/2976749.2978378.

[24] M. Lyu, H. H. Gharakheili, C. Russell, and V. Sivaraman, "Hierarchical anomaly-based detection of distributed DNS attacks on enterprise networks," *IEEE Trans. Netw. Service Manage.*, vol. 18, no. 1, pp. 1031–1048, Mar. 2021.

[25] D. Madariaga, J. Madariaga, M. Panza, J. Bustos-Jiménez, and B. Bustos, "Detecting anomalies at a TLD name server based on DNS traffic predictions," *IEEE Trans. Netw. Service Manage.*, vol. 18, no. 1, pp. 1016–1030, Mar. 2021.

[26] H. Yin, D. Song, M. Egele, C. Kruegel, and E. Kirda, "Panorama: Capturing system-wide information flow for malware detection and analysis," in *Proc. 14th ACM Conf. Comput. Commun. Secur.* New York, NY, USA: Association for Computing Machinery, Oct. 2007, pp. 116–127, doi: 10.1145/1315245.1315261.

[27] Ahelms. (2022). *VMware vSphere 7.0 Release Notes*. [Online]. Available: https://docs.vmware.com/en/VMware-vSphere/7.0/rn/vsphere-esxi-vcenter-server-70-release-notes.html
[28] Mandiant. (2017). *Dissecting One of APT29's Fileless WMI and PowerShell Backdoors (POSHSPY) | Mandiant*. [Online]. Available: https://www.mandiant.com/resources/dissecting-one-ofap
[29] MITRE. (2017). *Web Service, Technique T1102–Enterprise | MITRE ATT&CK*. [Online]. Available: https://attack.mitre.org/techniques/T1102/
[30] S. Sivakorn, K. Jee, Y. Sun, L. Korts-Parn, Z. Li, C. Lumezanu, Z. Wu, L.-A. Tang, and D. Li, "Countering malicious processes with process-DNS association," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2019, pp. 1–15.
[31] Z. Ying, Y. Zhang, S. Xu, G. Xu, and W. Liu, "Anteater: Malware injection detection with program network traffic behavior," in *Proc. Int. Conf. Netw. Netw. Appl. (NaNA)*, Dec. 2022, pp. 169–175. [Online]. Available: https://ieeexplore.ieee.org/document/9985093
[32] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," *J. Artif. Intell. Res.*, vol. 16, pp. 321–357, Jun. 2002. [Online]. Available: https://www.jair.org/index.php/jair/article/view/10302
[33] F. Weimer, "Passive DNS replication," in *Proc. 1st Conf. Comput. Secur. Incident*, vol. 98, 2005, pp. 1–14.
[34] M. Antonakakis, R. Perdisci, D. Dagon, W. Lee, and N. Feamster, "Building a dynamic reputation system for DNS," in *Proc. 19th USENIX Secur. Symp. (USENIX Secur.)*, 2010, pp. 1–17.
[35] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi, "Exposure: Finding malicious domains using passive DNS analysis," in *Proc. NDSS*, 2011, pp. 1–17.
[36] M. Antonakakis, R. Perdisci, W. Lee, N. Vasiloglou II, and D. Dagon, "Detecting malware domains at the upper DNS hierarchy," in *Proc. 20th USENIX Secur. Symp. (USENIX Secur.)*, 2011, pp. 1–16.
[37] 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 *Proc. 21st USENIX Secur. Symp. (USENIX Secur.)*, 2012, pp. 491–506.
[38] H. Choi, H. Lee, H. Lee, and H. Kim, "Botnet detection by monitoring group activities in DNS traffic," in *Proc. 7th IEEE Int. Conf. Comput. Inf. Technol. (CIT)*, Oct. 2007, pp. 715–720.
[39] R. Perdisci, I. Corona, D. Dagon, and W. Lee, "Detecting malicious flux service networks through passive analysis of recursive DNS traces," in *Proc. Annu. Comput. Secur. Appl. Conf.*, Dec. 2009, pp. 311–320.
[40] D. Plohmann, K. Yakdan, M. Klatt, J. Bader, and E. Gerhards-Padilla, "A comprehensive measurement study of domain generating malware," in *Proc. 25th USENIX Secur. Symp. (USENIX Secur.)*, 2016, pp. 263–278.
[41] K. Bartos, M. Sofka, and V. Franc, "Optimized invariant representation of network traffic for detecting unseen malware variants," in *Proc. 25th USENIX Secur. Symp. (USENIX Secur.)*, 2016, pp. 807–822.
[42] M. Christodorescu and S. Jha, "Static analysis of executables to detect malicious patterns," in *Proc. 12th USENIX Secur. Symp. (USENIX Secur.)*, 2003, pp. 1–18.
[43] 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 *Proc. Int. Conf. Inf. Syst. Secur.* Cham, Switzerland: Springer, 2008, pp. 1–25.
[44] A. Moser, C. Kruegel, and E. Kirda, "Limits of static analysis for malware detection," in *Proc. 23rd Annu. Comput. Secur. Appl. Conf. (ACSAC)*, Dec. 2007, pp. 421–430.
[45] M. Egele, T. Scholte, E. Kirda, and C. Kruegel, "A survey on automated dynamic malware-analysis techniques and tools," *ACM Comput. Surv.*, vol. 44, no. 2, pp. 1–42, Feb. 2012.
[46] I. You and K. Yim, "Malware obfuscation techniques: A brief survey," in *Proc. Int. Conf. Broadband, Wireless Comput., Commun. Appl.*, Nov. 2010, pp. 297–300.
[47] U. Bayer, C. Kruegel, and E. Kirda, *TTAnalyze: A tool for Analyzing Malware*. Princeton, NJ, USA: Citeseer, 2006.
[48] A. Vasudevan and R. Yerraballi, "Cobra: Fine-grained malware analysis using stealth localized-executions," in *Proc. IEEE Symp. Secur. Privacy (S&P)*, May 2006, p. 15.
[49] M. Sharif, A. Lanzi, J. Giffin, and W. Lee, "Automatic reverse engineering of malware emulators," in *Proc. 30th IEEE Symp. Secur. Privacy*, May 2009, pp. 94–109.
[50] (Mar. 2017). *Uncovering Cross-Process Injection With Windows Defender ATP*. [Online]. Available: https://www.microsoft.com/security/blog/2017/03/08/windows-defender-atp/
[51] (Jan. 2019). *Firefox Will Block DLL Injections gHacks Tech News*. [Online]. Available: https://www.ghacks.net/2019/01/21/firefox-will-block-dll-injections/
[52] (2022). *Google Chrome 72's Code Injection Blocking Detailed*. [Online]. Available: https://news.softpedia.com/news/google-chrome-72-s-code-blocking-detailed-524759.shtml
[53] M. E. Blog. (Nov. 2015). *Protecting Microsoft Edge Against Binary Injection*. [Online]. Available: https://blogs.windows.com/msedgedev/2015/11/17/microsoft-edge-module-code-integrity/

YANGZONG ZHANG is currently pursuing the Ph.D. degree with the School of Data Science, City University of Macau, China. His research interests include network security and cyber security.

WENJIAN LIU received the B.S., M.S., and Ph.D. degrees from the School of Electrical and Information Engineering, South China University of Technology, Guangzhou, China, in 1996, 2001, and 2010, respectively. He is currently an Associate Professor and the Vice Dean of the Faculty of Data Science, City University of Macau. His current research interests include intelligent transportation, intelligent healthcare, CV, Gen AI, and big data in psychology.

KAIIAN KUOK received the B.S. degree in applied physics from Jinan University and the M.S. degree in software engineering from the University of Macau. He is currently pursuing the Ph.D. degree in data science with the City University of Macau. As the Founder of Region Technologies Company Ltd., Macau, he has played a pivotal role in its growth and success. His current research interests include computer vision processing, sports technology, and big health.

NGAI CHEONG (Senior Member, IEEE) received the bachelor's degree in electronics engineering from the South China University of Science and Technology, China, in 1986, the master's degree in electronics engineering from South China Normal University, China, in 1989, and the Ph.D. degree in electrical and electronics engineering from the University of Macau, in 2001. He is currently a Professor with the Faculty of Applied Sciences, Macao Polytechnic University, Macau, China. His research interests include knowledge-based recommender systems in conceptual learning, computer-aided design, and theory of e-government. He is a member of IET and the Macao Association for Promotion of Science and Technology (MAPST).