JUGAAD: Comprehensive Malware Behavior-as-a-Service

Sareena Karapoola
sareena@cse.iitm.ac.in
Indian Institute of Technology Madras
Chennai, India

Nikhilesh Singh
nik@cse.iitm.ac.in
Indian Institute of Technology Madras
Chennai, India

Chester Rebeiro
chester@iitm.ac.in
Indian Institute of Technology Madras
Chennai, India

V. Kamakoti
kama@cse.iitm.ac.in
Indian Institute of Technology Madras
Chennai, India

ABSTRACT

An in-depth analysis of the impact of malware across multiple layers of cyber-connected systems is crucial for confronting evolving cyber-attacks. Gleaning such insights requires executing malware samples in analysis frameworks and observing their run-time characteristics. However, the evasive nature of malware, its dependence on real-world conditions, Internet connectivity, and short-lived remote servers to reveal its behavior, and the catastrophic consequences of its execution, pose significant challenges in collecting its real-world run-time behavior in analysis environments.

In this context, we propose JUGAAD, a malware behavior-as-a-service to meet the demands for the safe execution of malware. Such a service enables the users to submit malware hashes or programs and retrieve their precise and comprehensive real-world run-time characteristics. Unlike prior services that analyze malware and present verdicts on maliciousness and analysis reports, JUGAAD provides raw run-time characteristics to foster unbounded research while alleviating the unpredictable risks involved in executing them. JUGAAD facilitates such a service with a back-end that executes a regular supply of malware samples on a real-world testbed to feed a growing data-corpus that is used to serve the users. With heterogeneous compute and Internet connectivity, the testbed ensures real-world conditions for malware to operate while containing its ramifications. The simultaneous capture of multiple execution artifacts across the system stack, including network, operating system, and hardware, presents a comprehensive view of malware activity to foster multi-dimensional research. Finally, the automated mechanisms in JUGAAD ensure that the data-corpus is continually growing and is up to date with the changing malware landscape.

CCS CONCEPTS

• **Security and privacy → Malware and its mitigation**.

KEYWORDS

Dynamic Analysis, Malware, Run-time Behavior, Real-world, Testbeds

ACM Reference Format:
Sareena Karapoola, Nikhilesh Singh, Chester Rebeiro, and V. Kamakoti. 2022. JUGAAD: Comprehensive Malware Behavior-as-a-Service. In *Cyber Security Experimentation and Test Workshop (CSET 2022), August 8, 2022, Virtual, CA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3546096.3546108

CSET 2022, August 8, 2022, Virtual, CA, USA

ACM ISBN 978-1-4503-9684-4/22/08…$15.00
https://doi.org/10.1145/3546096.3546108

1 INTRODUCTION

Malware attacks are increasing at an alarming scale. The ramifications of these attacks vary widely, ranging from data breaches to business disruptions, reputation damage, financial loss, and even sabotage of critical infrastructures. With millions of malware variants reported every year, malware is continually evolving in potential and sophistication, posing significant challenges to security researchers. Confrontation of such an ever-evolving threat landscape requires an in-depth understanding of malware behavior in the wild, including their objectives, functionalities, and consequences. In fact, behavioral analysis of malware has recently gained traction in the arms race against malware due to its potential to detect zero-day malware.

Researchers glean insights into malware behavior from the run-time trails observed by executing malware samples on analysis frameworks. This demands access to a large corpus of recently reported *live* malware samples. The aspect of being live is important in the malware context, as its execution heavily depends on its communications to live remote servers called command-and-control (C&C) servers. These servers are short-lived and are typically pulled down in a few months after the malware is first reported, warranting a *timely* execution of the sample to elicit its real-world behavior.

Currently, malware research adopts two approaches, supported mainly by private enterprises, to address the demand for live samples. Given a hash of a sample to be tested, the first approach provides the outcome of analysis done by the Anti-virus (AV) engines housed by these enterprises [55]. The outcome includes the inference of the maliciousness of the sample, signatures, and reports from the analyses. However, these signatures and reports are limited by the capabilities of the available AV engines, whereas fostering unbounded research requires access to the raw behavioral data of these samples. The second approach supplies live malware samples to researchers for execution and subsequent analyses [39]. Such a model has multiple limitations. First, the distribution of live

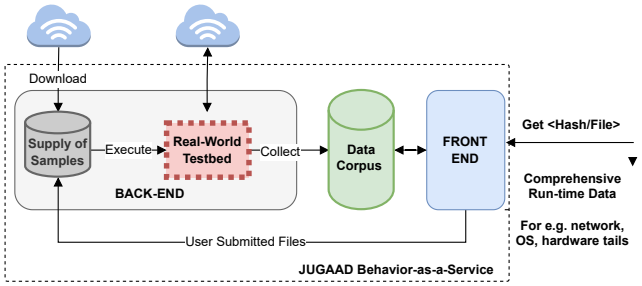

Figure 1: JUGAAD Framework for Malware Behavior-as-a-Service

samples is highly vulnerable to accidental execution and can be catastrophic if stringent access policies and processes do not govern the handling of these samples. Any leakage of samples can lead to potential misuse, warranting policies for ensuring accountability. Second, due to the potential risk and liabilities involved, such services are largely restricted and monopolized by a few private enterprises, incurring a high cost for a regular supply of new samples. Third, executing malware and eliciting their real-world behavior in a laboratory setting is challenging. Researchers typically prioritize safety and rely on non-connected virtualized frameworks for analyzing malware. However, modern malware can easily identify artifacts of such test environments and choose not to execute, thus evading analysis [40]. Consequently, the data collected does not represent malware behavior in the wild. In fact, a *precise* collection of close-to-real-world behavior requires timely execution of malware in unrestricted real-world conditions connected to the Internet, when its C&C servers are likely to be still active.

We propose an alternate model of malware *behavior-as-a-service* to meet the demands for the safe execution of samples. Such a model enables the users to submit hashes of malware samples for analysis and retrieve their precise and comprehensive run-time trails. We argue that such a *comprehensive* view of raw run-time data can replace the distribution of live malware samples, as behavioral research primarily relies on passively observable run-time trails rather than the malware executable contents.

To this end, we present JUGAAD, a framework to facilitate a comprehensive *behavior-as-a-service* for malware research. The framework shown in Figure 1 consists of a *front-end* that responds to user requests, a *growing data corpus* of precise and comprehensive malware behavior, and a *back-end* that continually feeds the data-corpus. The front-end provides API for users to submit program hashes or files and in return, outputs the corresponding data retrieved from the corpus. In cases where the data corresponding to the requested hash/file is not present in the corpus, the front-end submits the request to the back-end for processing. The back-end executes a *supply* of live malware samples on a *real-world testbed* and updates the collected behavior to the data-corpus. The testbed with Internet connectivity ensures real-world conditions for malware to operate while containing their malicious ramifications. Thus, JUGAAD alleviates the risks of handling and executing malware to the research community by facilitating the out-sourcing of the precise collection of malware behavior. The sustenance of the growing data corpus relies upon a continual supply of malware samples that is augmented by a regular feed from online repositories and files uploaded by users who use the JUGAAD service.

Following are the major contributions of this paper:

(1) A first-of-its-kind behavior-as-a-service model to provide precise and comprehensive real-world malware behavior for research, instead of the distribution of the risky malware samples. Unlike prior malware behavioral datasets [17, 18, 43, 52, 56], the growing corpus of malware behavior keeps JUGAAD up to date with the changing malware landscape.
(2) A real-world testbed ensuring close-to-real-world heterogeneous compute and Internet connectivity, with sufficient triggers for malware execution, demonstrated with 515 off-the-shelf devices.
(3) A framework with mechanisms for a comprehensive view of run-time malware activity observable across network, OS, and hardware. Unlike prior malware testbeds that support the collection of network and OS trails alone [5], JUGAAD provides simultaneous capture of hardware behavior along with these run-time trails.
(4) A framework with mechanisms for timely and large-scale execution of malware samples, tested up to 255 samples per day per network (58.6% faster than prior malware testbeds).

Following is the organization of rest of the paper. Section 2 provides the necessary background for the paper. Section 3 presents the related work. Section 4 discusses how a comprehensive behavior-as-a-service model can replace the need for distribution of malware samples. Section 5 describes the framework. Section 6 presents the implementation details. Section 8 discusses the limitations and future work in JUGAAD. Finally, section 9 concludes the paper.

2 BACKGROUND

Malware detection takes two broad directions based on the data they employ for analysis. *Static analysis* examines the contents of malware executable binaries to extract signatures and imply its maliciousness. However, such static signatures can be easily thwarted by techniques such as packing and obfuscation that change the malware binary without affecting its functionality. An alternate approach to malware detection is *dynamic analysis*, wherein maliciousness is inferred using the run-time behavior of malware. As the detection relies on observable behavior, dynamic analysis is immune to techniques that typically evade static analysis.

Behavioral Analysis. Dynamic analysis adopts two approaches to analyze malware. Active techniques [32, 42] repeatedly instrument the malware binary before execution to explore all execution paths in the malware, whereas passive techniques [4, 10, 57] merely execute malware and observe the behavioral trails. While such passive behavioral analysis can analyze the executed path alone, they are immune to evasive malware that can easily detect the instrumentation done by active techniques and choose not to execute [36].

Artificial intelligence (AI) driven run-time behavioral analysis has recently gained traction due to its upper edge in defense against evolving malware. Such techniques model good behavior and attempt to detect anomalies, thus facilitating zero-day malware detection. Primary to fostering such research is the availability of

ground-truth of malware behavior in the wild. However, collecting a precise representation of real-world malware behavior in a laboratory setting is challenging.

Conditions for Executing Malware. Malware execution can be catastrophic. Consequently, researchers typically rely on *virtualized non-connected* analysis environments to execute malware. However, modern malware is also evasive. They look for real-world conditions before they reveal their offensive behavior. Hence, they can quickly identify artifacts of analysis environments to detect them and remain dormant. Further, malware communications to its C&C servers are vital for its execution, calling for an active Internet connection while executing malware. Thus, eliciting real-world behavior requires executing malware in *real-world connected environments* while ensuring the *containment* of its ramifications.

Challenges in Large-Scale Analysis. Malware also poses challenges to large-scale evaluations. Specifically, malware execution impacts the system state in the analysis frameworks. Hence, each sample should be evaluated in a *clean initial state* of the frameworks. Resetting the analysis frameworks to their initial state makes large-scale analysis of thousands of samples time-consuming. Further, malware execution can cause frequent system crashes and halts beyond recovery using remote commands, requiring hard power restarts that significantly affect the large-scale automated analysis.

3 RELATED WORK

The demand for malware behavioral data in research is addressed using three approaches, namely, download of old malware samples, supply of live malware samples, or behavioral datasets, as discussed next.

Downloading older malware samples. Many online repositories allow researchers to download a corpus of malware samples that are a few years old [9]. However, with their C&C servers not available, most malware samples quit execution prematurely without exhibiting their real-world behavior.

Supply of live malware samples. Private enterprises such as VirusTotal provide premium services to download live malware samples [33, 39]. While such services are highly-priced at 82K$ for 12K malware samples a year, the main challenge lies in the safe handling and execution of these live malware samples.

Datasets of malware behavior. Multiple prior datasets present malware behavior for research [4, 15–18, 23, 35, 41, 43, 53, 56]. However, most of these works rely on virtualized analysis environment, thus lacking precise real-world behavior [16–18, 23, 35, 43, 52, 56]. On the other hand, datasets generated in real-world conditions are not open to the community [4, 15, 41, 53]. Additionally, these datasets are static with no mechanisms to keep pace with the evolving malware landscape.

In contrast, JUGAAD facilitates behavioral data-as-a-service to researchers. While it offloads the safe execution of live samples, it facilitates a dynamic and growing data-corpus that is regularly augmented with lately reported live samples.

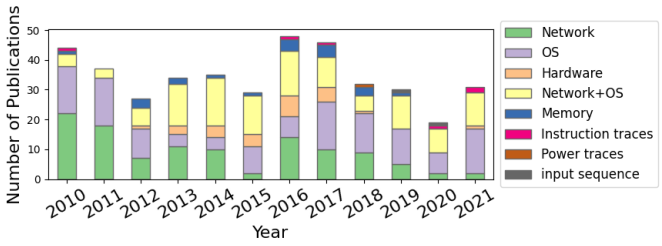

Figure 2: Behavioral data employed by prior research in the past decade

4 COMPREHENSIVE BEHAVIORAL DATA AS AN ALTERNATIVE TO MALWARE SAMPLES

JUGAAD advocates the distribution of malware behavioral data instead of live malware samples. The natural question is whether data can replace the need for distributing live samples in malware behavioral research. To answer this question, we first investigate the vast body of literature on dynamic malware detection to identify run-time characteristics that were used. Further, we show that any two programs can be distinguished by the run-time behavior.

Usage of samples in prior works. Figure 2 provides a summary of the run-time characteristics employed by 400 most cited prior research in dynamic malware detection since 2010. These works either rely on available datasets or generate data by executing malware for their research. The run-time trails of malware are observable at network (for instance, [1, 2, 4, 7, 8, 12, 18, 25–27, 31, 38, 43]), operating system (OS) (for instance, [6, 7, 10, 15, 17, 24, 26, 31, 34, 44, 56]), or, hardware (for instance, [3, 11, 19–21, 37, 41, 45, 49, 58, 59]). The network trails capture malware communications, including that to its command-and-control (C&C) servers. On the other hand, OS trails present the system calls (for e.g. file or registry operations) made by the malware. In contrast, hardware trails include the micro-architectural events (e.g., number of cache misses) triggered during malware execution. These hardware events are measurable using hardware performance counters (HPC) available in modern processors [41]. More recently, researchers have explored the potential of memory snapshots to detect malware using a technique called volatile memory acquisition (for instance, [13, 47, 50, 51]).

Change in the program leads to change in behavioral trails. We argue that any change in a program leads to a change in run-time characteristics that can be visible in the artifacts captured during malware execution. These differences are evident in a comprehensive view of malware behavior. To verify the same, we consider a corpus of 10,000 programs containing an even distribution of benign, ransomware, downloader, cryptominer, deceptor, potentially unwanted applications, spyware, and backdoor programs. Figure 3 plots the distribution of pair-wise dissimilarity between the behavioral features observed in three example artifacts captured across the system stack, namely, network, OS, and hardware. The dissimilarity between two programs p_1 and p_2 is defined as:

$$\texttt{dissimilarity}(p1, p2) = 1 - \texttt{cosine\_similarity}(p1, p2) \ , \quad (1)$$

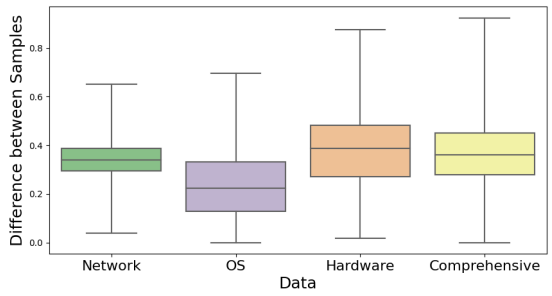

Figure 3: The dissimilarity between samples across the different run-time data trails. The rectangular boxes highlight the range of 50% of values in the distribution (i.e., the values between the first (Q1) and third (Q3) quartile). The horizontal line through the box indicates the median. The whiskers from each box represent the minimum and maximum in the distribution. The values between the minimum and Q1 (lower edge of the box) represent 25% of the values. Similarly, the values between the maximum and Q3 (upper edge of the box) indicate the remaining 25% of the distribution.

which measures the difference between the behavioral feature vectors of the two programs[1]. A dissimilarity of 0 indicates that the two programs have identical run-time trails, whereas 1 suggests that the two are distinct. Thus, most programs (>= 75%) differ in their network and OS behavior by a dissimilarity measure of $0.3-0.7$ and $0.15-0.7$, respectively. On the other hand, the hardware trails are comparatively more distinguishable with a dissimilarity of $0.3-0.9$ for most programs. While different artifacts can individually distinguish programs in varying capacities, a comprehensive view of these artifacts can capture most differences between the programs.

In the current implementation, JUGAAD provides simultaneous capture of three artifacts, including network, OS, and hardware behavior, that are predominantly used (Figure 2) for their effectiveness and decreased overheads in dynamic malware detection. For comprehensiveness, we intend to include memory snapshots and other trails such as instruction and power traces in JUGAAD, which we leave for future work.

5 JUGAAD FRAMEWORK

In this section, we discuss the JUGAAD framework, including its front-end and back-end, as shown in Figure 1. The front-end handles the user requests, whereas the back-end is responsible for the precise and comprehensive collection of malware behavior.

5.1 Front-end

The front-end presents the following Application Programming Interface (API) calls to the users to access the JUGAAD behavior-as-a-service as described in Figure 4.

Get data for hash. The API `GetDataForHash (hash h)`, allows the users to submit the hash `h` of a program and request for its behavioral data. In response, the front-end extracts the corresponding behavioral data from the data-corpus and returns it to the user as

[1]Cosine similarity measures the similarity between two vectors, and is measured by the cosine of the angle between two vectors.

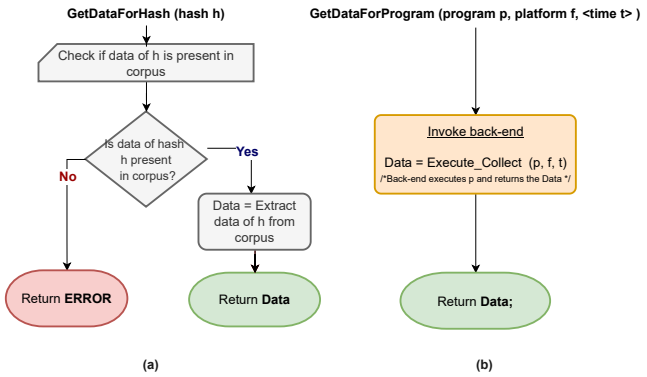

Figure 4: The APIs presented by JUGAAD front-end to the users. The API `GetDataForFolder` is similar to `GetDataForProgram`, where the former takes a folder of programs as input.

shown in Figure 4a. In cases where the data corresponding to the hash is not present in the data-corpus, the front-end returns an error.

Get data for a program. The API `GetDataForProgram (program p, platform f, ⟨ time t ⟩)`, allows the users to submit a program executable and request for the corresponding behavioral data. The input includes the program executable `p`, the platform `f` (e.g., Linux, Windows, Android) on which the program needs to be executed, and optionally, the time duration `t` for which the program execution should be observed while collecting the behavioral trails. In response, the front-end raises a request to the back-end, which executes the program for a time duration `t` and collects its behavioral trails. The collected trails are saved to the data-corpus and returned to the user. By default, the time duration `t` is configured as 2 minutes at the back-end, which is considered to be sufficient to elicit most of the malicious behaviors of malware [22].

Get data for a folder of samples. Alternatively, users may want to upload multiple files at once for the collection of behavioral trails. To this end, the API

`GetDataForFolder (program_folder F , platform f, ⟨ time t⟩)`, allows users to submit a folder of programs, along with specifying the platform and time for executing each sample in the folder. The front-end invokes the back-end to execute and collect the behavior of the samples and return the data to the user.

5.2 The Back-end

The primary functionality of the back-end is to supply *precise* close-to-real-world and *comprehensive* malware behavior to the data-corpus, which is used to serve the users. JUGAAD ensures precise behavioral data by facilitating: **(1)** timely execution of malware when their short-lived remote command and control (C&C) servers are highly likely to be active, and, **(2)** connected, yet contained real-world environment for the malware to execute. On the other hand, it facilitates a comprehensive view of malware activity with simultaneous capture of run-time trails across the system stack. Figure 5 illustrates the back-end of JUGAAD. The update and test engines together ensure a regular and timely update of the data-corpus to service the user requests. On the other hand, the real-world testbed

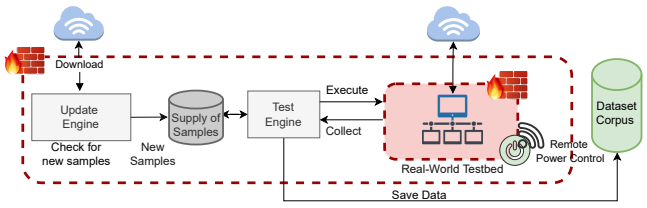

Figure 5: Back-end of JUGAAD.

provides connected real-world conditions for malware to operate while containing its malicious repercussions, and mechanisms to observe its comprehensive behavior.

5.2.1 **Timely Analysis of Malware.** Algorithm 1 describes the working of the update and test engines. The update-engine periodically crawls public malware repositories for newly reported malware and downloads them to its local supply of samples (Step 3-8). Further, the test-engine executes these samples immediately after the download on the real-world testbed. The timely execution ensures that the samples are executed when their C&C servers are still available. During the execution, the test engine collects multiple run-time artifacts (e.g., network communications, OS system calls, and micro-architectural events) across the system stack, which are later added to the data-corpus (`Execute_and_Collect` in Step 9). The default time for analyzing each sample is set as 2 minutes based on prior research on the minimum time window required to observe most of the malicious behaviors of malware [22]. Additionally, the test-engine services the user-submitted requests for data collection (Steps 10-15). It executes the programs submitted by the users and collects the behavioral trails, which are stored in the data-corpus and returned to the users (Step 15). Thus, the Algorithm 1 ensures the timely execution of a regular feed of new malware samples reported in the wild to maintain a growing data-corpus.

5.2.2 **Real-world Environment.** Modern malware are known to look for real-world conditions such as a network of diverse physical machines and Internet connectivity before they reveal their malicious behavior (refer Section 2). To this end, the real-world testbed provides a heterogeneous network of machines that can be employed as a *profiler* to execute malware. The testbed consists of desktop computers and single-board embedded platforms with varying operating systems (e.g., Linux, Windows). The diversity in machines and software environments not only provides sufficient triggers for malware to execute, but also enables JUGAAD to execute malware of diverse platforms. The testbed with auto-configuration capability is open, wherein new machines with specific environments can be added to the testbed seamlessly. Though these machines are connected in a bus topology, the testbed also can facilitate user-specific topologies for advanced analysis, such as the study of malware propagation.

Internet Connectivity and Containment. To provide connectivity while containing the ramifications, the back-end uses a dedicated Internet connection (ERNET [14]) that is isolated from the university network. Further, it connects to the Internet via a two-level firewall, as highlighted in Figure 5, which ensures containment of

Algorithm 1: JUGAAD Back-end

```
1  begin
2      while true do
           /* Update Engine                                   */
3          Crawl online repositories for newly reported samples
4          if updates are available then
5              NewhashList ← Hashes of newly reported malware
                samples
6              for h ∈ NewHashList do
7                  p ← Download hash h
8                  Supply-of-Samples ← p
                   /* Test Engine                             */
9                  Data-Corpus ← Execute_Collect (p)
10         Check for requests from front-end
11         if requests queued from front-end then
12             ListOfPrograms ← List of programs submitted by
                user
13             for p ∈ ListOfPrograms do
14                 Supply-of-Samples ← p
                   // Test Engine
15                 Data-Corpus ← Execute_Collect (p)
```

the malicious impact of executing malware, while allowing the malware to operate. The firewalls permit incoming communications to allow the malware to communicate with its C&C servers, while extensively scrutinizing outgoing communications to prevent malicious behavior from permeating outside the testbed. Implementing the two levels with different firewall models has advantages. The malware would need to compromise two separate firewalls to infect machines outside the testbed, which are less likely to be susceptible to the same malware. Likewise, external attackers would need to compromise two firewalls to attack the testbed.

While the firewalls allow initial handshakes of connections, it limits the rate or drops packets when the following scenarios/triggers from the testbed cross their respective thresholds: **(i)** DoS attempt: a high rate of outgoing packets from any machine; **(ii)** TCP Scan: a significant number of half-open TCP connections over time; **(iii)** SPAM: the number of email messages from the testbed; **(iv)** UDP Scan: the ratio of UDP packets from the malware to the unsuccessful responses (e.g., Internet Control Message Protocol port unreachable) received. While there is a possibility of some attacks like DoS to persist when network traffic patterns do not match the firewall rules, the risk is not unacceptably high. This is because the running time of every sample is restricted to a threshold, typically 2 minutes based on prior research [22]. After the execution, all machines in the testbed are reset to their clean initial state, which reduces the risk of spam, DoS, or unpredictable behavior to a small-time duration defined by the threshold. Finally, these rules are not exhaustive and would require continual monitoring and updates based on the malware classes that are being analyzed.

Stateless Evaluations. Each malware sample should be evaluated in a clean initial state or *baseline* of the testbed (refer Section 2). Unlike virtual machines, which can be easily reset to their baselines,

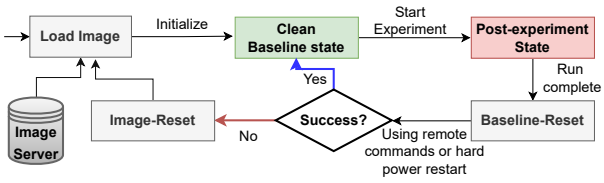

Figure 6: Two-level reset mechanism in JUGAAD.

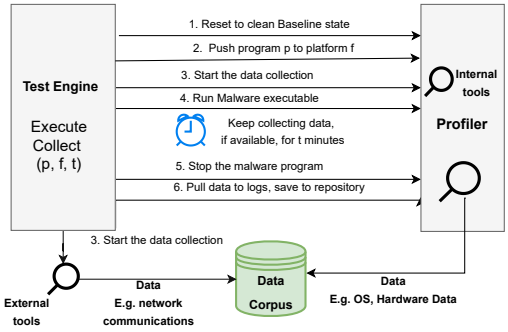

Figure 7: Data Collection in the testbed (Execute_and_Collect)

resetting all physical machines in the testbed for every sample execution is not trivial. To this end, JUGAAD employs a two-level reset mechanism as shown in Figure 6. The first level involves a quick low-overhead *baseline-reset* using software methods to restore a physical machine to its saved clean baselines on restart. JUGAAD initiates a baseline-reset with a restart of all machines using remote commands. In cases where the malware makes the system inaccessible remotely, JUGAAD uses the smart power switches to hard-restart the machines. Further, critical faults may arise, corrupting the baselines such that the baseline restore fails. Only in such scenarios JUGAAD performs an *image-reset*, which involves reloading of respective OS images from an image server.

5.2.3 **Comprehensive collection of malware behavior.** Figure 7 illustrates the working of the test-engine (`Execute_and_Collect`) to collect the comprehensive behavior of malware. First, it begins by resetting all the testbed machines to their clean baselines. Second, it chooses an appropriate machine in the testbed as a *profiler* to execute malware, and pushes the malware sample to it. For instance, it chooses a Windows machine to evaluate a Win32 malware. Third, it initiates the data collection. It starts the corresponding tools to capture artifacts inside the profiler. For instance, to capture OS system call trails, it starts the process monitoring tool [29] at the profiler. However, it is beneficial to observe some artifacts from outside the profiler. For instance, observing the network communications at the gateway that connects the testbed to the Internet, can capture not only the communications from the profiler but also malware interactions and their impact on other testbed machines. Accordingly, the test-engine starts such external tools at corresponding vantage points. Fourth, the test-engine executes the sample for a configured duration or provided by the user. Fifth, it stops the execution and the data collection tools. Finally, it saves

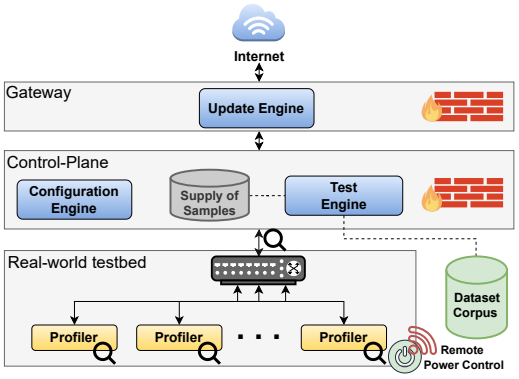

Figure 8: The implementation of back-end in JUGAAD

all the collected trails to the data-corpus along with the IP address of the profiler, and the process ID of the malware executable at the profiler.

6 IMPLEMENTATION

In this section, we discuss the implementation details of the JUGAAD framework.

We implement the front-end as an HTTPS web server in python to provide a public web interface to submit the program hash or files. The server also provides an HTTP-based public API to enable users to script submissions in any programming language. Inspired by software-defined networking, we divide the back-end into planes, namely gateway, control-plane, and testbed, to support customization and reconfiguration, as shown in Figure 8. The gateway connects the back-end to the Internet via a dedicated IP address in ERNET [14]. It also houses the update engine that feeds the supply of samples. The control-plane contains the configuration and test engines to initialize, operate and automate the back-end. We implement these modules using `bash` scripts and `Python`. While the configuration engine initializes the testbed (for e.g. software environment), the test engine automates the execution and data collection on the testbed. Table 1 lists the gateway, control-plane and testbed machines that are connected as in Figure 8 to realize the JUGAAD back-end. We next explain the implementation of timely analysis and the comprehensive data collection discussed in Section 5.2.

6.1 Timely Analysis

To facilitate access to newly reported malware, JUGAAD has subscriptions to premium services from online malware repositories such as Virustotal [54] that provide a daily feed of requested samples. We implement the back-end algorithm (Algorithm 1) in `Python`, which accesses the APIs provided by Virustotal to crawl for the availability of samples and download them. These samples are immediately executed by the test engine to ensure timely analysis.

Test engine (Execute-Collect). We implement the test engine in `Python`. For Windows-based profiler machines, the test engine uses `PowerShell Remoting` and `psexec` tool to remotely trigger the malware execution [28]. However, when remote execution is automated, the malware executes in the background without popping

Table 1: Devices in the open testbed

Device	Description	OS	Count	Role
Desktop	Intel i7, 8 cores, 64GB RAM, 2 1G Ethernet ports	Ubuntu v17.10	1	Gateway
Desktop	Intel i7, 8 cores, 64GB RAM, 2 1G Ethernet ports	Ubuntu v18.04	1	Control Plane
Desktop	Intel i5, 8 cores, 16GB RAM , 1G Ethernet port	Win 10 Pro	2	Testbed
Desktop	Intel i7, 8 cores, 64GB RAM , 2 1G Ethernet ports	Ubuntu v16.10	1	Testbed
Desktop	Intel x86 Atom, 1 Core, 2GB RAM, 1G Ethernet port	Win 7 Win 10	10	Testbed
Raspberry Pi 3 B+ 2	ARM Quad core, 1GB RAM, 1G Ethernet	Linux v3.14	2	Testbed
Galileo Gen 2	Intel Quark SoC 256MB RAM, 1G Ethernet	Linux v3.14	500	Testbed

Figure 9: The testbed in JUGAAD

its GUI window. To get the malware executed in the foreground, we configure the `psexec` tool to run the malware in the system account. For Linux-based machines, the test engine uses `SSH` service to push and execute a malware sample. The test engine also monitors the health of machines with a dummy `SSH connect` request. In case of a failure, it notifies the administrators with an email and resets the testbed using the smart power switch.

6.2 Real-world testbed

Figure 9 shows the real-world testbed we built in our lab in the back-end. We connect all machines in the testbed (refer Table 1) in a hierarchical bus topology using 28-port D-Link DGS-1210-24 and 24-port HPE-1920s switches. The testbed is *open* and *scalable*, as any new hardware with an Ethernet port can be connected to the switch to include in the network. It is also powered using WiFi-enabled smart switches to enable *remote hard-restart* of the testbed in cases of failures. Each machine in the testbed is assigned its management IP address at power-on by the Dynamic Host Control Protocol (DHCP) server at the control-plane. The control-plane configures and manages the testbed using the management IP addresses of the devices.

In the current implementation, all machines in the testbed are under one network. It is important to note that simultaneous execution of more than one malware sample would affect the precision of data as each sample can affect the system state in the entire network. The number of samples analyzed per day can be increased if we can divide the testbed into isolated virtual local area networks (VLAN). With the infrastructure of smart network switches (HPE-1920s switches), we intend to implement VLANs in future to enable parallel analysis of multiple samples. Further, we intend to enable custom topologies and forwarding behavior in the testbed in the future to facilitate long-term analysis such as malware propagation.

Heterogeneity. As evident from Table 1, the testbed has a heterogeneous mix of hardware (Raspberry Pi, Intel x86 Atom, Quark, i5, and i7 machines) and operating systems (different versions of Linux and Windows) to realize close-to-real-world conditions.

Isolation and Containment. JUGAAD uses a combination of `Snort` Intrusion Detection System [46] and `Zeek` network analysis framework [60] at the control-plane and gateway. While Snort can detect known exploits, Zeek can detect protocol violations and malformed headers using the built-in and custom scripts.

Stateless Evaluations. For baseline-reset, JUGAAD uses software like `Reboot Rx` on Windows and `SystemBack` on Linux [48]. The test-engine initiates a baseline-reset with an `SSH` command or a power-restart using the smart power switches in cases of `SSH` failures. During the operation of JUGAAD, we observed the need for such power-restart in several instances. In either case, the baseline-reset of all machines finishes in $\approx$ 2.64 minutes. For image-reset, JUGAAD loads the affected device with the OS image (created using `Clonezilla` disk imaging software) from a locally maintained image-server. However, image-reset is very rare, as we did not encounter any need for it during the analysis of more than 10K samples.

6.3 Comprehensive collection

In the current implementation, JUGAAD provides simultaneous capture of three artifacts, including network, OS, and hardware behavior. The control-plane captures network communications using `tshark` [30]. While external communications pass through the control-plane, inter-device communications in the testbed are mirrored to the control-plane using *port-mirroring* feature of the managed switches. At the profiler machine, we use `process monitor` for Windows, and `strace` for Linux to capture OS events of the malware process [29].

For HPCs, we use different interfaces based on the environment. While Linux provides `perftool` APIs to configure and fetch the counters from the userspace, the Windows OS requires some modifications. We design a custom Windows 10 Driver to read HPCs, configurable as per the underlying hardware and available events. Based on empirical observations, we configure the frequency of event logging to 10ms. It is critical to note that while the hardware may support hundreds of micro-architectural events, the number of HPC registers available physically is limited to 4-6 on most architectures. While time-multiplexing these events across the registers is a work-around, it can induce significant noise in the data. We

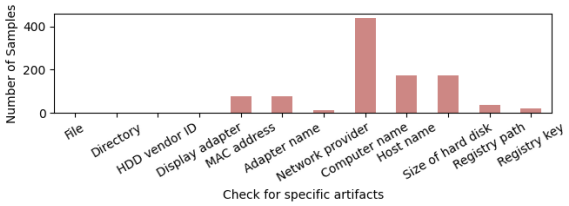

Figure 10: Evasion techniques found in 1000 randomly selected samples. Malware samples check for specific artifacts to identify virtualized analysis environments.

address this challenge with multiple binary executions, where limited events are counted during each execution. Finally, to prevent the corruption of collected OS and hardware data by malware like ransomware, they are temporarily stored in system folders (such as `C:\Windows\Temp\`) before being moved to the data corpus.

7 EVALUATION

In this section, we evaluate the precision of data collection in JUGAAD, the time taken for analysis, and the storage requirements of the data-corpus.

Precise collection of behavior. Modern malware are evasive and adopt diverse techniques to remain dormant in virtualized analysis environments [40]. We determine if JUGAAD is able to execute and observe the behavior of such malware. We take a random set of 1000 malware samples. We verify their system call traces to determine if these samples query for any specific information that aids in identifying analysis environments, such as differentiating virtual and physical machines. Figure 10 plots the count of samples using a subset of 12 techniques typically used for evasion [40]. We observe that at least 17% of samples had the signature system call to check if the hard-disk drive size and free space are small. More than 50% of samples verified the network MAC address, adapter name, and provider before continuing execution. The real-world conditions and the Internet connectivity enabled JUGAAD to ensure malware continues execution beyond these checks in their code.

Analysis Time. Table 2 computes the time taken by the back-end to analyze a given program sample. Analysis of each sample takes ≈ 338 seconds which includes steps 1-7 in Figure 7. The test-engine takes ≈ 158 seconds to reset the state of all testbed machines to their clean baselines in step 1. Hence, JUGAAD can analyse ≈ 255 malware samples per day (refer Table 2). The table also compares the time for state reset in JUGAAD with techniques used in public testbeds like DETER[12]. The two-level reset feature in JUGAAD (refer Section 5.2.2, Figure 6) enables 58.6% times faster reloads compared to DETER. The shorter time taken for state resets enables more number of sample analysis (255 per day per network) in JUGAAD as compared to DETER (154 per day per network).

We intend to increase the number of samples analyzed per day in JUGAAD by dividing the testbed into isolated VLANs. The current infrastructure of smart network switches (HPE-1920s switches) can easily enable such configurations to enable parallel analysis of multiple samples.

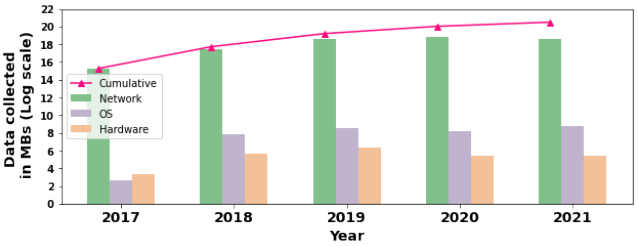

Figure 11: Growing data-corpus over years

Table 2: Time taken by JUGAAD back-end per sample (refer Figure 7)

Testbed	Baseline-reset (Step 1)	Experiment (Steps 2-7)	Time/ sample	Samples/ /day
JUGAAD	2 m 38 s	3 m	5 m 38 s	255
DETER	6 m 23 s	3 m	9 m 23 s	154

Growing data-corpus. To date, the data-corpus has 2.7 TB of data and 22M behavioral snapshots of 10,432 malware samples, including 7M network packets, 11.3M operating system call traces, and 3.3M micro-architectural events from hardware for 8 classes of malware. Table 3 shows the distribution of malware samples collected in the growing dataset. Figure 11 plots the growing storage requirements of the data-corpus in JUGAAD.

Table 3: Distribution of malware classes in the data-corpus

Class	%	Class	%	Class	%
Backdoor	13.5%	Spyware	16%	Ransomware	7.5%
Banker	14%	Benign	7%	Downloader	19.4%
PUA	10.8%	Deceptor	10.8%	Cryptominer	4.9%

8 DISCUSSION

We present a discussion on the comprehensiveness of the data collection and the sustenance of JUGAAD behavior-as-a-service model.

Comprehensive malware behavior. Malware behavior manifests through diverse artifacts that can be captured across the system stack during its execution. While network, OS, and hardware trails have been widely employed for their improved detection capabilities, many other run-time artifacts can be employed for detecting malware. Recently, the potential of memory snapshots, register contents, instruction opcode traces, and power traces have been explored to detect malware. Such collection modules can be easily plugged into the data collection framework of JUGAAD in Figure 7. We leave the inclusion of other such modules and novel artifacts for malware detection into JUGAAD as future work.

Sustenance. Sustaining the service model relies on a continual supply of newly reported malware samples. JUGAAD bootstraps such a supply with a premium subscription with private enterprises for downloading samples. However, with the continual operation and increasing user base, we envision a constant supply of malware samples from the users to ensure a growing data-corpus of malware behavior.

9 CONCLUSION

This paper presents JUGAAD, a malware behavior-as-a-service to present precise and comprehensive malware run-time characteristics for research. The beneficiaries of JUGAAD are malware researchers in academia and industry. It offloads the time and efforts of setting up a real-world evaluation infrastructure for comprehensive data collection, while alleviating the high risks involved in handling and executing potent malware. Prior efforts that provide malware analysis services present inferences on maliciousness of the user-submitted samples, that are limited by the capabilities of available state-of-the-art detection engines. In contrast, JUGAAD provides an unbiased comprehensive view of real-world malware behavior, enabling researchers to quickly explore and compare detection mechanisms to counter the evolving malware landscape.

ACKNOWLEDGMENTS

This research was supported by the Information Security Education and Awareness (ISEA) project from the Ministry of Electronics and Information Technology, Government of India. The authors thank the reviewers and the technical committee for reviewing the manuscript and providing constructive comments.

REFERENCES

[1] Blake Anderson, Andrew Chi, Scott Dunlop, and David McGrew. 2019. Limitless HTTP in an HTTPS World: Inferring the Semantics of the HTTPS Protocol without Decryption *(CODASPY '19)*. Association for Computing Machinery, New York, NY, USA, 267–278. https://doi.org/10.1145/3292006.3300025

[2] Blake Anderson and David A. McGrew. 2017. Machine Learning for Encrypted Malware Traffic Classification: Accounting for Noisy Labels and Non-Stationarity. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada, August 13 - 17, 2017*. ACM, 1723–1732. https://doi.org/10.1145/3097983.3098163

[3] Mohammad Bagher Bahador, M. Abadi, and Asghar Tajoddin. 2014. HPCMalHunter: Behavioral malware detection using hardware performance counters and singular value decomposition. *2014 4th International Conference on Computer and Knowledge Engineering (ICCKE)* (2014), 703–708.

[4] Karel Bartos, Michal Sofka, and Vojtech Franc. 2016. Optimized Invariant Representation of Network Traffic for Detecting Unseen Malware Variants. In *25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016*. 807–822. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/bartos

[5] Terry Benzel et al. 2007. Design, Deployment, and Use of the DETER Testbed. In *DETER Community Workshop on Cyber Security Experimentation and Test*.

[6] Davide Canali, Andrea Lanzi, Davide Balzarotti, Christopher Kruegel, Mihai Christodorescu, and Engin Kirda. 2012. A quantitative study of accuracy in system call-based malware detection. In *International Symposium on Software Testing and Analysis, ISSTA 2012, Minneapolis, MN, USA, July 15-20, 2012*, Mats Per Erik Heimdahl and Zhendong Su (Eds.). ACM, 122–132. https://doi.org/10.1145/2338965.2336768

[7] Tanmoy Chakraborty, Fabio Pierazzi, and V. S. Subrahmanian. 2020. EC2: Ensemble Clustering and Classification for Predicting Android Malware Families. *IEEE Trans. Dependable Secur. Comput.* 17, 2 (2020), 262–277. https://doi.org/10.1109/TDSC.2017.2739145

[8] Yehonatan Cohen and Danny Hendler. 2018. Scalable Detection of Server-Side Polymorphic Malware. *Knowledge-Based Systems* 156 (2018), 113–128.

[9] Da2dalus. 2022. *The Malware-Repo*. Retrieved March 2, 2022 from https://github.com/Da2dalus/The-MALWARE-Repo

[10] Sanjeev Das, Yang Liu, Wei Zhang, and Mahinthan Chandramohan. 2016. Semantics-Based Online Malware Detection: Towards Efficient Real-Time Protection Against Malware. *IEEE Trans. Inf. Forensics Secur.* 11, 2 (2016), 289–302. https://doi.org/10.1109/TIFS.2015.2491300

[11] John Demme, Matthew Maycock, Jared Schmitz, Adrian Tang, Adam Waksman, Simha Sethumadhavan, and Salvatore J. Stolfo. 2013. On the feasibility of online malware detection with performance counters. In *The 40th Annual International Symposium on Computer Architecture, ISCA'13, Tel-Aviv, Israel, June 23-27, 2013*. 559–570. https://doi.org/10.1145/2485922.2485970

[12] Xiyue Deng et al. 2017. Understanding malware's network behaviors using fantasm. In *The {LASER} Workshop: Learning from Authoritative Security Experiment Results ({LASER} 2017)*. 1–11.

[13] Farhood Norouzizadeh Dezfouli, Ali Dehghantanha, Ramlan Mahmoud, Nor Fazlida Binti Mohd Sani, and Solahuddin bin Shamsuddin. 2012. Volatile memory acquisition using backup for forensic investigation. In *2012 International Conference on Cyber Security, Cyber Warfare and Digital Forensic, CyberSec 2012, Kuala Lumpur, Malaysia, June 26-28, 2012*. IEEE, 186–189. https://doi.org/10.1109/CyberSec.2012.6246108

[14] ERNET. 2022. *Education & Research Network*. Retrieved March 2, 2022 from https://ernet.in/

[15] Hsien-De Huang, Tsung-Yen Chuang, Yi-Lang Tsai, and Chang-Shing Lee. 2010. Ontology-based intelligent system for malware behavioral analysis. In *FUZZ-IEEE 2010, IEEE International Conference on Fuzzy Systems, Barcelona, Spain, 18-23 July, 2010, Proceedings*. IEEE, 1–6. https://doi.org/10.1109/FUZZY.2010.5584325

[16] Impact. 2013. *IMPACT CyberTrust*. Retrieved March 2, 2022 from https://research.unsw.edu.au/projects/adfa-ids-datasets

[17] Mohammad Imran, Muhammad Tanvir Afzal, and Muhammad Abdul Qadir. 2015. Using hidden markov model for dynamic malware analysis: First impressions. In *12th International Conference on Fuzzy Systems and Knowledge Discovery, FSKD 2015, Zhangjiajie, China, August 15-17, 2015*. IEEE, 816–821. https://doi.org/10.1109/FSKD.2015.7382048

[18] Jae-wook Jang, Jiyoung Woo, Aziz Mohaisen, Jaesung Yun, and Huy Kang Kim. 2016. Mal-Netminer: Malware Classification Approach based on Social Network Analysis of System Call Graph. *CoRR* abs/1606.01971 (2016). arXiv:1606.01971 http://arxiv.org/abs/1606.01971

[19] Mikhail Kazdagli, Vijay Janapa Reddi, and Mohit Tiwari. 2016. Quantifying and improving the efficiency of hardware-based mobile malware detectors. In *2016 49th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 1–13.

[20] K. N. Khasawneh, N. Abu-Ghazaleh, D. Ponomarev, and L. Yu. 2017. RHMD: Evasion-Resilient Hardware Malware Detectors. In *2017 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 315–327.

[21] Khaled N. Khasawneh, Meltem Ozsoy, Caleb Donovick, Nael B. Abu-Ghazaleh, and Dmitry V. Ponomarev. 2015. Ensemble Learning for Low-Level Hardware-Supported Malware Detection. In *Research in Attacks, Intrusions, and Defenses - 18th International Symposium, RAID 2015, Kyoto, Japan, November 2-4, 2015, Proceedings*. 3–25. https://doi.org/10.1007/978-3-319-26362-5_1

[22] Alexander Küchler, Alessandro Mantovani, Yufei Han, Leyla Bilge, and Davide Balzarotti. 2021. Does Every Second Count? Time-based Evolution of Malware Behavior in Sandboxes. In *28th Annual Network and Distributed System Security Symposium, NDSS 2021, February 21-25, 2021*. The Internet Society. https://dx.doi.org/10.14722/ndss.2021.24475

[23] Statosphere lab. 2013. *The CTU-13 Dataset*. Retrieved March 2, 2022 from https://www.stratosphereips.org/datasets-ctu13

[24] Andrea Lanzi, Davide Balzarotti, Christopher Kruegel, Mihai Christodorescu, and Engin Kirda. 2010. AccessMiner: using system-centric models for malware protection. In *Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010, Chicago, Illinois, USA, October 4-8, 2010*. 399–412. https://doi.org/10.1145/1866307.1866353

[25] Chaz Lever, Platon Kotzias, Davide Balzarotti, Juan Caballero, and Manos Antonakakis. 2017. A Lustrum of Malware Network Communication: Evolution and Insights. In *IEEE Symposium on Security and Privacy (SP), 2017*. IEEE, 788–804.

[26] Chih-Hung Lin, Hsing-Kuo Pao, and Jian-Wei Liao. 2018. Efficient dynamic malware analysis using virtual time control mechanics. *Computers & Security* 73 (2018), 359–373.

[27] Chih-Hung Lin, Chin-Wei Tien, Chih-Wei Chen, Chia-Wei Tien, and Hsing-Kuo Pao. 2015. Efficient spear-phishing threat detection using hypervisor monitor. In *2015 International Carnahan Conference on Security Technology (ICCST)*. IEEE, 299–303.

[28] Microsoft. 2022. *PowerShell*. Retrieved March 2, 2022 from https://docs.microsoft.com/powershell

[29] Microsoft. 2022. *Process Monitor*. Retrieved March 2, 2022 from https://docs.microsoft.com/procmon

[30] Microsoft. 2022. *tshark*. Retrieved March 2, 2022 from https://www.wireshark.org

[31] Aziz Mohaisen, Omar Alrawi, and Manar Mohaisen. 2015. AMAL: high-fidelity, behavior-based automated malware analysis and classification. *computers & security* 52 (2015), 251–266.

[32] Andreas Moser, Christopher Kruegel, and Engin Kirda. 2007. Exploring multiple execution paths for malware analysis. In *2007 IEEE Symposium on Security and Privacy (SP'07)*. IEEE, 231–245.

[33] Yuval Nativ. 2022. *theZoo - A Live Malware Repository*. Retrieved March 2, 2022 from https://github.com/ytisf/theZoo/tree/master/malware/Binaries

[34] Matthias Neugschwandtner, Christian Platzer, Paolo Milani Comparetti, and Ulrich Bayer. 2010. Danubis–dynamic device driver analysis based on virtual machine introspection. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 41–60.

[35] University of Victoria. 2010. *ISOT Botnet and Ransomware Detection Datasets*. Retrieved March 2, 2022 from https://www.uvic.ca/ecs/ece/isot/datasets/botnet-ransomware/index.php

[36] Ori Or-Meir, Nir Nissim, Yuval Elovici, and Lior Rokach. 2019. Dynamic Malware Analysis in the Modern Era - A State of the Art Survey. *ACM Comput. Surv.* 52, 5 (2019), 88:1–88:48. https://doi.org/10.1145/33297

[37] Huicheng Peng, Jizeng Wei, and Wei Guo. 2016. Micro-architectural Features for Malware Detection. In *Advanced Computer Architecture - 11th Conference, ACA 2016, Weihai, China, August 22-23, 2016, Proceedings (Communications in Computer and Information Science, Vol. 626)*, Junjie Wu and Lian Li (Eds.). Springer, 48–60. https://doi.org/10.1007/978-981-10-2209-8_5

[38] Roberto Perdisci, Wenke Lee, and Nick Feamster. 2010. Behavioral Clustering of HTTP-Based Malware and Signature Generation Using Malicious Network Traces. In *Proceedings of the 7th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2010, April 28-30, 2010, San Jose, CA, USA*. USENIX Association, 391–404. http://www.usenix.org/events/nsdi10/tech/full_papers/perdisci.pdf

[39] VirusTotal Premium. 2022. *VT Intelligence*. Retrieved March 2, 2022 from https://www.virustotal.com/gui/intelligence-overview

[40] Checkpoint Research. 2022. *Evasion Techniques*. Retrieved March 2, 2022 from https://evasions.checkpoint.com/

[41] Hossein Sayadi, Nisarg Patel, Sai Manoj P. D., Avesta Sasan, Setareh Rafatirad, and Houman Homayoun. 2018. Ensemble learning for effective run-time hardware-based malware detection: a comprehensive analysis and classification. In *Proceedings of the 55th Annual Design Automation Conference, DAC 2018, San Francisco, CA, USA, June 24-29, 2018*. ACM, 1:1–1:6. https://doi.org/10.1145/3195970.3196047

[42] Edward J Schwartz, Thanassis Avgerinos, and David Brumley. 2010. All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask). In *2010 IEEE symposium on Security and privacy*. IEEE, 317–331.

[43] Giorgio Severi, Tim Leek, and Brendan Dolan-Gavitt. 2018. Malrec: compact full-trace malware recording for retrospective deep analysis. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 3–23.

[44] Madhu K. Shankarapani, Kesav Kancherla, Subbu Ramamoorthy, Ram S. Movva, and Srinivas Mukkamala. 2010. Kernel machines for malware classification and similarity analysis. In *International Joint Conference on Neural Networks, IJCNN 2010, Barcelona, Spain, 18-23 July, 2010*. IEEE, 1–6. https://doi.org/10.1109/IJCNN.2010.5596339

[45] Baljit Singh, Dmitry Evtyushkin, Jesse Elwell, Ryan Riley, and Iliano Cervesato. 2017. On the Detection of Kernel-Level Rootkits Using Hardware Performance Counters. In *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, AsiaCCS 2017, Abu Dhabi, United Arab Emirates, April 2-6, 2017*. 483–493. https://doi.org/10.1145/3052973.3052999

[46] Snort. 2022. *SNORT*. Retrieved March 2, 2022 from https://www.snort.org/

[47] Joe Sylve, Andrew Case, Lodovico Marziale, and Golden G. Richard III. 2012. Acquisition and analysis of volatile memory from android devices. *Digit. Investig.* 8, 3-4 (2012), 175–184. https://doi.org/10.1016/j.diin.2011.10.003

[48] Horizon Data Sys. 2022. *Reboot Restore Rx*. Retrieved March 2, 2022 from https://horizondatasys.com

[49] Adrian Tang, Simha Sethumadhavan, and Salvatore J. Stolfo. 2014. Unsupervised Anomaly-Based Malware Detection Using Hardware Features. In *Research in Attacks, Intrusions and Defenses - 17th International Symposium, RAID 2014, Gothenburg, Sweden, September 17-19, 2014. Proceedings*. 109–129. https://doi.org/10.1007/978-3-319-11379-1_6

[50] Jacob Taylor, Benjamin P. Turnbull, and Gideon Creech. 2018. Volatile Memory Forensics Acquisition Efficacy: A Comparative Study Towards Analysing Firmware-Based Rootkits. In *Proceedings of the 13th International Conference on Availability, Reliability and Security, ARES 2018, Hamburg, Germany, August 27-30, 2018*, Sebastian Doerr, Mathias Fischer, Sebastian Schrittwieser, and Dominik Herrmann (Eds.). ACM, 48:1–48:11. https://doi.org/10.1145/3230833.3232810

[51] Vrizlynn L. L. Thing and Zheng Leong Chua. 2013. Smartphone Volatile Memory Acquisition for Security Analysis and Forensics Investigation. In *Security and Privacy Protection in Information Processing Systems - 28th IFIP TC 11 International Conference, SEC 2013, Auckland, New Zealand, July 8-10, 2013. Proceedings (IFIP Advances in Information and Communication Technology, Vol. 405)*, Lech J. Janczewski, Henry B. Wolfe, and Sujeet Shenoi (Eds.). Springer, 217–230. https://doi.org/10.1007/978-3-642-39218-4_17

[52] UCI. 2017. *Dynamic Features of VirusShare Executables Data Set*. Retrieved March 2, 2022 from https://archive.ics.uci.edu/ml/datasets/Dynamic+Features+of+VirusShare+Executables

[53] Jorge Maestre Vidal, Ana Lucila Sandoval Orozco, and Luis Javier García-Villalba. 2017. Alert correlation framework for malware detection by anomaly-based packet payload analysis. *J. Netw. Comput. Appl.* 97 (2017), 11–22. https://doi.org/10.1016/j.jnca.2017.08.010

[54] VirusTotal. 2022. *VirusTotal*. Retrieved March 2, 2022 from https://www.virustotal.com/

[55] VirusTotal. 2022. *VTAPI Getting Started with v2*. Retrieved March 2, 2022 from https://developers.virustotal.com/v2.0/reference/getting-started

[56] Qi Wang, Wajih Ul Hassan, Ding Li, Kangkook Jee, Xiao Yu, Kexuan Zou, Junghwan Rhee, Zhengzhang Chen, Wei Cheng, Carl A. Gunter, and Haifeng Chen. 2020. You Are What You Do: Hunting Stealthy Malware via Data Provenance Analysis. In *27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020*. The Internet Society. https://www.ndss-symposium.org/ndss-paper/you-are-what-you-do-hunting-stealthy-malware-via-data-provenance-analysis/

[57] Xueyang Wang, Sek Chai, Michael Isnardi, Sehoon Lim, and Ramesh Karri. 2016. Hardware Performance Counter-Based Malware Identification and Detection with Adaptive Compressive Sensing. *ACM Trans. Archit. Code Optim.* 13, 1, Article 3 (March 2016), 23 pages. https://doi.org/10.1145/2857055

[58] Xueyang Wang and Ramesh Karri. 2013. NumChecker: detecting kernel control-flow modifying rootkits by using hardware performance counters. In *The 50th Annual Design Automation Conference 2013, DAC '13, Austin, TX, USA, May 29 - June 07, 2013*. ACM, 79:1–79:7. https://doi.org/10.1145/2463209.2488831

[59] Xueyang Wang and Ramesh Karri. 2016. Reusing Hardware Performance Counters to Detect and Identify Kernel Control-Flow Modifying Rootkits. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 35, 3 (2016), 485–498. https://doi.org/10.1109/TCAD.2015.2474374

[60] Zeek. 2022. *The Zeek Network Security Monitor*. Retrieved March 2, 2022 from https://www.zeek.org/