MEDUSA: Malware detection using statistical analysis of system's behavior

Muhammad Ejaz Ahmed
Data61 CSIRO
Sydney, Australia
ejaz.ahmed@data61.csiro.au

Surya Nepal
Data61 CSIRO
Sydney, Australia
surya.nepal@data61.csiro.au

Hyoungshick Kim
Sungkyunkwan University
Suwon, South Korea
hyoung@skku.edu

***Abstract*—Traditional malware detection techniques have focused on analyzing known malware samples' codes and behaviors to construct an effective database of malware signatures. In recent times, however, such techniques have inherently exposed limitations in detecting unknown malware samples and maintaining the database up-to-date, as many polymorphic and metamorphic malware samples are newly created and spread very quickly throughout the Internet.**

To address the limitations of existing signature-based malware scanners, we take a different view and focus on designing a novel malware detection framework, called MEDUSA (MalwarE Detection Using Statistical Analysis of system's behavior), for building a model for a system's behaviors with normal processes. Unlike traditional approaches for malware detection, MEDUSA has the potential to effectively detect unknown malware samples because it is designed to monitor a system's behavior and detect significant changes from the system's normal status. In this paper, we specifically discuss several important considerations that must be taken into account to successfully develop MEDUSA in practice.

***Index Terms*—malware detection, anomaly detection, system behavior, system profile, system artifacts.**

I. Introduction

Traditionally, malware detection tools use signature and/or suspicious behaviors of malware, which can be generated by program analysis techniques such as static analysis[1] and dynamic analysis[2] [1], [13]. Even though malware analysis still requires a manually intensive task for analysts, several automated malware detection techniques have been introduced [1]–[4], [6]–[9]. In recent years, the adoption of machine learning algorithms has particularly received significant attention for developing more effective malware scanners that can effectively reduce human analysts' workload [1], [2]. Typically, a massive number of malware samples is collected from several different sources and (automatically) analyzed to construct a database of malware samples for detecting malware on the systems [5]. This approach, however, only provides a partial solution to prevent known malware samples, and is not effective against new malware samples [1]. Because new malware samples showing-up everyday are recently grown up to 285,000 [12], existing anti-malware tools are exposed to their inherent limitations. In this paper, we call these traditional approaches using known malware samples as a *malware-centric* approach.

In this paper, we suggest a *system-centric* approach using a deviation-based outlier detection model. Our key idea is to quantify the degree of deviation of a system's behavior by constantly monitoring various system artifacts (e.g., frequency of I/O operations) on a target system. We surmise that a malware infected system may generate suspicious or abnormal events that can impact the behavior of the target system.

To implement this approach, we need to create a target system's several profiles (i.e., models) by learning the system's status with normal applications alone. There have been numerous studies to model a system's normal behavior to detect malware [6], [7], [11]. However, mostly such approaches leverage only a specific dimension of the system, such as CPU [9], network [11], [13], [14], hardware features [7], registry [9], or Windows prefetch files [8], etc, to detect malware activities. In practice, the system's behavior is often not reliable and may evolve over time, e.g., applications are installed/removed which may effect a normal behavior, thus such approaches may struggle with accuracy. Motivated by the above mentioned problems, we present MEDUSA: **M**alwar**E** **D**etection **U**sing **S**tatistical **A**nalysis of a system's behaviors for building a model with normal processes' usage patterns on the system's resources. For MEDUSA, we monitor system's event logs, file operations, networking, and registry activities of applications/processes to generate the system's normal profiles and detect suspicious activities. The advantage of the proposed system-centric approach is that unlike malware-centric approaches, it does not need to be trained on whole families of malware; instead it focuses more on the system specific artifacts and features to detect abnormalities.

The organization of this paper goes as follows: Section II summarizes the background and related work. Section III presents the motivation and problem statement. Section IV explains how a system's artifacts could be used in profiling the system's behaviors. Section V discusses our proposed approach for malware detection. In Section VI, we conclude this paper.

[1]Static analysis is the testing and evaluation of a program by examining the code alone without executing the program.

[2]Dynamic analysis is the testing and evaluation of a program by running the program in a controlled environment (e.g., sandbox, virtual machine, emulator, etc).

II. Related work

A. Background

Most state-of-the-art malware detection systems rely on automated analysis of malware samples using machine learning techniques. Feature extraction is one of the key component of automated analysis in which static and/or dynamic code analysis is popularly used to extract features and patterns of malware samples [1]–[6]. Based on the extracted features, machine learning models are leveraged to perform classification. In the following, we briefly provide an overview of static and dynamic analysis techniques for malware detection.

1) Static analysis: Static analysis refers to the analysis of program codes without executing them. In order to analyze a program at the binary code level, it is first decompressed/unpacked if it is compressed by third-party tools such as UPX, and ASPack shell. In case of window executable files, they need to be disassembled using dis-assembler tools (e.g., IDA pro, PEiD, CFF Explorer, Hiew, or Feddler, etc.) to display malware codes as Intel x86 assembly instructions. Memory dumping tools play an important part in obtaining the protected codes from the main memory and dump them to a file. These tools are effective in cases when the program source or binary files are packed and are difficult to disassemble. For static analysis, most of the literature rely on Windows API calls, byte n-grams, opcodes, strings, and control flow graphs (CFGs) [1].

2) Dynamic analysis: Dynamic analysis executes malware in a controlled environment (e.g., sandboxes) to observe its behavior. The observation points in dynamic analysis are function input/output parameters analysis, function call monitoring, information flow tracking, etc. The most common dynamic analysis tools include strace (Linux tool), Valgrind, Pin and DynamoRIO. The following execution environments are used for performing dynamic analysis:

- Debugger: Debuggers such as Windbg, GDB, or Softice are used in performing fine-grained analysis of executable files at an instruction level. However, recently developed sophisticated malware are able to detect analysis environments by observing the changes made to their codes, and thus can bypass the detection by altering their behavior. To counter such anti-debug malware, VAMPiRE can support stealthy breakpoints by leveraging the virtual memory.
- Virtual machine: Virtual machines or sandbox environments are used to observe malware behavior by executing it in a virtual environment. However, there are certain types malware that are able to detect sandbox/virtual environments by exploiting system features such as "Recently Open Files" and the lack of sufficient number of processes. To counter such evasive malware, system artifacts are used to mimic characteristics of real systems [2], [3].
- Simulator: These tools run malware in a controlled environment while monitoring their actions. Detours and CWSandbox are examples of such tools.

Features extracted from static and/or dynamic analysis are employed by machine learning models for malware detection.

B. Literature overview

1) Virtualization artifacts: Most cyber-security solutions rely on dynamic analysis techniques that load potentially malicious content in a controlled environment for analysis purposes [13]- [14].

Kirat et al. in [4] proposed a system called "BareCloud" which automatically detects malware by performing analysis with numerous samples on a bare-metal reference system. Their key idea is to detect deviations in the dynamic behavior of a sample when executed in several analysis environments. Malware samples are first filtered using Anubis (malware analysis framework) to select interesting samples exhibiting environment-sensitive behavior. The filtered samples are then executed on the cluster of bare-metal dynamic analysis hosts and on three other malware analysis systems namely, Ether, Anubis, and Cuckoo Sandbox. The execution of a malware sample is performed under different analysis systems. The behavioral patterns are extracted from each of these analysis systems, and behavior deviation scores are computed to detect malware samples.

2) Record and replay-based attack provenance detection: Many provenance tracking systems or attack investigation tools [15], [16] monitor and log interesting system events to identify the processes that are secretly interacting with unknown remote hosts and/or accessing/modifying sensitive files. The previous and current behavior of suspicious processes are compared to determine if they are compromised.

RAIN, a Refinable Attack INvestigation system [16] is one of such systems that record all system-call events during execution. It performs instruction-level dynamic information flow tracking (DIFT) during replay to effectively detect fine-grained causal relationships between processes and objects (e.g., between files and network endpoints) constructed during the execution of user processes. RAIN consists of two main components: the target host and the analysis host. In the target host, the kernel module of the RAIN logs all system calls that user processes have requested, including the return values and parameters from the system calls. The log record is leveraged to generate a provenance graph. The target host then sends all the collected system calls log to the analysis host. Analysis host utilizes all the received system call logs to construct provenance graph of the whole system that contains many security causality events. In the analysis host, RAIN first detects triggering points that may represent suspicious events in the provenance graph (e.g., accessing a sensitive file). Next, it performs reachability analysis, i.e., downstream, upstream, and point-to-point analysis, to construct security-sensitive provenance subgraphs. Lastly, the provenance graph is refined to detect the actual behavior and the damages caused due to the sophisticated attack.

Similarly, Panorama [15] exploits malware-specific information access and processing behavior to detect numerous malware categories breaching users' privacy. Panorama performs

the following three steps to detect stealthy malware samples: test, monitor, and analyze. Their approach examines a malware sample by first loading it into an analysis environment and running a series of automated tests on it. Each test generates events that introduce sensitive information into the system in a way that is not destined for the sample under analysis. Panorama monitors and records the behavior of a sample during different tests. Finally, analysis is performed on the recorded behavior of a test sample to detect malware. Based on the generated taint graphs, Panorama performs an automatic detection and analysis of malicious code from numerous categories.

3) Cloud-based detection: Dynamic analysis based approaches such as Crowdroid [17] analyze application's behavior as a mean for detecting maliciousness in Android platform. Crowdroid is based on collection of traces from many real users based on crowdsourcing. Crowdroid monitors all Kernel-level system calls and sends them to a remote server. The remote server after receiving the data parses them, and creates a system call vector for each user. Based on this, the behaviour data are created for each application. Finally, a clustering algorithm is deployed on the remote server to partition the dataset into benign and malicious applications.

4) Graph-based semantic malware detection: Semantic malware detection approach [18] is a technique for automatically extracting and learning malware signatures from a small set of samples of a malware family. Their key idea is to find a maximally suspicious common subgraph (MSCS) that is shared among all known instances of a malware family. Malware semantics represented by the MSCS depicts the shared functionality between multiple Android applications in terms of inter-component call relations and respective metadata. Their approach identify such maximally suspicious common subgraphs. Once semantic malware signature is learned, their approach uses a combination of static analysis and signature matching algorithms to determine whether the application under consideration matches the semantic signature (already given) characterizing a given malware family.

5) Heuristic and signature-based detection: Heuristic-based malware detection techniques [19] are based on rules/patterns determined by domain experts to discriminate malware samples and benign files. These rules/patterns should be generic enough to be consistent with variants of the same malware threat, but not falsely matched on benign files.

The signature-based methods are widely employed by malware detection antivirus tools to recognize various threats. A signature is a short sequence of bytes, which is often unique to each known malware, allowing newly encountered files to be correctly identified with a small error rate.

III. Motivation and problem statement

Most literature on malware detection rely on analyzing malware samples in a controlled environment to observe their behavior and extract insights to aid malware detection process. In the following, we discuss shortcomings relating to the above mentioned approaches, and provide our proposal for malware detection.

Malware analysis in a controlled environment (e.g., sandboxes, emulators, or virtual machines, etc.) widely used by cyber-security community faces the challenge that an environment-aware malware sample alters its behavior once it detects that it is being executed in a virtual environment. For instance, Najmeh et al. in [2] presented a sandbox evasion technique which exploits the "wear and tear" that inevitably occurs on real systems due to its normal use. They rely on the fact that by moving beyond how realistic a system looks like to how realistic its past use looks like, malware samples can effectively evade the detection system. They investigated the feasibility of this evasion strategy by leveraging the wear-and-tear artifacts collected from real user devices. They demonstrated that by using simple decision trees derived from the analyzed data, malware samples can detect an artificial environment with an evasion accuracy of 92.86%. Therefore, state-of-the-art malware detection systems are required to take into account this challenging problem while designing malware detection systems.

Typically, multiple monitoring environments are leveraged to perform various tests on given malware samples. For instance, BareCloud [4] used four malware analysis environments including virtualization, emulation, hypervisor, and bare-metal in which each malware sample is run over these environments to fully understand its behavior. Similarly, RAIN [16] leverages two hosts, i.e., the target host and the analysis host to collect system logs and perform analysis on the collected logs, respectively. Panorama [15] uses a separate analysis environment to observe the behavior of a malware under the controlled environment. Using multiple environments (e.g., analysis hosts) incur extra maintenance and management cost. Moreover, numerous samples from different malware families are required to form policies for detection systems.

Record and replay approaches rely on deep examination of malware behaviors using dynamic analysis techniques [15], [16]. However, record and replay-based approaches consume large amount of computational resources for analysis and may not be feasible due to exponential growth in malware samples over time.

Traditional signature-based malware detection approaches face big challenges due to the fact that they can be outpaced by the malware writers. For instance, according to Symantec's report, about 1.8 million malware signatures were released in 2008 which resulted in 200 million detection per month. However, in 2013, the suspicious files collected by the cyber-security lab of Kingsoft reached over 120 million, out of which 41.26 million (34%) were detected as malware. Malware signatures may change over time, e.g., even though numerous malware samples have been detected and blocked, there still remains a significant number of such samples that have been generated/mutated resulting in evasion of traditional signature-based anti-virus scanning tools. Moreover, due to economic benefits, malware authors are now resorting to

different techniques such as instruction virtualization, packing, obfuscation, polymorphism, emulation, and metamorphism to bypass detection systems.

Heuristic-based detection are often based on some heuristics such as the insertion of hooks into a certain library or system interfaces or the monitoring of modifications to the registry. Such heuristics are not exclusive to malware behaviour; they can result in high false positive and false negative rates. For example, many legitimate programs access and modify registry entries. Thereby, just because an application creates hooks in the registry does not necessarily mean that it is malicious. Furthermore, a malware may try to hook a library or system call interfaces that the detector does not monitor in order to evade the detection system. Making it even worse, as many rootkits hide in the kernel, most of such heuristics cannot detect them as they may not necessarily modify any visible registry entries, library or system call interfaces.

To that end, instead of focusing on malware-centric approach, we take a radically different view and focus on a system-centric approach in which system artifacts are used to model the system's baseline behavior. System artifacts include features specific to process, file, network, and registry activities. These features can fingerprint the system's behavior, and any deviation from a baseline could be further analyzed to detect malware.

Employing system artifacts for malware detection provide the following benefits: first, rather than learning signature/behaviour pattern of numerous malware samples, the proposed system-centric approach relies only on system artifacts to detect abnormal behavior of the system. Since our approach is not based on a dedicated infrastructure for malware analysis, it is efficient in terms of computational cost, and management requirements. Second, new and unknown malware samples can be detected by analyzing system artifacts. Third, since system's baseline behavior is user-specific, the machine learning model is also expected to be user-specific because if a user interact with the system moderately, the machine learning model will be fairly lightweight and simple.

IV. System artifacts and deployment challenges

We aim to learn system's normal behavior by exploiting system artifacts during its normal operation. Here we discuss an indicative set of main categories of the system artifacts that can be used to model its baseline behavior. Any deviation from the system's baseline profile may be the result of either benign or malicious samples. Our list of artifacts presented here to model the system's baseline profile is clearly not comprehensive and we are almost certain that there are many good candidate sources of artifacts that may be included.

A. System artifact categories

System behavior depends on the user's pattern of system usage. For example, in case of Microsoft Windows host, several system artifacts are available to model user's activity pattern such as the number of active processes, the number of applications installed, the usage behaviour within each application, resources utilization, etc. Such artifacts could be used to build system's baseline usage profile. In the following, we present a brief description for the main system artifacts that could be leveraged to model the system's baseline profile. The complete set of system artifacts is given in Table I.

TABLE I
List of system artifacts.

Category	Name	Description
Process	Total processes	# of processes
	sysevt	# of system events
	CreationDate	Date the process begins executing.
	syssrc	Sources of system events.
	sysapp	Sources of systems' application events.
	ExecutablePath	Path to the executable file of the process.
	ExecutionState	Current operating condition of the process.
	InstallDate	Date an object is installed.
File	ReadOperationCount	# read operations performed.
	ReadTransferCount	Amount of data read.
	WriteOperationCount	#of write operations performed.
	WriteTransferCount	Amount of data written.
	PeakPageFileUsage	Maximum amount of page file space used during the life of a process.
	VirtualSize	Current size of the virtual address space that a process is using.
	PageFileUsag	Amount of page file space that a process is using currently.
Network	ARPCacheEntries	# of entries in the ARP cache.
	tcpConnections	# of active TCP connections.
	AvgBent	Average # of bytes per second sent by the process in the last minute.
	AvgBSeceived	Average # of bytes per second received by the process in the last minute.
	AvgBTransferred	Average # of bytes transferred (sent and received) per second by the process in the last minute.
	LPortConnection	Local port of the connection.
	RPortConnection	Remote port of the connection.
	LPort	Local port number in use by a process.
	RPort	Remote port number in use by a process.
	Pro	Network protocol used by a process.
Registry	installCount	# of registered software installers.
	autoRunCoun	# of programs that automatically run at system startup.
	totalSharedDlls	Legacy DLL reference count.
	totalAppPaths	# of registered application paths.
	totalActiveSetup	# of Active Setup application entries.
	usrassistCount	# of entries in the UserAssist cache (frequently opened applications).
	FireruleCount	# of rules in the Windows Firewall.

1) Process: System properties such as the number of running processes with corresponding activity patterns are directly related and could be used to model the system's baseline profile. The event log of Windows systems is yet another rich source of information about the current state and past activity of the system. In Microsoft Windows operating system, various types of events are recorded in the event

log, that includes application, setup, security, and system-level events of different administrative ratings. With the continuous use of a system, the number of processes/applications are expected to increase over time. Consequently, a large number of records will have been accumulated in the events log. Besides the simple count of system events, additional aspects indicative of past use could also be considered, such as the number of read operations performed, the amount of data read since the process is created, the number of events from user applications, and the time difference between the first and last event.

Newly created process can be monitored by analyzing the list of events it performed since its creation. Some recent malware, e.g., cryptojacking uses the resources of the victim's computer secretly to mine cryptocurrency. Since mining cryptocurrency is a CPU intensive job, the victim may clearly observe degrading performance when a mining algorithm is running in the background.

2) File: Users activities may involve file operations including file creation, deletion, and modification, etc. File system related activities of a user play an important role in profiling the user's normal activity. Under certain circumstances, such file-related operations play crucial role to detect abnormal activities, e.g., suspicious processes may aggressively manipulate file system. An I/O operation contains the process name which requested it, time stamp, operation type, file system path and the pointers to the data buffers along with the corresponding entropy information in read/write requests [20]. All I/O operations are converted to a sequence of I/O requests.

File operations under the normal system use does not seem to fluctuate a lot. However, certain malware such as ransomware perform massive file and encryption operations that clearly show deviation from the system's baseline behavior. Additionally, stealthy malware such as key-loggers rely on writing logs to files.

3) Network: Network activities of a user is a strong indicator of the system's baseline behavior. When a host sends a packet to a remote destination, it consults the system's address resolution protocol (ARP) cache to find the physical address, i.e., MAC, of the gateway, or the address of another host in the local network. System's network activities can also be extracted from the events log. Network activities play an important role in profiling the system's normal behavior. Any anomaly in the network activity pattern may result due to certain malware such bot, spyware, etc.

4) Registry: The Windows registry is a big source of information about numerous aspects of a system. When an application or driver is installed on a Windows platform, it typically stores many key-value pairs in the registry.

Registry artifacts that are considered here are related to the footprints of the applications that have been installed in the system (e.g., a number of installed applications), as well as those that have not been fully uninstalled, such as orphaned entries left-over from the removed programs and the listed DLLs that do not exist on disk. We believe that these artifacts can help in fingerprinting the usage patterns of a user.

The above mentioned artifacts will suffice to model the system's baseline profile. In the following, we discuss salient features of the proposed system artifacts-based malware detection.

B. Advantages of the system artifacts-based detection

Here we discuss major advantages of the proposed system artifacts-based malware detection.

1) Reduced learning space: Unlike malware-centric approaches that require large number of samples from various malware families to learn signatures and/or behavior patterns, our system-centric approach mainly focuses on the system-level artifacts that drastically reduce the learning space. If an anomaly in the system's resource use is detected, the system will result raise a warning flag which is further analyzed to detect a potential malicious activity.

Additionally, by focusing only on interesting processes for detecting an abnormal behavior, the analysis overhead can further be reduced. For instance, previous works focus on pruning the process's behavior graph [16] or filtering interesting processes to save processing. System-centric approach can significantly reduce the learning complexity, and be much more practical to be deployed on real systems.

2) Activity pattern-based model: As discussed earlier, a system's usage profile is user-specific, i.e., the number of installed applications, the usage patterns of different applications, and the intra-application usage patterns are unique for each user. As a result, the learned machine learning model based on the system's usage pattern is also expected to be user-specific, and may not be highly complex under moderate system usage. Unlike traditional malware detection approaches that deploy complex machine learning models based on a huge number of samples from all malware families, the proposed system-centric based approach relies on simple and efficient models for malware detection.

3) Reduced infrastructure: Unlike dynamic analysis approaches that perform analysis in a controlled environment requiring multiple analysis environments [4], [13], [15], [17], the proposed system-centric approach relies only on system artifacts. Anomalies in terms of system artifacts once detected are analyzed to perform the detection.

4) Resilient against obfuscated, packed files, and zero day attacks: A major shortcoming of static analysis techniques is that they are not much effective when binary or source code files are obfuscated or packed. However, in a system-centric approach, malware actual behavior could be analyzed by monitoring system's behavior, and therefore, obfuscation and packing have no major effect on malware detection.

Since, environment-aware malware are capable of detecting analysis environment, in a system-centric approach such malware are unable to hide their operational behaviors which means that conventional evasion techniques are no more effective.

C. Deployment challenges

While focusing on a system-centric approach to accurately model system's baseline profile, there are few challenges that

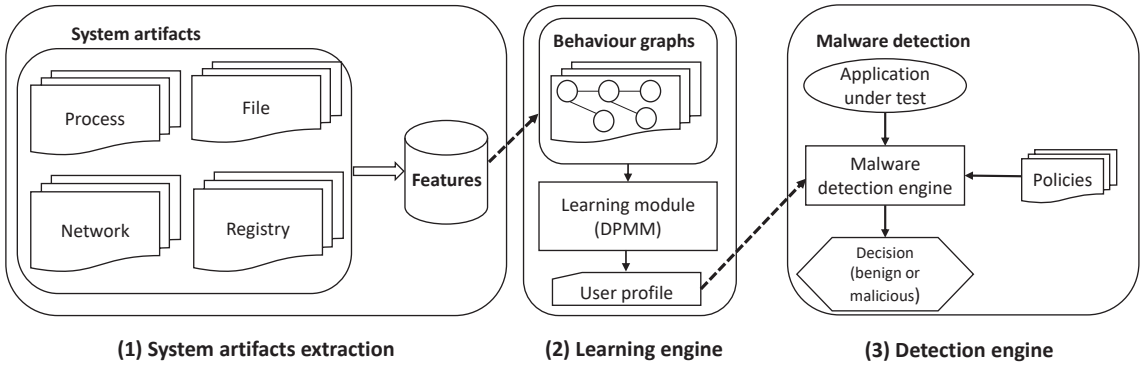

Fig. 1. High-level design of MEDUSA based on deviation from baseline profiles.

need to be handled in order to effectively deploy such systems. One of the challenges is to accurately model system behavior such that it is flexible and does not incur false alarm, because a system's behavior is not fixed and may change over time, e.g., a user may install or delete certain applications, or the network activity may increase under certain circumstances, etc. The proposed malware detection model should be flexible enough to accommodate such genuine changes and not incur false positives under such variations.

Another major challenge is that the proposed system-centric approach should be able to detect zero day attacks. As system-centric approach fingerprints the system's baseline behavior, it is obvious that the learned model will incorporate only the system's current behavior as normal, and anything else is considered to be malicious. It is therefore a challenge for a system-centric approach to distinguish between benign and malicious programs efficiently. Also, it is critically important to detect unseen zero day attacks without producing excessive false alarms.

Moreover, the proposed model should perform malware detection in real-time, i.e., once a malware-specific behavior is detected, the process responsible such behavior should be detected in a timely manner to prevent it from causing a potential damage to the system.

V. MEDUSA: MALWARE DETECTION USING STATISTICAL ANALYSIS OF A SYSTEM'S BEHAVIORS

Here we discuss our proposal to model the system's baseline behavior using system-artifacts for malware detection. Our proposed framework consists of the following components: system-artifacts extraction, behavior graphs generation, learning engine, and detection engine. The framework of MEDUSA is given in Figure 1.

A. System artifacts extraction

The system artifacts extraction module collects the following system artifacts: process, file, network, and registry, from the underlying system resources manager, events log, and process monitor. The extracted artifacts are given in Table I.

To gather a representative dataset of the system artifacts from the real user's system, a simple tool that collect artifacts information can be implemented. For instance, Najmeh et al. in [2] developed a C++ based probe tool which consists of a single Windows 32-bit file that does not need installation and does not have any dependencies besides already available system APIs. Most artifacts are easy to collect by accessing the appropriate log files and directories by using the available Windows APIs. However, the challenging aspect probably is to maximize compatibility with different Windows versions, from Windows XP to Windows 10 [2]. For instance, besides the differences in the paths of various system files and directories across major versions, different system APIs could also be used for Windows XP compared to more recent Windows versions.

B. Learning Engine

The learning engine component of MEDUSA consists of the following components: behavior graphs generation, Dirichlet process mixture model (DPMM) based learning algorithm, and user profile.

1) Behaviour graphs generation: The behaviour graph generator component generates graphs from the system artifacts for each process. This component collects all the raw system features and constructs a behavior graph which reflects the system's usage pattern. For instance, it uses the process's transitioning between different system call relating to file, networking, registry, or process execution, etc. Key system calls such as open(), write(), or socket(), are invoked by a number of processes for accessing important system resources. Our aim is to examine the intermediate states of these resource-requesting processes. These observations truly reflect the behaviour of an on-going process.

2) Learning using DPMM: In general, machine learning algorithms used for classification are implemented in the learner engine. The behaviour graphs are sent to the learner module of the learning engine to build a classification model. DPMM-based classification models [21] are the state-of-the-art statistical machine learning models for density estimation that rely on Dirichlet processes to model the dynamic behavior of a system. The only variables known in DPMM are the observations (system artifacts). Our aim is to estimate parameters from the observations.

The proposed DPMM is a Bayesian nonparametric approach for clustering where the term nonparametric means that the number of behavior patterns are unknown and may grow over time. The proposed system can flexibly be adapted to various system usage situations. Also, no prior knowledge is required about the number of clusters as DPMM model infers it based on the data provided. We model the feature space of a cluster as a multivariate Gaussian distribution with unknown parameters. For multiple clusters where the number of clusters is not known in advance, the DPMM employs an infinite Gaussian mixture model.

The clusters obtained from DPMM represents the behavior of processes. A cluster constitutes processes exhibiting similar behaviors in terms of system features. Note that DPMM is able to create new clusters as more data is observed. This feature of DPMM is useful in detecting zero day attacks because a suspicious program may show totally new behavior and is assigned a new cluster.

3) User profile generation: After learning the user's operational patterns, the last component is the profile generator that encodes the learning results in the form of learned distribution which will help to generate a user profile.

C. Detection Engine

To test whether a running process in the system is benign or compromised by malware, the real-time detection engine authenticates whether the program behaviors follow the system's normal profile and the detection policy. A set of policies is input to the malware detection engine to decide if a particular running process is malware or benign. Applications are examined based on the existing system's profiles and their access to important system resources will be denied if they deviate from the system's normal profile to a certain degree.

VI. Conclusion and Future Work

Existing malware detection techniques mainly rely on the analysis of known malware samples to generate a database of malware signatures. However, the effectiveness of those techniques is limited as the number of new variants of malware increases. To overcome this limitation of existing techniques, we propose a new malware detection method named `MEDUSA` approach, which is a paradigm shift in malware detection that focuses on a target syhstem's behaviors instead of the malware properties. We expect that `MEDUSA` has the potential to more effectively detect newly created poloymorphic and metamorphic malware compared with signature based anti-malware tools. To show the feasibility of the proposed system, we designed a statistical model based on the Bayesian nonparametric clustering approach which takes into account system artifacts and discussed several deployment challenges.

As a part of future work, we plan to implement MEDUSA for Microsoft Windows operating systems with the system artifacts in Table I. To build an effective anomaly detection system, we consider several statistical models such as DPMM.

References

[1] Y. Ye, T. Li, D. Adjeroh, S. S. Iyengar, "A survey on malware detection using data mining techniques," ACM Computing Surveys, vol. 50, no. 3, pp. 41, 2017.

[2] N. Miramirkhani, M. P. Appini, N. Nikiforakis, M. Polychronakis, "Spotless sandboxes: Evading malware analysis systems using wear-and-tear artifacts," IEEE Symposium on Security and Privacy, pp. 1009–1024, 2017.

[3] C. Willems, R. Hund, A. Fobian, D. Felsch, T. Holz, and A. Vasudevan, "Down to the bare metal: Using processor features for binary analysis," in Proceedings of the 28th Annual Computer Security Applications Conference, pp. 189–198, 2012.

[4] D. Kirat, G. Vigna, and C. Kruegel, "Barecloud: Bare-metal analysis-based evasive malware detection," in Proceedings of the 23rd USENIX Security Symposium, pp. 287–301, 2014.

[5] T. LANE and C. E. Brodley, "Approaches to online learning and concept drift for user identification in computer security," in KDD, 1998.

[6] R. Luh, S. Schrittwieser, S. Marschalek, and H. Janicke, "Design of an Anomaly-based Threat Detection & Explication System," In Proceedings of the 3rd International Conference on Information Systems Security and Privacy, pages 397-402, 2017.

[7] A. Tang, S. Sethumadhavan, and S. Stolfo, "Unsupervised Anomaly-based Malware Detection using Hardware Features," In international Workshop on Recent Advances in Intrusion Detection, 2014.

[8] B. Alsulami, H. Dong, and S. Mancoridis, "Lightweight Behavioral Malware Detection for Windows Platforms," In 12th International Conference on Malicious and Unwanted Software, pp.75-81, 2017.

[9] F. Apap, A. Honig, S. Hershkop, E. Eskin, and S. Stolfo, "Detecting Malicious Software by Monitoring Anomalous Windows Registry Accesses," RAID, LNCS, pp. 3653, 2002.

[10] J. M. H. Jiménez, J. A. Nichols, K. Goseva-Popstojanova, S. Prowell, R. A. Bridges, "Malware detection on general-purpose computers using power consumption monitoring: A proof of concept and case study," arXiv preprint arXiv:1705.01977, 2017.

[11] Liang Xie , Xinwen Zhang, Jean-pierre Seifert, and Sencun Zhu, "pBMDS: a Behavior-based Malware Detection System for Cellphone Devices," in Proceedings of the third ACM conference on Wireless network security, pp. 37–48, 2010.

[12] The Exponential Growth of Malware, https://www.pandasecurity.com/mediacenter/malware/2017-figures/, Jan 2018.

[13] C. Willems, R. Hund, A. Fobian, D. Felsch, T. Holz, and A. Vasudevan, "Down to the bare metal: Using processor features for binary analysis," in Proceedings of the 28th Annual Computer Security Applications Conference, pp. 189198, 2012.

[14] A. Kapravelos, Y. Shoshitaishvili, M. Cova, C. Kruegel, and G. Vigna, "Revolver: An automated approach to the detection of evasive web-based malware," in Proceedings of the 22nd USENIX Security Symposium, pp. 637652, 2013.

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

[16] Y. Ji, S. Lee, E. Downing, W. Wang, M. Fazzini, T. Kim, A. Orso, and W. Lee, "Rain: Refinable Attack Investigation with On-demand Inter-Process Information Flow Tracking," in Proceedings of the ACM conference on computer and communications security, 2017.

[17] I. Burguera, U. Zurutuza, S. Nadjm-Tehrani, "Crowdroid: behavior-based malware detection system for android," in Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices, pp. 15–26, 2011.

[18] Y. Feng, O. Bastani, R. Martins, I. Dillig, S. Anand, "Automated synthesis of semantic malware signatures using maximum satisfiability," arXiv preprint arXiv:1608.06254, 2017.

[19] I. Firdausi, C. Lim, A. Erwin, and A. S. Nugroho, "Analysis of machine learning techniques used in behavior-based malware detection," in Proceedings of Advances in Computing, Control and Telecommunication Technologies (ACT), 2010.

[20] A. Kharraz, S. Arshad, C. Mulliner, W. Robertson, and E. Kirda, "UNVEIL: A Large-Scale, Automated Approach to Detecting Ransomware," in USENIX Security Symposium, pp. 757–772, 2016.

[21] D. Blei and M. Jordan, "Variational inference for Dirichlet process mixtures," in *Bayesian Analysis*, vol. 1, no. 1, pp. 121-144, Aug. 2006.