Evasion-Resistant Malware Signature Based on Profiling Kernel Data Structure Objects

Ahmed F.Shosha*, Chen-Ching Liu,
Pavel Gladyshev*
* School of Computer Science and Informatics,
School of Electrical, Electronics and Communication Engineering.
University College Dublin.
Ahmed.Shosha@ucdconnect.ie
{Liu, Pavel.Gladyshev}@ucd.ie

Marcus Matten
Avira Research Department,
Avira Operations GmbH& Co. KG.
Marcus.Matten@avira.com

Abstract— **Malware authors attempt in an endless effort to find new methods to evade the malware detection engines. A popular method is the use of obfuscation technologies that change the syntax of malicious code while preserving the execution semantics. This leads to the evasion of signatures that are built based on the code syntax. In this paper, we propose a novel approach to develop an evasion-resistant malware signature. This signature is based on the malware's execution profiles extracted from kernel data structure objects and neither uses malicious code syntax specific information code execution flow information. Thus, proposed signature is more resistant to obfuscation methods and resilient in detecting malicious code variants. To evaluate the effectiveness of the proposed approach, a prototype signature generation tool called SigGENE is developed. The effectiveness of signatures generated by SigGENE evaluated using an experimental root kit-simulation tool that employs techniques commonly found in rootkits. This simulation-tool is obfuscated using several different methods. In further experiments, real-world malware samples that have different variants with the same behavior used to verify the real-world applicability of the approach. The experiments show that the proposed approach is effective, not only in generating a signature that detects the malware and its variants and defeats different obfuscation methods, but also, in producing an execution profiles that can be used to characterize different malicious attacks.**

Keywords*: *Malware Behavior Profiling, Malware Signature, Signature-Based Detection, Kernel Data Structure.

I. INTRODUCTION

Traditional signature-based detection is one of the most popular approaches to detect known malware in the anti-virus (AV) industry. It relies on extracting sequences of bytes from malicious code binaries that form a signature used to detect it. Unfortunately, advances in malware development have led to a variety of methods to evade malware detection signatures that rely on byte sequence pattern matching.

A prevalent feature that is commonly used in modern malware to bypass signature-based engines is employing code obfuscation and packing technology [1, 2]. The term obfuscation describes the process of intentional tampering and manipulation of the malicious code syntax while preserving the malicious behavior semantics. Practically, packing, code re-ordering and junk code insertions are the most commonly used methods to subvert and evade signature-based detection engines [3, 4]. Further obfuscation methods include API obfuscation, in which unnecessary API calls are inserted in malicious binaries to impede malicious code analysis process and encounter code emulation [2]. Unfortunately, employing these methods in malware code, not only, hinder malware analysis and malware forensic investigation, but also, various malicious code variant programs can easily be generated. These malicious programs are capable of executing the original malicious payload, while being transparent to the original detection signatures.

In essence, the intent of these obfuscation methods is to subvert features input used in signature development process. As a result, created signatures will be ineffective in detecting obfuscated malicious code. Particularly, signatures developed based on features prone to manipulation and obfuscation cause signature detection failures, whereas signatures developed based on features sensitive to tampering are resistant to obfuscation methods and evasions techniques.

In this paper, a novel method is proposed to develop an evasion-resistant malware signature based on features that are sensitive to tampering and robust in detecting malware behavior. In the proposed approach, the characteristics of operating system kernel data structure objects [5] are used to develop malware signature instead of traditional signature that relies on byte sequence matching.

The operating system kernel manages several data structure objects that describe and manage the operations of the programs being executed. The syntax and semantic of such data structure objects are defined by the operating system code. Tampering or modifying these kernel objects properties while programs are being executed can cause the operating system to crash or produces unpredictable behaviors. Furthermore, kernel objects are considered to be an analogous representation of code executed in the operating system kernel. Therefore, characteristics of kernel objects' features are a potential source for deriving evasion-resistant malware signatures.

The key idea of the proposed approach is to profile the invariant values of the kernel objects' features that represent malicious code execution during malware dynamic analysis process in a controlled profiling environment. Profiled features, then, will be used to derive a robust malware detection signature.

The process of features profiling is based on monitoring the malware information flow at different execution states, i.e. system call invocations. In each monitored execution state, there exists a unique pattern of features' values in the kernel objects that characterize malware behaviors and values of these features uniquely describe the semantics of the malicious code execution state. As such, by aggregating all the values of kernel object's features that are profiled during the malicious code execution process, we can detect invariants that precisely represent the malware execution. These invariants are, then, used to develop a unique malware signature that is robust, sensitive to manipulation, and can detect malware variants and obfuscated malware samples.

In this work, malware signature is developed based on profiling `EPROCESS,` a dynamic kernel object that is used to represent a running process in Windows operating systems. However, the proposed approach can easily be extended to profile other kernel objects types that represent various aspects of the program behavior. We monitor the dynamic changes to the `EPROCESS` object related to a malware in memory while malware's code is being executed. Thereafter, an invariant identification technique called "*data structure invariant detection"* [7] is utilized to aggregate the profiled `EPROCESS` object throughout different execution states and determine invariants values from profiled object's features. Determined invariants describe different properties of monitored object that hold over the life time of malware execution. An invariant in profiled `EPROCESS` object can, for example, be a specific value of a security `Token` that represents control access to a process object. Further, invariant can be a specific value of `EPROCESS Flags` that represent process execution flags.

To evaluate the proposed approach, we implemented a prototype malware signature generation tool called SigGENE. The tool profiles values of `EPROCESS` kernel object features in a dynamic analysis environment [6]. This involves hooking the operating system API call table [8, 9] and building a custom kernel device driver in order to capture dynamic changes to `EPROCESS` features in the profiling phase.

In the experimentation phase, we developed detection signatures for several real-world malware variants that belong to five different malware samples families and obfuscated using different techniques. In addition, we evaluated the proposed approach on our developed kernel-mode rootkit-simulation program that features user-space process hiding. Developed rootkit-simulator has been obfuscated using three different techniques to verify the effectiveness of the generated signature in defeating different obfuscation methods. A number of the test malware variants effectively evade two different AV scanners that could not detect all malware variants. However, the proposed approach successfully detected all obfuscated variants in both real and simulated malicious samples. We further analyzed profiled data of kernel objects for each malware sample at each execution state, we argue that each profiled kernel object maintains a unique pattern of data traces that describe and determine the state currently being executed. Thus, by utilizing this observation, forensic identification of previously executed system calls is likely to be possible.

At last, we state the contributions of this paper as follow:

- An approach is proposed to develop a robust malware detection signature based on detected kernel data structure invariants that is evasion-resistant to obfuscation techniques.
- This approach automates the process of malware signature generation based on profiling kernel data structure objects monitored during the malware execution process.
- A prototype malware signature generation tool is implemented to automate development of malware signatures through dynamic analysis and profiling of dynamic kernel data structure objects.

The remainder of the paper is organized as follows: In section two, the system architecture of SigGENE is described. Section three presents the evaluation of the proposed approach. In section four, a brief discussion of the proposed approach is provided. In section five, we discuss the related work. Finally, section six concludes the proposed approach.

II. System Overview

In this section, the proposed malicious kernel objects profiling approach is presented and assisted with the design and implementation of SigGENE. SigGENE is a prototype malware signature generation engine that profiles malicious kernel objects for executed malware sample and determines invariant kernel objects' features values during malicious code execution. To perform malware behavior profiling, SigGENE monitors the kernel objects that belong to the malicious code in memory throughout utilizing a *Virtual Machine Monitoring and Introspection* (VMI) techniques [10]. Fundamentally, VMI employment in the proposed approach allows in monitoring the dynamic changes to the kernel objects' features and forms the basis for profiling malicious code behavior and monitoring malware execution. An overview of SigGENE system design is depicted in Figure 1.

The design model of SigGENE is comprised of two complementary modules. The first module is designated to: 1) Identify features in kernel objects that effectively contribute to robust signatures development. 2) Monitor dynamic changes to kernel object features in the context of malicious code execution and develop kernel objects' profiles. The module functionalities are implemented in Virtual Machine Monitor

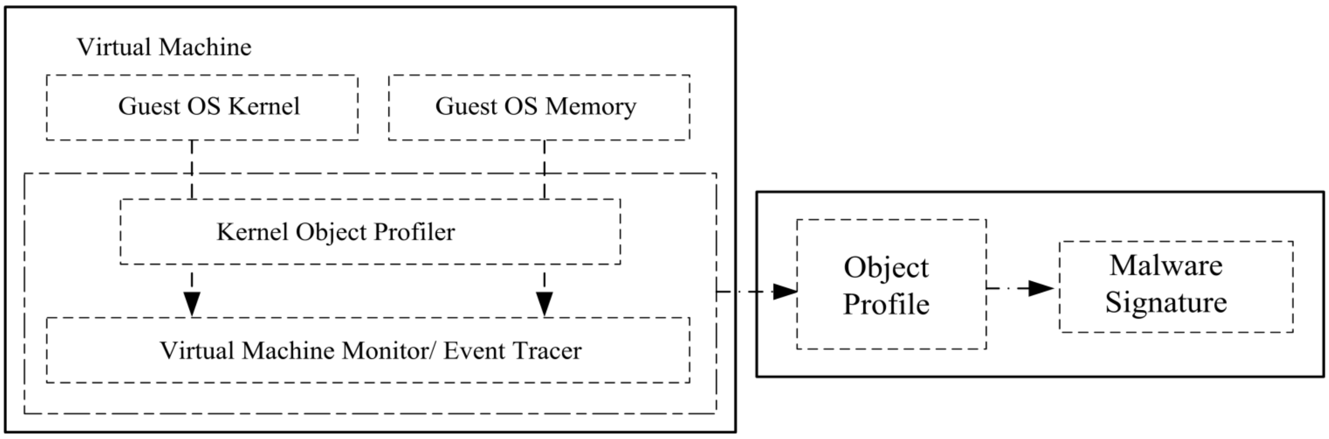

Figure 1: System Overview

(VMM) and Kernel Object Profiler components in Fig 1. The inputs to previously mentioned components are definitions to kernel objects data structure as defined in the guest operating system code and locations of the kernel objects instances in the guest OS memory. The second module utilizes developed kernel object profiles during dynamic monitoring and introspection of malicious code execution and determines invariants values over kernel object's features to generate the evasion-resistant malware signature.

A. Robust Features Identification

Robust features are properties in monitored malicious kernel objects that effectively contribute to the execution of malicious code and assist in producing evasion-resistant detection signature. Identifying robust features is the core component in the proposed signature development approach. Since numerous features may be considered as viable candidates to the signature development process, only a limited number of features are effectively contribute to robust signature development. For example, `EPROCESS` kernel object and its substructure objects hold up to 2000 features based on the OS version [11]. Some of these features are unused or used in specific circumstances and others are prone to manipulation by malicious code. That is, unused features or features prone to manipulation threat the signature integrity and assist in producing signatures that can be evaded if an appropriate evasion technique employed. Thus, locating features in kernel objects that allow in robust signature development is an inevitable portion of the proposed approach.

In our implementation, a derivate of dynamic monitoring of kernel data structures using Virtual Machine Introspection (VMI) technique proposed in [5] has been used to identify robust features in the kernel objects. The key idea of the robust feature identification process is based on how important these features are to malicious code execution. Features that are accessed or modified while malware is being executed are more likely to be relevant to the malicious code execution, relative to features that are never accessed. Similarly, features that, if modified, will cause malware to misbehave, are more likely to have a strong relevancy to malicious code execution, than features, if changed, do not alter malware behavior.

To identify robust features in `EPROCESS` kernel object that used in signature development process, we developed a dynamic monitoring component for `EPROCESS` kernel object features using (VMI) to identify whether `EPROCESS`'s features have been accessed or modified during malware execution process. We have customized a version of QEMU emulator [12], a fast processor emulator using dynamic code translation, to implement a kernel object memory monitoring. We instruct the customized QEMU to create an *Event Traces* for memory reads and writes routines that are part of the Virtual Machine Monitor (VMM) component shown in Figure 1. Event tracer allows tracing dynamic changes on kernel object features through monitoring memory *Reads* and *Writes* operations over memory regions allocated to the kernel object's features. Thus, if a memory region represents a malicious code kernel object is accessed or modified, an event is triggered to describe the offset of the memory region and the operation used to access the offsets. These logs are examined later and mapped to the definition of the `EPROCESS` kernel data structure to determine what features were accessed or modified and how often. Based on the results of tracing dynamic changes to memory regions allocated to kernel objects' features, VMI event tracer determines robust features per malware sample that will be considered in the profiling process and will contribute in producing an evasion-resistant malware detection signature.

B. Malicious Kernel Objects Profiling

In the profiling process, we profile kernel object features during malicious code execution. These profiles primarily describe the characteristics of monitored malicious kernel objects' features that are analogous to malware execution semantics in different execution states.

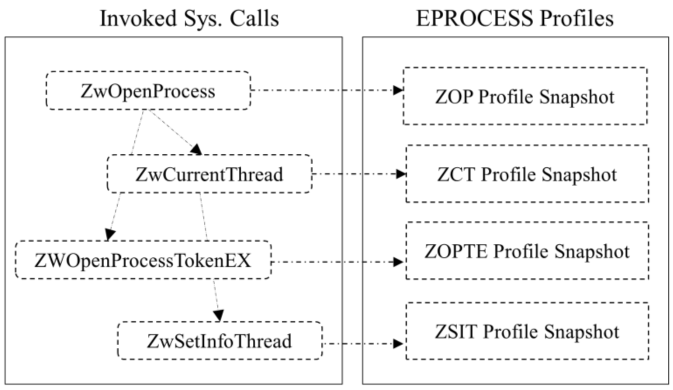

Figure 2: Kernel Object Profiling During Malicious Code Execution

This process encompasses the execution of malicious code binaries in a controlled environment [13-16] to identify the malicious code information flow and executed system calls (*syscalls*). Since system calls are the main interface for programs to interact with the operating system kernel, we use system calls invocation procedure as a trigger for the profiling procedure. Unlike other systems that model malware behavior by specifying system calls execution sequences, we only use system calls invocation to trigger the process of profiling identified robust features in kernel objects. Unfortunately, systems relying on malware behavior profiling based on, only, system calls sequences are prone to different attacks such as insertion of irrelevant calls or call sequence re-ordering [17]. Thus, to avoid such shortcoming, the proposed profiling process solely consider system calls invocations as an initiate to an execution state with no regards to the calls execution sequence.

In this research, we defined a formalism to describe the process of kernel objects profiling in the context of malicious program execution. Malicious code control flow is represented in a finite state automata [17] model $\mathcal{M}$ that describes the malicious program behavior, where each state is labeled with a system call used by malware code to interact with the operating system kernel and edges are transitions that represent the dynamic control flow and determine the dependencies between states.

Definition. A malicious code behavior $\mathcal{M} = (S, \longrightarrow, L)$ is a finite sequence of states $s_1, s_2, s_3, \ldots, s_n$ in S such that s' and $s \in S$ with $s \longrightarrow s'$, and L is labeling procedure $L: S \longrightarrow \psi(\varphi)$, in which $\psi(\varphi)$ is an atomic proposition that is true at the execution of the system call $\varphi \in$ syscalls.

In essence, invocation of a system call φ causes changes to the control properties of a kernel object O, in which the operating system kernel changes the object O in way to permit it to execute φ. Hence, we define invocation of φ as a function that stimulates changing values of various properties (robust features) $\{\rho_1, \rho_2, .. \rho_n\}$ in malicious kernel object o_{mal} from $\{j_1, j_2, j_3, .. j_n\}$ into $\{k_1, k_2, k_3, .. k_m\}$, such that, the features values hold after invocation of φ uniquely define the characteristics of o_{mal} robust feature at state *s(φ)*. The profiling procedure used to capture changes to o_{mal} at invocation of φ is defined as $\mathcal{F}\varphi_x \in \mathbb{F}$, where the function $\mathcal{F}\varphi_x$ over φ_x is defined as $\mathcal{F}\varphi_x : j_1, .., j_m \longrightarrow k_1, .., k_m$. Finally, the result of o_{mal} robust feature profiling process are stored in the object snapshots profile repository $\mathbb{R}$, such that, $\mathbb{R} = [\![o_{mal}\langle\varphi_1\rangle, o_{mal}\langle\varphi_2\rangle, .., o_{mal}\langle\varphi_x\rangle]\!]$.

Based on the previous discussion, we define a malicious code profiling process $\mathbb{P}$, more formally, as: $\mathbb{P} := \langle S, \mathbb{F}, \mathbb{R} \rangle$, such that:

- S is a set of malicious code execution states where each state labeled with a system call φ.
- $\mathbb{F}$ is a profiling procedure that monitor the dynamic changes to the features of a malicious kernel object o_{mal} and capture the characteristics of object o_{mal} after invocation of φ.
- $\mathbb{R}$ is a repository of profiles related to a malicious kernel object o_{mal} that hold at different system calls invocations.

Intuitively, the profiling process encodes the characteristics of malicious kernel object features at system call invocations. This means extracted profiles will represent malicious code execution from the kernel object data structure perspective. These profiles do not include specific information about malicious code syntax or execution sequence of malicious system calls. That is, the profiles are less vulnerable to obfuscation methods and evasion techniques that rely on manipulating code syntax.

Practically, the process of malicious kernel object profiling is implemented at the Kernel Object Profiler (KOP) module as shown in Figure 1. KOP is a set of kernel device drivers that monitor invocation of the kernel systems calls invoked from malicious processes in memory throughout system call table hooking [18, 19]. This includes, monitoring systems calls that used in different aspects of the malicious programs execution, such as process and thread creation, malicious DLL loads and file, registry, network operations. Once malicious system call is invoked, the locations of the robust features belongs to malicious kernel object in memory are requested from the VMM, followed by an acquisition of the robust features' values from the guest operating system's memory. Note that, an assistant procedure defined as a provisional suppression of malicious code execution after system call invocations until the acquisitions of the robust features' values is defined to allow consistent profiling and protect the features' values of being overwritten before the completion of features' values acquisition.

Figure 2 depicts an excerpt from a profiled malicious kernel

Malicious Kernel Objects' Profiles Characteristics	Sample Family 1	Sample Family 2	Sample Family 3	Sample Family 4	Sample Family 5	Lab Rootkit-Simulator Family
#Average Robust features	253	348	211	283	147	118
#Average Obtained profile snapshots	277	513	293	375	256	188
#Average read operations over robust features	9365	11975	7132	9844	8401	6121
#Average write operations over robust features	2499	3730	1981	2373	1192	958
# Signature detection false positives/negatives	0	0	0	0	0	0

Table 1: Malicious Kernel Objects Profiling Results

object while malicious code is being executed. In this graph, one can see various invoked system calls by the malicious code to obtain handle of a malicious process and get an access to the process token to probe a processes-space in memory. The profiling procedure in KOP is stimulated, once the VMM notify KOP with a notification routine of invoked system call and its arguments, to profile a snapshot of kernel objects' robust features that represent malicious code in memory. Finally, the profiling procedure adds profiled kernel object features' snapshot to the kernel object repository space. This repository space is a set of profiles representing characteristics of malicious kernel objects in the context of malware execution at different system calls invocations.

Note that, the vector length of a profile snapshot is determined through the robust feature identification process, as previously explained. For example, the length of a single snapshot varies from 100 robust features to 500 features. Similarly, the length of a malicious object repository space is determined based on the number of invoked system calls by malicious code.

C. Signature Generation

During malicious kernel object profiling process, numerous profile snapshots are obtained that uniquely characterize malicious object at each invoked system call. However, each profile snapshot represents a timely-specific characteristic of the malicious code behavior. In other words, it preserves the malicious kernel data structure characteristics at specific execution state. Thus, to aggregate obtained profiles and represent all execution states, an aggregation process is proposed to assemble the profile snapshots, and detect invariants features' values obtained in the profiles acquisition process. In the profiles aggregation process, we used the concept of *Dynamic Invariant Detection* [20]. This concept is proposed to detect likely invariants in a user space program execution by instrumenting the source programs to trace variables of interest through program execution over a set of test cases. In the proposed approach, we use a simplified version of a dynamic invariant detector [20], where inputs to the detector is comprised of obtained profile snapshots from malicious code profiling process. The profile snapshots are examined by several test cases using different constraints and examination templates to detect invariant values of monitored robust features. An example for a constraint used to detect invariants in profiled malicious kernel object is a constant value for a specific robust feature, or a linear relationship between two features, given that they are present in all snapshot profiles. Another example is a value for a specific feature being in a specific range during malicious code information flow. Consequently, applying invariant detection process on obtained profile snapshots produces a profile that represents a unique invariants characteristic of malicious kernel objects features that holds over the life time of malware execution.

Finally, produced profile is used as a signature to detect malware programs throughout scanning the dynamic kernel objects belong to a malicious executable in memory and matching characteristics of scanned dynamic kernel object with produced profiles.

III. Evaluation

Several test cases performed to evaluate the efficiency of SigGENE and to prove that produced profiles are resistant to evasion and obfuscation techniques.

Two different experiments were conducted to evaluate the proposed approach. In the first experiment, a program simulating a kernel-mode rootkit is developed to launches a dynamic kernel object manipulation attack (DKOM) [21] and hides a different malicious processes running in user-mode. Additionally, developed rootkit-simulator has been obfuscated using 3 different methods to determine if generated profile signature is capable of detecting obfuscated variants. Note that, used code obfuscation methods were obfuscation by encryption, code re-ordering and instruction substitution [2]. The result of code obfuscation process was generation of 13 different variants of developed rootkit-simulator. The second experiment was conducted on real-world malware samples [22] from different samples families. Each sample family has up to 17 variants and executes the same functionalities, with a total number of 63 test samples. All samples have been scanned with two different AV detectors; the detectors, however, failed to detect 19 samples variants while developed profile signatures successfully detected all samples variants and obfuscated versions of the lab rootkit-simulator.

The signature development process has been verified in different versions of Windows operating system to evaluate the kernel object profiling accuracy in different object's definitions.

The profiling process was developed in Windows XP SP3 and Windows 7 SP1.We observed that identified robust features in Windows 7 were more comprehensive in describing the characteristics of malicious kernel object. The vector length of features included in profiled snapshots was extended since `EPROCESS` kernel object definition in Windows 7 is slightly different compared to previous Windows versions and contains more flags that controls programs execution.

The results of the experiments, given above in Table 1, describe and present the outputs of the profiling process for the test malware samples and developed rootkit-simulator used in SigGENE evaluation. The robust features results section presents features that are determined to be used in kernel object profiling process based on their contribution in malicious code execution process. Obtained profile snapshots section shows produced number of profile snapshots upon invocation of various system calls per malicious code execution run. As such, malicious code execution run represents the process of monitoring malicious code execution starting from the creation of a malicious process until malicious process termination. Finally, *Read* and *Write* operations over robust features section provides an indication on how determined features are relevant to the execution of malicious process and describe the numbers of dynamic changes of robust features' values in malicious kernel object throughout malicious code execution. Note that, presented numbers is the average of sample evaluation and its variants.

In the verification process, generated profiles used as a signature were also verified using several benign kernel objects representing user-mode programs and malicious kernel objects representing malware samples that were not a part of test cases, as well. All test cases did not produce false positives or negatives and generated profiles produced accurate results in detecting intended samples.

Throughout the profiling process, we analyzed obtained profile snapshots for each test sample. A core observation was that each profile snapshot has a unique set of feature values that uniquely characterize the execution state itself. For example, while investigating the test lab rootkit-simulator snapshot profiles, no profile snapshot was identical to other snapshots and values of at least 20 robust features are unique compared to other profile snapshots and to other objects profile spaces. Thus, we argue that we can, not only, develop a signature to detect malware and its variant based on kernel object profiles, but also, identify system calls that have been invoked by malware, if profile snapshots information employed in malware forensic investigation analysis.

Furthermore, in the profiling process we observed similarities between profiles extracted while invocation of system calls belong to same group family, i.e. networking or memory related system calls. For example, system calls used to probe or attach to user address space of other processes in memory such as `KeStackAttachProcess` [11] changes the values of `Token` feature and `debug` flag to a unique value that enabled us to determine that a process probe related system calls have been invoked.

By utilizing these observations, we can characterize malicious attacks based on similarities of profiles obtained during the attack execution. For example, in an experiment designated to analyze three different rootkit samples launching a DKOM attack, we observed partial similarities between profiles extracted through execution of system calls related to the attack. Therefore, we argue that the proposed approach can be extended to detect unknown samples based on profiling the characteristics of malicious attacks.

IV. Discussion

SigGENE is a signature-based malware detection approach, primarily designed to detect obfuscated malware and malware variants through profiling malicious dynamic kernel objects. Although signatures developed based on kernel object profiles demonstrate promising results in the evaluation phase, SigGENE prototype is confronted with a number of limitations, which are being addressed in our on-going work.

- *Profiling Performance*: SigGENE traces memory access using a VM monitoring module as a basis for the robust feature identification process. This process is both time-consuming and computationally expensive in profiling stage. Thus, our work in-progress includes a lightweight process memory monitor based on tracing memory page access on page fault errors from the operating system memory manager, instead of tracing read and write operations directly from the VM using Event Tracer.
- *Behavior Monitoring*: We monitor malware execution by hooking the operating system calls. However, some anti-analysis methods employed by malware do detect monitoring-based hooking. Thus, SigGENE may produce inaccurate profiles if the sample employs such methods. To overcome this shortcoming, we are moving the kernel object profiler and monitor component from the operating system kernel internals to outside by implementing the monitoring functionalities in the VM monitoring layer. This type of monitoring and profiling will as a result be transparent to malware samples under investigation.
- *Kernel Object*s: The current scope of the proposed approach is limited to profile `EPROCESS` kernel data structures; one suggested improvement is to include additional kernel objects such as `FILE_OBJECTS` and `VAD.` We believe that inclusion of different kernel objects will yield to deeper and unique profiles generation, which will lead to improve malware behavior characterization.
- *Profiling Samples and Attack*: Currently proposed approach profiles kernel malware samples and its variants. The evaluation results, however, demonstrated the possibility of profiling malicious attacks through the observation of kernel objects while invoking malicious system calls to perform a specific attack. Thus, our future work will includes an extension to the proposed approach to include malicious attacks profiling.

V. Related Work

While signature-based detection has been studied for decades, malware detection based on behavioral profiling and defeating code obfuscation has become increasingly important in recent years. Various approaches have been proposed to characterize malware behaviors based on code execution flow [23-25]. However, such approaches were confronted with different obfuscation methods that elude malware analysis and traditional signature-based detectors.

Panorama proposed a malicious code information tracking approach using taint-based information flow method to understand how data can be manipulated by the malicious code. However, this approach suffered from control flow evasion attacks that break a taint-based information flow method [26], hence, Panorama's detection engine will not be able to detect variants employing this evasion method. Similarly, K-Tracer proposed a backward-forward slicing techniques on simulated kernel event traces to extract malware goals and functionalities [27]. However the proposed method requires prior determination of the data on which to perform the slicing operation. Another approach that profiles malware behavior was PoKer [28] which proposed a context tracking method to trace rootkit execution and extract a behavioral profile based on these execution traces. Although the approach can effectively profile different rootkit behaviors, extracted profile is, unfortunately, based on execution syntax and vulnerable to obfuscation methods.

An improved method to profile rootkit behavior was proposed in [29]. DataGene proposed a memory data access pattern extraction approach to characterize the malware behavior. The main motivation behind DataGene was to avoid dependence on control flow execution to develop behavioral profiles. Thus, DataGene proposed a monitoring mechanism to access patterns of data resident in memory that belongs to the malicious code and extract unique access patterns that characterize malware execution. The limitation of this approach is that data access pattern is subject to the execution constraints and its environmental parameters, and hence it is not robust enough to be used as a malware signature. A similar profiling approach based on data access patterns was proposed in [30]. KILMAX correlates memory write patterns to normal distribution of user-issued key stokes to profile and detect key-logger malware.

Perhaps the most research work relevant to the proposed approach was presented in [31]. Gibraltar by Baliga et al. takes advantage of data structure invariant inferences by generating a graph of kernel objects in memory and, then, derives constraints over the object data. Observed deviations from the inferred invariants are considered attacks against the kernel data structure. In essence, the goal and a number of limitations that were discovered in Gibraltar make our approach and our implementation substantially different. Whereas Gibraltar profiles attacks to be able detect it, our approach profiles malware semantics to produce a robust detection signature and defeat obfuscation methods. Additionally, Gibraltar fetches the kernel data structure from the memory without filtering kernel object features based on relevance to the attack semantics. Consequently, a number of unnecessary features will be included in behavior profiling which is an issue regarding the precision of the generated profiles.

VI. Conclusion

Traditional signature based detection techniques can be bypassed using malicious code obfuscation or packing, since features used in signature development are vulnerable to manipulation and tampering by malicious code.

In this research paper, we propose a novel method to develop a malware signature that is resistant to obfuscation techniques. The proposed signature is based on kernel object characteristics while avoiding dependency on specific malicious code information that may utilize to evade developed signatures. In addition, a method is proposed to identify kernel object's features that effectively contribute to the development of a robust malware detection signature. Kernel object profiling and an invariant detection method are, also, proposed to assist the process of evasion-resistant signature development.

To support the proposed approach, a prototype tool is developed to produce malware detection signatures based on obtained profiles. Experiments using real-world obfuscated malware samples show the effectiveness of developed signatures in detecting malware variants and obfuscated malicious code.

Acknowledgment

The authors would like to thank Avira's GmbH research department and detection engine team for their useful ideas, advices and contribution in this research. We would also like to thank our anonymous reviewer for their useful comments and suggestions. The authors would like to acknowledge support from EU FP7 project, "A Framework for Electrical Power Systems Vulnerability Identification, Defence and Restoration (AFTER)," at University College Dublin.

This research work is a part of on-going effort to produce a novel computational methods and detection engine to detect targeted malware intrusions to the software infrastructure of physical security systems.

References

[1] Guo, F., P. Ferrie, and T. Chiueh, "A Study of the Packer Problem and Its Solutions", In 11th International Symposium on Recent Advances in Intrusion Detection, 2008, p. 98-115.

[2] You, I. and K. Yim, "Malware Obfuscation Techniques: A Brief Survey", In International Conference on Broadband, Wireless Computing, Communication and Applications, 2010, IEEE Computer Society. p. 297-300.

[3] Sharif, M., et al. "Impeding Malware Analysis Using Conditional Code Obfuscation". In Network and Distributed System Security Symposium, 2008.

[4] Yan, W., Z. Zhang, and N. Ansari, “Revealing Packed Malware”, In IEEE Security and Privacy, 2008, p. 65-69.

[5] Dolan-Gavitt, B., et al., “Robust Signatures for Kernel Data Structures”, In 16th ACM Conference on Computer and Communications Security, 2009, p. 566-577.

[6] Egele, M., et al., “A Survey on Automated Dynamic Malware-Analysis Techniques and Tools”, ACM Comput. Surv., 2008. Vol 44(2): p. 1-42.

[7] Ernst, M., et al., “Quickly Detecting Relevant Program Invariants”, In 22nd International Conference on Software Engineering, 2000, p. 449-458.

[8] Mutz, D., et al., “Anomalous System Call Detection”, In ACM Trans. Inf. Syst. Secur., 2006.Vol 9(1): p. 61-93.

[9] Forrest, S., S. Hofmeyr, and A. Somayaji, “The Evolution of System-Call Monitoring”, In Annual Computer Security Applications Conference, 2008, p. 418-430.

[10] Nance, K., M. Bishop, and B. Hay, “Virtual Machine Introspection: Observation or Interference?” IEEE Security and Privacy, 2008. Vol 6(5): p. 32-37.

[11] Microsoft, “System-Defined Data Structures”, 2012 [cited May 2012]; Available from: http://msdn.microsoft.com/en-us/library/windows/hardware/ff564540(v=vs.85).aspx.

[12] Bellard, F., “QEMU, A Fast and Portable Dynamic Translator”, In USENIX, 2005, p. 41-41.

[13] C, W., T. Holz, and F. Freiling, “Toward Automated Dynamic Malware Analysis Using CWSandbox”. In IEEE Security and Privacy, 2007. Vol 5(2): p. 32-39.

[14] Dinaburg, A., et al., “Ether: Malware Analysis via Hardware Virtualization Extensions”, In 15th ACM conference on Computer and Communications security, 2008, p. 51-62.

[15] Vasudevan, A. and R. Yerraballi, “Cobra: Fine-grained Malware Analysis using Stealth Localized-executions”, In IEEE Symposium on Security and Privacy, 2006, p. 264-279.

[16] Newsome, J. and D. Song, “Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software”. In Network and Distributed System Security Symposium, 2005.

[17] Kolbitsch, C., et al., “Effective and Efficient Malware Detection at the End Host”, In 18th Conference on USENIX Security Symposium, 2009, p. 351-366.

[18] Yin, H., Z. Liang, and D. Song, “HookFinder: Identifying and Understanding Malware Hooking Behaviors”. In Distributed System Security Symposium, 2008.

[19] Hunt, G. and D. Brubacher, “Detours: Binary Interception of Win32 Functions”, In 3rd Conference on USENIX Windows NT Symposium, Vol 3, p. 14-14.

[20] Ernst, M., et al., “The Daikon System for Dynamic Detection of Likely Invariants”. Sci. Comput. Program., 2007, Vol 69(1-3): p. 35-45.

[21] Hoglund, G., “Rootkits: Subverting the Windows Kernel”, 2005: Addison-Wesley Professional. 352.

[22] Offensive Computing. [cited May 2012]; Available from: http://www.offensivecomputing.net/.

[23] Moser, A., C. Kruegel, and E. Kirda, “Exploring Multiple Execution Paths for Malware Analysis”, In IEEE Symposium on Security and Privacy, 2007, p. 231-245.

[24] Xuan, C., J. Copeland, and R. Beyah, “Toward Revealing Kernel Malware Behavior in Virtual Execution Environments”, In 12th International Symposium on Recent Advances in Intrusion Detection, 2009, p. 304-325.

[25] Preda, M., “Code Obfuscation and Malware Detection by Abstract Interpretation”, In Dipartimento di Informatica, 2010.

[26] Yin, H., et al., “Panorama: Capturing System-wide Information Flow for Malware Detection and Analysis”, In 14th ACM Conference on Computer and Communications Security, 2007, p. 116-127.

[27] Lanzi, A., M. Sharif, and W. Lee, “K-Tracer: A System for Extracting Kernel Malware Behavior”, In 16th Annual Network and Distributed System Security Symposium, 2009.

[28] Riley, R., X. Jiang, and D. Xu, “Multi-aspect Profiling of Kernel Rootkit Behavior”, In 4th ACM European Conference on Computer Systems, 2009, p. 47-60.

[29] Rhee, J., Z. Lin, and D. Xu, “Characterizing Kernel Malware Behavior with Kernel Data Access Patterns”, In 6th ACM Symposium on Information, Computer and Communications Security, 2011, p. 207-216.

[30] Ortolani, S., C. Giuffrida, and B. Crispo. “KLIMAX: Profiling Memory Write Patterns to Detect Keystroke-Harvesting Malware”. In International Symposium on Recent Advances in Intrusion Detection (RAID 2011).

[31] Baliga, A., V. Ganapathy, and L. Iftode, “Detecting Kernel-Level Rootkits Using Data Structure Invariants”. In IEEE Trans. Dependable Secur. Comput., 2011. Vol 8(5): p. 670-684.