IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 17, Issue 6, Ver. VI (Nov – Dec. 2015), PP 01-07
www.iosrjournals.org

Novel Malware Clustering System Based on Kernel Data Structure

Bhandare Trupti Vasantrao[1], Pramod B. Mali[2]
[1]*PG Student, Department of Computer Engineering, Smt. Kashibai Navale College of Engineering, Vadgaon Bk, Pune, India*
[2]*Assistant Professor, Department of Computer Engineering, Smt. Kashibai Navale College of Engineering, Vadgaon Bk, Pune, India*

Abstract *: An operating system kernel is the prime of system software, responsible for the integrity and conventional computer system's operations. Traditional malware detection approaches have based on the code-centric aspects of malicious programs, e.g. injection of unauthorized code or the control flow patterns of malware programs.*

In response to these malware detection strategies, modern malware focus on advanced techniques such as reusing existing code or complicated malware code to circumvent detection. A new perspective is introduced to detect malware which is different from code-centric approaches. The data centric malware defense architecture (DMDA) is introduced which models and detects malware behavior. This architecture is based on properties of the kernel data objects that are targeted during malware attacks. This architecture requires external monitoring. External monitor resides outside the monitored kernel and ensures temper-resistance. This architecture consists of three core system components that enable inspection of the kernel data properties and depending upon these properties from malware cluster. The system clusters malware depending upon the kernel data objects.

Keywords: *Kernel data structure, malware signature, malware clustering, kernel malware classification.*

I. Introduction

An operating system kernel is the core of system software which is responsible for the integrity and operations of a conventional computer system. Developers of malicious software (malware) have been continuously exploring various attack vectors to tamper with the kernel. Traditional malware detection approaches have focused on the code centric aspects of malicious programs, such as the injection of unauthorized code or the control flow patterns of malware programs. However, in response to these malware detection strategies, modern malware is employing advanced techniques such as reusing existing code or complicated malware code to circumvent detection.

Kernel-level malware is one of the most dangerous threats to the security of users on the Internet, so there is an urgent need for its detection. The most popular detection approach is misuse-based detection. However, it cannot catch up with today's advanced malware that increasingly apply polymorphism and obfuscation. In this thesis, we present our integrity-based detection for kernel-level malware, which does not rely on the specific features of malware.

Modern malware use a variety of techniques to cause divergence in the attacked program's behaviour and achieve the attacker's goal. Traditional malicious programs such as computer viruses, worms, and exploits have been using code injection attacks which inject malicious code into a program to perform a nefarious function. Intrusion detection approaches based on such code properties effectively detect or prevent this class of malware attacks [10], [11], [12].

Data-centric approaches require neither the detection of code injection nor malicious code patterns. Therefore they are not directly subvertible using code reuse or obfuscation techniques. However, detecting malware based on data modifications has a unique challenge that makes it distinct from code based approaches. Unlike code, which is typically expected to be invariant, data status can be dynamic. Correspondingly, conventional integrity checking cannot be applied to data properties. In addition, monitoring data objects of an operating system (OS) kernel has additional challenges because an OS may be the lowest software layer in conventional computing environments, meaning that there is no monitoring layer below it.

Traditional malware detection and analysis approaches have been focusing on code-centric aspects of malicious programs, such as detection of the injection of malicious code or matching malicious code sequences. However, modern malware has been employing advanced strategies, such as reusing legitimate code or complicated malware code to circumvent the detection. As a new perspective to complement code-centric approaches, we propose a data-centric OS kernel malware characterization architecture that detects and characterizes malware attacks based on the properties of data objects manipulated during the attacks [1].

To address the challenges of relying on only code in malware defence, we propose new approaches based on the properties of data objects that are targeted in malware attacks. These approaches do not require the detection of the injected code or the specific sequence of malicious code. Therefore, they are not directly subject to attacks targeting the approaches based on code properties.

II. Literature Survey

Data invariants are one important class of integrity properties that concerns the expected values of program variables (other integrity properties include control flow integrity and information flow integrity). Since the behaviour of a program can largely depend on its variables, manipulation of data has been an important malware attack technique.

Malicious code or malware has long been recognized as the major threat to the computing world [7]. According to Mohamad Fadli and Aman Jantan, all malware have their specific objective and target, but the main purpose is to create threats to the data network and computer operation [8]. In general, incoming files are considered as malware if they perform suspicious attack and might harm computer systems. These files can break into vulnerable systems from many ways such as via internet (HTTP), email, removable media, Peer-2-Peer (P2P) and Instant Messaging (IM). Anti-malware products in host machine will perform scanning process to detect and identify the attack. If unknown kinds of attack are found, the samples will be sent to anti-malware vendors to be analyzed. Once analysis process is completed, vendors will update the virus signature database server and the latest update can be downloaded from host machine side. Classification part will classify the attack sample into the correct malware classes and prediction as well as removal part will remove, clean or quarantine the attacks.

Many malware detection mechanisms rely on the properties of malware code such as the injection of unauthorized code [7][14] and the patterns of malicious code sequences [8][9]. While these approaches are effective for classic malware, emerging malicious programs are introducing advanced techniques such as return/jump oriented programming [4][15][16], code obfuscation [17], and code emulation [18] to elude those malware detection mechanisms. In this paper, we have presented a new approach for detecting kernel malware based on the properties of kernel data objects.

Compared to dynamic kernel objects, static objects have memory addresses that are predetermined at the compilation time. The manipulation of static objects is observed as write accesses to their unique addresses. If such memory access patterns are observed specifically during malware execution, they are extracted as malware signatures. For example, system call hijacking is implemented as the manipulation of the system call table that is a static object. The manipulation of this object by other than the legitimate initialization code is rare in benign execution. Thus, this attack pattern is automatically extracted as a signature.

III. Proposed System

We present a novel scheme that addresses these challenges and enables OS kernel malware detection approaches based on kernel data properties. The monitoring system should be designed in a way that cannot directly be altered by potentially malicious code therefore; we use an external monitor to observe the target OS kernel. An external monitor has a challenging task in identifying the data object information of the monitored kernel, which is known as a semantic gap. Such information should be reconstructed externally in the monitor. We propose a classification approach for identification of the malware characterization over kernel data structure and clustering using data object properties to detect kernel malware, which consists of two main components.

The first component is a kernel object mapping system that externally identifies the dynamic kernel objects of the monitored OS kernel at runtime, and our aim is to observe memory accesses to kernel data objects. This component is essential because it enables an external monitor to recognize the data objects that are targeted by the accesses. As well as being an infrastructure to recognize data objects, this system provides effective applications such as the detection of data hiding kernel malware attacks and the analysis of malware behavior targeting dynamic kernel objects.

In addition to the kernel data mapping system, we propose a new approach that detects malware by matching memory access patterns that specifically occur during malware attacks. Dynamic kernel analysis can produce effective malware signatures that can suppress frequent false positives in typical workloads by extracting malware memory reference patterns specific to malware attacks.

IV. System Implementation

This section describes the system designs in the form system architecture, modules. It is also explain the software and hardware requirement to perform the system execution.

4.1 System Architecture

We first present how we can generate the data information and present our approach to characterize malware based on kernel data patterns. In this paper we propose identification of the malware characterization over kernel data structure, which models and detects malware behavior using the properties of kernel data objects.

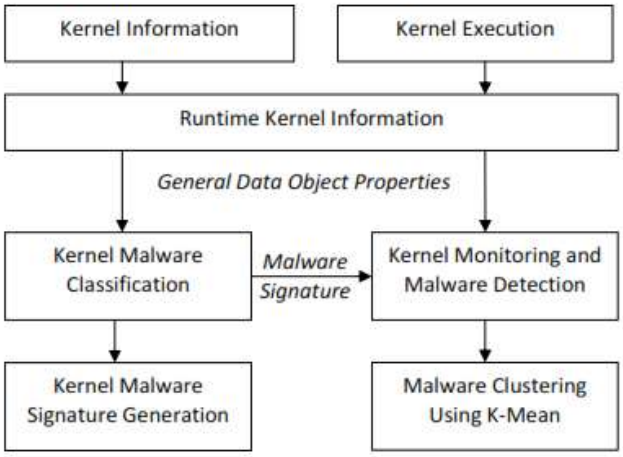

Fig. 4.1: Architecture Design Identification of the Malware Characterization

4.2 System Modules

Based on the design architecture in Fig. 4.1 it has two modules:

1. Malware Signature Generation
2. Malware Detection

4.2.1 Malware Signature Generation

This module works on the collected kernel information data from the kernel behavior on different type of system activities. The collected information is cleaned using a data cleaning mechanism to remove the noise information and run a rule-based classification approach to generate an activities based signature patterns known as data behavior pattern (DBP). A signature generally is a pattern of instruction sequence or system calls sequence performed for executing or completing a job. The generated activity signature patterns are stored for runtime testing.

To generate a malware signature for a malicious kernel run *j* with malware *M*, we apply set operations on *n* malicious kernel runs and *m* kind runs as follows.

- Let's assume $D_{M,j}$, and $D_{B,k}$ → represents a data behavior profile for a kind of kernel execution *k*, and S_M is called a data behavior signature for a malware *M*.

$$S_M = \bigcap_{j \in [1,n]} D_{M,j} - \bigcup_{k \in [1,m]} D_{B,k}$$

- This formula represents that S_M is the set of data behavior that consistently appears in *n* malware runs, but never appears in *m* kind runs. The underlying observation from this formula is that kernel malware will consistently perform malicious operations during attacks.

4.2.2 Malware Detection

This module detects the malware activities performed at runtime in a kernel execution in support with the malware signature pattern generated. It characterized the malicious data behavior by measuring the behavior instruction difference in relates to the generated signature pattern for an activity. In order to reliably characterize the data behavior of kernel malware in dynamic execution, we test multiple kernel runs over the signature generated for matching a malware signature.

- The likelihood that a malware program *M* is present in a tested run *r* is determined by deriving a set of data behavior elements in S_M which belong to the data behavior profile, D_r.
- This set *I* corresponds to the intersection of S_M and D_r *i.e.,*

$$I = \{i \mid i \in S_M \wedge i \in D_r\}$$

- S_M is a data behavior signature of a data behavior profile that is a set of data behavior elements derived by the intersection and union of data behavior profiles. If I has a variation from a define threshold limit then it can considered as malware detection. The detected malware are clustered using k-mean algorithm for further analysis.

4.3 Mathematical Model for Classification and Clustering

4.3.1 Classification Using Rule-Based Classification

Rules are a good way of representing information or bits of knowledge. A rule-based classifier uses a set of IF-THEN rules for classification. IF-THEN rules can be extracted directly from the training data using a sequential covering algorithm. The name comes from the notion that the rules are learned sequentially (one at a time), where each rule for a given class will ideally cover many of the tuples of that class.

Sequential covering algorithms are the most widely used approach to mining disjunctive sets of classification rules, and form the topic of this subsection. Note that in a newer alternative approach, classification rules can be generated using associative classification algorithms, which search for attribute-value pairs that occur frequently in the data. These pairs may form association rules, which can be analyzed and used in classification.

Algorithm: Sequential covering. Learn a set of IF-THEN rules for classification.

Input:

1. D, a data set class-labeled tuples;
2. Att-vals, the set of all attributes and their possible values.

Output: A set of IF_THEN rules.

Method:

1. Rule_Set = {}; //Initial set of rules learned is empty
2. For each class c do
3. Repeat
4. Rule = Learn_One_Rule(D, Att_vals, c);
5. Remove tuples covered by Rule from D;
6. Until terminating condition;
7. Rule_Set = Rule_Set + Rule; // add new rule to rule set
8. End for
9. Return Rule_Set

A basic sequential covering algorithm is shown in Algorithm. Here, rules are learned for one class at a time. Ideally, when learning a rule for a class, C_i, we would like the rule to cover all (or many) of the training tuples of class C and none (or few) of the tuples from other classes. In this way, the rules learned should be of high accuracy. The rules need not necessarily be of high coverage. This is because we can have more than one rule for a class, so that different rules may cover different tuples within the same class. The process continues until the terminating condition is met, such as when there are no more training tuples or the quality of a rule returned is below a user-specified threshold. The Learn One Rule procedure finds the "best" rule for the current class, given the current set of training tuples.

4.3.2 Clustering Using K-Mean Approach

The *k*-means algorithm takes the input parameter, *k*, and partitions a set of *n* objects into *k* clusters so that the resulting intracluster similarity is high but the intercluster similarity is low. Cluster similarity is measured in regard to the *mean* value of the objects in a cluster, which can be viewed as the cluster's *centroid* or *center of gravity*.

We define *k* main clusters and under each cluster we have sub-cluster as shown in Table 4.1,

Table- 4.1: k main clusters and sub clusters.

k Main Cluster	Sub-Clusters
1_DATA_ATTACKS	Ipsweep Mscan Nmap portsweep Saint Satan spy
2_APPLICATION_ATTACKS	guess_passwd ftp_write Imap multihop Named Phf Sendmail snmp Getattack snmpguess Warezmaster warezclient Worm Xlock xsnoop
3_SYSTEM_ATTACKS	buffer_overflow httptunnel Loadmodule perl rootkit ps sqlattack xterm
4_DOS_ATTACKS	Apache2 Back Land mailbomb Neptune pod Processtable SYNFlood Smurf Teardrop Udpstrom

We run, *K* parameters of data → KERNEL_TRAIN_DATA which already classified with class label, to create *n* Objects for each sub-clusters with it features data for the further classification → Files in Trained_Classification_Output Folder.

V. Evaluation

To measure the performance of the proposal we compute the Classification Accuracy by measuring True Positive, False Negative, False Positive, True Negative values. Ideally, malware detection systems (MDS) should have an attack detection rate (DR) of 100% along with false positive (FP) of 0%. Nevertheless, in practice this is really hard to achieve. The most important parameters involved in the performance estimation of malware detection systems are shown in Table-5.1.

Table-5.1: Parameters for performance estimation of MDS.

Parameters	Definition
True Positive (TP) or Detection Rate (DR)	Attack occur and alarm raised
False Positive (FP)	No attack but alarm raised
True Negative (TN)	No attack and no alarm
False Negative (FN)	Attack occur but no alarm

Malware behavior analysis has discovered 20 suspicious malware behaviors. We observed the executed malware by focusing it to the specific target and operation behavior in windows environment systems. We also classify the malware specific operation into 4 main classes which are Data, Application, System and DoS. Malware that are attacked File is group under Data class while malware that attacked Browser is group under Application class. Malware that are attacked Kernel, Operating System and Registry is group under System class. Lastly, malware that are attacked CPU, Memory and Network is group under DoS class as shown in Table-5.2.

Table-5.2: Machine learning classifier in malware classification

Class Target Operation (CTO)	Rank	Attack Example	Affected Example
Data	1	Malware attack office and adobe file	.doc, .ppt, .xls, .pdf etc
Application	2	Malware attack application audio and video application	Microsoft Office, Winamp and Windows Media Player
System	3	Malware attack the entire Operating System	Windows XP, Windows 7 and Windows Server 2008
DoS	4	Malware attack the physical hardware and entire machine	CPU usage, memory and network access

These four malware classes are related with each other in rank 1 to 4 starting with Data class and end with DoS class. These classes are actually inspired from basic fundamental of computer physical architecture. According to Ivy, physical architecture of host-based computer consists of application, operating system and hardware.

For this work we used 37042 trained samples data and 25 malware samples files for the evaluation. The control value in this experiment is called as —"Threshold" Threshold is the total or the value of training data that has been trained. This work trained the training sample first and the control value is set from 50 % to 90% trained samples. There are 25 sample malware data files for testing purposes and the same samples is used for each Threshold values. In order to review the classifier, accuracy are calculated by using formula as follows,

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

The experiment is conducted by using our new MDS Classifier. 5 sample malware files have been tested by adjusting Threshold value from 0.5 to 0.9. Table 5.3 and Fig. 5.1 show the relationship between Threshold value and the accuracy.

Table 5.3: Classifier Results Measures with different threshold

Threshold	TP	TN	FP	FN	Accuracy
50	1080	618	288	685	63.57169599
60	982	519	108	502	71.1037423
70	886	399	86	387	73.09442548
80	590	280	55	209	76.71957672
90	485	382	14	110	87.48738648

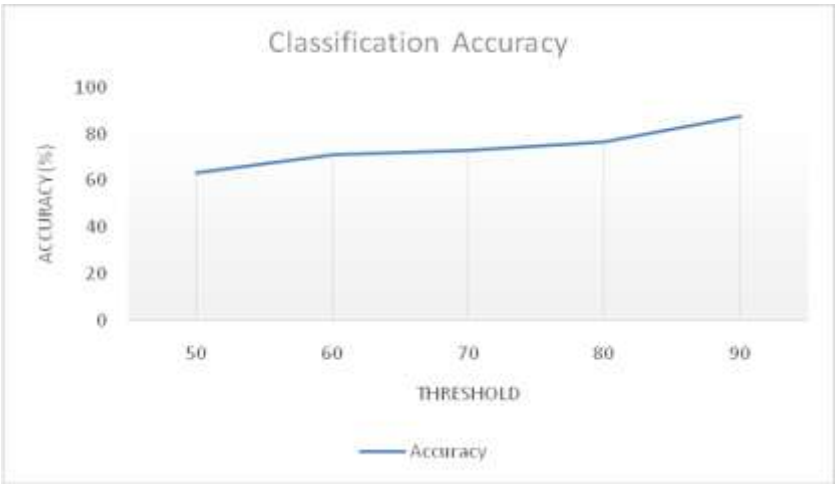

Fig. 5.1: Classification Accuracy

We observed the values of Parameters for performance estimation of MDS by using different Threshold values from 0.5 to 0.9. The Fig. 5.2 shows the graphical representation of different threshold values and these parameters.

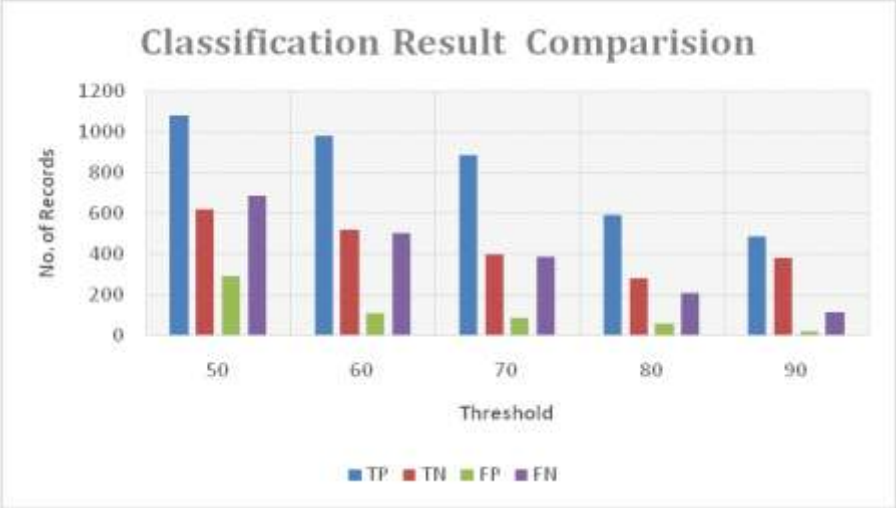

Fig. 5.2: Classification Result Comparison

VI. Conclusion

Many malware detection mechanisms rely on the properties of malware code such as the injection of unauthorized code and the patterns of malicious code sequences. While these approaches are effective for classic malware, emerging malicious programs are introducing advanced techniques such as return/jump oriented programming, code obfuscation, and code emulation to elude those malware detection mechanisms. In this work, we have presented a new approach for classifying kernel malware based on the properties of kernel data objects. This system is composed of two modules asMalware Signature Generation and Malware Detection which helps to design a classifier which enables to detect accurate malware on the kernel memory access patterns. Experiment result shows higher accuracy with increasing classification threshold.

References

[1] Junghwan R, R Riley, Z Lin, X Jiang, and D Xu, "Data-Centric OS Kernel Malware Characterization" IEEE Transactions on Information Forensics And Security, Vol. 9, No. 1, January 2014.
[2] Kaan Onarlioglu, Leyla Bilge, Andrea Lanzi, Davide Balzarotti, and Engin Kirda. G-Free: Defeating Return-oriented Programming through Gadget-less Binaries. In Proceedings of the 26th Annual Computer Security Applications Conference (ACSAC'10), 2010.
[3] H. Etoh, "GCC Extension for Protecting Applications From Stack-smashing Attacks". http://www.trl.ibm.com/projects/security/ssp/. Accessed May 2011.
[4] Jinku Li, Zhi Wang, Xuxian Jiang, Michael Grace, and Sina Bahram. Defeating Return-oriented Rootkits with"Return-Less" Kernels. In Proceedings of the 5th European Conference on Computer Systems (EUROSYS'10), 2010.
[5] Phrack Magazine. Linux On-the-fly Kernel Patching without LKM. http:// www.phrack.com/issues.html?issue=58&id=7. Accessed May 2011.
[6] M. W. Lucas Davi and Ahmad-Reza Sadeghi. Ropdefender: A Detection Tool to Defend against Return-oriented Programming Attacks. 2010. Technical Report HGI-TR-2010-001.
[7] Ryan Riley, Xuxian Jiang, and Dongyan Xu. Guest-Transparent Prevention of Kernel Rootkits with VMM-based Memory Shadowing. In Proceedings of 11th International Symposium on Recent Advances in Intrusion Detection (RAID'08), 2008.
[8] Davide Balzarotti, Marco Cova, Christoph Karlberger, Christopher Kruegel, EnginKirda, and Giovanni Vigna. Efficient Detection of Split Personalities in Malware. In Proceedings of the 17th Annual Network and Distributed System Security Symposium (NDSS'10), 2010.
[9] Clemens Kolbitsch, Paolo Milani Comparetti, Christopher Kruegel, Engin Kirda, Xiaoyong Zhou, and Xiao Feng Wang. Effective and Efficient Malware Detection at the End Host. In Proceedings of the 18th Usenix Security Symposium (Security'09),2009
[10] R. Riley, X. Jiang, and D. Xu, "An architectural approach to preventing code injection attacks," IEEE Trans. Dependable Secure Comput.,vol.7, no. 4, pp. 351–365, Dec. 2009.
[11] A. Seshadri, M. Luk, N. Qu, and A. Perrig, "SecVisor: A tiny hypervisor to provide lifetime kernel code integrity for commodity OSes," in Proc. 21st SOSP, Oct. 2007, pp. 1–17.
[12] C. Cowan, C. Pu, D. Maier, J. Walpole, P. Bakke, S. Beattie, et al., "StackGuard: Automatic adaptive detection and prevention of buffer-overfow attacks," in Proc. 7th USENIX Sec. Conf., Jan. 1998, pp. 63–78.
[13] Nergal, "The advanced return-into-lib(c) exploits: PaX case study," Phrack, vol. 11, no. 58, article 4, Dec. 2001.
[14] Arvind Seshadri, Mark Luk, Ning Qu, and Adrian Perrig. SecVisor: A Tiny Hypervisor to Provide Lifetime Kernel Code Integrity for Commodity OSes. In Proceedings of 21st Symposium on Operating Systems Principles (SOSP'07).ACM, 2007.
[15] Erik Buchanan, Ryan Roemer, HovavShacham, and Stefan Savage. When Good Instructions Go Bad: Generalizing Return-Oriented Programming to RISC. In Proceedings of CCS 2008, pages 27–38. ACM Press, October 2008.
[16] Hovav Shacham. The Geometry of Innocent Flesh on the Bone: Return-into-libc without Function Calls (on the x86). In Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS'07), pages 552–561. ACM, 2007.
[17] Monirul Sharif, Andrea Lanzi, Jonathon Giffn, and Wenke Lee. Impeding Malware Analysis Using Conditional Code Obfuscation. In Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS'08), 2008.
[18] Monirul Sharif, Andrea Lanzi, Jonathon Giffn, and Wenke Lee. Automatic Reverse Engineering of Malware Emulators. In Proceedings of the 2009 30th IEEE Symposium on Security and Privacy,2009