International Journal of Advanced Research in

Computer Science Engineering and Information Technology

Volume: 5 Issue: 2 09-Apr-2015,ISSN_NO: 2321-3337

HYBRID MALWARE DETECT MEMORY MAPPER IN KERNEL CENTRIC VM BASED GREEN COMPUTING WORLD

V. Nandhini

Student, Dept. of. Information Technology, Veltech University, India[1].

S.Deepan

Assistant Professor, Dept. of. Information Technology, Veltech University, India[1].

ABSTRACT—*Kernel malwares will offer user level-malware characteristics with an extra chances of concealment their malicious activities by sterilization the legitimate kernel behavior of operating system. Several analysis take on malware hook behavior and defense and preventive actions for constant. Still, an automatic analysis of the particular malicious goals and fix the behavior isn't investigated properly.Hybrid Malware detect memory Mapped provides an optimized solution to analyze windows kernel-level code and extract malicious behaviors from root kits, including sensitive data access, modification and triggers A new technique that provides a mix of backward slicing choice to check the mapped memory by slicing step by step within the kernel level. It will determine the malware influenced sensitive information and attainable resolution for this drawback.*

Keywords— Kernel, sterilization, triggers.

1. INTRODUCTION:

Network Security is that the method of taking physical and computer code preventative measures to guard the underlying networking infrastructure from unauthorized access, misuse, malfunction, modification, destruction, or improper revealing, thereby making a secure platform for computers, users and programs to perform their permissible critical functions within a secure environment. A new technique that provides a mix of backward slicing choice to check the mapped memory by slicing step by step within the kernel level. Malware observe memory Mapped provides associate optimized answer to research windows kernel-level code and extract malicious behaviors from root kits, additionally as

sensitive data access, modification and triggers. It's going to establish the malware influenced sensitive data and potential declare this downside.

2. OVERVIEW OF EXISTING SYSTEM:

The existing system goes together with the malware identification pattern victimisation the existing hardware Services (kernel) that identifies by detecting the Malware in order to clean up the worms and viruses by providing temporary protection to system. Existing approach projects towards Memory Performance Check, Memory management Leaks and Interoperability between managed (New version of Microsoft Language like c#) and unmanaged code (Older version of Microsoft Language like VC++). System call through Widows level coding invokes the malicious malware specification matching up with the suspicious system calls arises with existing malicious activity in the virtual operating system. Memory Mapping / Leaks leads to memory leakage in the virtual machine which leads to handle the files with the inappropriate usage of Application call for Kernel Mode Services Irregular memory wastage and improper properties of exe files while accessing the VM access program are some of the disadvantages of the existing approach.

2.1 ALGORITHM:

DKOM –DIRECT KERNEL OBJECT MALWARE ALGORITHM:

This method was presented for the malware manipulation process. The object malware algorithm detects the system and enables out-of-the box, tamper-resistant malware detection without losing the semantic view. In general this algorithm prevents the system comprises at least one guest operating system and at least one virtual machine, where the guest operating system runs on the virtual machine. Having virtual resources, the virtual machine resides on a host operating system. The virtual resources include virtual memory and at least one virtual disk where it acts to avoid the malware. DKOM along with a virtual machine inspector, a guest function extrapolator, and a transparent presenter, the virtual machine examiner resides outside the virtual machine.

2.2Drawbacks of the Existing System:

- We proclaim to propose a projected approach of how to find the malware initiated programs and prevent the malwares in the virtual machine.

3. PROPOSEDAPPROACH:

In our planned approach, the malware within the virtual machine is being detected and also tends to be monitored with the assistance of malware detector. Observation application execution involves Memory Management Leaks, Memory Performance Checks, Unmanaged Code execution and Listing down the malware and fixing it by implementing over some testing analysis like Malware bytes Anti-Malware (MBAM) scanner. Dynamic detection of malware activity in virtual surroundings detects the vulnerable activity in kernel power-assisted with proof concluding over the injected malware code and memory leaking mechanism.

3.1Merits:

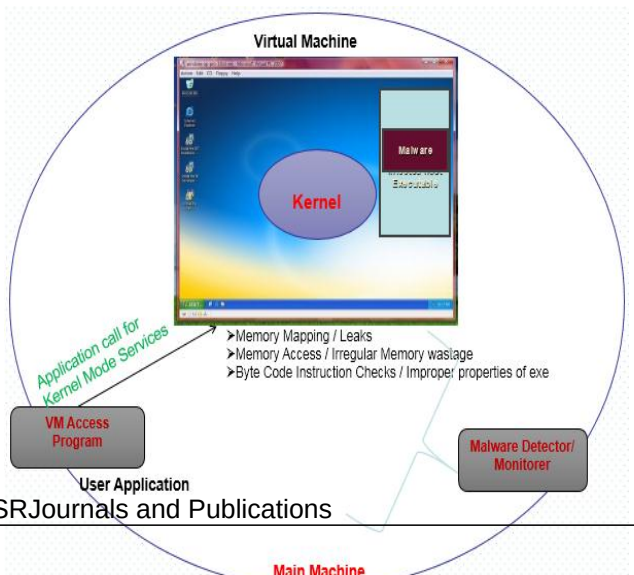
Malware detector and malware monitoring will eradicate and prevent the vulnerable malwares in the virtual machine by providing anti-malware protection.

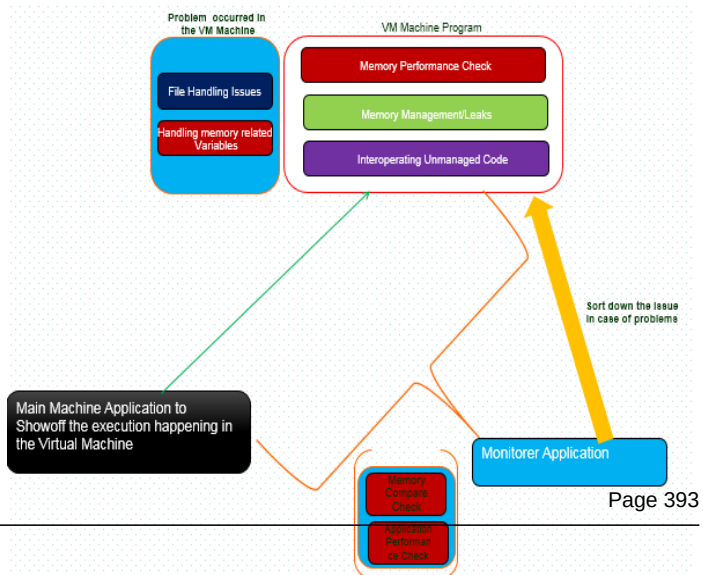
4.ALGORITHM AND DESIGN:

4.1 MALWARE BYTES ANTI-MALWARE SCANNER ALGORITHM:

The Malware anti-malware scanner algorithm is organized to use the interpreted virtual memory states and the interpreted virtual disk states to notice system's malware and also the affected files. The directions executed from outside of the virtual machine, comprising files to retrieve improper exe within the virtual machines internal. Based on non-intrusive virtual machine introspection without disturbing their execution, the virtual resources extrapolating guest functions by interpreting the memory states and also the virtual disk states.

4.2 ARCHITECTURAL AND FUNCTIONALITY ARCHITECTURE DIAGRAM:





5, IMPLEMENTATION DETAILS:

5.1, Malware creation:

The malware creation will be taken place in the virtual machine where the process of sophistication malware will affect the virtual environment. The malware created in the virtual machine will create improper properties of exe files, memory leakages, and other issues .In order to check these malware issues, at first no analysis of the core file will be done.

5.2, Main machine app calling:

A system call is a mechanism that is used by the application program to request a service from the operating system. In this module the malware program will enable the operating system to interact with a hardware device.The kernel takes responsibility for deciding many running programs. In the main machine app calling module.

5.3, Monitored Invoke:

User program running under guest OS will create issues in kernel call instruction. When guest OS (virtual os) returns from system call, the monitoring mechanism will be done by the virtual memory management (VMM) to invoke the malware. In this module the subset facilities of the underlying machine will invoke and monitor the memory management with extra mechanisms implemented by the operating system.

5.4, Memory leak check:

In this module the memory leaks in virtual machine by the malware will be identified in order to analyze the impact in the main machine. The memory leak check module will also identify the other mal functionalities (improper file handling) that occur in the virtual environment.

5.5, Memory Management Mapping:

The control of kernel operations will be mapping up with the overall machine memory in order to optimize the use of RAM, where there is a Memory Management issue. The Memory Manager includes memory-mapped files. Memory-mapping can speed-up sequential file processing due to the fact the data is not sought randomly, and it provides a mechanism for memory-sharing between processes. Scheduling of computing time and memory management is also part of the virtual machine monitors responsibilities

5.6, Invalid file properties validation:

In this module the invalid file properties of the malware will be monitored and identified to rectify the issues in the main machine. Validating the invalid files will be done with the help of the malware detector monitoring mechanism to avoid the misleading of file activities.

5.7, Patch applying:

A patch is a small text document containing a delta of changes between two different versions of a source. Patches are created with the `different ‘program in the kernel. In this patch applying module, the Patches for the kernel are generated relative to the parent directory holding the kernel source dir.

5.8, Performance evaluation:

In this module we will evaluate the performance of the malware monitoring system. The performance evaluation module will eradicate the overall performance of the virtual machine and the main machine after the malware were fixed and rectified.

6. CONCULSION:

The system works by building a live kernel object map that may faithfully observe info activity rootkit attacks due to its world organization tampered read of kernel objects. The map is then utilized together with an observation agent to trace operation patterns on kernel info objects. Supported these access patterns, we tend to propose a replacement malware signature approach victimization consistent patterns specific to malware attacks. Malware detector and malware observation will eradicate and stop the vulnerable malwares inside the virtual machine by providing anti-malware protection.

7. FUTURE ENHANCEMENT:

We imagine cell-phones equipped with sophisticated sensors that stream information to a centralized service for processing. Traditional operating systems have a fundamental mismatch with such networked stream-oriented applications. The job of a traditional OS is to run user tasks, not store and serve data. We propose a different approach in which the data-centric semantics of an application are exposed to the operating system. Our system, called Data-centric OS, enables a model where the application can simply declare its monitoring semantics by instantiating a Query File. Once an application has created a Query File, the application is no longer in the critical path of filtering, processing or storing a network stream. Instead, the Query File System plays an active role in efficiently processing and storing a network stream according to application semantics. And the Exposing application-level semantics to the OS, helps to filter unimportant data early and up calls to user-space are made only for semantically meaningful information. Application-specific processing paths can be set up to efficiently process incoming network packets.

Also, OS resources may be allocated and adapted with varying data rates according to application semantics and different operations on the same data can be integrated.

REFERENCES:

[1] M. Abadi, M. Budiu, Ú. Erlingsson, and J. Ligatti, "Control-flow integrity: Principles, implementations, and applications," in *Proc. 12thACM Conf. CCS*, Nov. 2005, pp. 1–4.

[2] A. Baliga, V. Ganapathy, and L. Iftode, "Automatic inference and enforcement of kernel data structure invariants," in *Proc. 24th ACSAC*, Dec. 2008, pp. 77–86.

[3] D. Balzarotti, M. Cova, C. Karlberger, C. Kruegel, E. Kirda, and G. Vigna, "Efficient detection of split personalities in malware," in *Proc.17th Annu. NDSS*, Feb. 2010, pp. 1–17.

[4] U. Bayer, P. Milani Comparetti, C. Hlauscheck, C. Kruegel, and E. Kirda, "Scalable, behavior-based malware clustering," in *Proc. 16thSymp. NDSS*, Feb. 2009, pp. 1–26.

[5] F. Bellard, "QEMU: A fast and portable dynamic translator," in *Proc. USENIX Annu. Tech. Conf.*, Mar. 2005, pp. 41–46.

[6] E. Buchanan, R. Roemer, H. Shacham, and S. Savage, "When good instructions go bad: Generalizing return-oriented programming to RISC," in *Proc. 15th ACM Conf. CCS*, Oct. 2008, pp. 27–38.

[7] J. Butler. (2012, Dec. 12). *DKOM (Direct Kernel Object Manipulation)* [Online]. Available: http://www.blackhat.com/presentations/winusa- 04/bh-win-04-butler.pdf.

[8] (2010). *Bypassing Non-Executable-Stack During Exploitation Using Return-to-Libc* [Online]. Available: http://www.citeulike.org/user/rvermeulen/author/C0ntex

[9] M. Carbone, W. Cui, L. Lu,W. Lee, M. Peinado, and X. Jiang, "Mapping kernel objects to enable systematic integrity checking," in *Proc. 16thACM Conf. CCS*, Nov. 2009, pp. 555–565.

[10] P. Chen, H. Xiao, X. Shen, X. Yin, B. Mao, and L. Xie, "DROP: Detecting return-oriented programming malicious code," in *Proc. 5thICISS*, Dec. 2009, pp. 163–177.