MAS: Malware Analysis System Based on Hardware-Assisted Virtualization Technology

Taehyoung Kim, Inhyuk Kim, Changwoo Min, and Young Ik Eom

School of Information and Communication Eng., Sungkyunkwan University,
300 Cheoncheon-dong, Jangan-gu, Suwon, Gyeonggi-do 440-746, Korea
{kim15m,kkojiband,multics69,yieom}@ece.skku.ac.kr

Abstract. There are many analysis techniques in order to analyze malicious codes. However, recently malicious codes often evade detection using stealthy obfuscation techniques, and attack computing systems. We propose an enhanced dynamic binary instrumentation using hardware-assisted virtualization technology. As a machine-level analyzer, our system can be isolated from almost the whole threats of malware, and provides single step analysis environment. Proposed system also supports rapid system call analysis environment. We implement our malware analysis system (referred as MAS) on the KVM hypervisor with Intel VT-x virtualization support. Our experiments with benchmarks show that the proposed system provides efficient analysis environment with low overhead.

Keywords: Malware, Dynamic Binary Instrumentation, Virtualization Technology, Intel VT, KVM.

1 Introduction

System security is the one of important issues to manage critical computing systems. Especially, malicious codes could leak confidential information of the individual and the enterprise, and infect hosts. Malware abuses incompleteness of system software and utilizes mistakes of system manager [1][2][3]. Thus, analyzing various malicious codes is essential to guarantee system security. However, complexity of obfuscation techniques has been increased continuously and newly modified malicious codes are generated exponentially. Therefore, more efficient and accurate analysis methods are required for secure system environments.

Recently, malware analysis methods which utilize virtualization technology are introduced such as Win32 API hooking, emulator, and full virtualization. Win32 API hooking shows excellent performance in the system call tracing, but can't trace instruction and memory access. Emulator, which realizes hardware function as software, can trace instruction and memory access, but it is too much slow and vulnerable to anti-debugging [4][5][6][7][8][9]. The full virtualization based on the binary translation is

© Springer-Verlag Berlin Heidelberg 2010

faster than emulator, but it is also vulnerable to anti-debugging. In case of full virtualization based on hardware-assisted VT, it shows better performance and escapes anti-debugging methods easily. Therefore, we propose the enhanced dynamic malware analysis system based on hardware-assisted VT.

2 Background

Dynamic code analysis method is utilized for various purposes such as observing behavior of process, developing applications, and so on. We design and implement dynamic code analysis system for analyzing malware. Therefore, in this section, we describe representative dynamic code analysis methods.

Because emulating environments is isolated from the real system, the emulator is utilized for various purposes. The emulator provides a virtual computing system through only software implementation without hardware support. Certainly, linux and other operating systems also can be executed on the emulator without code modification. Representative emulators are QEMU and Bochs. Also, representative dynamic code analyzers based on the emulator are BitBlaze [10], Renovo [11], VMScope [12], TTAnalyzer [13]. However, there is limitation to utilize emulator because of considerably slow executing speed.

In case of programs executing on general operating system, only general registers and user memory region are accessible. And another resource can be used through system call. On the other hand, the user-level dynamic code analyzer manage user context using shadow register, memory, and system call redirection. So, the user-level code analyzer can monitor all contexts without system call procedure in kernel context. Representative user-level code analyzers are Valgrind and Pin [14][15]. Even though user-level code analyzer is faster than emulator, it needs binary translation and can be easily exposed to anti-debugging.

Dinaburg proposed a machine-level dynamic code analyzer, Ether [16]. Ether is Xen based dynamic code analyzer using hardware-assisted function. Utilizing obfuscation break point of VAMPiRE, Ether supports memory trace, system call trace, and limited process trace. However, Ehter cannot analyze detail context such as DLL loading and API call information.

3 MAS

In this section, we introduce MAS: a transparent malware analysis system based on hardware-assisted virtualization technology. MAS provides two analysis phases: a single step phase, and a system call phase. Each analysis phase has special features. In the single step phase, MAS supplies detail analysis results of a whole process behavior. Especially, the single step phase is useful to analyze specific process. Simply observing behavior of process during user context, MAS provides an efficient analysis environment. Then, in the system call phase, MAS shows high performance close to a non-analysis phase.

3.1 Structure of MAS

Fig. 1 shows the structure of MAS. Using VT of Intel x86 processor, the full virtualization is implemented based on a KVM hypervisor. Memory virtualization is implemented as a shadow paging, and the device virtualization is implemented as the QEMU emulator.

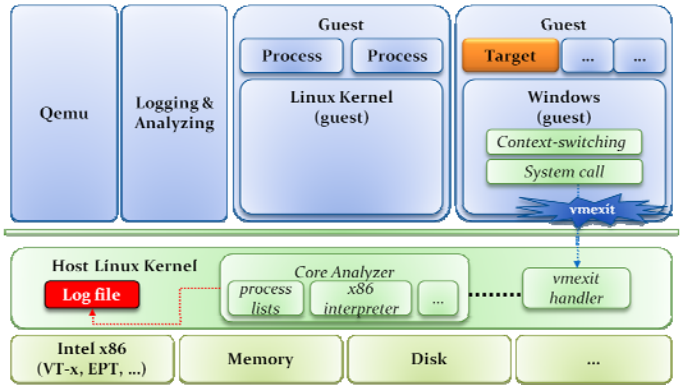

Fig. 1. Structure of MAS

We set a target process which runs on Windows guest OS. When the target process is switched by the scheduler, MAS begins analysis of target process. Invoking VMEXIT, MAS is able to analyze a whole context of guest depending on its purpose. Each VMEXIT handler carries out their duties such as page fault, general protection fault, and debug exception. A core analyzer component interprets instruction and manages process list which contains target process name. Also, the results of analyses are recorded whenever needed.

3.2 Single Step Phase

In the single step phase, the target process is observed by a unit of the instruction. For a complete single step analysis, MAS is designed to occur VMEXIT when guest OS access to cr register, or when debug exception is raised. The beginning of single step is to find a target process among all processes executing on guest OS. By the initialization setting, MAS invokes VMEXIT when the guest OS accesses to CR3 register for the context switching. In this case, the VMEXIT handler searches for a current process name in process lists to conform whether a current process is the target process or not. If the current process is the target process, MAS performs the single step analysis. Therefore, repeating the context switching check, we can select our concerned process. Fig. 2 shows the operation of the single step phase.

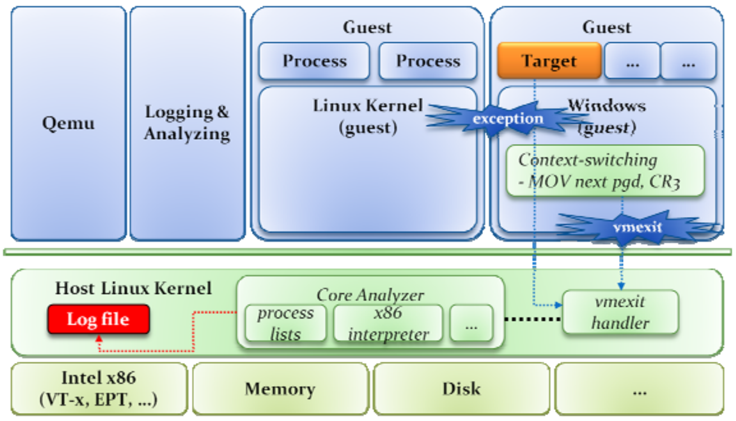

Fig. 2. Operation of single step phase

Once the target process is detected during context switching procedure, MAS sets up a trap flag of EFLAG register on x86 CPU. If the trap flag is turned on, guest OS will raise a debug exception immediately in the next instruction execution. According to the initialization setting of MAS, VMEXIT is occurred when the debug exception is raised. Then, the core analyzer interprets current context such as process id, instruction, register, stack, and memory. After logging necessary information, the trap flag is turned on once again. Therefore, the single step is continued until the not-target process is switched.

Furthermore, we can distinguish the current privileged level between the user level and the kernel level. Generally, it is required to monitor the user level context during the process execution for the malware analysis. Furthermore, malware frequently accesses disk and connects network. In many cases, these kinds of job require kernel services. And, malware which downloads new codes from the internet may require the procedure of network initialization. During the network connection, establishing connection is completed within a certain period of time. If the connection time is delayed by the process monitoring procedure, it is difficult to analyze the behavior of malware accurately. Therefore, our ring level detection scheme is powerful to monitor various malicious codes.

Ring level detection scheme is designed utilizing general protection fault. By setting guest machine to load a invalid descriptor during the context switching, we lead guest machine to invoke VMEXIT by the general protection fault. When the user mode is changed to kernel mode, and vice versa, VMEXIT is invoked. Then, we set the single step phase on/off state suitably. Thus, we can monitor only the user context.

Using single step, MAS supports several functions to meet requirements for dynamic malware analysis. Basically, instruction analysis is supported. When instruction accesses memory to read/write, corresponding related information are recorded in the logging file. Also, if instruction calls API, arguments and a return value of API call are recorded in the logging file. Fig 3 shows the algorithmic flow of single step.

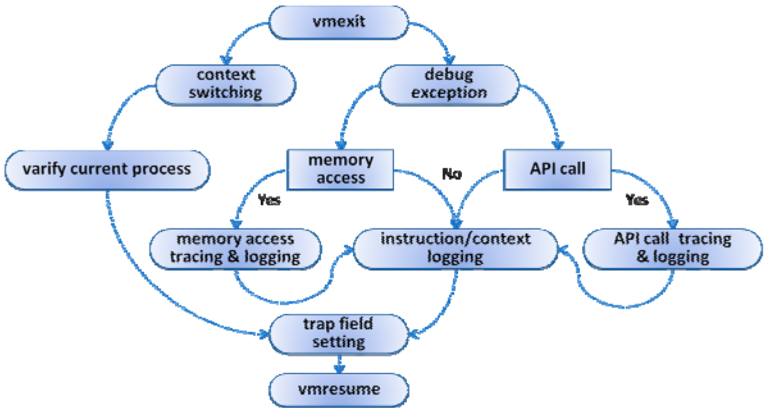

Fig. 3. Algorithmic flow of single step

3.3 System Call Phase

The system call tracing is possible in the single step phase. However, there is performance overhead in single step phase because the single step executes thousands of instruction to analyze one instruction. Therefore, MAS provides another analysis phase for tracing system call efficiently. Unlike the single step, the system call phase manages page fault by SYSENTER assembly instruction. As a fast system call, SYSENTER needs three MSR: SYSENTER_EIP_MSR, SYSENTER_CS_MSR, SYSENTER_ESP_MSR. MAS utilizes a feature of the SYSENTER operation. When the system call phase is set up, MAS injects invalid address to SYSENTER_EIP_MSR. Invalid address invokes page fault causing VMEXIT. Then, SYSENTER handler in hypervisor interprets system call information such as system call number and arguments. After logging related information, SYSENTER handler substitutes an invalid address of current EIP register by a valid address.

4 Implementation and Evaluation

Our proposed system, MAS, is implemented on the KVM hypervisor for the Intel x86 processor. KVM is a virtual machine monitor for Intel x86 architecture. KVM utilizes a modified version of QEMU for the device emulation. MAS is implemented as an extension version of KVM using Intel VT-x for the processor virtualization. In our implementation, we use Linux kernel 2.6.31. All the experiments are performed on a PC wih 2.8GHz Intel Core i7 processor and 4GB of physical memory. A guest OS is Windows XP with servicepack2.

In this section, we subscribe practical scenario by the single step phase of MAS and show experimental evaluation of MAS.

4.1 Practical Scenario of MAS

In this section, we verify that MAS provides powerful analysis environment by analyzing the typical malicious code. Above all, we analyze packing malware. Malware utilizes packing methods in order to make analyzers to take a long time for analyzing the executing code. Using packing methods, malware can practice their object until packing is unpacked. Therefore, if packed malware is unpacked quickly, propagation of malware is minimized and the damage could be decreased.

Using MAS, we analyze the malicious packing code. We utilize the feature of the packing method. Fundamentally, packing codes copy some codes to specific address to restore original code. After unpacking the code, the corresponding code is executed on original entry point. Namely, the unpacking time is when the written code is executed. Therefore, we monitor the packing malware using a single step phase. We can analyze every instructions, memory accesses, and API calls. Especially, we trace a memory write operation. When there is the memory write operation, we record values of corresponding value of EIP, other registers, and instruction. Then, if the instruction on the logged address is executed, unpacked time is found finally. As above, MAS supports efficient malware analysis environment.

4.2 Performance Evaluation

The purposes of our experiments are to evaluate overhead of the system call phase and demonstrate effectiveness in the ring level detection scheme of the single step phase. In order to evaluate performance overhead, we utilize SPEC CPU 2006 benchmark. SPEC CPU 2006 is a CPU-intensive benchmark suite. We use 10 benchmarks among 29, which is composed of 12 integer benchmarks and 17 floating point benchmarks. Fig. 4 shows performance comparison of the system call phase in the proposed system. Baseline is the normalized running time when analysis phase is not applied. Total overhead of the whole benchmark is small, 1.33%. High overheads of some cases which need a lot of system call are lower than 6%.

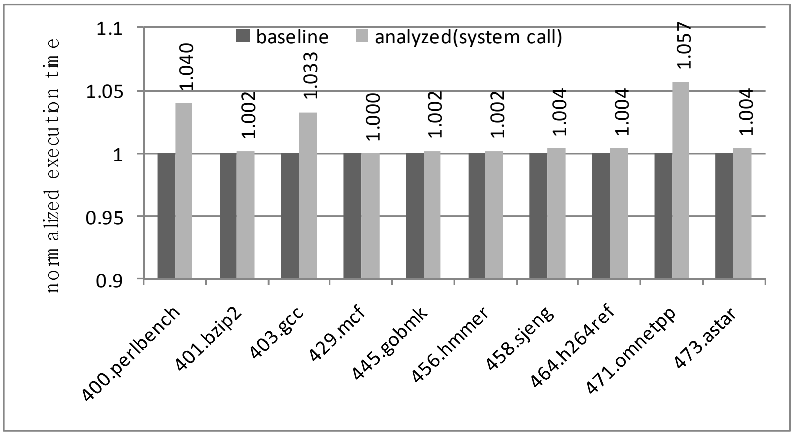

Fig. 4. Performance evaluation using SPEC 2006

In order to demonstrate effectiveness in the ring level detection scheme of the single step phase, we illustrate normalized execution time of representative GNU tools. Fig. 5 shows performance comparison of the single step phase in our system. Baseline is the general single step phase in which the overall execution context including both user context and kernel context are analyzed. User-only is the analysis mode which analyze only user context by the ring level detection scheme. Execution time of tar, md5sum, gzip, and wget are reduced significantly. Their performance gains over baseline are 48%, 32%, 14% and 42% respectively. In overall performance, the proposed system achieves 21% performance improvement.

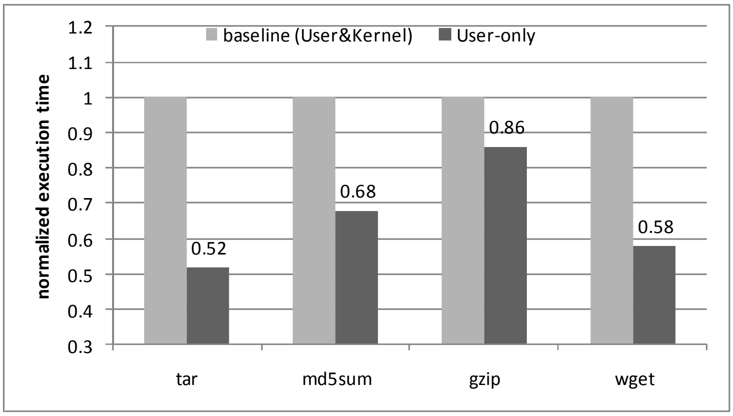

Fig. 5. Performance evaluation using representative GNU tools

5 Conclusion

In this paper, we introduced the enhanced malware analysis system based on hardware-assisted virtualization technology. Our system provided two analysis phases: single step phase and system call phase. In the single step phase, the detailed and various analyses are supported such as instruction tracing, memory tracing, and API call tracing. Especially, the ring level detection scheme strengthened efficiency of the single step phase. In case of the system call phase, it provides more efficient analysis environment. We implemented our system on KVM hypervisor using Intel VT-x. Throughout typical malware analyses, we show that our system provides the appropriate and efficient analysis environment. Our performance evaluations show that the system call phase of MAS has small performance overhead and the single step phase based on the ring level detection scheme significantly improves performance of analysis according to exclude analysis of kernel context.

Acknowledgement

This work is financially supported by the Ministry of Education, Science and Technology(MEST), the Ministry of Knowledge Economy(MKE) through the fostering project of HUNIC.

References

1. Idika, N., Mathur, A.P.: A Survey of Malware Detection Techniques. Research, Dept. of Computer Science, Purdue Univ. (2007)
2. Carvey, H.: Malware analysis for windows administrators. Digital Investigation 2, 19–22 (2005)
3. Pfleeger, C.P., Pfleeger, S.L.: Security in Computing. Prentice Hall, Englewood Cliffs (2003)
4. Garfinkel, T., Adams, K., Warfield, A., Franklin, J.: Compatibility is Not Transparency: VMM Detection Myths and Realities. In: Proc. of 11th Workshop on Hot Topics in Operating Systems (2007)
5. Ferrie, P.: Anti-unpacker tricks. In: CARO Workshop (2008)
6. Ferrie, P.: Attacks on Virtual Machines. In: AVAR Conf., pp. 128–143 (2006)
7. Listion, T., Skoudis, E.: On the Cutting Edge: Thwarting Virtual Machine Detection. SANS Internet Storm Center (2006)
8. Chen, X., Andersen, J., Mao, Z.M., Bailey, M., Nazario, J.: Towards an Understanding of Anti-virtualization and Anti-debugging Behavior in Morden Malware. In: DSN 2008, pp. 117–186 (2008)
9. Xu, M., Malyugin, V., Sheldon, J., Venkitachalam, G., Weissman, B.: ReTrace: Collecting Execution Trace with Virtual Machine Deterministic Replay. In: Proc. of 2007 Workshop on Modeling, Benchmarking and Simulation (2007)
10. BitBlaze Binary Analysis Platform, `http://bitblaze.cs.berkeley.edu`
11. Kang, M.G., Poosankam, P., Yin, H.: Renovo: A Hidden Code Extractor for Packed Executables. In: Proc. of WORM (2007)
12. Jiang, X., Wang, X., Xu, D.: Stealthy Malware Detection Through VMM-Based Out-of-the-Box Semantic View Reconstruction. In: Proc. of CCS, pp. 128–138 (2007)
13. Bayer, U., Kruegel, C., Kirda, E.: TTanalyze: A Tool for Analyzing Malware. In: Proc. of EICAR, pp.180–192 (2006)
14. Instrumentation Framework for building dynamic analysis tools, `http://valgrind.org`
15. A Dynamic Binary Instrumentation Tool, `http://pintool.org`
16. Dinaburg, A., Royal, P., Sharif, M., Lee, W.: Ether: Malware Analysis via Hardware Virtualization Extensions. In: Proc. of ACM CCS (2008)