Research Paper

Volume : 2 | Issue : 6 | June 2013 • ISSN No 2277 - 8179

Virtualization Evolution For Transparent Malware Analysis

Computer Science

KEYWORDS : Hardware virtualization, transparent malware analysis, ether analyzer

Leenu Singh Jamia Hamdard University, Computer Science Department, New Delhi, India

Syed Imtiyaz Hassan Jamia Hamdard University, Computer Science Department, New Delhi, India

ABSTRACT

Reverse engineering is commonly used by the malware analysts to detect the runtime activities of the malicious codes. To study these malicious codes, security researchers are always in search of analyzers that can complete their analysis without malwares being aware of them. Hardware Virtualization provided a platform to analyze the malwares and to protect the host systems from their effects but this technique was not effective for longer period. Malware authors started developing the malicious codes that could detect the environment where the codes were executing and therefore virtualization alone could not provide a shield for analyzers to hide their environment from the malwares. Therefore, there was the need to create a transparent analyzing system that was not detected by the malicious code.

Numerous works by security researchers has been done in this field. In this paper, author will brief and compare many such techniques, but will mainly focus on the transparent virtual environment designs and related technical issues of Ether and its based systems like VERA, ETHERANNOTATE that combine the hardware virtualization techniques with software emulations and lastly will highlight the previously detected loop hole of the ether that rejects its successful implement as a transparent analyzer.

1. INTRODUCTION

Malware–the increasingly common vehicle by which criminal organizations facilitate online crime–has become an artifact whose use intersects multiple major security threats faced by information security practitioners [4].

" Malware is a piece of code which changes the behavior of either the operating system kernel or some security sensitive applications, without a user consent and in such a way that it is then impossible to detect those changes using a documented features of the operating system or the application" [3].

Different types of malwares exits, which can be easily be classified as [27] virus and worm. Viruses are the malicious codes that has to be attached with the some or the other application files which then in occurrence of any event replicate themselves or cause damage to the system hardware. Whereas worms are malicious codes that run on networks and replicate them by self-initialization. Beside there are many concealments like Trojan horse, Root kits, backdoors, botnets, spywares, backdoors, and exploits.

Malwares analysis today is becoming a serious concern for the analyzers. Malwares have become

smart enough to depict the virtual environments provided by the analyzers. Malware designers work hard in designing and implementing smart malwares. Therefore, modern malware contain anti-debugging, anti-instrumentation, and anti-Virtual Machine (VM) techniques that help the malwares to escape from the analyst observations. More on this, static code analyzers are frequently challenged by the run time defined malicious codes and researchers require these codes to get executed to learn their characteristics.

1.1 Types of Malware analysis

Before we discuss about the platforms used for analysis, we will throw some light on criteria of malware analysis. Analysis as defined by Casey is the process of organizing and structuring previously recovered, low-level data into higher-level evidence and narratives of behavior. [1]

Malware analysis can be categorized in any of the form:

- Temporal Analysis: This analysis considers the ordering of the recorded events and actions in time so that sequential study of records can be done.
- Relational Analysis: This type of analysis concentrates on linking the traces detected.
- Functional Analysis: It is an attempt to find out the technical moves and the effects of the execution of these codes.

1.2 Virtualization

Virtualization hides the physical characteristics of a computing platform from users, instead showing virtual abstract computing platform. Virtualization can host multiple guest operating systems. Each guest OS runs in its own domain and handles its own applications. Virtualization can be deployed in one of the two forms [2] [7]:

- Full virtualization: This type of implementation abstracts the complete underlying hardware and creates a virtual system where guest OS can run. In this environment, one system is considered as host (hypervisor), that consist of a thin OS that hosts
- VM's and another system is required that monitors the VM's running on another machine, termed as Virtual Machine Monitor (VMM) or Management Console.
- Paravitualization: In this, guest OS is aware that the hardware is virtualized as the virtual machine emulators. It implies that the guest operating systems need to be modified.

2. IMPLEMENTATION

The paper will first brief the architecture designs and implementations of all the transparent hardware virtualization environments developed so far and then will check their implementations with various static and dynamic tools.

2.1 Architecture Designs

2.1.1 Xen Hypervisor

Xen was published in 2003, in Cambridge University by a group of researchers. It is open source software that takes the advantage of x86 architecture which is used for implementation of hardware virtualization [6] [2].

Xen works on different levels of privileges that make this system a unique one. It allows the unmodified operating systems to run in isolated virtual machines. In xen, multiple domains are created and the lowest domain (Dom0) is given the highest privileges as an administrator that can have the direct access to the real hardware. Beside this domain all the other domains are used to reside operating systems and are referred as domU. All the other domains are at the same privilege level.

Xen works in two modes- VMX root mode and VMX non-root mode. The Xen hypervisor runs in VMX root mode. The domUs or guests, which we refer to as the analysis targets [6], run in VMX non-root mode.

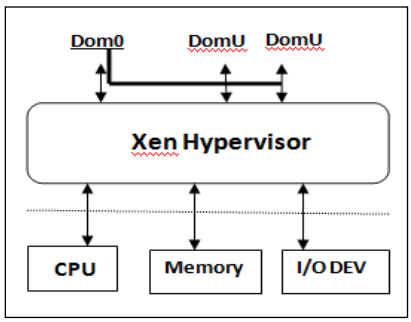

Figure 1. Xen logical layout

VM transitions are used to change the modes of operation between VMX root mode and non-root mode. These transitions can be classified as:

- VMEntry: this event is used to perform transition from VMX root mode to the VMX non-root mode.
- VMExit: This transition is used to transit from VMX non-root mode to the VMX root mode.

2.1.2 ETHER

Ether [5] designed by Dinaburg et al., runs as a component in the hypervisor layer, and as a user space component in Dom0. Ether uses the analysis target as Windows XP with Service Pack 2 and runs it in a DomU. Ether exploits the modes of X86 Intel VT architecture and the Xen hypervisor privileges to provide a transparent virtualization environment. Ether runs between the modes-VMExit and VMEntry .

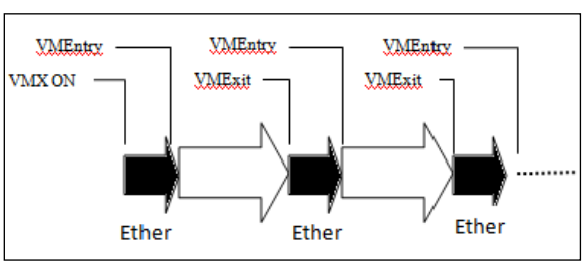

Figure 2. Ether operates between VMExit and VMEntry

Ether was designed by Dinaburg et al. in 2008. Ether was implemented by making the ether hypervisor component part of the Xen hypervisor that runs on the bare metal and the ether user-space component works in the Dom0, which provides ether the highest privileges in the system. Ether provides a transparent virtualization environment for malicious codes and the analysis is done with the help of Windows XP Service Pack2 running in DomU [5] that is user domain. Etherannotate designed by Joshua Michael Eads, also use ether as a platform for carrying out a transparent environment analysis and uses IDA Pro static analyzing tool (explained in section 2.1.4) Plug-in to detail the malware characteristics [6]. Daniel Quest and Lorie also published their work in 2010, in which they used the tool, VERA (Visualization of Executables for Reversing and Analysis) [8], which used a method for monitoring running programs via modifications to the Ether framework.

2.1.3 V2E Protocol

V2E Protocol [11] was designed in March 2012 by Yan et al. This implementation provided both transparency and extensibility. In this method, the malware execution is recorded with the help of HVM recorder by hardware virtualization and then replayed in software (via. software emulation). Here analyst runs the malware samples in an emulated execution environment like QEMU and analyze its behavior in a fine-grained manner.

2.1.4 Static and Dynamic Analysis

Malware analysis tools are designed for static and dynamic malware analysis, where [6] [8]:

- Static Analysis: Static analysis techniques use disassembler, decompiler, string searching tools and signature matching systems for static analysis of assembly language code. Few of the popular static analyzers are Ollydbg [18] that works as disassembler, IDA Pro that provides a graphical representation of the disassembled code that separates it from other disassemblers [6], Hex-Rays [19] that work as an plug-in for IDA Pro and acts as an decompiler to convert the disassembled instruction to pseudo code.
- Dynamic Analysis: This analysis technique monitors and controls the malicious codes in run-time environment [9]. Many dynamic malware analyzers are designed to detect the exact execution behavior of the malicious code. Analysts also have been working on providing the transparent virtualized environment to the malwares so that they do not detect the analyzer's environment. Example being Ollydbg, ProcMon, Emulators, Ether.

3. Evaluation of Analyzers

Hardware virtualization that act as the platform for dynamic malware analysis has been revealed to carry lot many loop holes and lacks in preventing the analyzer to be transparent to the malware. Attackers now-a-days design malwares that use anti-debugging, anti-instrumentation, and anti-VM techniques to hide or miss guide the runtime observations. Following Peter Ferrie [10], attacks can be categorized into [7]:

- Detection of VM presence to conceal malicious code activities
- Denial of service on the virtual machine emulator
- The most challenging and threatening type of attack VM escape.

Therefore, there was a requirement of the analyzer that works directly on the bare metal so that malwares do not detect any change in their execution environment and along with that it becomes possible for the analyst to read the run time behavior of the malicious codes. This section of my paper is based on the study of different ways in which analysts have used the hardware directly for the malware analysis so that the transparency of the analyzer is maintained.

3.1 Ether based analyzers

In Ether architecture, Dinaburg stated a definition [5],

"Assume E is a runtime environment, and A is E with malware analyzer PA running. PA is transparent if for any malware P containing any detection logic dP, dP (A) = dP (E) = 0". This definition states that the environment of a analyzer can be kept transparent from the malwares by implementing the detection logic, dP (by which malicious codes detects the difference between the environment of the simple hardware implemented system and the analyzer) similarly on the two different environments, one with the hardware and the other one with analyzer. Proof of this definition was given with the help of the privilege level (domains) defined in X86 Intel VT processor architecture [13] and by patching the ether with the Xen hypervisor kernel, that on combining these factors enables ether to reside in the Dom0 with higher privileges. Ether defines five requirements [5] that help ether to attain generalization of hardware and operating system factors like privilege levels, virtual memory, exception handling, and attaining transparency.

- Higher Privilege: This requirement states that the analyzer should have higher privileges than all the other programs including the one with the malicious codes. Many earlier malware analyzers like CWSandbox [16], VAMPiRE [17] do not satisfy this condition as they employ their user level or kernel level counterparts in the host which can be easily used for higher privileges by the rootkit method by the malwares to have access to the hardware resources that are emulated. The reduced privileged guests and full system emulation based virtualization approaches like VMware [14] and VirtualPC [15] for x86 satisfy this requirement. Therefore, ether, etherannotate, VERA and all other ether based systems also satisfies this requirement as they reside

in the Dom0 i.e. ring -1.

- No non-privileged side effects: This requirement states that all the side effects created by the analyzer in the system should be accessed and handled through exception handlers at the higher privileged levels than the general programs. Hardware virtualization can achieve this requirement as they completely isolate the analyzer's domain with the help of shadow page tables. Hardware virtualization extensions not only enable basic memory protections, but also offer privileged access to sensitive CPU registers and instructions including instructions that access time, such as RDTSC. Therefore, all ether based analyzers satisfy this requirement also.
- Identical Basic Instruction Execution Semantics: As defined in the second requirement above, the basic semantics do not involve any side-effects introduced by analyzer program. As hardware virtualization extensions rely on the same hardware execution semantics, therefore ether guarantees that the third requirement is satisfied.
- Transparent Exception Handling: This requirement states that to trap the action caused by the malicious code ether uses exception handling and this handling of exception should not be noticed by the malware. Requirement (3) stated above and (5) defined next supports to achieve this requirement. This requirement is also achieved by the ether as it guarantees that a debug exception occurs after the execution of each instruction by setting a trap flag in the analysis target.
- Identical measurement of Time: This requirement states that the timing information received by the instruction in the analyzer environment and system environment should be same.

In ether, the side effect on time is privileged because the instructions that access time (e.g., RDTSC) are privileged. In virtualization extensions, a separate execution cycle count in the hypervisor is maintained, which allows an analyzer to adjust a cycle value before it is given to the guest. Ether does this by adjusting the time of the system clock analyzer timings by causing a VMExit at every system call execution which passes on the control to VMX root mode; thereby target is never able to detect the analyzer presence in the system with the help of timing attacks. Ether's hypervisor component is responsible for detecting events such as system call execution, instruction execution, memory writes, and context switches. Ether protects the detection of these changes made to the guest to maintain the transparency of the system. Ether hides the trap flag by intercepting the instructions used to read the trap flag value and changes the result to what the analysis target should be given. Next it hides the shadow table modifications from the guest by only keeping the guest page tables in guest environment and making guest unaware of the shadow tables. Finally, to monitor the system call execution, ether makes modification in the register SYSENTER_EIP_MSR that stores the address of the privilege level. SYSENTER instruction is used in x86 mechanism processors to change the privilege level and it uses the SYSENTER_EIP_MSR to jump to the defined address for the privilege level. Ether prevents the system call execution by storing a false address in the SYSENTER_EIP_MSR and whenever target attempts for system call it is swapped to an address that does not exists and thus, causing a page fault. By using the accepting handling procedure ether changes the address to the one expected by the target and now the target easily resumes its functioning.

3.1.1 VERA (Visualization of Executables for Reversing and Analysis)

Daniel in work on VERA [8] have defined that visualizing compiled executables for malware analysis with the help of reverse engineering process greatly reduces the time to extract the key features of an executable. Open Graph Drawing Framework (OGDF) is used to organize the graph defined from the parse trace files which are in turn generated using Ether. Reverse engineering extends the Xen-Ether framework to deobfuscate a binary file with faster speeds and gives the flow and better understanding of the compiled executable. The graph layout obtained from OGDF is passed on to the display engine tool, VERA. VERA uses OGDF GML tool to translate the 2-D view to the 3-D view for better zooming options. As VERA is an extension to ether framework hence, VERA provides a transparent framework for malware analysis. This work is now added to the ether framework.

3.1.2 ETHERANNOTATE

Joshua designed EtherAnnotate [6] using ether as analyzer where the malicious are executed and a log file is generated that keeps the record of all the instructions executed, list of all register values referenced in the instruction and other referenced memory locations. These records are considered as EtherAnnotate Instruction Trace file. This file is then subjected to the system where IDA Pro, dynamic malware analysis tool with EtherAnnotate plug-in is installed. IDA Pro disassembler and the IDA Python EtherAnnotate plug-in annotate all instructions using the trace file previously generated EtherAnnotate is based on ether framework therefore it is capable of providing the transparent execution environment to the analysis target.

3.1.3 nEther

Analysts at Crysys (Laboratory of Cryptography and System Security) implemented an application framework named as nEther [20] that was successful in detecting the Ether framework. In their work new detection attacks were used like in-guest timing attack and verification based on the CPU specific design defects. In-guest timing is detected here by a contradiction raised by clock cycle manipulation that is done by the analyzer to hide its presence by making the adjustments in the CPU cycles. Secondly, the CPUID instruction that is used to get information regarding processor identifications and features is used. Though, it is quoted in Ether work that it leaves no in-memory and in-register presence of itself but still nEther have detected that it alters few bits of information. Lastly, due to design deficiencies of processor, nEther was not able to detect the presence of Ether but this approach can be used to detect hardware assisted virtualized environments.

3.2 V2E Implementation

An analysis platform exploiting both the hardware virtualization and software emulation was implemented by Yan et al. recently in March'2012 [11]. In aiming goal to achieve the transparency and extensibility, this platform was designed with the essence of precise heterogeneous replay. Ether proved to provide a transparent base for analysis but as stated in V2E work, it lacks the flexibility of instrumentation support and the analysis efficiency. V2E believes to provide such a flexible platform for code instrumentation as it uses the software emulation implementation where it is easier for the analyst to add-on the various code instrumentation. Yan et al. used the hardware virtualization to implement a transparent malware analysis environment where they used a recorder to collect all execution moves of the malware. This recording is then subjected to TEMU [28], dynamic binary analysis tools (QEMU with some modifications) to conduct a fair analysis of the malicious code.

The heterogeneous replay was first proposed and implemented in Aftersight [12], which records the virtual machine execution from VMware and replays it in QEMU. V2E lacks in several aspects like its single core support and denial of service attack on recorder.

4. RELATED WORK

A study by Garfinkel et al. [21] proposed a lightweight VMM (namely MAVMM) that is designed specifically for a single job: malware analysis. MAVMM does not implement unnecessary virtualization features commonly found in general purpose hypervisors, including virtual device emulation. Getu et al. [22] proposed a malware analyzer that can detect different layers of packing and is capable of extracting and reconstructing both syntactic and semantic behaviors of the packed malware. Garfinkel et al. [23] have argued that it is not always possible to change the timing information to avoid timing attacks. Azure [24] was implemented by Paul Royal that implemented KVM on Intel VT for a transparent malware analysis. Martin [7] in his paper on virtual security has discussed the pros and cons

of the virtualized environment for anti-virtualization malicious codes. Tavis Ormady [25], in his work stated the success rates for popular virtual machine implementations for x86 systems by applying the combination of source code auditing and black-box random testing to assess the security exposure to the hosts of hostile virtualized environments. BareBox [26], implementation of malware analysis framework on bare metal is designed by Kirat et al. in which a fast and reboot less system restore technique is defined.

5. SUMMARY

The malware analyzers discussed so far in the paper are summarized in this section. The criteria's used to pen down the different characteristics are firstly, according to the features of tools, that is discussed in table 1 and secondly, the analyzing tools are evaluated on basis of their outcomes and shortcomings, displayed in table 2.

Table 1. Features of Tools

Tool	Transparency Technique	Tracer	Analyzer
Ether	H/w virtualization using Xen and patching ether to Xen	EtherTrace coarse-grain tracing	EtherUnpack fine-grained tracing (unpacking of code)
VERA	Ether based architecture	Trace file is created by making modifications to ether.	1. Trace files are parsed to generate graphs using OGDF by reverse engineering. 2. VERA displays 2-D view into 3-D space using OGDF GML technique.
Etherannotate	Ether based architecture	EtherTrace coarse-grain tracing	IDA Pro-static code analysis tool
V2E	H/w virtualization using KVM in linux kernel 2.6.32. Techniques used are memory management, logging and event landmark	Recorder realm control and enforcement logic implemented in top-page-fault function.	Replayer implemented on TEMU, a dynamic analysis platform.

Table 2. Evaluation of Tools

Tool	Positive outcomes	Shortcomings
Ether	Remains transparent on malware executions on all anti-debugging and anti-VM's attacks by completely residing outside of the target OS environment.	Application framework. nEther detected: 1.ether on basis of timing information, CPUID info 2. Due to CPU errata, h/w virtualization could be detected but not the ether presence.
VERA	Transparent on all anti-debugging and anti-VM attacks and enhances speed of reverse engineering and reduces time to reverse engineer an executable	1. nEther detection 2. Requirement for clearer visualization of loops.
Etherannotate	Transparent on all anti-debugging and anti-VM attacks and statically view what values variables and memory addresses the program held during its actual runtime	1.nEther detection 2.complex register referencing and direct memory access
V2E	Transparent and extensible platform tested successfully for malware analysis by recording the execution traces and analyzing by replaying outside the OS environment	1. In case of emulation bugs, successful replay cannot be done. 2. denial –of-service attacks possible 3. supports only single core guest environment 4. weak landmark mechanism used

6. CONCLUSION

In the paper, basically extensions of hardware virtualization implementations that provide a platform for transparent malware analysis were considered and among all the implementations Ether is given most limelight as it is used by many analysts for transparent hardware virtualization along with other testing tools to analyze binary codes. After study of these out-of-guest analyzers, it is found that most of the work was successful in implementing the transparent analyzer for analysis as compared to other emulation based (in-guest) and static and dynamic malware analysis, as shown in summary tables 1 and 2 (section IV). The shortcomings detected by nEther are processor design deficiencies and can be improved by using other processor like AMD-V that overcomes the design deficiencies of Intel-VT architecture [5] [20]. Author leaves this implementation for future work.

REFERENCE

Chalmers, M. and Chitson, P., Bead, "Explorations in Information Visualization", in Proc. Of the 15th Annual Int'l ACM SIGIR Conference on Research and Development in Information Retrieval, p330-337, 1992 | [2] Virtualization Guide: RedHat Virtualization https://access.redhat.com/knowledge/docs/en-US/Red_Hat_Enterprise_Linux/6/html/6.0_Release_Notes/documentation.html | [3] Joanna Rutkowska, "Introducing Stealth Malware Taxonomy", Version 1.01, COSEINC Advanced Malware Labs, November 2006. | [4] Malwares, http://en.wikipedia.org/wiki/Malware | [5] Artem Dinaburg, Paul Royal, Monirul Sharif, Wenke Lee, "Ether: Malware Analysis via Hardware Virtualization Extensions", ACM CCS 2008 | [6] JOSHUA MICHAEL EADS, "Etherannotate: A transparent malware analysis tool for integrating dynamic and static examination", THESIS Presented to the Faculty of the Graduate School of MISSOURI UNIVERSITY OF SCIENCE AND TECHNOLOGY in Partial Fulfillment of the Requirements for the Degree MASTER OF SCIENCE IN COMPUTER SCIENCE 2010 | [7] Martin Wimmer Siemens AG, "Virtual Security: About the Security Pros and Cons of Server Virtualization, Corporate Technology", CT IC CERT D-80200 Munich, Germany martin.r.wimmer@siemens.com | [8] Daniel A. Quist, "Visualizing Compiled Executables for Malware Analysis", Vezsec, 2009. | [9] A. Vasudevan and R. Yerraballi. "Spike: Engineering malware analysis tools using unobtrusive binary-instrumentation". Volume 48, Hobart, Tasmania, Australia, Australian Computer Society, Inc., January 2006. | [10] P. Ferrie, "Attacks on More Virtual Machine Emulators". Technical report, Symantec Research. February 2007. | [11] Lok-Kwong Yan, Manjukumar Jayachandra, Mu Zhang Heng Yin, "V2E: Combining Hardware Virtualization and Software Emulation for Transparent and Extensible Malware Analysis", Department of Electrical Engineering & Computer Science, Air Force Research Laboratory, Syracuse University, Syracuse, New York, USA , March 2012. | [12] J. Chow, T. Garfinkel, and P. Chen. "Decoupling dynamic program analysis from execution in virtual environments". In Proceedings of 2008 Usenix Annual Technical Conference (ATC'08), June 2008. | [13] Enabling Intel® Virtualization Technology Features and Benefits, Maximizing the benefits of virtualization with Intel's new CPUs and chipsets www.intel.es/.../virtualization-enabling-intel-virtualization-technology | [14] VMWare. http://www.vmware.com. | [15] VirtualPC. http://www.microsoft.com/windows/products/winfamily/virtualpc/. | [16] C. Willems, T. Holz, and F. Freiling. "Automated Dynamic Malware Analysis Using CWSandbox", IEEE Security and Privacy, 5(2), 2007. | [17] Amit Vasudevan and Ramesh Yerraballi, "Stealth Breakpoints", IEEE exporer, Dec 2005. | [18] O. Yuschuk, Ollydbg debugger and disassembler. Product Description Page. http://www.ollydbg.de/. | [19] Hexrays. Ida pro disassembler and debugger. Product Description Page. | http://www.hex-rays.com/idapro/. | [20] Gábor Pék, Boldizsár Bencsáth, Levente Buttyán, "nEther: In-guest Detection of Out-of-the-guest Malware Analyzers", at Laboratory of Cryptography and System Security (CrySyS), Budapest University of Technology and Economics in April 2011. | [21] AM Nguyen, N Schear, HD Jung, "Mavmm: Lightweight and purpose built vmm for malware analysis", ACSAC'09, 2009 - ieeexplore.ieee.org, 2009 | [22] Getu, Tewfik Adem, "Comparison and benchmarking of automatic malware unpacking techniques", December'2010 | [23] T. Gar_nkel, K. Adams, A. War_eld, and J. Franklin, "Compatibility is Not Transparency: VMM Detection Myths and Realities", in Proceedings of the 11th Workshop on Hot Topics in Operating Systems (HotOS-XI), May 2007. | [24] P. Royal, "Alternative medicine: The malware analyst's blue pill", Aug 2008. | [25] Tavis Ormandy, "An Empirical Study into the Security Exposure to Hosts of Hostile Virtualized Environments", taviso@google.com Google, Inc., Feb 2007. | [26] Dhilung Kirat, Giovanni Vigna, Christopher Kruegel, "BareBox: Efficient Malware Analysis on Bare-Metal", Acsac'2011. | [27] Tala Tafazzoli and Seyed Hadi Sadjadi, "Malware fuzzy ontology for semantic web", IJCSNS International Journal of Computer Science and Network Security, VOL.8 No.7, July 2008, Pg 154. | [28] Heng Yin, Dawn Song, "TEMU: Binary Code Analysis via Whole-System Layered Annotative Execution", Technical Report No. UCB/EECS-2010-3. |