Collabra: A Xen Hypervisor based Collaborative Intrusion Detection System

Saketh Bharadwaja, Weiqing Sun, Mohammed Niamat
University of Toledo
Toledo, Ohio, USA

Fangyang Shen
Northern New Mexico College
Albuquerque, New Mexico, USA

Abstract—**In this paper, we introduce Collabra, a distributed intrusion detection platform based on Xen hypervisors to maintain the security of the cloud based on virtualized network. While the concept of virtual machine monitor (VMM) signifies implementing an abstraction layer between the underlying host and the guest operating system (OS) to enforce security, its kernel is required to be free of vulnerabilities that intruders can use to compromise the host. In Xen, guest applications make resource requests through the hyper-call API to transfer the privilege to the VMM kernel for executing privileged operations. On a cloud scale, there exist hundreds of VM networks and thousands of guest operating systems (OSes) running on virtual domains. There is every possibility of intruders trying to misuse the hyper-call interface to compromise guest OS kernels and finally the host OS kernel itself. Sophisticated attacks can be launched in the distributed and collaborative style thereby bypassing most current intrusion detection systems. Collabra acts as a filtering layer which is completely integrated with every VMM. It scans through each call by incorporating integrity checking and collaborative detection mechanisms. It exists in multiple instances, and acts concurrently over a VMM network interacting with other instances to detect (possibly collaborative) attacks and prevent illicit access to the VMM and the host. An admin version of Collabra exists on a privileged domain in the VM network to perform filtering of malicious add-ons to hyper-calls at the guest OS level itself before routing the call to the VMM.**

Keywords-Virtual machine; distributed intrusion detection system; intrusion detection architecture; cloud computing; hyper-call

I. INTRODUCTION

Virtualization is a significant element of cloud computing. High performance hardware can be effectively utilized to host multiple virtual machines which form virtual organizations. Such virtual networks facilitate the management, lower the cost, and therefore have seen increasingly popular usage nowadays. Similar to the traditional host based networks, an IDS (intrusion detection system) component is typically deployed inside each individual virtual host within the virtual network, as proposed in [1]. However, questions are raised about the effectiveness of this type of IDS deployment. Although it is capable of detecting attacks at the level of operations within the virtual host, it's insufficient in protecting the virtual networks belonging to the organization. Because sophisticated attacks may target at the underlying infrastructure that host the virtual networks. For instance, the attacker might take advantage of the vulnerability in VMM, and in turn subvert the physical host, part of the underlying physical networks, and eventually all the virtual networks supported by them. Approaches that deploy IDS at the VMM layer helps to mitigate this type of threat. But to make things even worse, attackers can take control of multiple virtual hosts in the virtual network, and then launch distributed attacks on the underlying physical infrastructure. The individual VMM based IDS module lacks the global view in general, and has difficulty in identifying the correlations between multiple operations that are part of the correlated attack. There's a possibility that sophisticated attacks bypass the existing IDS module within the virtual host, and the IDS module at the VMM layer. This calls for an innovative and effective approach to prevent this type of attacks.

Xen is one of the most popular virtualization software. We use Xen hypervisor [2] for our research because it is open source and more friendly to virtualization research. The main objectives of the Xen hypervisor are support for a wide variety of commodity OSes, isolation of resources, least performance overhead of virtualization and huge support of VMs per host [2, 3]. In Xen, guest OSes make resource requests via hyper-calls. It has only 35 hyper-calls compared to 324 system calls for the Linux 2.6.22 kernel [4]. The functionality of a hyper-call is similar to that of system-calls in an OS. It is a software trap from a domain or a virtual machine to Xen just as a system call is a software trap from an application to the kernel [4]. As a hyper-call transfers control to a more privileged state in the VMM, guest VMs use hyper-calls to request privileged operations like low level memory requests. The hyper-call like the system call is synchronous but the return path from Xen to the guest VM uses event channels [2].

In order to successfully compromise the host kernel, the intruder needs to compromise the guest VM kernel, the privileged domain (Dom 0), followed by the VMM and finally the host kernel. On a virtualized infrastructure on the cloud scale, a VMM is hosted on each host which emulates the underlying hardware and creates a virtualized VM network. As discussed above, hyper-calls are utilized for applications in guest VMs to gain a more privileged state. We therefore propose Collabra to act as an interfacing layer between the VMM and Dom 0 in the guest VM network. In addition, an admin version of Collabra exists in a domain alongside Dom 0 to check security critical events and

concurrent requests from guest OSes. To prevent attacks of the VMM by multiple hyper-call requests from multiple VMs in the network, this layer performs integrity authentication of hyper-calls dynamically and then transfers them for privileged execution by the VMM. Collabra is integrated with each VMM and has embedded communication protocols through which it interacts with its instances on other VMMs in the subnet to make a collaborative effort to analyze call patterns and perform integrity checking in real-time. This architecture helps to prevent not only attacks targeting at individual VMM and physical host, but also those attacks that are distributed and collaborative in nature.

Subsequently in the paper, we provide the background for our work in Section II and related work in Section III. The Collabra architecture is described in Section IV, and its effectiveness is discussed in Section V. Finally, we conclude our work in Section VI.

II. BACKGROUND

Successful cloud security approaches involve actively performing intrusion detection of various calls made by each application in every VM, thereby ensuring integrity of the whole infrastructure and a fail-safe transaction process. The existing Xen VMM architecture and privilege model have implications for security and enable an administrator to take complete control of all VMs running on every host. This makes it improbable for the VM users to completely trust in the confidentiality and integrity of the virtual machine [5]. The traditional representation of Xen architecture as illustrated in Figure 1 shows several guest virtual machines placed concurrently on top of the hypervisor [6]. The privileged domain Dom 0 is called the management or control VM which contains the control plane software like drivers. On the Xen hypervisor, the privilege levels are defined as follows: The host kernel runs in ring 0 since it's the most privileged level and the guest VM applications run in ring 3, the least privileged level. The Xen VMM and the Dom 0 which form the most privileged of the TCB (Trusted Computing Base), run in ring 0. The kernels of the unprivileged VMs run in ring 1 since they are para-virtualized.

Guest applications in ring 3 raise hyper-call requests requiring the host kernel's privilege in ring 0 for executing some processes and hence, the call is routed through the Dom 0 and the VMM. The VMM checks for the authenticity of the call and enables its execution on a high privilege level. Hyper-call forms the most primitive mechanism for communication. They are configured to build higher level communication primitives such as in facilitating inter VM communications; Xen provides event channels and a mechanism to establish shared memory regions [1]. As mentioned above, the hyper-call is the counterpart of the system call: the former involves a software trap from the guest VM to XEN and the latter, from an application to the host's kernel. Xen hyper-call interface plays a significant role in Xen security. Most security incidents presently happen in operating systems involving system calls directly or indirectly [3].

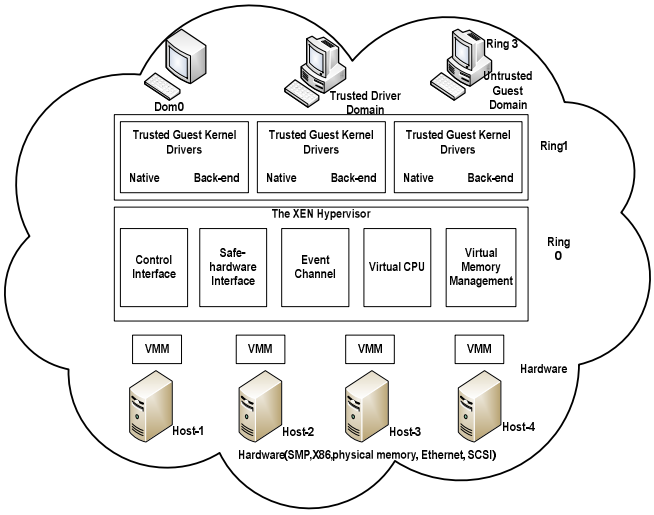

Figure 1. Xen hierarchical architecture [4].

Since hyper-calls are highly privileged, it could become an attractive vector for attackers to perform effective attacks. The attacker could apply security objectives like denying services to guests, introduction of low-level root-kits which find their way under the host kernel to generate passive attacks on a long run. These root-kits may have the capability of sniffing the guest's information or destroying the complete virtualized setup.

III. RELATED WORK

There have been works that focused on protecting the virtualization setup by inspecting hyper-calls and preventing intrusions effectively, but none have yet focused on making use of communication protocols for implementing a collaborative filtering mechanism in a virtualized atmosphere. Hoang [3] proposes an enlightened approach in analyzing the security properties of the Xen architecture which shows that hyper-call attacks could be a real threat to the architecture. The paper demonstrates that hyper-calls could be exploited to inject malicious code into the VMM by a compromised guest OS. Research done by Cully [6] and Ormandy [7] focuses on the aspect that randomized hyper-call attacks can corrupt the virtualization platform prompting security issues that need to be resolved urgently.

As observed, a typical hyper-call attack occurs in the following sequence:

1: The attacker carefully scans all the applications in the guest VM network and compromises one of them with the most vulnerability.

2: The existing vulnerabilities are not taken into consideration since the guest VM is connected to the outside world and explicitly open to attacks.

3: Once compromised, the attacker may then enhance his privileges by system call based attack methods.

4: As the attacker gains privilege to the kernel of the guest VM which exists in ring 1, he can then launch hyper-call based attacks to compromise the VMM.

Hyper-calls to VMM are similar to system calls to OS; therefore, we can draw on research study on system call

based approaches. According to [9], most operating system security violations involve the invocation of system calls at some point. Control data attacks and non-control data attacks are two prominent system call attacks. The former refer to data loaded into a program counter during program execution such as stack pointers, function pointers, which could be emulated in compromising the VMM code by modifying hyper-calls [10, 11]. Non-control data attacks such as configuration data and user input, involve targeting other data types instead of code pointers to achieve their malicious objectives as in [12, 13, 14]. On the other hand, [15] focuses on reducing the size of the trusted computing base thereby reducing the attack surface significantly. This helps in controlled monitoring of hyper-calls and the possible vulnerability exploiting code it may carry. [16] is a good example of anomaly detection using system call stack information by using the *VtPath* method which extracts return address information extracted from the call stack. This is one of the techniques adopted by Collabra in performing successful screening and backtracking. Finally, to address buffer overflow attacks by sending a deluge of hyper-calls after compromising the VM kernel, mitigation techniques including 1: non-executable stack, heap, data sections, 2: address space layout randomization, and 3: stack smashing protection have been used in [17].

IV. COLLABRA DESIGN

A. System Architecture

As shown in Figure 2, Collabra instances are integrated with the VMM of each host in a multiple-host virtualized environment. We design this setup to implement a collaborative filtering mechanism for guest VM initiated hyper-calls. Our objective is to prevent intrusions via hyper-calls from compromised guest VMs in order to protect the VMM and maintain uninterrupted services to other guest VMs. Since there are no documented hyper-call attacks to date, we focus the implementation of our security system on anomaly detection instead of misuse detection mechanisms. The latter are very well known to have a very low false alarm rate and have better performance, but anomaly detection mechanisms are helpful in detection of unknown attacks and are more appropriate for hyper-call screening. Approaching hyper-call anomaly detection completely based on system calls anomaly detection processes is not viable since there are subtle differences between the underlying operating system and the virtual machine monitor architecture. Most documented system call attacks as described in [18, 19, 20] are predominantly based on the sequences in which the attacker exploits the vulnerability (for example, opening a file, writing into a file, etc.) These methods may not be effective in anomaly detection of hyper-call attacks since there is no certainty of attack mechanism which could be employed. However, there are documented system call attacks [21, 22] which could be emulated in the case of hyper-call attack scenarios.

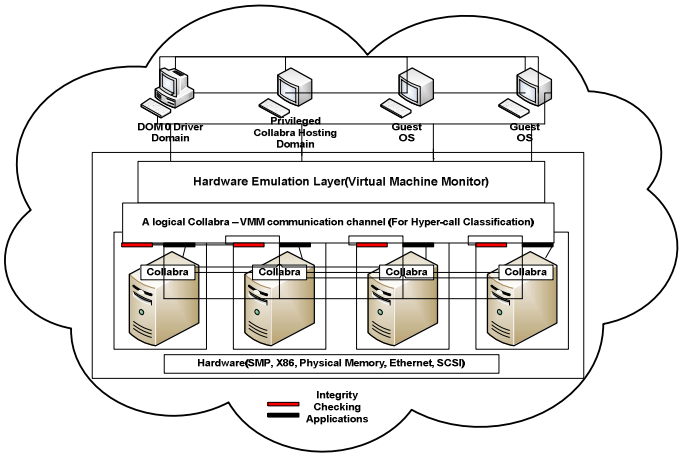

Figure 2. Modified Xen Architecture with Collabra System Integrated within the VMM of Each Host.

Guest VMs are exposed to the outer world and are inherently vulnerable since their hosted intrusion detection and prevention systems have the limited scope of protecting their own kernels from getting compromised. By dynamically exploiting vulnerability in the VMM architecture, a compromised VM can try taking control of its associated VMM. As the VMMs are separated semantically, even if one is completely compromised, all the other VMMs and VMs supported by this VMM remain safe. This situation would get off limits, as the exploited vulnerability in the architecture would apply to all the VMMs in the virtualized atmosphere. Collabra integrated with each VMM also hosts a solid isolation feature. When an intrusion is detected, it dynamically isolates the affected module which hosts the communication protocols for interacting with other VMMs.

B. Hyper-call Classification and Integrity Check

TABLE I. HYPER-CALLS WITH VULNERABILITY LEVELS

Vulnerability Level	Hyper-calls
1	sched_op
1	nmi_op
1	iret
1	multicall
1	get_debugreg
1	physdev_op_compat
1	update_vamapping_otherdom
2	platform_op
2	fpu_taskswitch
2	stack_switch
2	get_gdt
2	mmu_update
2	xsm_op

A selected set of hyper-calls available in the Linux kernel version 2.6.22 is summarized in Table I based on the vulnerability levels of the hyper-calls. The higher the level, the more vulnerable is a particular hyper-call.

Our intrusion detection system is designed to be able to communicate with multiple instances of itself hosted on each VMM. It first monitors each anomalous call routed by the admin version in the VM network. The admin forms a trusted access point. The type and origin of the hyper-call will be recorded and communicated to the VMM directly by the privileged Collabra domain for the specific instance to identify among the numerous calls routed through it. It then, immediately notifies all the other instances through logical domain channels (discussed in Section IV.C) about the attack characteristics and notifies the specific VMM to be sanitized. The hyper-call classification model below will help to get a better understanding of the attack characteristics through a concept called ***anomaly score***.

1) Hyper-call Classification Mechanism

Let's consider an element m_i responsible for a hyper-call. Its task is to assign an *anomaly score* of a_i for each invocation of a hyper-call. a_i calculates the probability of the occurrence of anomalous calls against a set number. Based on the anomaly score output $\{a_i \text{ where } i = 1 \ldots k\}$ of k elements $M = \{m_1 \ldots m_k\}$ and possibly additional information I, a conclusion must be met whether a certain hyper-call invocation is malicious (anomalous) or legitimate (normal). This process is called hyper-call classification. The classification could be defined more formally as a function S which, for a certain hyper-call h with a set of arguments, accepts as input the corresponding element outputs (anomaly scores) $\{a_i \text{ where } i = 1 \ldots k\}$ and additional information I. The result of this classification function is a binary value that identifies the hyper-call h as normal or anomalous. So, for a certain hyper-call h the function call classification S is as follows.

$$S(a_1, a_2, ..., a_k, I) = \{normal, anomalous\} \quad (1)$$

In anomaly based intrusion detection systems, S is a simple function that calculates the sum of the anomaly scores a_i and compares it to I that represents threshold. So, S is defined as follows.

$$S(a_1, a_2, ..., a_k, I) = \{normal : \sum_{1}^{k} a_i \le I, anomalous : \sum_{1}^{k} a_i > I\} \quad (2)$$

However, there is a possibility that a certain argument might not be suitable to distinguish between attacks and legitimate behaviors. The same values of this argument may appear in both legitimate and malicious behaviors. In such cases, it is useful to reduce the influence of an element's output on the final decision.

Our assumption is that all guest virtual machines initially start with a clean state which is not compromised by the intruders to launch malicious hyper-call based attacks. The aim is to ensure that the privileged domain hosting the Collabra admin version is safe from being compromised and is at the same privilege level of Dom 0. For successful implementation of our system, we employ reactive mechanisms to detect and respond to attacks. In particular, we equip Collabra architecture with two key security components, *Hyper-call Integrity Check* and *Hyper-call Origin Access*.

2) Hyper-call Integrity Check

In the virtualized infrastructure, there exist hundreds of VMs in the virtual network with each group of VMs hosted by a VMM in a subnet. When a hyper-call is raised by a guest VM, the specified policy for that call and a message authentication code (MAC) in cryptographic format are cross checked by Collabra against the repository. It is an integrity checking station with two data repositories. A trusted call is allowed to the root kernel and an unknown call is stalled and cross checked with other instances through the communication channel for hyper-call classification. It basically checks for the privileges of the guest VM, the type of hyper-calls its applications can raise and whether the calls are legitimate. The message authentication code helps to maintain integrity of the VM network supported by each VMM. Each VM in a network supported by a VMM is given a MAC code computed over a byte string using AES [23] based message authentication.

3) Hyper-call Origin Access

In large networks of VM running numerous guest applications, most of them raise hyper-call requests to gain kernel level privilege. There is a possibility of some calls not being screened due to privilege constraints, argument constraints and coverage problems. Our aim is to make Collabra fail-safe and coherently thwart malicious hyper-call requests. But it is possible that the filtering mechanism be overwhelmed due to the sheer magnitude of requests coming in. Therefore, the admin version of Collabra is designed to perform the task of identifying the call origins of legitimate hyper-calls. Calls made from other locations are labeled *untrusted* and the corresponding Collabra instance notified about the details of the call such as the type of application which raised it, whether the VMs in that network have the privilege to raise such requests and how many requests are concurrently being raised. In our architecture, the calls are routed from the Collabra privileged domain in each VM network to the Dom 0, to the specific Collabra instance lying upon the VMM and finally to the VMM for privileged execution. This design enables us to distinguish hyper-calls invoked illegally by malicious code from those legally invoked as part of the guest VM's normal execution.

C. Communication Mechanism

Each Collabra instance is considered as a logical domain and the channel that one Collabra instance establishes with other instances within the same virtual organization is considered as a logical domain channel (LDC). These channels provide a virtual link layer abstraction that is designed as a point to point communication channel between each instance. The logical domain channels provide an

encapsulation protocol onto which higher level transports such as TCP/IP and PPP are built. Within a logical domain, a logical device channel is instantiated as a single endpoint. Thus, there exists only one channel for the two-way communication between Collabra instances.

When the admin version of Collabra in the privileged domain identifies an anomaly, it notifies the specific instance through Dom 0. If the specific instance of Collabra hosted on the VMM is not able to identify the characteristics of the hyper-call raised by the guest VM, it communicates with other instances in the network through a simple packet based data transfer mechanism.

Packets of analysis data can be transferred by encapsulating it into LDC packets and then sending them directly from one instance to another. As mentioned above, the communication link layer protocol provides the flexibility to choose any of the mechanisms. The link layer protocol is responsible for fragmenting the messages to be reassembled at the destination. Additional header information is added to each packet to indicate the start and the end of each fragmented packet. We use this mechanism mainly for sending important short messages since this helps in ensuring complete invisibility from the intruder. When a packet of data is to be sent from an instance of Collabra, it first checks if there is a logical channel which is already established with other instances in the VMM network. It is not allowed to create multiple channels to different instances since they are already busy detecting anomalies.

V. Collabra Security Analysis

When a virtual machine has completely compromised its VMM, it is exposed to a plethora of privileged accesses sufficient to compromise the entire setup. However, as mentioned earlier, the system considers the VMM as an ordinary application with no special rights. Each VMM runs in its own address space and as such does not have any means of communication with other VMMs. Collabra bridges this barrier by acting as a filtering layer atop each VMM which talks to its instances through the device drivers provided by the hypervisor. This calls urgently to the requirement that the drivers need to be actively secured as they act as the root means of communication across the virtualization infrastructure. Device drivers, however, have the ability to shut down the communication channel to a VMM in case it accepts too many requests overwhelming the system.

Authors in [23] point out that drivers can be very large such as those for software modems and wireless cards. It makes gaining confidence in them improbable. Their code tends to be of questionable quality in most kernels and is the main source of security issues. The Collabra instances first need to be protected from these drivers. This can be achieved by confining drivers using hardware memory protection and restricting their access to the domain channels. Due to their high vulnerability level, the drivers are not included in the TCB of our architecture, as recommended in [24].

VI. Conclusions and Future Work

In the paper, we present a resilient intrusion detection architecture called Collabra for dynamic filtering of malicious hyper-calls in a virtualized environment to defeat sophisticated distributed attacks against the virtual networks. The key techniques involve designing the Collabra architecture, classifying hyper-calls and performing integrity check on hyper-calls.

For future work, we plan to provide a complete prototype Collabra implementation based on Xen hypervisors and evaluate its overall effectiveness and performance impact. In addition, we plan to incorporate other IDS models into the Collabra system for detailed comparison and further enhancements.

References

[1] P. Chen, B. Noble. When virtual is better than real. In Proceedings of the 8th workshop on Hot Topics on Operating Systems, page 133, Washington D. C., USA, 2001.

[2] B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, I. Pratt, A. Warfield, P. Barham, and R. Neugebauer. Xen and the art of virtualization. In Proc. of the ACM Symposium on Operating Systems Principles (SOSP), Oct 2003.

[3] D. Kuhlmann, R. Landfermann, H. Ramasamy, M. Schunter, G. Ramunno and D. Vernizzi. An Open trusted Computing Architecture: Secure virtual machine enabling user-defined policy enforcement. Technical report, OpenTC consortium, 2006.

[4] C. Hoang. Protecting Xen hyper-calls. Intrusion Detection/Prevention in a Virtualized Environment. MS Thesis, University of British Columbia. Jul 2009.

[5] D. G. Murray, G. Milos, S. Hand, Improving Xen Security through Disaggregation. VEE'08, March 5-7, Seattle, Washington, USA, 2008.

[6] B. Cully. The virtual monkey monitor. Technical report, University of British Columbia, Jun 2006.

[7] T. Ormandy. An empirical study into the security exposure to hosts of hostile virtualized environments. Technical report, Google Inc., 2007.

[8] Cert security advisories. http://www.cert.org/advisories/

[9] Microsoft Security Bulletin. www.microsoft.com/technet/security/bulletin/ms10-mar.mspx

[10] United States Computer Emergency Readiness Team. Technical cyber security. http://www.us-cert.gov/

[11] C. Cowan, S. Beattie, J, Johansen and P. Wagle. PointGuardTM: Protecting pointers from buffer overflow vulnerabilities. In Proc. of the 12th Usenix Security Symposium, Aug 2003.

[12] G. Edward Suh, Jae W. Lee, D. Zhang, and S. Devadas. Secure program execution via dynamic information flow tracking. In ASPLOS-XI: Proceedings of the 11th international conference on Architectural support for programming languages and operating systems, pages 85–96, New York, NY, USA, 2004.

[13] W. D. Young and J. McHugh. Coding for a believable specification to implementation mapping. In Proc. of IEEE Symposium on Security and Privacy, pages 140–148, Oakland, CA, USA, Apr 1987.

[14] U. Steinberg, B. Kauer. NOVA: A Micro hypervisor-Based Secure Virtualization Architecture. EuroSys' 10, Apr 13-16, 2010, Paris, France.

[15] H. H. Feng, O. M. Kolesnikov, P. Fogla, W. Lee, W. Gong. Anomaly Detection Using Call Stack Information. In Proceedings of the 2003 IEEE Symposium on Security and Privacy, 2003.

[16] H. Fritsch, Buffer Overflows on linux-x86-64. For BlackHat, Apr 16, 2009.

[17] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff. A sense of self for Unix processes. In Proc. of the 1996 IEEE Symposium on Research in Security and Privacy, pages 120–128. IEEE Computer Society Press, 1996.

[18] A. Ghosh A. Schwartzbard. A study in using neural networks for anomaly and misuse detection. In 8th USENIX Security Symposium, pages 141–151, 1999.

[19] C. Warrender, S. Forrest, and B. A. Pearlmutter. Detecting intrusions using system calls: Alternative data models. In IEEE Symposium on Security and Privacy, pages 133–145, Oakland, CA, 1999.

[20] G. S. Kc and A. D. Keromytis. e-nexsh: Achieving an effectively non-executable stack and heap via system-call policing. In Proc. of the 21st Annual Computer Security Applications Conference, Dec 2005.

[21] C. M. Linn, M. Rajagopalan, C. Collberg, S. Baker, S. K. Debray, and J. H. Hartman. Protecting against unexpected system calls. In Proc. of the 14th USENIX Security Symposium, Oct 2005.

[22] T. Iwata and K. Kurosawa. Omac: One-key cbc mac, Pre-proceedings of Fast Software Encryption, FSE 2003.

[23] T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, D. Boneh. Terra: A Virtual Machine-Based Platform for Trusted Computing. *SOSP'03*, Bolton Landing, New York, USA. Oct 2003.

[24] B. Leslie and G. Heiser. Towards untrusted device drivers. Technical Report 0303, University of New South Wales, Mar. 2003.