Francesco Gadaleta, Raoul Strackx, Nick Nikiforakis,
Frank Piessens, Wouter Joosen*

On the effectiveness of virtualization-based security

Protecting commodity operating systems and applications against malware and targeted attacks has proven to be difficult. In recent years, virtualization has received attention from security researchers who utilize it to harden existing systems and provide strong security guarantees. This has lead to interesting use cases such as cloud computing where possibly sensitive data is processed on remote, third party systems. The migration and processing of data in remote servers, poses new technical and legal questions, such as which security measures should be taken to protect this data or how can it be proven that execution of code wasn't tampered with. In this paper we focus on technological aspects. We discuss the various possibilities of security within the virtualization layer and we use as a case study *Hello Rootkitty*, a lightweight invariance-enforcing framework which allows an operating system to recover from kernel-level attacks. In addition to *Hello Rootkitty*, we also explore the use of special hardware chips as a way of further protecting and guaranteeing the integrity of a virtualized system.

I.   Introduction

Virtualization is the set of technologies that together allow for the existence of multiple running virtual machines on-top of a single physical machine. While initially all of the needed mechanisms for virtualization were created in software, the sustained popularity of virtualization, lead to their implementation in hardware, providing the desired speed that was lacking in their software counterparts. Today virtualization is gaining more and more interest in IT as well as the business world. The ability of virtualization to easily consolidate and migrate virtual machines between physical machines, allows corporations to outsource their IT infrastructure while reducing the cost of maintenance, power consumption and required infrastructure. This practice is called *cloud computing*. While virtualization is now mostly a technology for server farms, it is expected that it will soon affect both the mobile device and desktops market. Companies are already offering products such as VMWare's Mobile Virtualization platform (Barr et al., 2010) and Citrix's XenDesktop which utilize virtualization to increase productivity, decrease costs and allow easier maintenance of mobile and desktop devices in corporate environments.

From a security point of view, researchers have already identified a number of attractive properties of virtualization that can be used to provide stronger security guarantees in server, desktop as well as mobile environments. The property that has received the most attention is the guaranteed isolation between virtual

*IBBT-DistriNet, Katholieke Universiteit Leuven, 3001 Leuven, Belgium

machines on top of the same physical machine and the isolation of the code managing the virtual machines, called the hypervisor, from the virtualized operating systems. This makes the hypervisor and separate virtual machines ideal locations for security measures. A strong isolation between the layer where a security mechanism resides and the layer which it protects allows the security mechanism to continue to operate correctly even in the presence of an attack against the protected layer. A number of virtualization-utilizing security mechanisms have been proposed, ranging all the way from rootkit detectors to anti-virus products (Chiueh et al., 2009; Gadaleta et al., 2011, 2009; Qub, n.d.). Unfortunately virtualization-utilizing security measures are usually affected by consistent overhead since they typically require interactions between the hypervisor and the virtualized operating system that would not be present in traditional environments.

In this paper we discuss two very active research tracks. First, one track focuses on the application of virtualization techniques to increase the security of the overall system against malicious software. As a use case we describe *Hello Rootkitty*, an in-hypervisor invariance-enforcing framework used to detect kernel-level rootkits. While *Hello Rootkitty* significantly elevates security of the overall system without requiring any hardware or software modification, it is not able to guarantee isolation of sensitive information against a determined attacker.

Second, we discuss recent research results of another research track to provide formally provable security for small, specially tailored software modules. These strong security guarantees come at a significant cost of partitioning applications in security sensitive and insensitive parts. These results build on the technology of the Trusted Platform Module (TPM), a low-budget chip that is currently shipping with newer computers that provides a limited number of security features in hardware.

The rest of the paper is structured as follows. Section II. discusses how virtualization can be used to rethink security from the ground-up and use it to develop strong defenses for modern computing. Section A. explores *Hello Rootkitty* and shows how a hypervisor can be used as an invariance enforcing framework in order to identify malicious and unexpected modifications in a kernel's data structures. Section B. presents the technology of TPMs and discusses how this hardware chip can be employed to offer provable security, followed by our conclusions in Section III..

II.   Rethinking security

In order to take advantage of cloud computing in a corporate environment, a strong isolation between workloads of different parties is required. Modern operating systems, such as Microsoft Windows Server 2008, are not able to

provide sufficient isolation between different applications, or even between applications and the operating system itself. According to the National Institute of Standards and Technology (NIST), that keeps track of all reported vulnerabilities of commercial available software packets, there were 128 new vulnerabilities found in Windows Server 2008 in 2010 alone (*National Vulnerability Database (NVD) CVE Statistics*, 2011). For 2011, this figure was already surpassed at the time of writing. The size of the kernel, ranging into millions lines of code, make it infeasible to make it reliably secure. Subtle bugs(One, 1996) can be exploited by an attacker to gain kernel-level access. As this is the most privileged level, sensitive information stored by another party's applications can now be easily accessed.

Virtualization techniques, allow us to create another, even more privileged, layer. Using this additional layer, research has focused on two distinct approaches. First, the layer can be used to offer stronger protection of the operating system running on top of it without any need to modify any source code or binary. Second, security measures can be implemented in this layer to protect the execution and isolation of code even in the presence of malware. While the latter approach is able to offer stronger security guarantees, it requires a significant modification of source code.

A.   Hello Rootkitty: an invariance-enforcing framework

Rootkits are pieces of malicious software deployed on a compromised operating system with the chief purpose of concealing the presence of other malicious applications (such as a keylogger, or a backdoor) from the users and administrators of that system. The two most common rootkit classes are user-mode and kernel-mode and essentially signify the privilege level where the rootkit resides. User-mode rootkits have a relatively limited impact on the system because they compromise a single application at a time and can be easily detected and removed by security mechanisms residing either in the userspace or the kernel-space of the operating system. Kernel-mode rootkits however, are much more insidious, with a higher impact on the system and harder to detect and remove.

A countermeasure deployed within the same layer of the system that it protects might be circumvented and is susceptible to attacks. No isolation can be guaranteed if the countermeasure that protects and the kernel that is to be protected, are both part of the attack surface. *Hello Rootkitty* takes advantage of isolation provided by virtualization in order to protect a target kernel and mitigate the problem of rootkits. We assume that a rootkit can be introduced in a system through a Loadable Kernel Module (LKM) [1], by overwriting mem-

---

[1]LKMs are regularly used to install new hardware or extend the kernel with new features. Unfortunately, an inexperienced user can easily install a malicious LKM which masks itself as a benign

ory directly via kernel-exposed interfaces, or by exploiting a vulnerability in the kernel that allows execution of arbitrary code. A common characteristic of most rootkits is that they overwrite locations in memory in order to change the control-flow inside the kernel. The majority of these locations have values that do not change during normal execution. Thus, any sign of variance can be used to detect the presence of rootkits. We name these target memory areas as invariant "critical kernel objects" because compromising such locations is essential to change the control-flow of the kernel and execute injected code.

The literature provides methods to detect invariant critical kernel objects as described in Baliga et al. (2010); Dolan-Gavitt et al. (2009); Carbone et al. (2009); Wang et al. (2009). These methods differ depending on the type of kernel object. We identify three types of objects:

1. Static kernel objects at addresses hard-coded and not dependent on kernel compilation

2. Static kernel objects whose addresses depend on kernel compilation

3. Dynamic kernel objects allocated on the kernel heap via kernel-specific memory allocation functions

Once the locations of invariant kernel objects have been collected, *Hello Rootkitty* can check their integrity regardless of their type. The minimal information required to enable protection without dealing with false positives is the address of the object within guest memory and its size in bytes.

Part of *Hello Rootkitty* is a trusted module which operates in the guest operating system at boot time and provides such information to the hypervisor. We consider boot time our root of trust. This is a realistic assumption especially in the case of production servers, such as mail and web servers, in which the environment does not change after their installation. After the first boot, the system is considered to operate in an untrusted environment and integrity checking will be enforced by the hypervisor.

A schema of *Hello Rootkitty* is provided in Figure 1. Given the list of invariant kernel objects, the trusted module sends this data to the hypervisor via a hypercall[2]. The hypervisor will checksum the contents at the provided addresses, store the computed hashes to a private memory area, not accessible by the guest, and will force the trusted module to unload. After this point, an attacker can no longer tamper with the countermeasure: the trusted module is not part of the attack surface and isolation between the guest and the hypervisor is guaranteed by virtualization-enabled hardware.

---

application.

[2]Virtual machines communicate to the underlying hypervisor via hypercalls, the equivalent of system calls used by regular processes to communicate with the underlying kernel.
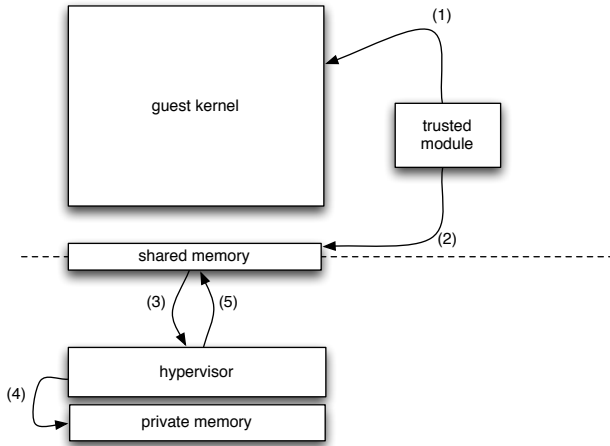
Figure 1: **High level view of trusted module-hypervisor interaction in *Hello Rootkitty* framework**

Integrity checking is needed to detect if any of the protected objects has been compromised. In order to detect such changes, the hypervisor needs to access the contents within the guest, compute their hash and finally compare it against the hash stored in its private memory area.

Hello Rootkitty executes checking by taking into account the regular interaction of the hypervisor and the guest running on top. In a virtualized environment the guest runs on a logical processor in a privilege level lower than the virtualized machine, called VMX non-root mode. In this mode certain instructions or events triggered by the guest kernel will cause a VMExit and control is given to the hypervisor. The hypervisor will handle the exception and return to the guest upon termination. We found that writing to control registers[3] is the most convenient event to check the integrity of kernel objects. This event is strategic because with virtual addressing enabled, Control Register 3 (CR3) becomes the page directory base register. On Intel architecture switching between two running processes will change CR3. This gives direct information about the current guest system load and allows to implement a countermeasure that scales accordingly.

When the hypervisor detects that the signature of a protected object does not match the one computed the first time, the system will report an ongoing attack. *Hello Rootkitty* is also able to repair the compromised object, if a copy of it has

---

[3]MOV CR* for Intel x86 Architecture

been provided by the trusted module.

Executing integrity checking outside of the target operating system can have consistent cost that affects performance overhead and limit the deployment in production systems. Moreover, the number of critical kernel objects to be protected is usually high and checking the integrity of the entire list could make the system unusable. Thus, integrity checking of a large list of objects is spread over a certain number of events trapped by the hypervisor. This problem relaxation considerably improves the performance overhead, although it comes at a cost in terms of security and detection time. However, the detection ability of this countermeasure remains strong also in a realistic scenario.

While performance benchmarks show negligible overhead and the memory footprint is proportional to the number of protected objects, *hello Rootkitty* has some limitations. Since it depends on invariance inference engines to provide an accurate list of invariant critical kernel objects, *hello Rootkitty* will be unable to detect attacks that occur in the non-reported ones. Moreover, this framework will not enable protection of objects whose values can legitimately change during regular usage. However, protecting a consistent amount of invariant kernel objects will dramatically reduce the attack surface and make the overall system more secure against rootkits.

B.   Offering provable secure isolation

While *Hello Rootkitty* significantly elevates the security of the overall system, it is not able to guarantee complete isolation of applications. Malware could still successfully exploit subtle bugs in the kernel by trying to restore invariants before they are checked or avoiding breaking invariants altogether. For at least a certain period of time, the malware may be able to access sensitive information of applications running on behalf of other parties.

An alternative research track attempts to protect sensitive information even in the presence of kernel-level malware. Security measures have been developed(Azab et al., 2011; Garfinkel et al., 2003; McCune et al., 2008, 2010; Singaravelu et al., 2006; Strackx et al., 2010) to offer such strong security guarantees by splitting applications in a security sensitive and a security insensitive part. The former part is executed in complete isolation from the rest of the system. This minimizes the size of the trusted computing base (TCB), the sum of all software that is relied upon to isolate sensitive information and calculate the correct result, up-to a point that it becomes feasible to formally verify(Jacobs and Piessens, 2008) the correctness of code. This proves with mathematical certainty that sensitive information will never leak to another party. While these security measures are able to provide very strong security guarantees, they are no longer binary-compatible with legacy applications. A significant effort is required to partition these applications in an security sensitive and insensitive

part. Moreover, it does not provide any availability guarantees. While malware is unable to access or modify sensitive information, it still can, for example, cause the system to freeze, preventing all parties to execute any application.

1. Root of trust

Accepting the presence of kernel-level malware causes significant problems to establish a root of trust. Malware may already have infected the system before the security measure is applied. Hence, the malware could influence the security measure's behavior or disable it altogether. For example, the kernel image stored on the hard drive, could have been modified to include the malicious code. To mitigate this problem, a low-cost security chip, called the Trusted Platform Module (TPM)(Trusted Computing Group, 2004) has been developed and is already shipped with most modern computers. Equipped with a slow but cheap processor and its own memory, it can be used to execute a fixed set of security-related tasks. To protect the chip itself from software attacks, it is shipped with all required software that under no circumstance can be modified.

The TPM chip is designed for a few specific tasks. First, it is able to record all software that is loaded on the system. Starting at power up, a measurement of the software is calculated and stored on the chip. Every time a new process is loaded, this measurement is extended with a measurement of the loaded software including the used configuration files. This is called a *Static Root of Trust Measurement* (SRTM). Similarly, a new measurement can be started after the system has already booted, called a *Dynamic Root of Trust Measurement* (DRTM). Second, the TPM chip is able to store a very limited amount of data, called *sealed storage*. On storage, the data is supplied together with a measurement. Only when software with this specific measurement is loaded, can the data be retrieved again. Finally, the chip is able to attest to a third party that a specific version of software has executed and outputted the specified results. Using cryptographic functions, it can prevent malware from making false claims such as specifying a different output.

Using a combination of the features directly provided by the TPM chip, strong security guarantees can be provided. Flicker(McCune et al., 2008), takes this approach. Applications are divided into security sensitive modules. Upon invocation, a new dynamic root of trust measurement is started, measuring the loaded module. This will also place the machine in an isolated state. During that time, only the module itself has control over the machine. Malware, possibly already present on the system, will not be executed. Hence, it is unable to access any sensitive information used by the module. After the module finished its execution, it will clear all sensitive information from memory and resume normal operation of the system. When information needs to be stored for other invocations of the same or other modules, it uses the sealed storage feature of

the TPM chip.

While malware is not able to modify the binary image of the module, the module must still be trusted. Subtle bugs in the module itself, may be exploited by an attacker by providing unexpected parameters. As a result, sensitive information may not be completely overwritten after the module finished its execution or an incorrect result may be provided. Hence, there is still a need to formally verify that modules behave correctly (Jacobs and Piessens, 2008). Given the very limited code size, contrary to entire monolithic kernels such as Windows and Linux, this approach is feasible.

While offering strong security guarantees, the Flicker security measure still has significant drawbacks. First, isolating security-sensitive parts of an application in modules can be difficult. *All* accesses to sensitive information must be encapsulated in the module in such a way that the result does not reveal any information unknown to an attacker. This could be achieved by, for example, encrypting the provided result. Another difficulty is that security-sensitive functionality provided by the operating system can no longer be used by the module. Second, to reduce the cost of the TPM chip, it is equipped with a low-budget processor that is much slower than the main processor of the system. As a result, executing a module incurs a significant overhead. Moreover, to isolate the module, other code is prevented from being executed and the user can experience a momentarily freeze of the system.

2. Increasing flexibility

Recent security measures(McCune et al., 2010; Sahita R, 2009) are able to significantly reduce Flicker's drawbacks while offering the same strong security guarantees. Using virtualization techniques, an additional protection layer can be added. This hypervisor layer is even more privileged than kernel layer and can be used to protect against kernel-level malware. As usual, formal verification is required to avoid malware infecting this most privileged level.

TrustVisor is an example of a security architecture that uses virtualization techniques to offer strong isolation of code and data of sensitive parts of an application. When the system is booted, a hypervisor is installed. This code executing at the highest privilege level, has two purposes. First, it is responsible to maintain binary compatibility with legacy code. Given the number and diversity of operating systems and legacy applications, any security measure that requires even minor changes to legacy software are infeasible in practice. Only applications that require use of the newly offered security guarantees should require minimal modification.

Second, the hypervisor must protect memory regions used by itself or protected modules. To be able to guarantee isolation of modules, TrustVisor must prevent read and write access from malware to these memory regions from the

legacy operating system or applications. Similarly, protected modules that are possibly specially crafted by an attacker must not be able access other protected modules or the implementation of the security measure itself.

Using these properties, TrustVisor implements and protects software-based TPM's, called μTPMs, to significantly increase performance. When the system is booted, the hardware TPM chip measures all loaded software. At the first execution of the security measure, a long term secret is created for the μTPMs and sealed by the TPM. For subsequent boots, only when the security measure is loaded correctly, access to the long-term secrets is granted. The μTPMs use this long-term secret to safely provide secure storage and attestation functionality without accessing the slow hardware TPM. When a security-sensitive part of an application request TPM functionality, a μTPM is accessed instead. This reduces the overhead of accessing the hardware TPM chip to the cost of crossing the kernel-hypervisor border. Since μTPMs are executed on the main processor which is significantly more performant than the low-cost hardware TPM, performance evaluation of TrustVisor shows a significant speedup compared to Flicker.

III.   Conclusion

Recent years corporations have been looking at cloud computing as a convenient model to outsource their IT infrastructure. The key technology to enable this is virtualization. It allows the execution of several virtualized machines on the same physical hardware decreasing the cost of maintenance, power consumption and required infrastructure. Virtualization, when supported by hardware, also comes with important features that can be used to implement strong security measures such as isolation and a low performance overhead.

In this paper we described two distinct active security research tracks. First, as a use case, we presented *Hello Rootkitty*, a lightweight security measure that mitigates the problem of kernel level rootkits. Upon detection of malware *Hello Rootkitty* alerts the administrator of the virtual machine and, in some cases, proceeds to repair the compromised kernel. This security measure has been implemented within a hypervisor. Using virtualization techniques, it doesn't require any modification of the target kernel nor of any legacy user application. Moreover performance overhead is low enough to be applicable to a large spectrum of systems. Although it significantly elevates security, it can't guarantee full isolation of applications and can leave sensitive information unprotected.

Secondly, we described recent results of another active research track that focuses on provable protection of sensitive information. The core idea of security measures in this research field, is to partition target applications into a security sensitive and an insensitive part. This is achieved by modifying the target application according to a detailed application specific functional analysis. For

most use cases a significant effort is required. Moreover to achieve complete isolation from malware, a root of trust needs to be established by supported hardware.

Most countermeasures originating from these two research tracks are complementary. Strong isolation of sensitive information and availability guarantees can be provided, even when multiple, possibly malicious, parties are executing on the same platform, as is the case of cloud computing.

References:

ALEPH ONE (1996), 'Smashing the stack for fun and profit', *Phrack magazine* 7(49).
URL: http://www.phrack.com/issues.html?issue=49&id=14

ARATI BALIGA AND VINOD GANAPATHY AND LIVIU IFTODE (2010), 'Detecting kernel-level rootkits using data structure invariants'.

AZAB, A.M. AND NING, P. AND ZHANG, X. (2011), Sice: a hardware-level strongly isolated computing environment for x86 multi-core platforms, *in* 'Proceedings of the 18th ACM conference on Computer and communications security', ACM, pp. 375–388.
URL: http://www4.ncsu.edu/~amazab/SICE-CCS11.pdf

BARR, KEN AND BUNGALE, PRASHANTH AND DEASY, STEPHEN AND GYURIS, VIKTOR AND HUNG, PERRY AND NEWELL, CRAIG AND TUCH, HARVEY AND ZOPPIS, BRUNO (2010), The vmware mobile virtualization platform: is that a hypervisor in your pocket?, Vol. 44, ACM, New York, NY, USA, pp. 124–135.
URL: http://doi.acm.org/10.1145/1899928.1899945

B. JACOBS AND F. PIESSENS (2008), 'The VeriFast program verifier', *status: published* .
URL: http://people.cs.kuleuven.be/~bart.jacobs/verifast/verifast.pdf

CHIUEH, T. AND CONOVER, M. AND LU,M. AND MONTAGUE, B. (2009), Stealthy deployment and execution of in-guest kernel agents, *in* 'Proceedings of the Black Hat USA Security Conference'.

DOLAN-GAVITT, BRENDAN AND SRIVASTAVA, ABHINAV AND TRAYNOR, PATRICK AND GIFFIN, JONATHON (2009), Robust signatures for kernel data structures, *in* 'Proceedings of CCS '09'.

GADALETA, FRANCESCO AND NIKIFORAKIS, NICK AND YOUNAN, YVES AND JOOSEN, WOUTER (2011), Hello rootKitty: A lightweight invariance-enforcing framework., *in* X. Lai, J. Zhou and H. Li, eds, 'ISC', Vol. 7001 of *Lecture Notes in Computer Science*, Springer, pp. 213–228.

GADALETA, FRANCESCO AND YOUNAN, YVES AND JACOBS, BART AND JOOSEN, WOUTER AND DE NEVE, ERIK AND BEOSIER, NILS (2009), Instruction-level countermeasures against stack-based buffer overflow attacks, *in* 'Proceedings of the 1st EuroSys Workshop on Virtualization Technology for Dependable Systems', VDTS '09, ACM, New York, NY, USA, pp. 7–12.
URL: http://doi.acm.org/10.1145/1518684.1518686

GARFINKEL, T. AND PFAFF, B. AND CHOW, J. AND ROSENBLUM, M. AND BONEH, D. (2003), 'Terra: A virtual machine-based platform for trusted computing', *ACM SIGOPS Operating Systems Review* 37(5), 193–206.
URL: http://www.ittc.ku.edu/~kulkarni/teaching/archieve/EECS800-Spring-2008/terra.pdf

JONATHAN M. MCCUNE AND BRYAN PARNO AND ADRIAN PERRIG AND MICHAEL K. REITER AND HIROSHI ISOZAKI (2008), Flicker: An execution infrastructure for TCB minimization, *in* 'Proceedings of the ACM European Conference in Computer Systems (EuroSys)', ACM, pp. 315–328.
URL: http://www.ece.cmu.edu/~jmmccune/papers/mccune_parno_perrig_reiter_isozaki_eurosys08.pdf

JONATHAN M. MCCUNE AND YANLIN LI AND NING QU AND ZONG-WEI ZHOU AND ANUPAM DATTA AND VIRGIL GLIGOR AND ADRIAN PERRIG (2010), TrustVisor: Efficient TCB reduction and attestation, *in* 'Proceedings of the IEEE Symposium on Security and Privacy'.
URL: http://www.ece.cmu.edu/~jmmccune/papers/MLQZDGP2010.pdf

LENIN SINGARAVELU AND CALTON PU AND HERMANN HÄRTIG AND CHRISTIAN HELMUTH (2006), Reducing tcb complexity for security-sensitive applications: three case studies, *in* 'EuroSys '06: Proceedings of the 1st ACM SIGOPS/EuroSys European Conference on Computer Systems 2006', ACM, New York, NY, USA, pp. 161–174.
URL: http://www.cs.kuleuven.ac.be/conference/EuroSys2006/papers/p161-singaravelu.pdf

MARTIM CARBONE AND WENKE LEE AND WEIDONG CUI AND MARCUS PEINADO AND LONG LU AND XUXIAN JIANG (2009), Mapping kernel objects to enable systematic integrity checking, *in* 'In ACM Conf. on Computer and Communications Security'.

National Vulnerability Database (NVD) CVE Statistics (2011).
URL: http://web.nvd.nist.gov/view/vuln/statistics

Qub (n.d.), 'QubesOS: Architecture Specification', http://qubes-os.org/files/doc/arch-spec-0.3.pdf.

SAHITA R, WARRIER U., DEWAN P. (2009), 'Protecting Critical Applications on Mobile Platforms', *Intel Technology Journal* 13, 16–35.
URL: http://www.cs.unh.edu/~it666/reading_list/Hardware/intel_techjounral_security.pdf

STRACKX, R. AND PIESSENS, F. AND PRENEEL, B. (2010), 'Efficient Isolation of Trusted Subsystems in Embedded Systems', *Security and Privacy in Communication Networks* pp. 344–361.
URL: https://lirias.kuleuven.be/bitstream/123456789/277417/1/paper.pdf

TRUSTED COMPUTING GROUP (2004), 'Design Principles Specification Version 1.2', *TCG Specification* 5.
URL: http://www.trustedcomputinggroup.org/resources/tpm_main_specification

WANG, ZHI AND JIANG, XUXIAN AND CUI, WEIDONG AND NING, PENG (2009), Countering kernel rootkits with lightweight hook protection, *in* 'Proceedings of CCS '09'.