RootkitDet: Practical End-to-End Defense against Kernel Rootkits in a Cloud Environment

Lingchen Zhang[1], Sachin Shetty[1], Peng Liu[2] and Jiwu Jing[3]

College of Engineering, Tennessee State University[1]
College of IST, Penn State University[2]
Institute of Information Engineering,
Chinese Academy of Sciences[3]

Roadmap

- Introduction
- Threat Model and Assumptions
- Design of RootkitDet
- Implementation
- Evaluation

Introduction

- **Cloud Computing** enable ubiquitous access to data and applications
- Cloud security issues have gained traction due to vulnerabilities in low level operating systems and virtual machine implementations resulting in novel denial-of-service attacks [Liu, CCSW 2010]
- **Kernel-Level Rootkits** has the potential to inflict maximum damage and can launch stealth attacks, which can be difficult to detect or eliminate by the administrator.
- Need for an **efficient**, **scalable** and easy to **deploy** Kernel- rootkits defense system in the cloud

Introduction

- Several Kernel-level rootkit defense systems to protect hypervisors have been reported
- VMWatcher [Jiang, CCS 08] and Lares [Payne, S&P 08] constructing semantics views of target VM
- SBCFI [Petroni, CCS 08] and HookSafe [Wang, CCS 09] check kernel data structures to detect and prevent rootkits
- SecVisor [Seshadri, SIGOPS 07] and NICKLE [Riley, RAID 08] preserve kernel code integrity by preventing insertion of unauthenticated code in kernel space
- However, the aforementioned systems are designed to protect attacks on a single VM and are not suitable to protect VMs in the cloud.

Introduction

- **Problem Setting**
  - User accesses service provided by cloud user
  - Vulnerabilities exist in application service and/or guest OS
  - Attacker may gain root privilege, install a kernel-level rootkit to launch stealth attack on VM
- **Requirements of defense system**
  - End-to-end defense
  - Light-weight / low overhead
  - Scalable
  - Easy to use

Agenda

- Introduction
- Threat Model and Assumptions
- Design of RootkitDet
- Implementation
- Evaluation

Threat Model and Assumptions

- **Threat model**
  - **External attacker installs rootkit in the OS kernel managing VMs by exploiting zero-day vulnerabilities in kernel and application software in VMs**
  - **Attacker gains control over multiple VMs to steal confidential data, modify memory, etc**
- **Assumptions**
  - **CPU supports NX-bit(Non-executable) feature, and Linux kernel utilizes this feature for memory protection**
  - **Kernel-level rootkits insert code into kernel space**
  - **VMM is not affected by rootkits**

Agenda

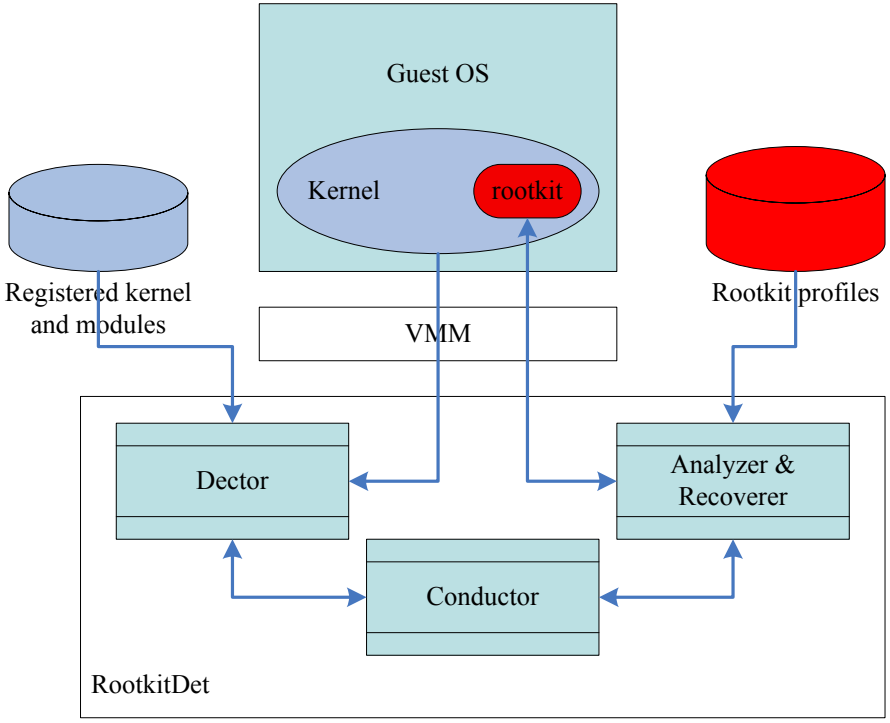
- Introduction
- Threat Model and Assumptions
- Design of RootkitDet
- Implementation
- Evaluation

Design of RootkitDet

- Overview
  - Detection
    - Registration procedures indicate legitimate code of guest OSes
    - Detection procedures find out suspicious code in the kernel space
  - Diagnosis
    - Perform static analysis on the code of rootkits to collect characteristic information
    - Categorize by matching to profiles of known rootkits
  - Recovery

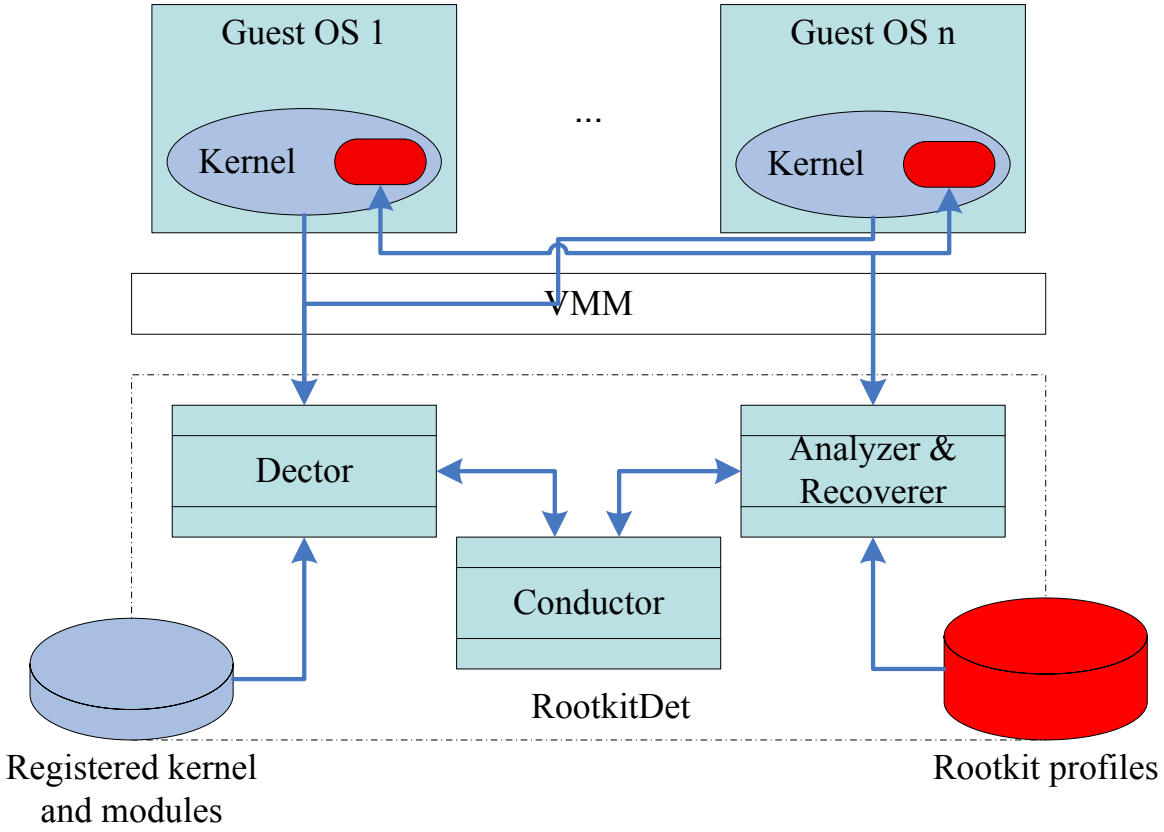
Design of RootkitDet

- **Basic Architecture**



Design of RootkitDet

- **Scalable Architecture**



Detection - Design of RootkitDet

- Registration procedures
  - Registration of kernel
  - Registration of loadable kernel modules (LKMs)
- Detection procedures
  - Detect extra executable regions in kernel space
  - Detect code residing in unused space of LKMs
  - Detect malicious modifications to legitimate code
- Challenges
  - Inconsistency of executable regions when LKM is unloaded
    - Kernel frees unused virtual memory area in a lazy manner
  - Module's code is variable due to the relocation
    - Relocation address and symbols of itself
    - Symbols of main kernel, even other modules

Diagnosis - Design of RootkitDet

- **Static analysis**
  - Characteristic information of detected rootkits
    - External function calls
    - Global variables and dynamically allocated objects accessed by the code
- **Categorization**
  - Profiles of known rootkits
    - The tactic adopted by the rootkit to achieve its intention
    - Data structures that are modified according to its tactic
  - Detail the profiles
    - Translate symbols into addresses according to the running kernel

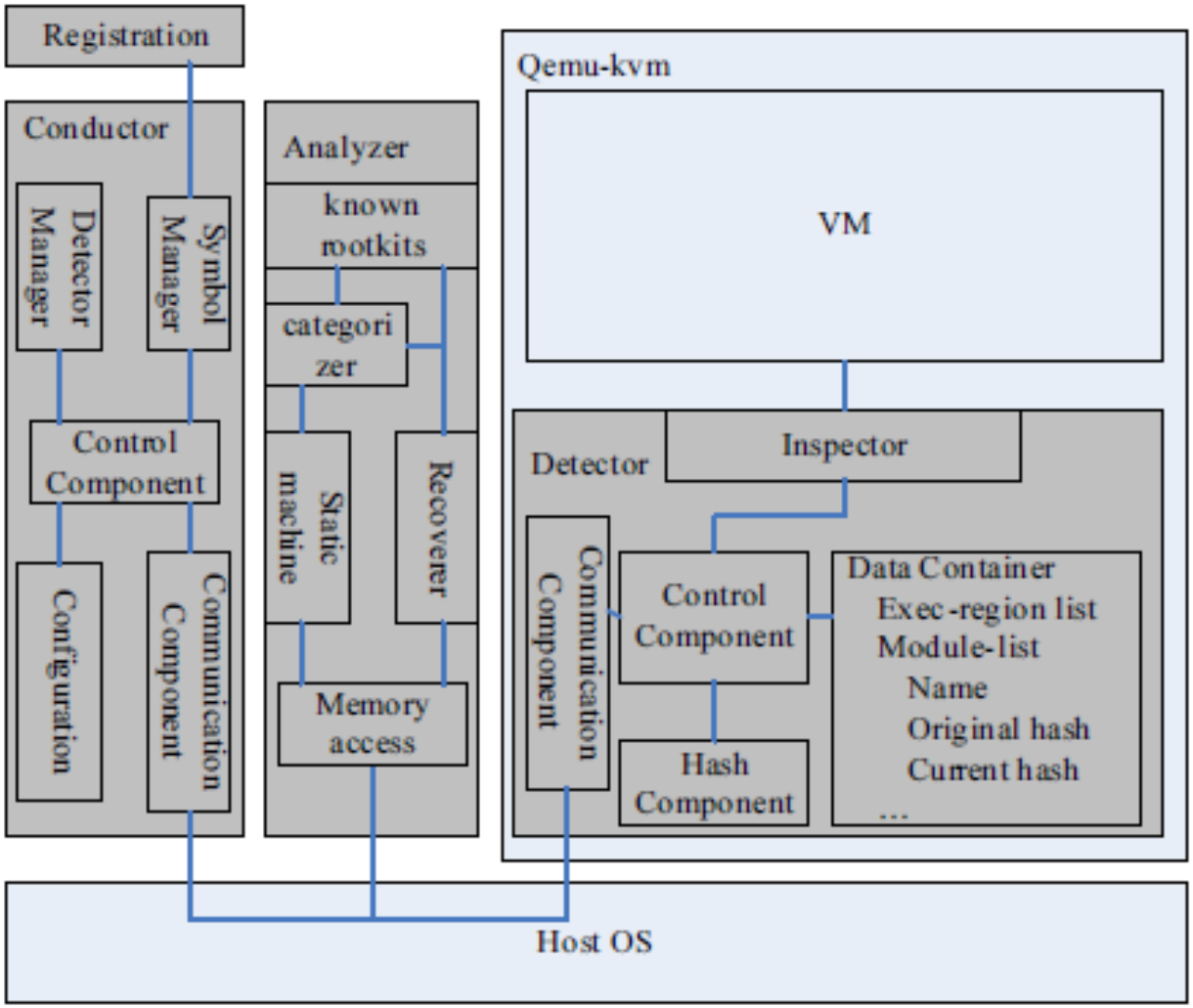
Recovery – Design of RootkitDet

- **Recovery-driven profile**
  - Derived from the profile of the rootkit and meta-data of the running kernel
- **Recovery of control data**
  - Expected values are constant and known
  - Tracking down from some global variable
- **Recovery of non-control data**
  - Expected values can be inferred if logical relations among non-control data and other objects in the kernel

Agenda

- Introduction
- Threat Model and Assumptions
- Design of RootkitDet
- Implementation
- Evaluation

Implementation



Implementation

- Conductor
  - Schedule detector and analyzer
  - Handle I/O
    - Connection from a new guest OS to be monitored
    - Response of detector
    - Configuration change
  - Sleep

Implementation

- **Detector –** *Inspector integrated*
  1. **Detect extra executable regions**
  2. **Detect code in unused space of kernel modules**
  3. **Detect modifications to the code of kernel and modules**

Implementation

- **Analyzer**
  - **Independent program**
  - **Scheduled by Conductor**
  - **Perform static analysis and categorize the detected rootkit**
  - **Perform recovery**

Agenda

- Introduction
- Threat Model and Assumptions
- Design of RootkitDet
- Implementation
- Evaluation

Evaluation

- **Effectiveness**

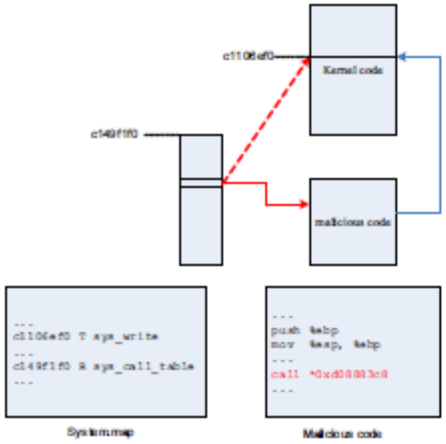


Fig. 4. hksc: hooking sys_write

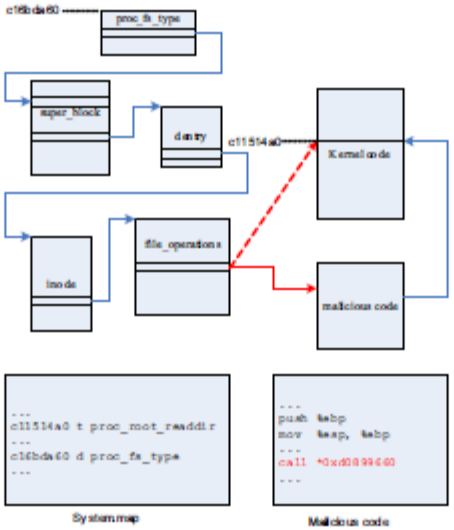


Fig. 5. hkproc: hooking proc filesystem

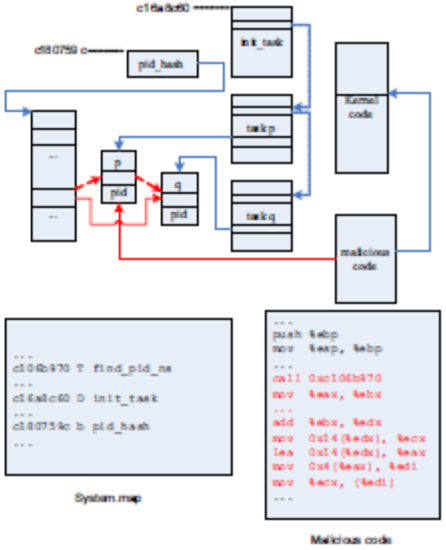


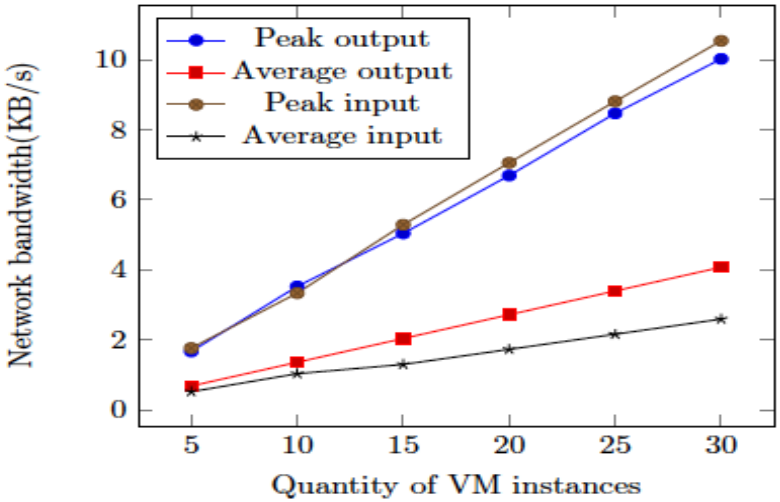
Fig. 6. hidepc: manipulating pid hash table

Evaluation

Detector overhead

Detection Procedure	Time consumed/ us
1	189
2	713
3	47139

Network overhead



Kernel-level rootkit detection

Rootkit	Method to insert code	DP
adore-ng	module	1
enyelkm	module and substitution	1, 3
icmp-cmd	executable region	1
icmp-cmd_v2	unused space	2

Evaluation

- **Overhead to guest OSes**

Table 2. Application-level benchmarks of overhead to guest OSes

Benchmark	W/o Performance	W/i Performance	Relative Performance
Dhrystone	6040580.1 lps	6045164.7 lps	1.001X
Whetstone	630.6 MIPS	629.9 MIPS	0.999X
Lmbench(pipe bandwidth)	3843.2 MB/s	3810.3 MB/s	0.991X
Apache Bench(throughput)	569.95 KB/s	568.67 KB/s	0.998X
Kernel decompression	21.343 s	21.529 s	0.991X
Kernel build	1300.4 s	1292.9 s	1.001X

- **Micro-benchmark of RootkitDet**

Table 3. Time of detection and recovery

Rootkit	Code size(byte)	detection time(ms)	analysis time(ms)	recovery time(ms)
hksc	407	< 1	14.6	3.7
hkproc	978	< 1	44.6	7.7
hidepc	565	< 1	29.1	204.8

Conclusion and Future Work

- **Conclusion**
  - Presented RootkitDet system, an efficient, scalable and easy to deploy kernel-level rootkit detection system in cloud
  - RootkitDet leverages the page directory of the kernel space in the guest OSes and the monitor functions provided by the VMM in the cloud detect rootkits
  - Experimental evaluation show that the RootkitDet system can effectively detect all of the kernel-level rootkits that insert code into kernel space with performance cost of less than 1%.
- **Future Work**
  - Migrate infected VM into QEMU after detection of “alien” code pages and detect control data or non-control data modifications

References

- H. Liu, ”A new form of DOS attack in a cloud and its avoidance mechanism,” CCSW, 2010
- Bryan D Payne, Martim Carbone, Monirul Sharif, and Wenke Lee. Lares: An architecture for secure active monitoring using virtualization. S&P 2008.
- Nick L Petroni Jr and Michael Hicks. Automated detection of persistent kernel control-flow attacks. ACM CCS 2008
- Ryan Riley, Xuxian Jiang, and Dongyan Xu. Guest-transparent prevention of kernel rootkits with vmm-based memory shadowing. In Recent Advances in Intrusion Detection, pages 1–20. Springer, 2008.
- Arvind Seshadri, Mark Luk, Ning Qu, and Adrian Perrig. Secvisor: a tiny hypervisor to provide lifetime kernel code integrity for commodity oses. In ACM SIGOPS 2007
- Zhi Wang, Xuxian Jiang, Weidong Cui, and Peng Ning. Countering kernel rootkits with lightweight hook protection, ACM CCS 2009
- Xuxian Jiang, Xinyuan Wang, and Dongyan Xu, Stealthy malware detection through vmm-based out-of-the-box semantic view reconstruction. ACM CCS 2007

Thank You and Questions