Scalable Intrusion Detection System in the Cloud

Sachin Shetty

Assistant Professor

Electrical and Computer Engineering

Tennessee State University

Collaborators

Lingchen Zhang (TSU), Peng Liu (PSU) and Mohan Malkani (TSU)

Heterogeneous VM Replication: A New Approach to Intrusion Detection, Active Response, and Recovery in Cloud Data Centers

Mohan Malkani, Sachin Shetty and Peng Liu

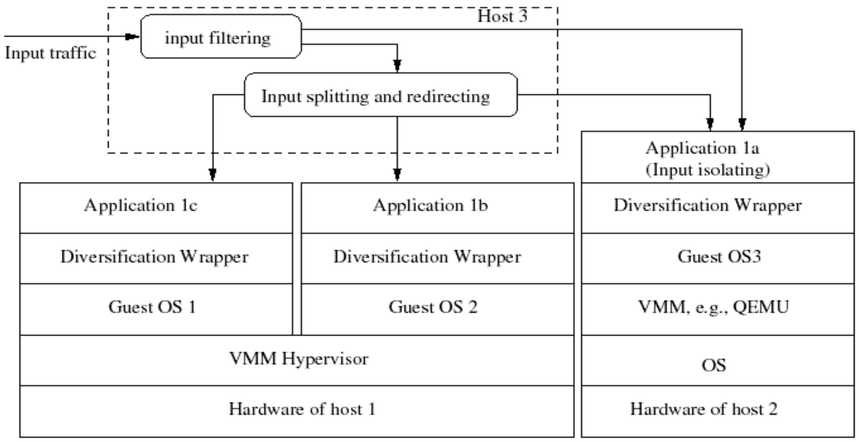
Project Goal

- Proactive defense techniques needed to combat against emerging and evolving cyber threats against clouds.
- This research focuses on development of a Moving Target Defense technique, H-VM-R (Heterogeneous VM Replication), to detect and prevent intrusion attempts in cloud data centers.

Research Approach

- Develop H-VM-R based intrusion detection approach by comparing heterogeneous VM images resulting from same execution history
- Develop cost-effective response by proactively setting up standby VM replicas
- Develop migration technique to move compromised VM replica to clean yet heterogeneous VM replica.

H-VM-R Architecture



Work Accomplished

- Developed and implemented Heterdevice system prototype to detect resource starvation attacks from compromised drivers (RAID 2012)
- Developed and implemented kernel-level rootkit detection system for cloud environment (in submission)
- Established cloud data center testbed at TSU comprising of 50 Dell PowerEdge M610 servers running OpenStack and Hadoop distributed computing platform

Kernel-Level RootKit Detection System in Cloud

- 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 detection system in the cloud

Kernel-Level RootKit Detection System in Cloud

- **Several Kernel-level rootkit detection 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 detection systems are designed to protect attacks on a single VM and are not suitable to protect VMs in the cloud.**

Kernel-Level RootKit Detection System in Cloud

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

Kernel-level Rootkit Classification

Name	Hijack Tech	Insertion Tech
adore-ng	Dynamic: inode_ops	Module
enyelkm	Code: system_call	Module
kbeast	Static: sys_call_table	Module
knark	Static: sys_call_table	Module
mood-nt	Static: sys_call_table	/dev/kmem
override	Static: sys_call_table	Module
phantax	Static: sys_call_table	/dev/mem
suckit v2	Static: sys_call_table	/dev/mem

Kernel-level Rootkit Classification

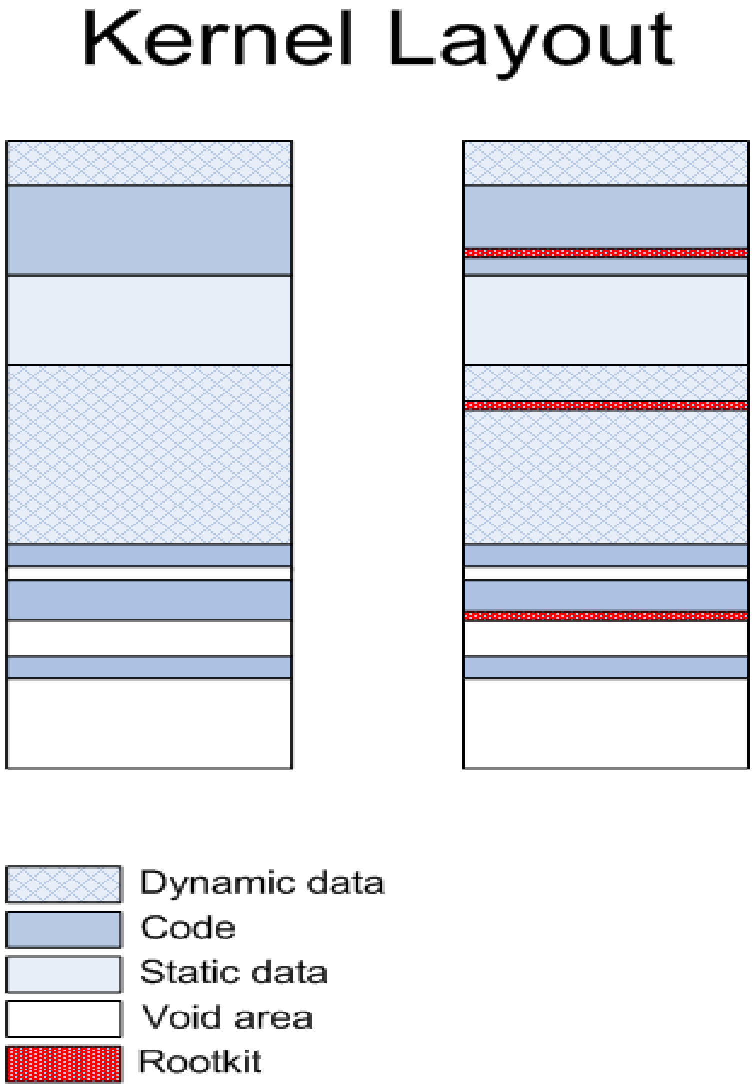
- Classification according to hijacking of control flow
  - Rootkits modifying kernel code
  - Rootkits modifying static data
    - Static data locates at the same address
    - Static data used by rootkits are usually read-only
  - Rootkits modifying dynamic data
    - Dynamic data used by rootkits are usually function pointers to kernel data structures
- Common characteristic
  - Code inserted into kernel space

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

Design of RootkitDet

- **Rootkit Location**
  - Code of Kernel/Module
  - Dynamic allocated area
  - Unused space of Modules
- **Characteristics**
  - Related page entries are marked executable (NX-bit is cleared)



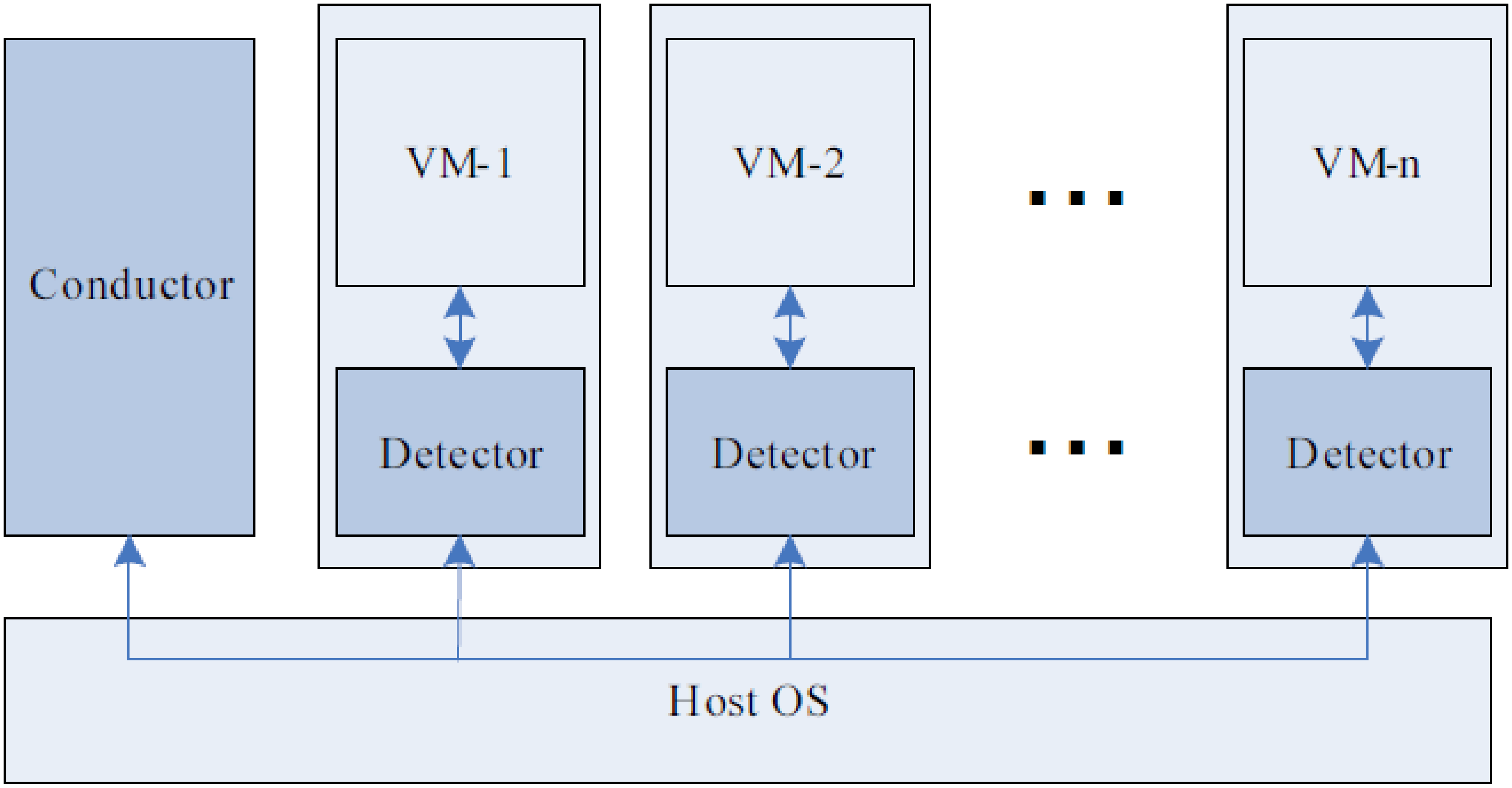
Design of RootkitDet

- Requirements
  - Extra executable regions in the kernel space
    - Read page directory entries of VM
  - Code in unused space of kernel modules
    - Read unused space of all kernel modules
  - Modifications of code of kernel and modules
    - Calculate hash values of code of kernel and modules
    - Original and current
- Challenges
  - Inconsistency of executable regions when kernel module 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

Design of RootkitDet

- Overview of RootKitDet
  - Memory access of VMs
    - Take advantages of VMM, read memory of VM when it is suspended
  - Modification of Linux kernel
    - Free virtual memory areas after a module is unloaded
    - This won’t affect the efficiency of the kernel
  - Generation of original hash values of kernel and modules
    - Original filesystem image of VMs
    - Do relocation as what the kernel does to load a module

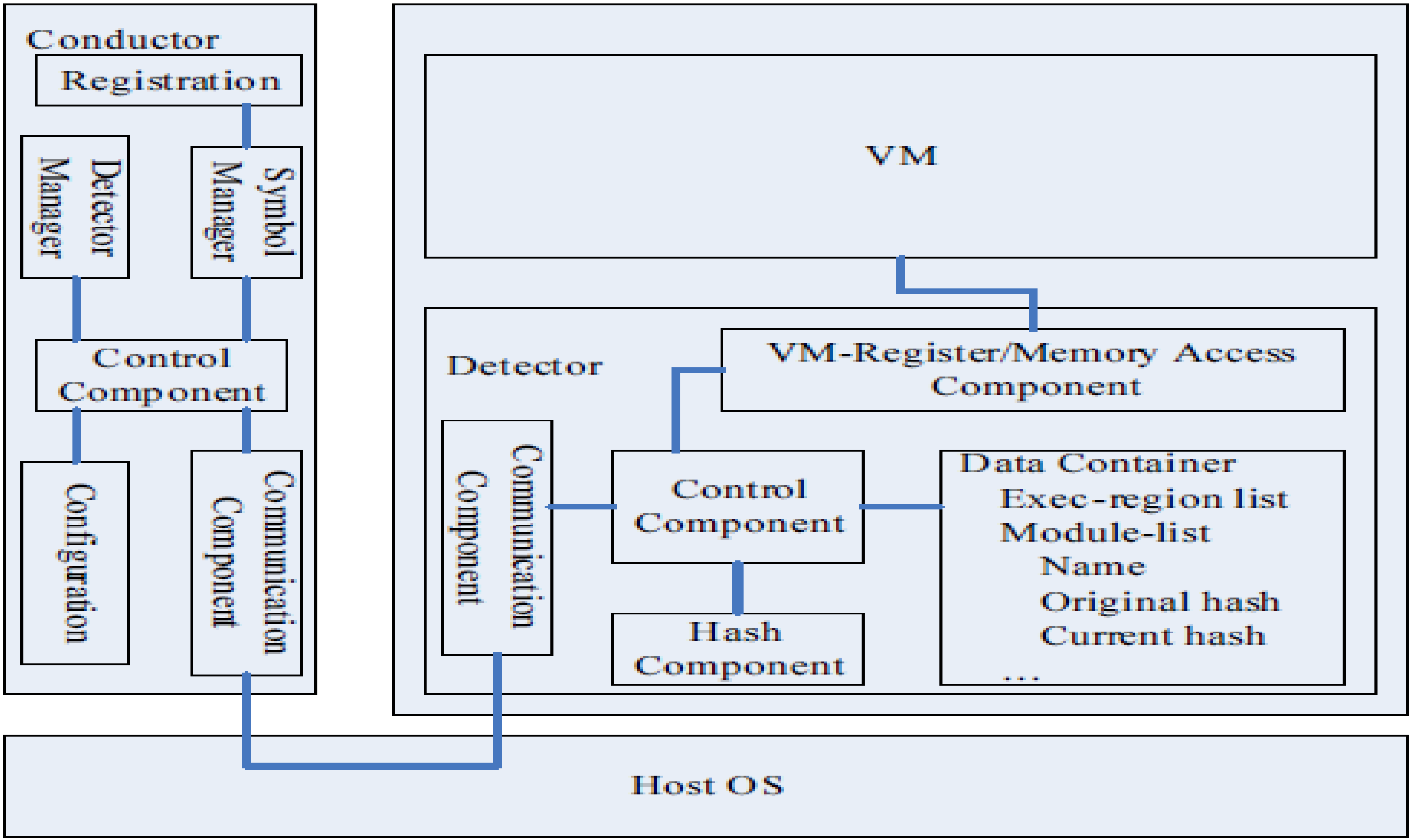
Design of RootkitDet



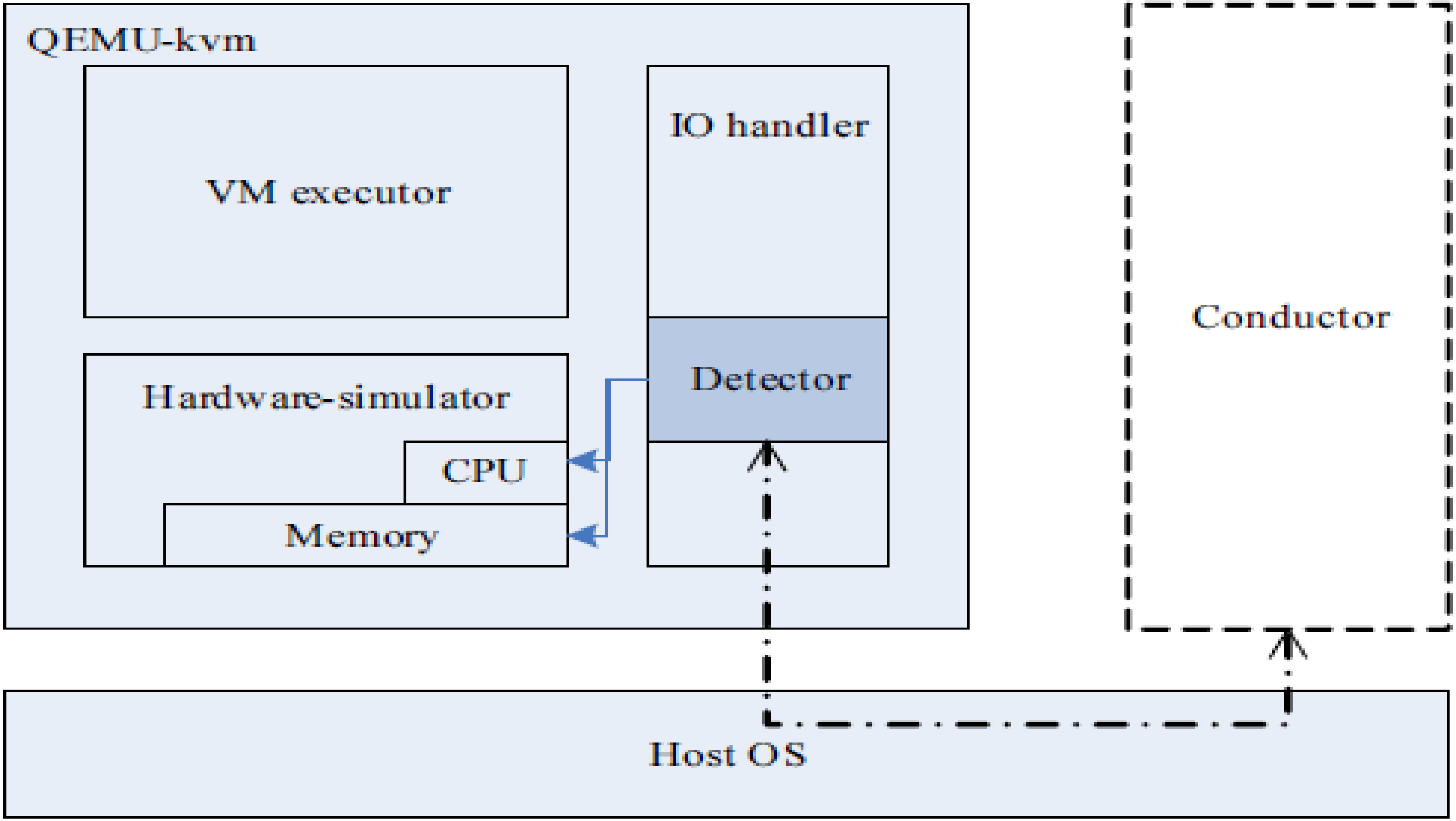
Design of RootkitDet

- Overview
  - Detector
    - Integrated into VMM
    - Detection of extra executable regions
    - Detection of code in unused space of kernel modules
    - Detection modification of code and kernel and modules
  - Conductor
    - User process independent of VMs
    - Communicate with Detector: get modules list, set original hash values
    - Calculate original hash values when necessary
    - Conduct the procedure of detection
    - Serve multiple Detectors

Design of RootkitDet

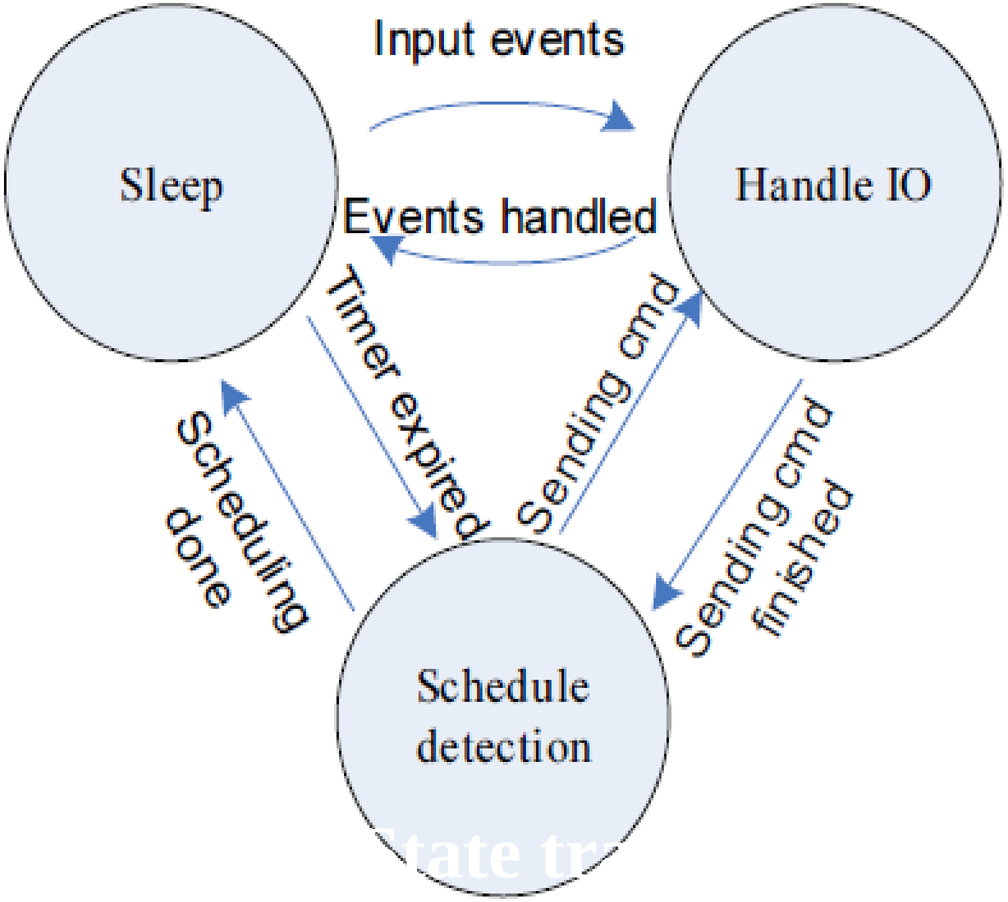


Implementation



Implementation

- **Conductor**
  - Schedule detector
  - Handle I/O
    - New connection
    - Response of detector
    - Configuration change
  - Sleep



Implementation

- **Detection procedures**
  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

Algorithm 1: Detection Procedure 1

```
Data: No input data
Result: Rootkit_is_found

Rootkit_is_found = false;
list1 = get_executable_regions();
sort_by_address(list1);
list2 = get_kernel_and_modules();
sort_by_address(list2);
while list1 and list2 are both not empty do
    element1 = list1.top();
    element2 = list2.top();
    list1.remove_top();
    list2.remove_top();
    if element1 not match element2 then
        Rootkit_is_found = true;
        break;
    end
end
if list1 or list2 is not empty then
    Rootkit_is_found = true;
end
return Rootkit_is_found;
```

Implementation

Algorithm 2: Detection Procedure 2

Data: No input data
Result: Rootkit_is_found

```
Rootkit_is_found = false;
list = get_modules();
while list is not empty do
    element = list.top();
    list.remove_top();
    if unused_space_is_not_zero(element) then
        Rootkit_is_found = true;
        break;
    end
end
return Rootkit_is_found;
```

Algorithm 3: Detection Procedure 3

Data: No input data
Result: Rootkit_is_found

```
list = get_kernel_and_modules();
while list is not empty do
    element = list.top();
    list.remove_top();
    element.calculate_hash_value();
    if element.new_hash not match element.original_hash then
        Rootkit_is_found = true;
        break;
    end
end
return Rootkit_is_found;
```

Evaluation

- **Kernel-level rootkit detection**

Rootkit	Way to insert code	Detection Procedure
adore-ng	module	1
enyelkm	module and substitution	1/3
icmp-cmd	executable region	1
icmp-cmd_v2	unused space	2

- **Overhead of Detector**

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

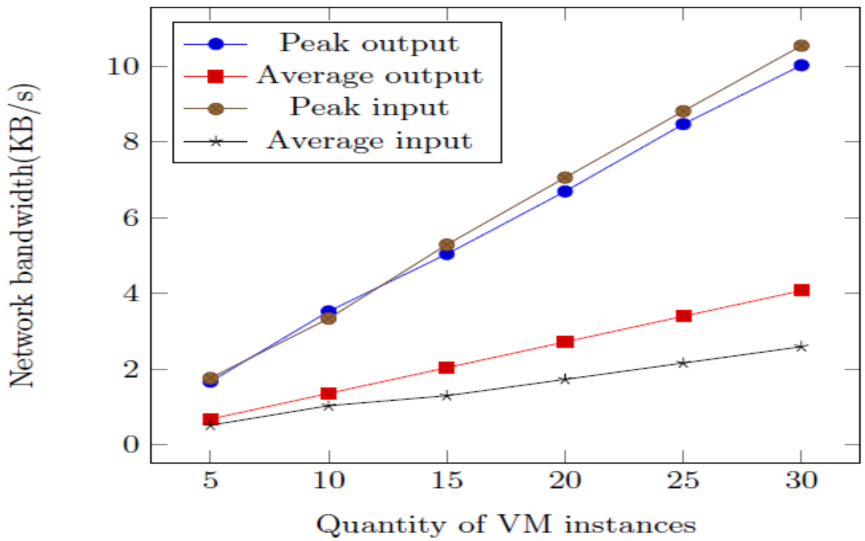
Evaluation

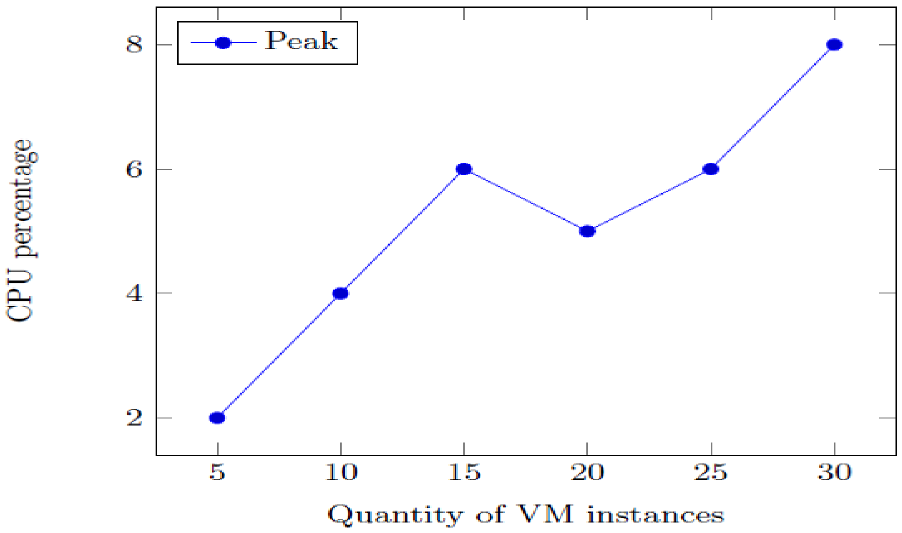
- **Overhead of Detector**
  - **Application benchmarks**

Benchmark	W/o Performance	W/i Performance	Relative Performance
Dhrystone	6040580.1 lps	6045164.7 lps	100.1%
Whetstone	630.6 MIPS	629.9 MIPS	99.9%
Lmbench(pipe bandwidth)	3843.2 MB/s	3810.3 MB/s	99.1%
Apache Bench(throughput)	569.95 KB/s	568.67 KB/s	99.8%
Kernel decompression	21.343 s	21.529 s	99.1%
Kernel build	1300.4 s	1292.9 s	100.1%

Evaluation

- **Conductor Performance**





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

Publications

- Shengzhi Zhang, et. al., "Assessing the trustworthiness of Drivers", in Proceedings of the 15th International Symposium on Research in Attacks, Intrusions and Defenses (RAID) 2012.
- Lingchen Zhang, Sachin Shetty, Peng Liu, Mohan Malkani, " A Scalable Kernel-Level Rootkit Detection System in the Cloud", in submission.

Thank You and Questions