End to End Defense against Rootkits in Cloud Environment

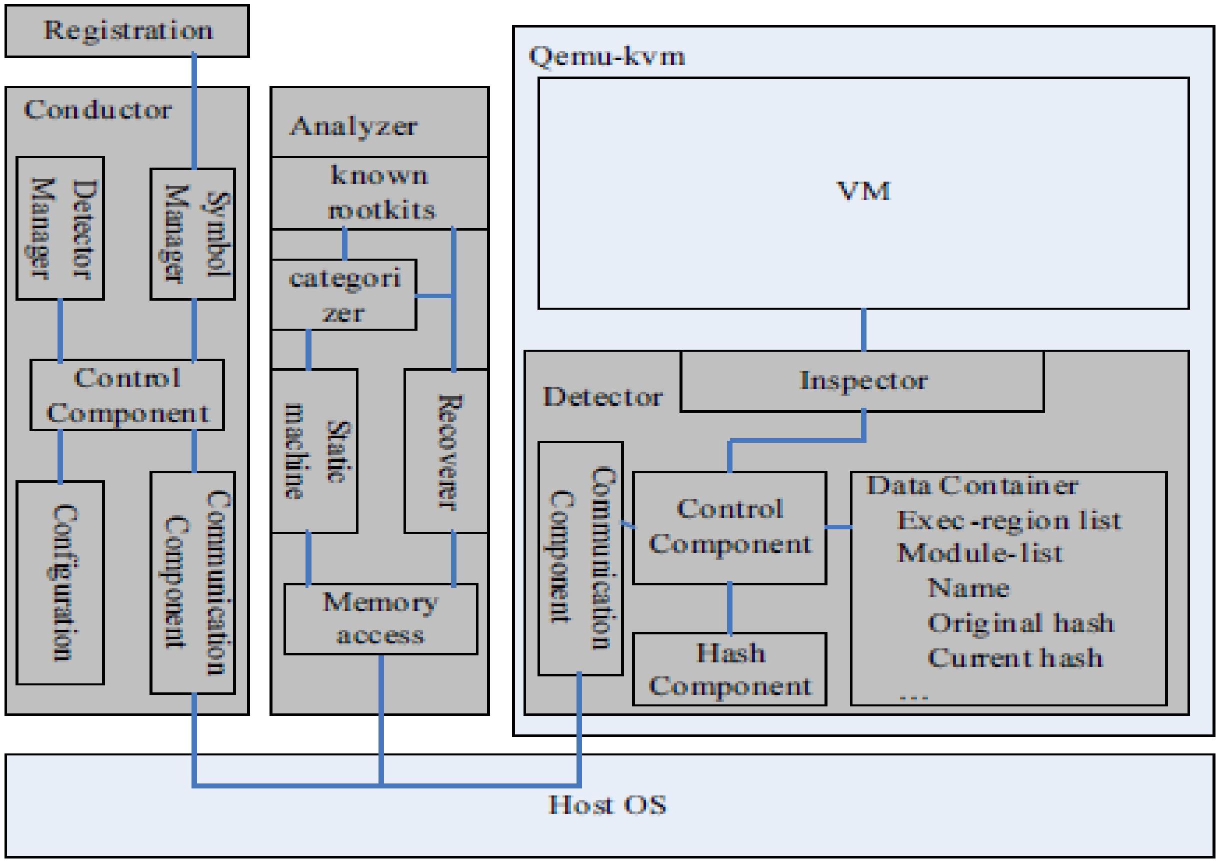
Implemenation and Evaluation

Sachin Shetty

Associate Professor
Electrical and Computer Engineering
Director, Cybersecurity Laboratory
Tennessee State University

Implementation

We use qemu-kvm-1.2.0 for creating instances of the guest OS, and compile linux-2.6.32.60 for the guest kernel



Implementation

- Detector.
  - We integrated the detector into qemu-kvm because the guest Oses are running as user processes on the host OS and the integration reduces interprocess communications.
  - Five components: inspector, data container, hash component, control component and communication component.
  - Inspector is responsible for reading the registers and memory of the VM.
  - Data container component constructs the necessary semantic data structures from the raw data of the VM's memory given by inspector according to the profile, and also stores data coming from the conductor through communication and control components.

Implementation

- Hash components is used for calculating the current hash values for the kernel and modules’ code.
- The communication component takes care of all of the communication with the conductor.
- The control component receives commands through the communication component from the conductor, then executes the commands and sends the response back to the conductor.

Implementation

- The conductor periodically schedules detectors for monitored guest OSes and initiates analyzer when rootkits are detected.
- Generates original checksums of registered modules
  - Performs the same relocation work as the guest kernel does.
- The correct relocation work of a module depends on
  - the original object file, the relocation address, the addresses of the used kernel symbols and the addresses of the used symbols of other modules.
- The conductor acquires the original object file of a module from the registration component and obtains its relocation address from the detector

Implementation

- The conductor can figure out the address of a kernel symbol by referring to the meta-data of the kernel.
- We create a database storing the relative addresses of symbols exported by registered modules.
  - The conductor can calculate the absolute address of a symbol exported by a module by looking it up in the database
- To resolve dependency among modules during the relocation, the conductor calculates original checksums after collecting the relocation addresses of all loaded modules from the detector.
- Consequently, the conductor can generate original checksums for all of the loaded modules and send them back to the detector.

Implementation

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

Figure 5: State transition of Conductor

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

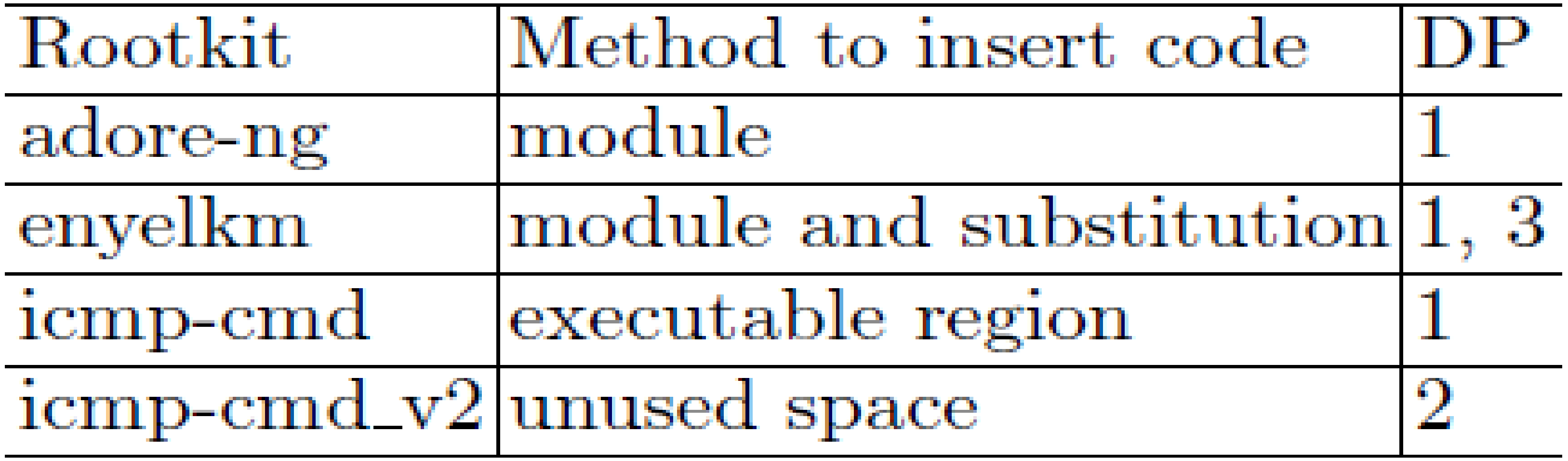
Implementation

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

Evaluation

- Evaluate RootkitDet's effectiveness for detecting kernel-level rootkits that compromise the code integrity of the OS kernel and recovering modified data
- Measure the overhead introduced to guest OSes and extra resources consumed by RootkitDet.
- Experiments are conducted on Dell PowerEdge M610 Server (2.40GHz Intel Xeon E5645 and 6GB memory.
  - The hypervisor is qemu-kvm-1.2.0 and host OS is Ubuntu-12.04.
  - We used Debian-squeeze with kernel version 2.6.32 as our guest OS.
  - Detector is integrated into qemu-kvm, and thus runs with the guest OS and conductor ran on another computer as a user process.

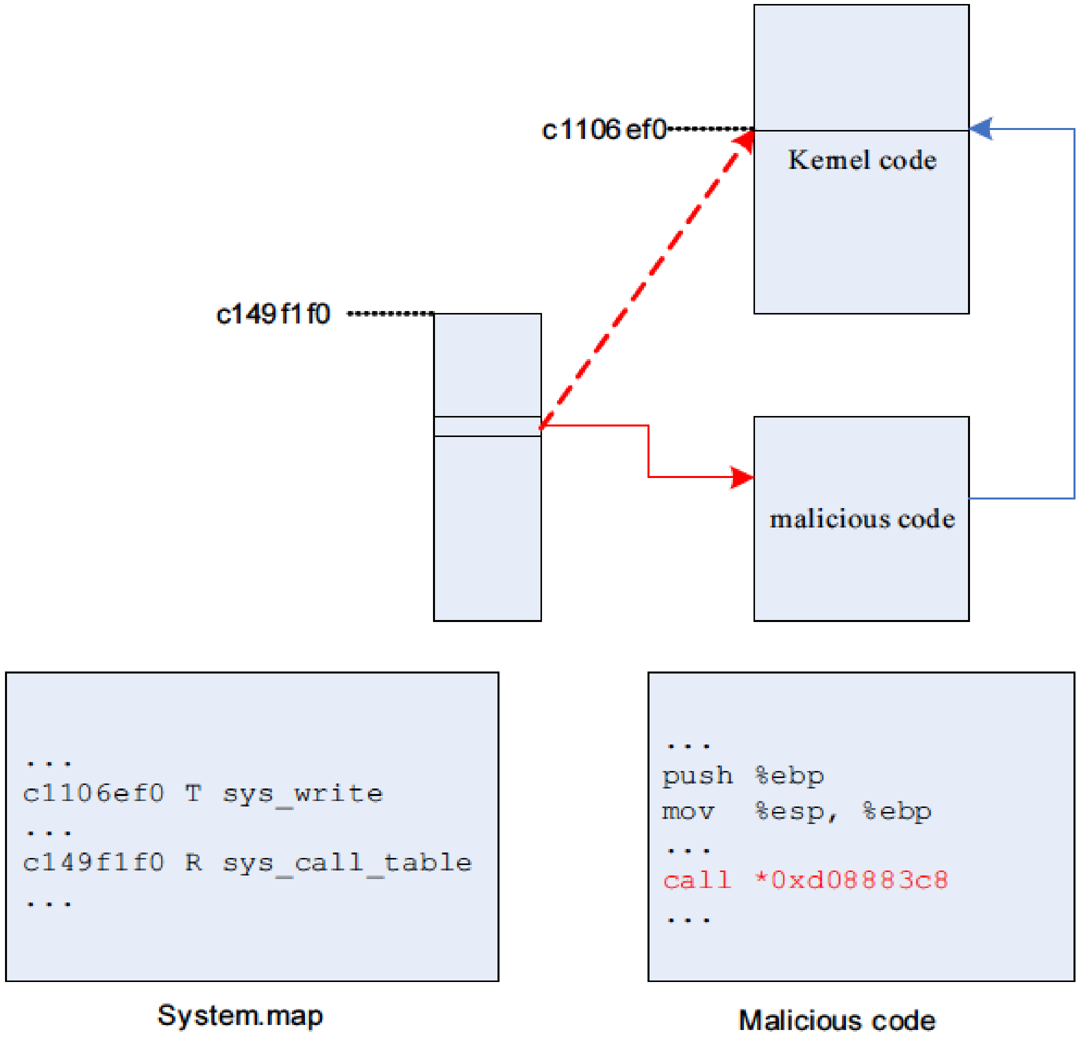
Evaluation



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

- Hijacking sys write system call to hide a specific process by tampering with what is displayed to the administrators of guest OSes.
- We recover the modifie system call table to eliminate the effect of this rootkit

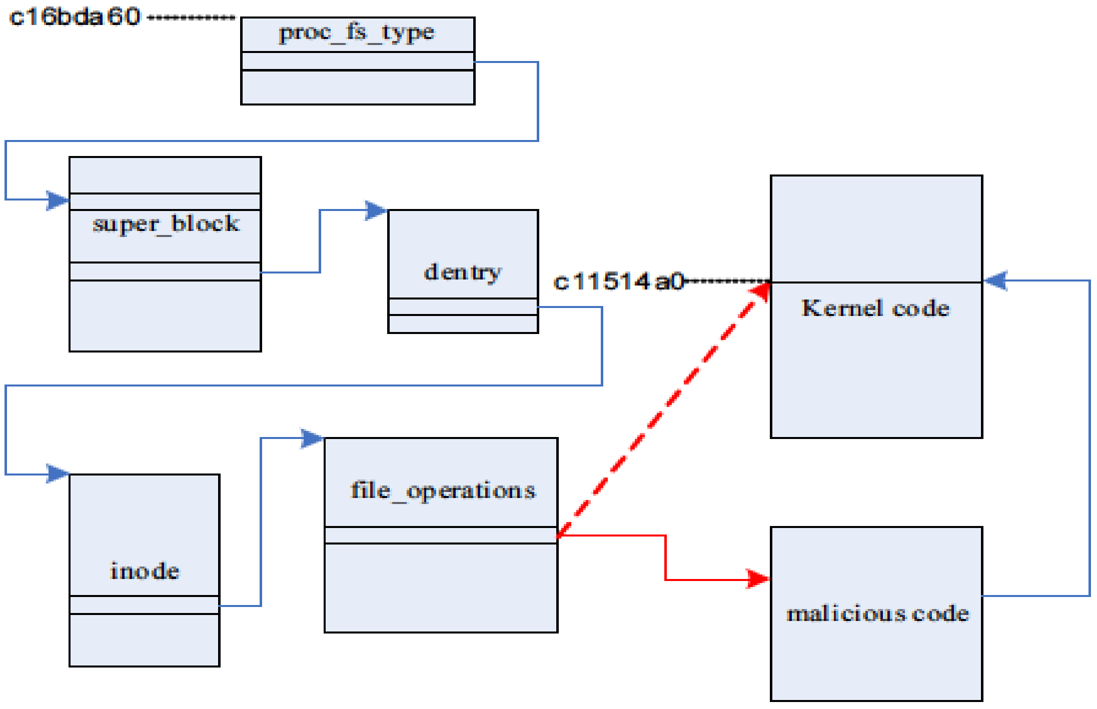


```
...
c1106ef0 T sys_write
...
c149f1f0 R sys_call_table
...
```

```
...
push %ebp
mov  %esp, %ebp
...
call *0xd08883c8
...
```

Evaluation

- Hooking the function pointer *proc root readdir* to hide a specific process by removing related pid entry in the proc file system.
- We find the hooked function pointer by tracking down from *proc fs* type , which is a global variable, and correct it with the real location of kernel function *proc root readdir*



```
...
c11514a0 t proc_root_readdir
...
c16bda60 d proc_fs_type
...
```

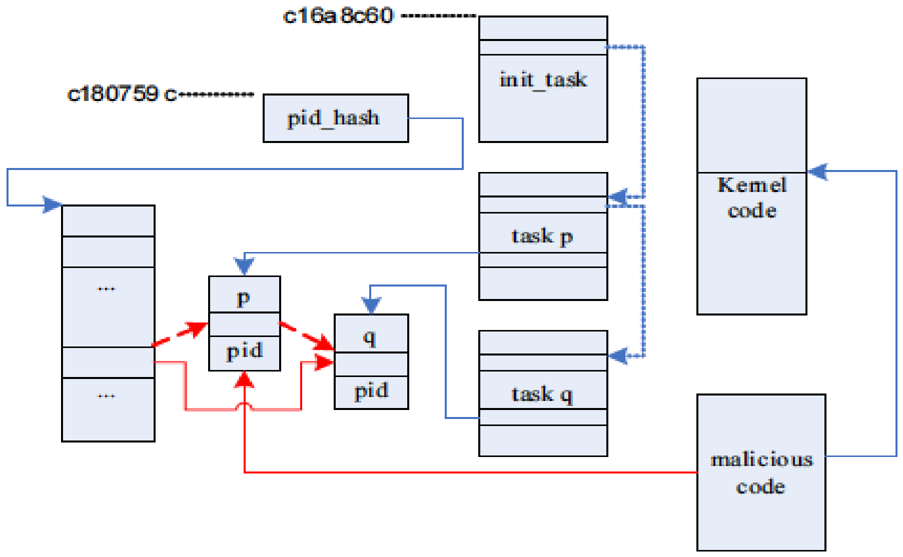
System.map

```
...
push %ebp
mov  %esp, %ebp
...
call *0xd0899660
...
```

Malicious code

Evaluation

- Manipulating pid hash table
- Hiding a specific process by removing related entry in the pid hash table.
- We first find the task struct of the hidden process by tracking down from init task , and then relink it into the pid hash table to reveal the hidden process.



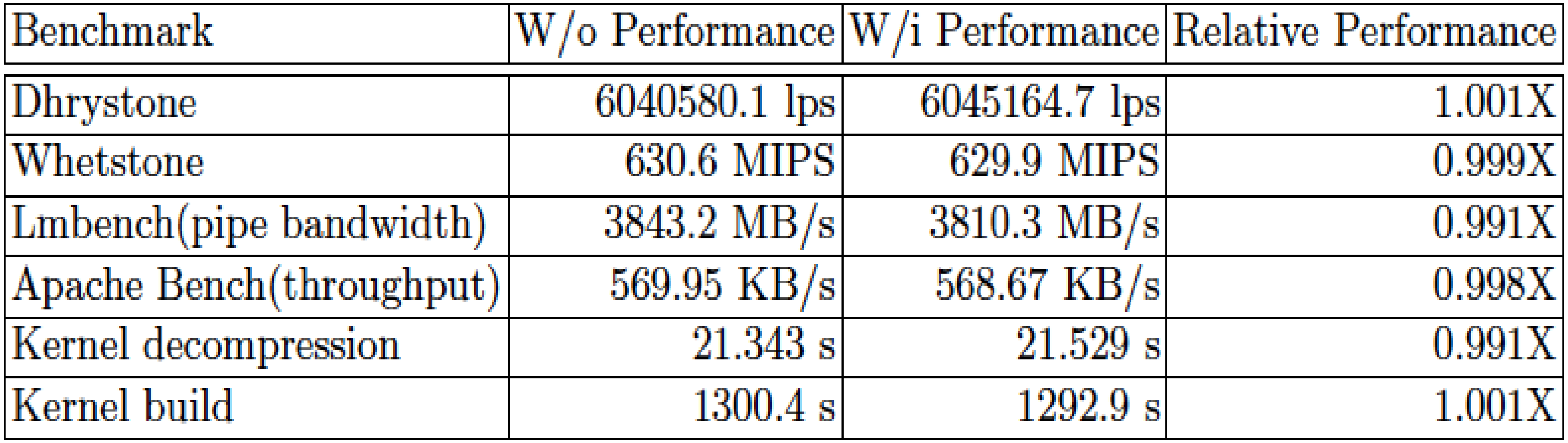
```
...
c106b970 T find_pid_ns
...
c16a8c60 D init_task
...
c180759c b pid_hash
...
```

System.map

```
...
push %ebp
mov  %esp, %ebp
...
call 0xc106b970
mov  %eax, %ebx
...
add  %ebx, %edx
mov  0x14(%edx), %ecx
lea  0x14(%edx), %eax
mov  0x4(%eax), %edi
mov  %ecx, (%edi)
...
```

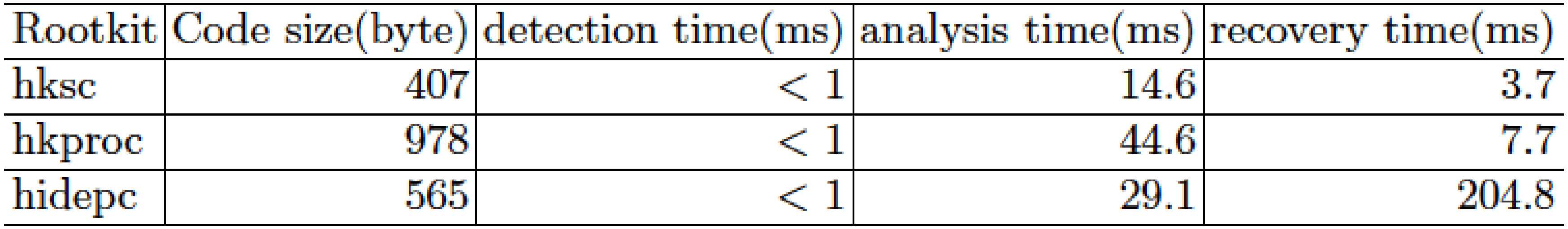
Malicious code

Evaluation



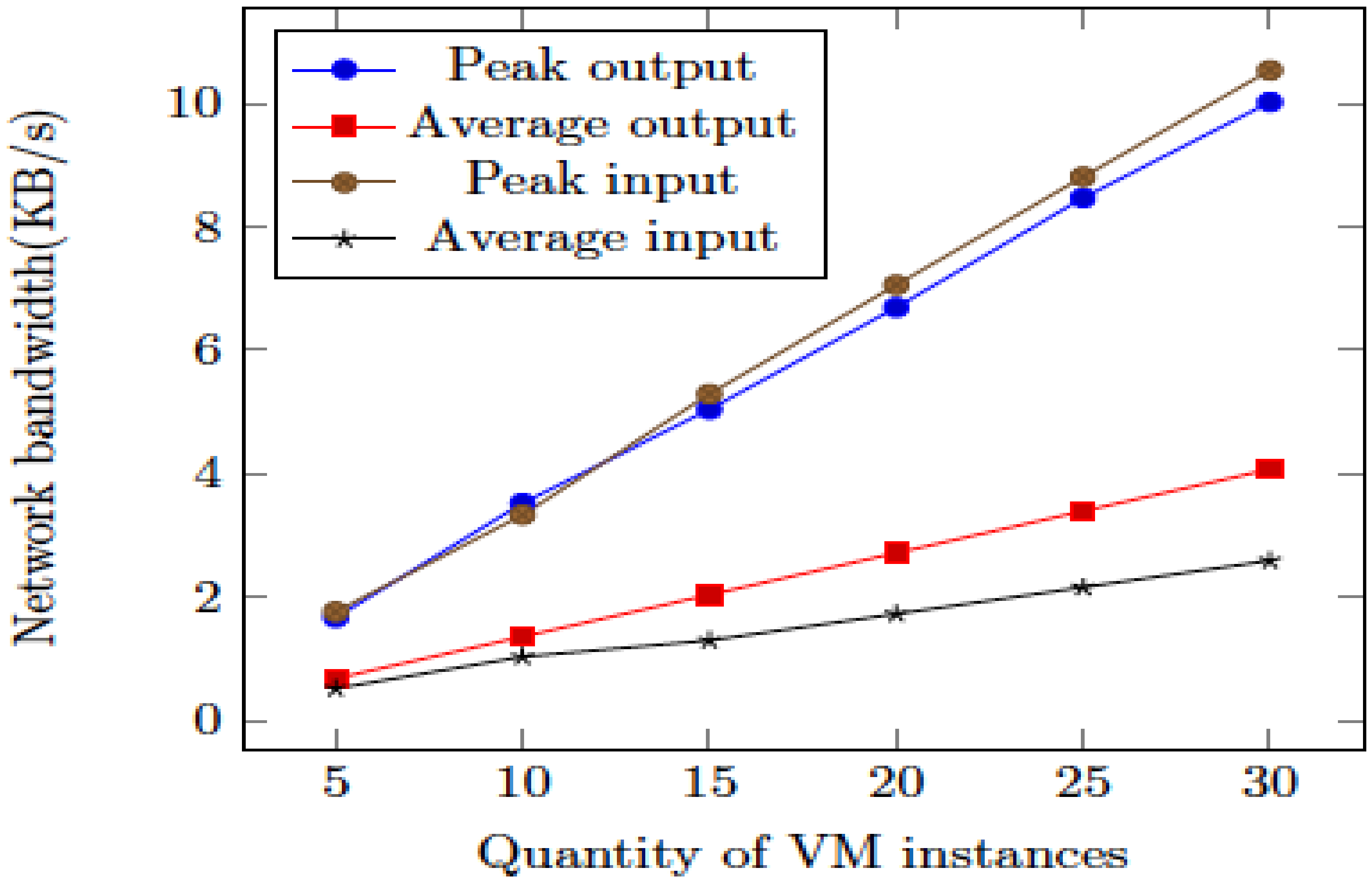
Benchmark	W/o Performance	W/i Performance	Relative Performance
Dhrystone	6040580.1 lps	6045164.7 lps	1.001X
Whetstone	630.6 MIPS	629.9 MIPS	0.999X
Lmbench(pipe bandwidth)	3843.2 MB/s	3810.3 MB/s	0.991X
Apache Bench(throughput)	569.95 KB/s	568.67 KB/s	0.998X
Kernel decompression	21.343 s	21.529 s	0.991X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

Evaluation



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

Evaluation


Peak output
Average output
Peak input
Average input
Network bandwidth(KB/s)
0
2
4
6
8
10
5
10
15
20
25
30
Quantity of VM instances

Discussions

- RootkitDet system faces some limitations.
- First, it cannot detect rootkits that are erased immediately after executed or that have no specific code in the kernel space, like return-oriented rootkits.
- Second, it may not detect all of the code of a rootkit if the rootkit hides part of its code by switching NX-bit of the corresponding pages, therefore our system may lose some characteristic information of the rootkit during analysis.
- Third, it cannot prevent the installation of the kernel-level rootkits although it detects rootkits and recovers the kernel if possible.

Discussions

- Fourth, it cannot certainly recover all modifications made by the rootkits, especially when categorization of the rootkits fails.
- Finally, the generation of instinct information of rootkits are not automatic.

Discussions

- RootkitDet can perform quick detection of kernel-level rootkits by only issuing detection procedure 1 and 2 because almost all of kernel-level rootkits in the wild introduce extra code into the kernel space and fewer and fewer of them modify the code of the kernel or modules.
- RootkitDet system provides the characteristic information of unknown rootkits to assist further investigation.
- In future work, we can focus on the analysis and recovery of novel and unknown rootkits and automatic generation of rootkits’ instinct information.

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