NumChecker:

A System Approach for Kernel Rootkit Detection and Identification

Xueyang Wang, Ph.D.
<u>Xiaofei (Rex) Guo, Ph.D.</u>

(xueyang.wang || xiaofei.rex.guo) *noSPAM* intel.com

Disclaimer

We don’t speak for our employer. All the opinions and information here are our responsibility including mistakes and bad jokes.

Executive Summary

- Malware continues to proliferate
  - Increasing in number
  - Stealthier
- Traditional software-level detection mechanisms have limited effectiveness
  - Most of them relies on the correct functioning of OS
  - VMM-based approaches has semantic gap
  - Performance constraints
- A new solution: NumChecker
  - Analyzing software behaviors with rich hardware events
  - Low performance overhead
  - Focus on kernel rootkit

Agenda

- Kernel Rootkits
- Hardware Performance Counter
- NumChecker Design
- Kernel Rootkit Detection
- Kernel Rootkit Identification
- Conclusion

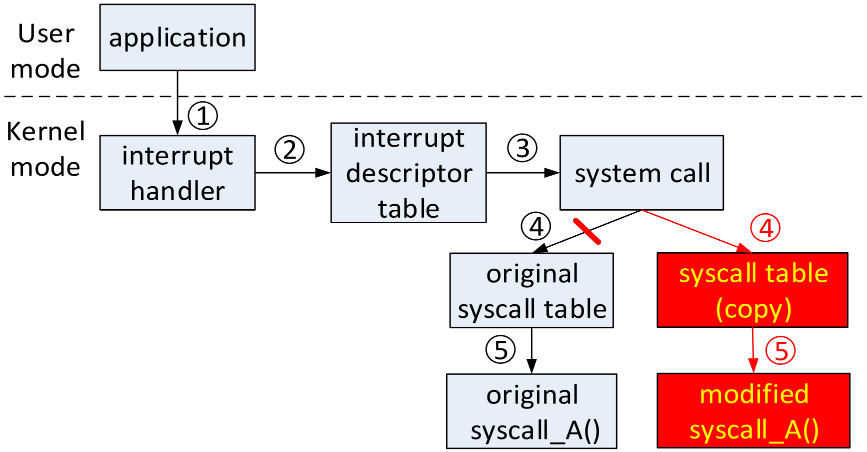
Kernel Rootkit

- Rootkit
  - Toolkits injected by attackers to hide malicious activities from the users and detection tools
- Kernel Rootkit
  - Rootkits that subvert the operating system kernel directly
  - Have unrestricted access to system resources
  - Used by attackers to hide their presence, open backdoors, gain root privilege, and disable defense mechanisms

Kernel Rootkit Behavior Classification

- Direct kernel object manipulation (DKOM)
  - Subvert the kernel by directly modifying data objects

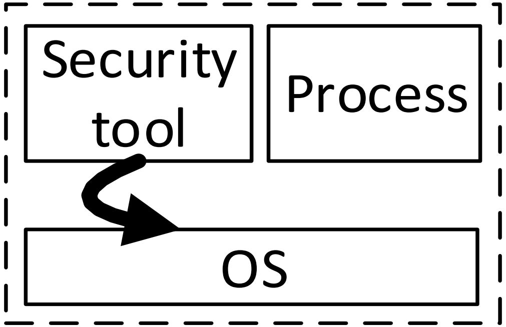
- Kernel Object Hooking (KOH)
  - Hijack the kernel control-flow
  - A majority of Linux kernel rootkits persistently violate control-flow integrity
  - Hijack the kernel static control transfers (e.g., SucKIT rookit)
  - Hijack the kernel dynamic control transfers (e.g., Adore-ng)



Known Kernel Rootkit Detection Approaches

Host-based rootkit detection

- Run inside the target they are protecting
- Check kernel static and dynamic objects



Known Kernel Rootkit Detection Approaches

Host-based rootkit detection

- Run inside the target they are protecting
- Check kernel static and dynamic objects

Challenges:
-Detection tools themselves might be tampered with by advanced kernel rootkits, which have high privilege and can access the kernel memory

Known Kernel Rootkit Detection Approaches

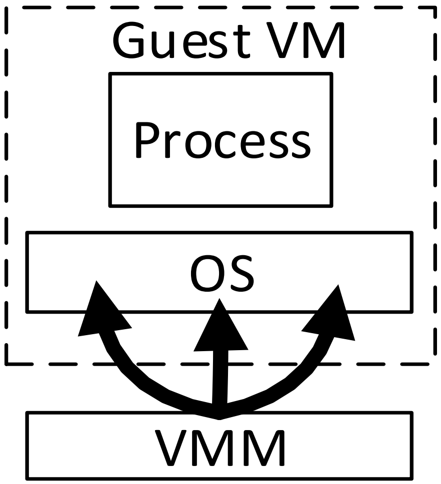
Host-based rootkit detection

- Run inside the target they are protecting
- Check kernel static and dynamic objects

Challenges:
-Detection tools themselves might be tampered with by advanced kernel rootkits, which have high privilege and can access the kernel memory

Virtual Machine Monitor (VMM) based rootkit detection

- Run at the VMM level
- Check kernel static and dynamic objects



Known Kernel Rootkit Detection Approaches

Host-based rootkit detection

- Run inside the target they are protecting
- Check kernel static and dynamic objects

Challenges:
-Detection tools themselves might be tampered with by advanced kernel rootkits, which have high privilege and can access the kernel memory

Virtual Machine Monitor (VMM) based rootkit detection

- Run at the VMM level
- Check kernel static and dynamic objects

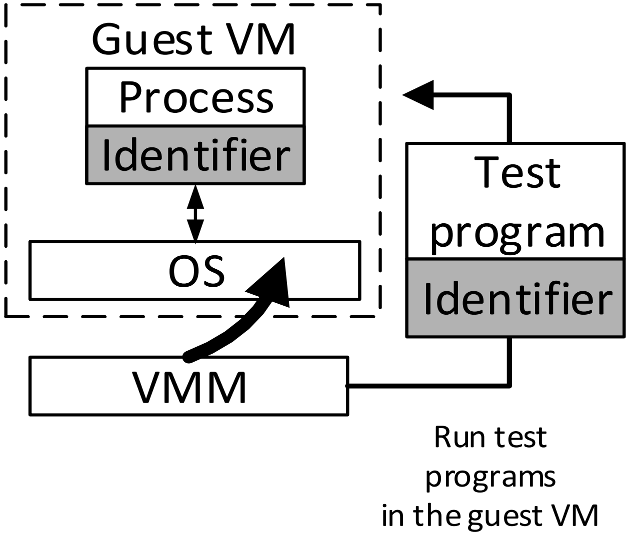
Challenges:
-“semantic gap” between the external and internal observation. The detection tools require detailed knowledge of the guest OS implementation

-Performance overhead

Hardware-Assisted Kernel Rootkit Detection

NumChecker: VMM-based kernel execution path checking using Hardware Performance Counters (HPCs)

- Runs at the VMM level
- Does not require detailed knowledge of the guest OS implementation
- Validates the execution path of guest system calls by checking the number of certain hardware events using HPCs

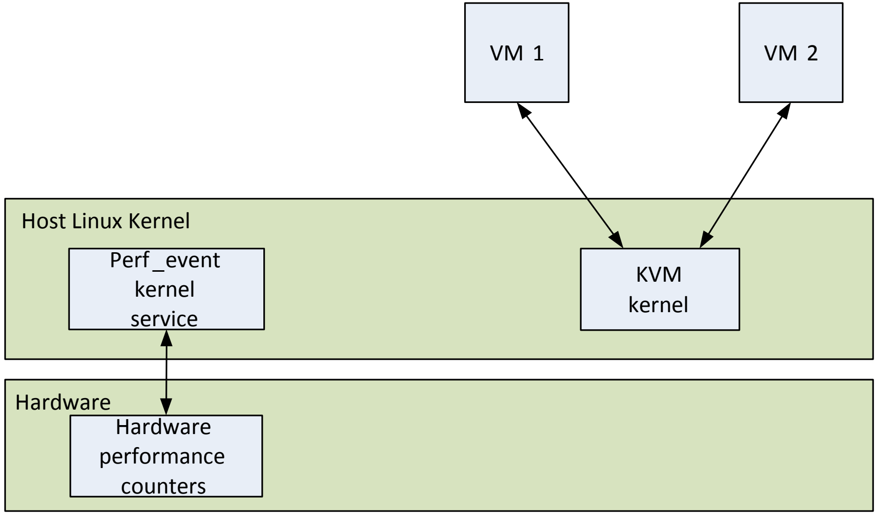


Hardware Performance Counters (HPC)

- Performance monitoring unit (PMU)
  - Originally used for performance tuning
  - Performance counters
    - Intel Core i7 (11 counters per core)
    - AMD Quad-Core Opteron 1356 CPU (4 counters per core)
  - Event selectors
- Automatically count hardware events at the process level
- Typical events include clock cycles, instruction retirements, cache misses, TLB misses (100+ events)
- Details in the developer's manuals
  - Intel® 64 and IA-32 Architectures Software Developer's Manual
  - BIOS and Kernel Developer's Guide (BKDG) for AMD Family 10h Processors

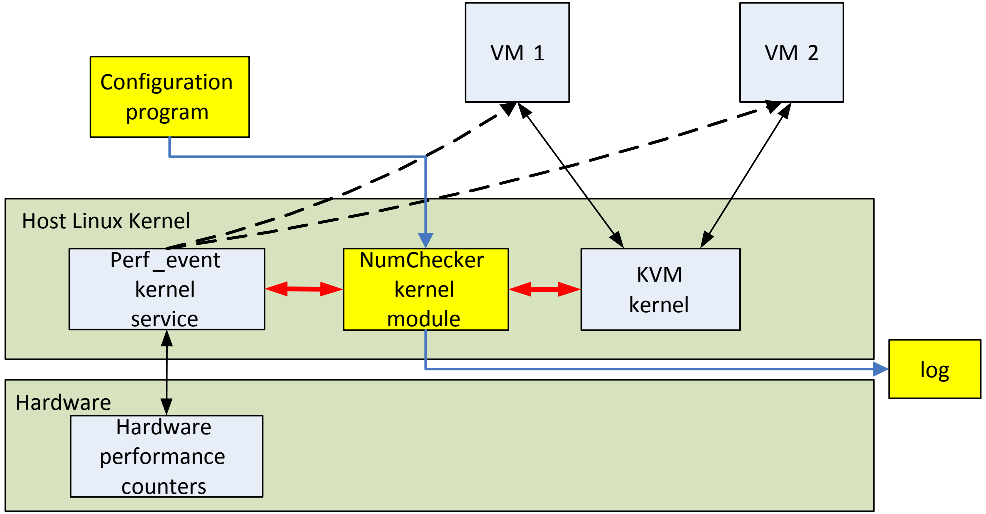
KVM in Linux

- Kernel-based virtual machine (KVM)
  - Based on Intel (VT) or AMD (SVM)
  - Guest mode and host mode
  - Each VM is an individual process
- KVM kernel module
  - Handles interception
- Linux Perf_event kernel service
  - Initializes, enables/disables, reads, and closes HPCs

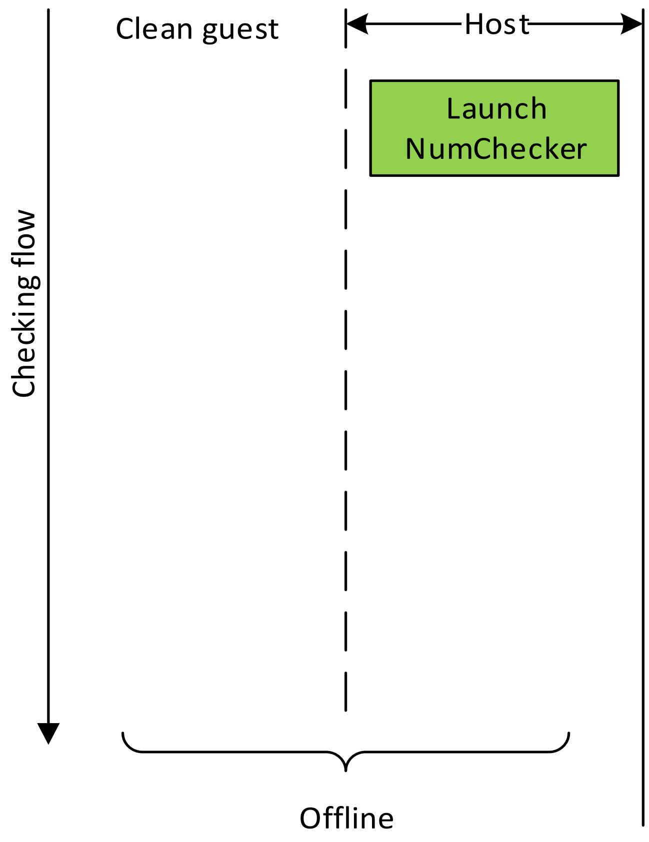


NumChecker Design

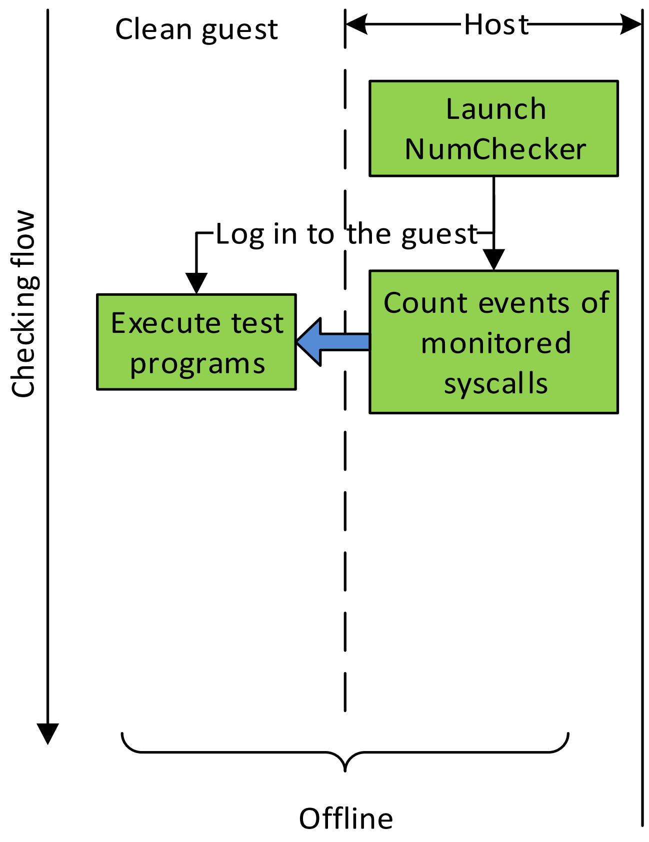
- NumChecker kernel module
  - Communicates with Perf_event kernel service and KVM kernel
- Configuration program
  - Dynamically configure the events and syscalls to be monitored
- Log
  - HPC results are stored and compared with the reference model



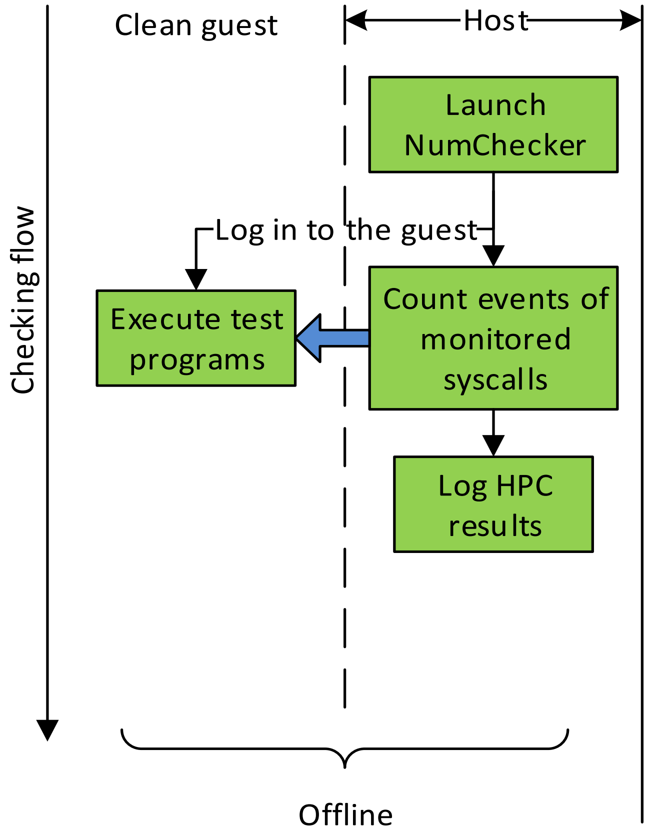
Two-Phase Detection and Identification



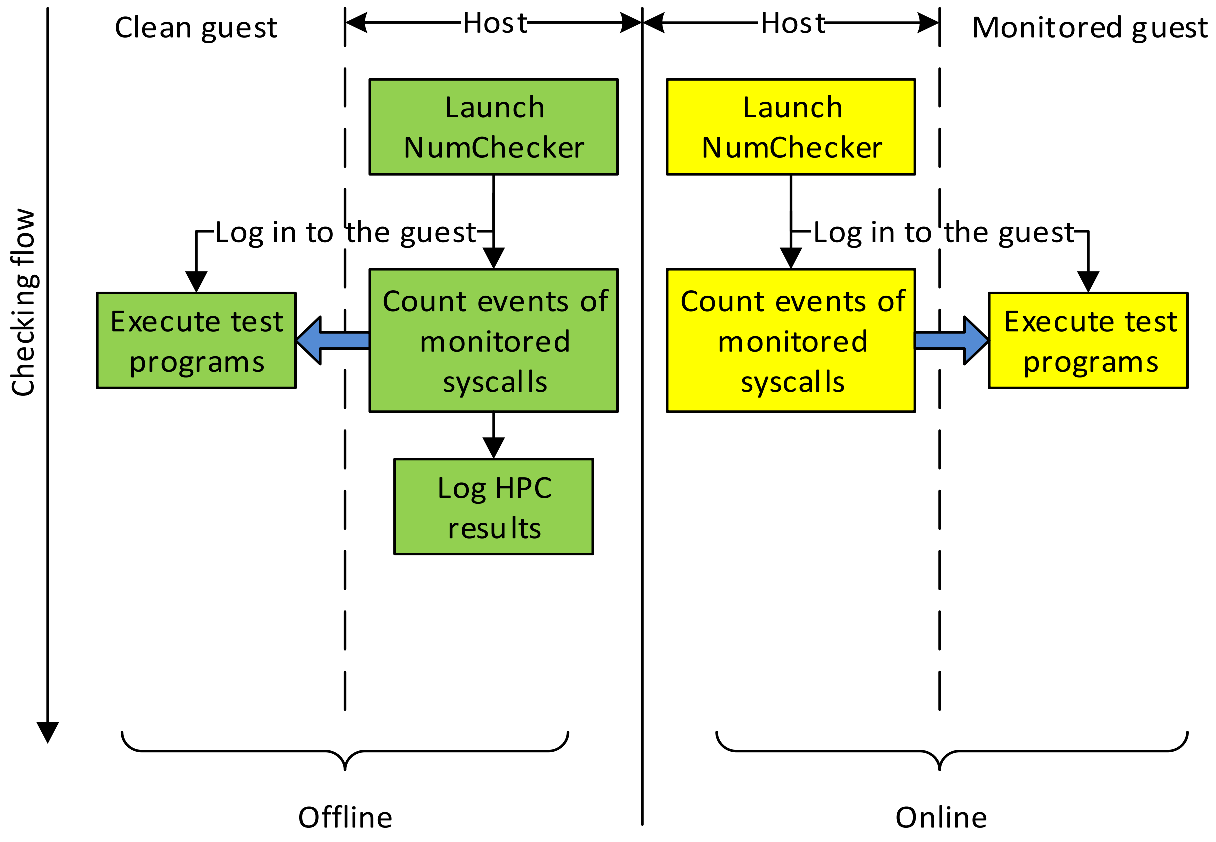
Two-Phase Detection and Identification



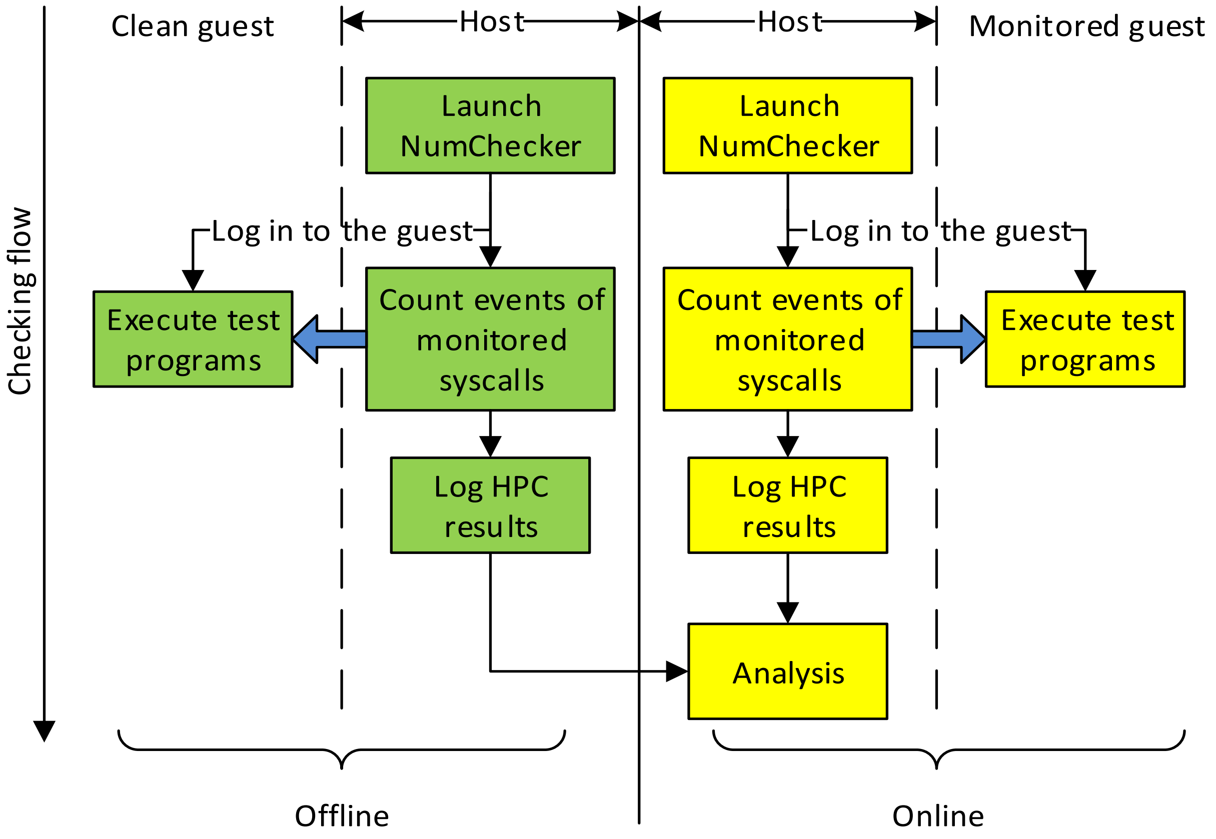
Two-Phase Detection and Identification



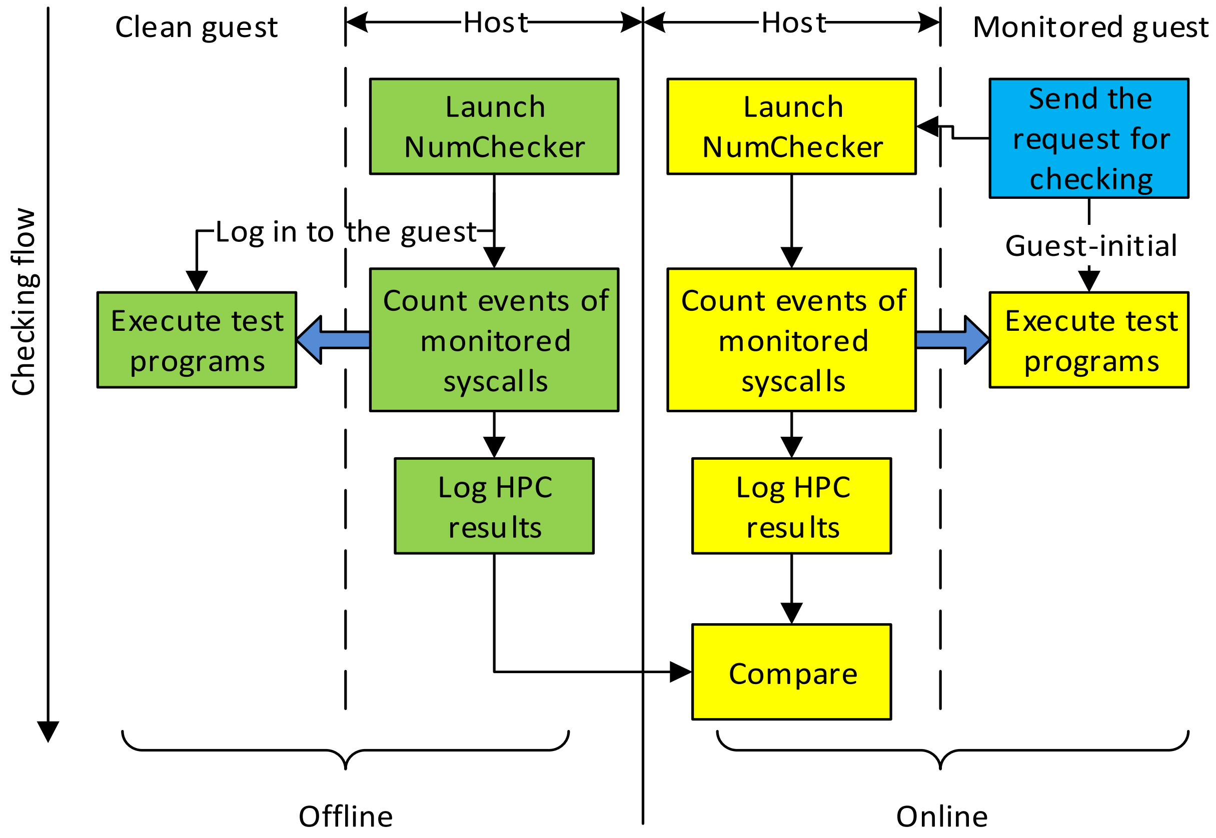
Two-Phase Detection and Identification



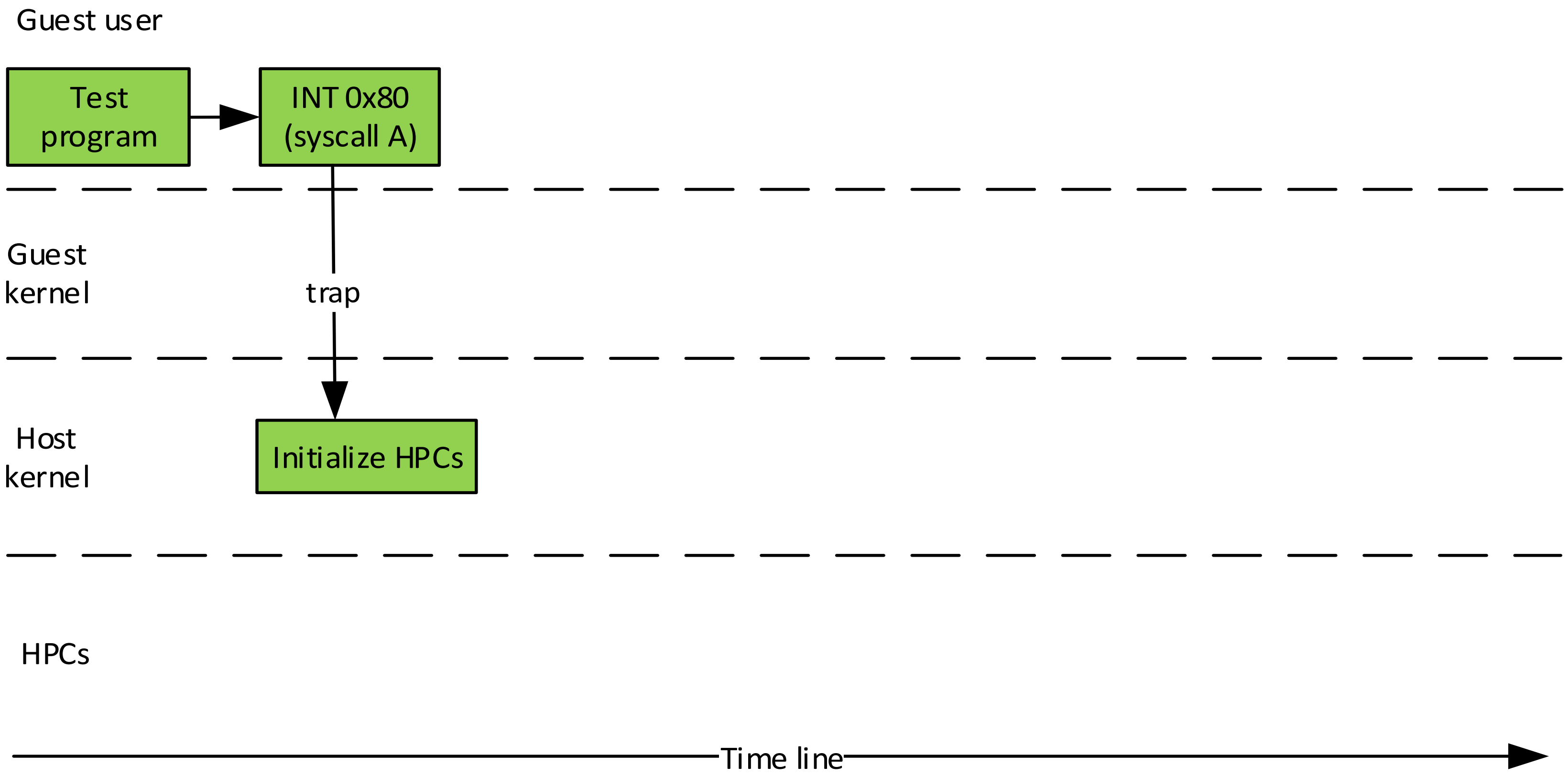
Two-Phase Detection and Identification



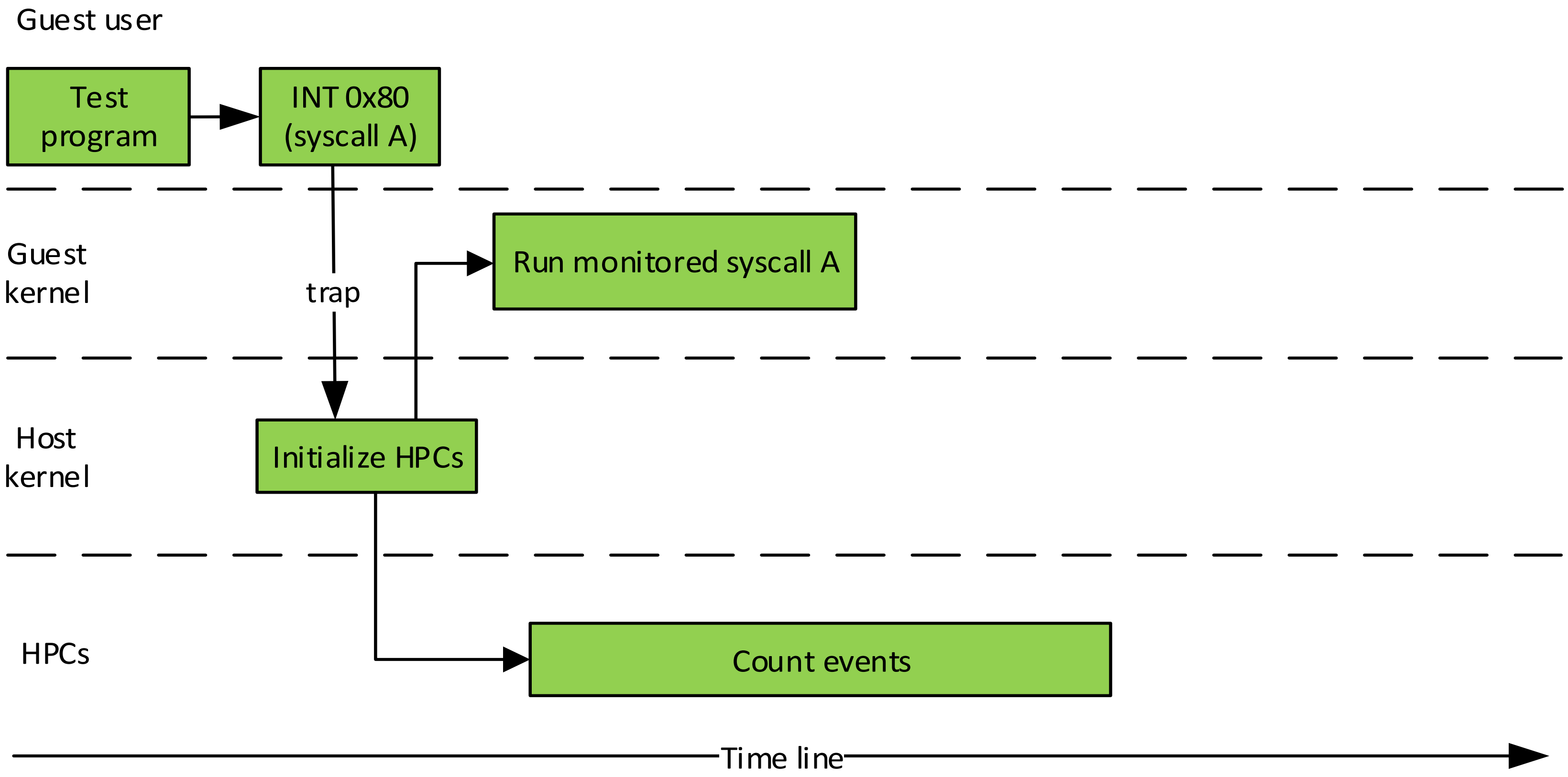
Two-Phase Detection and Identification



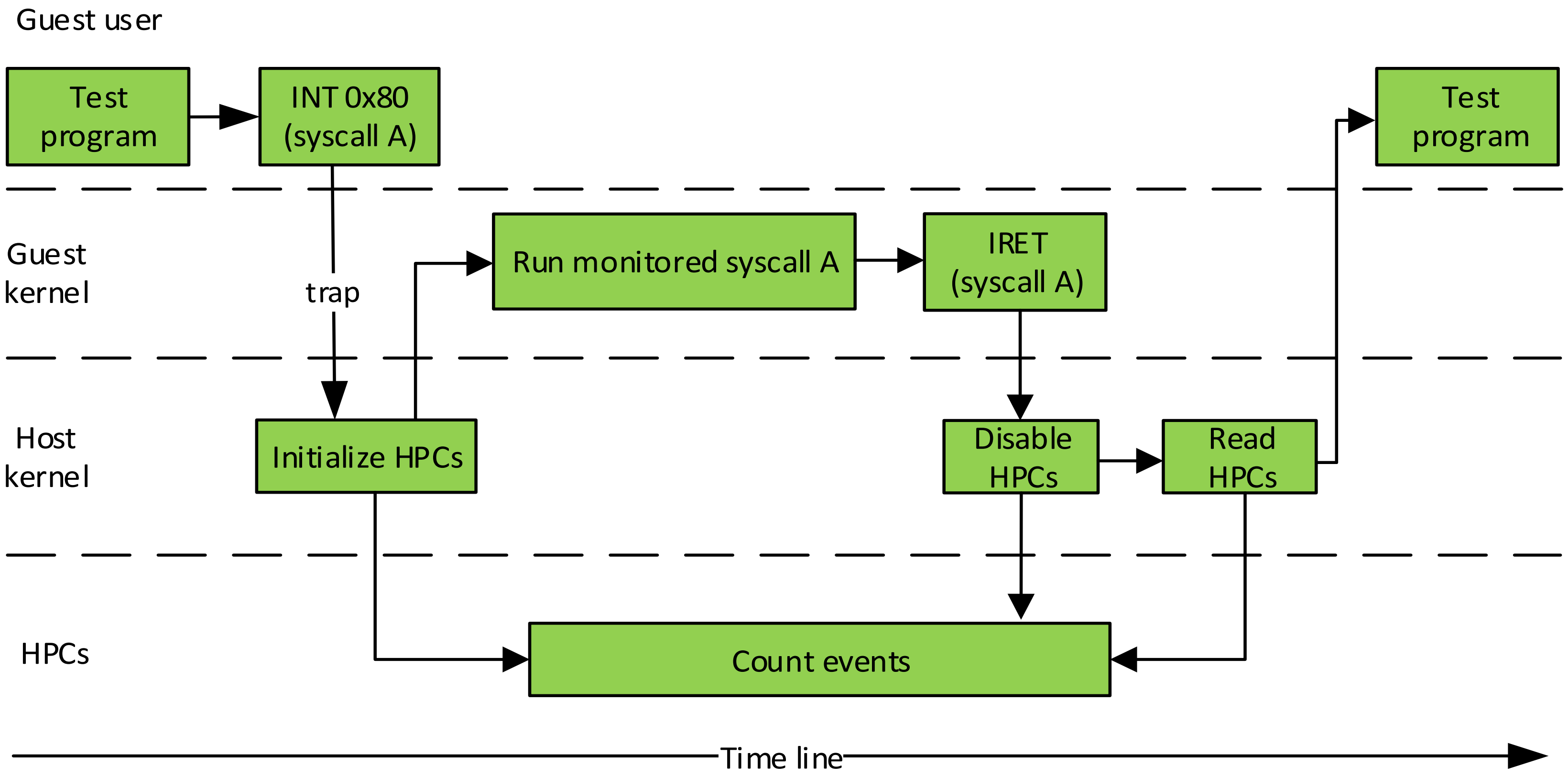
Syscall Measurement



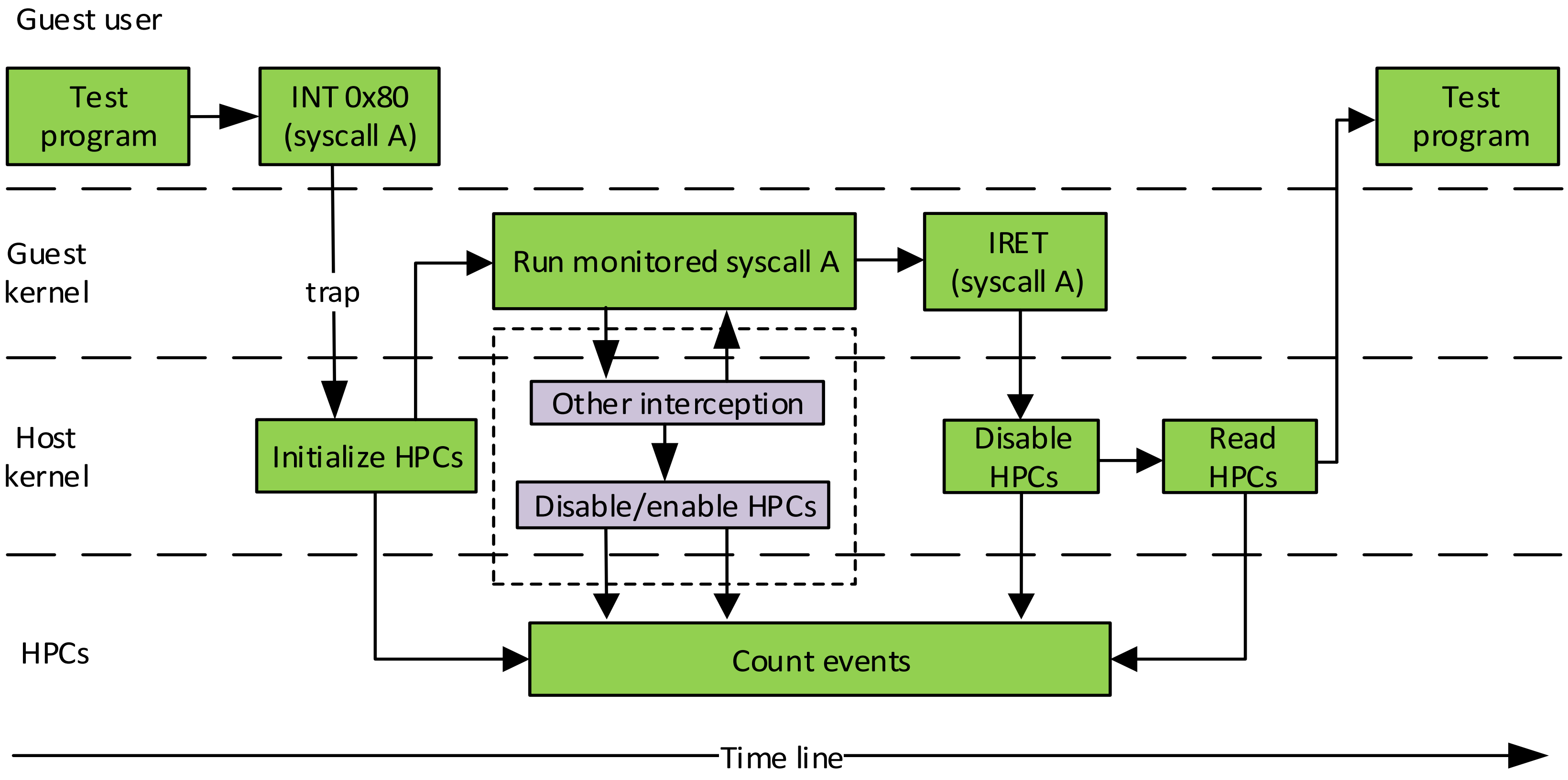
Syscall Measurement



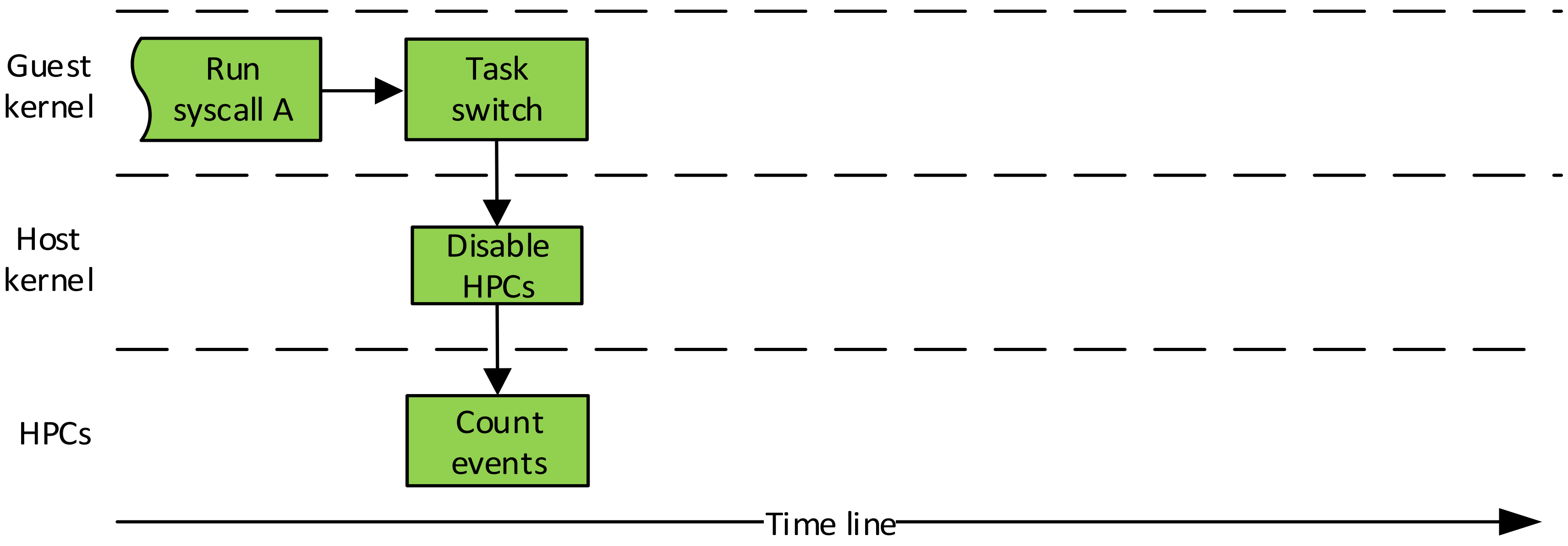
Syscall Measurement



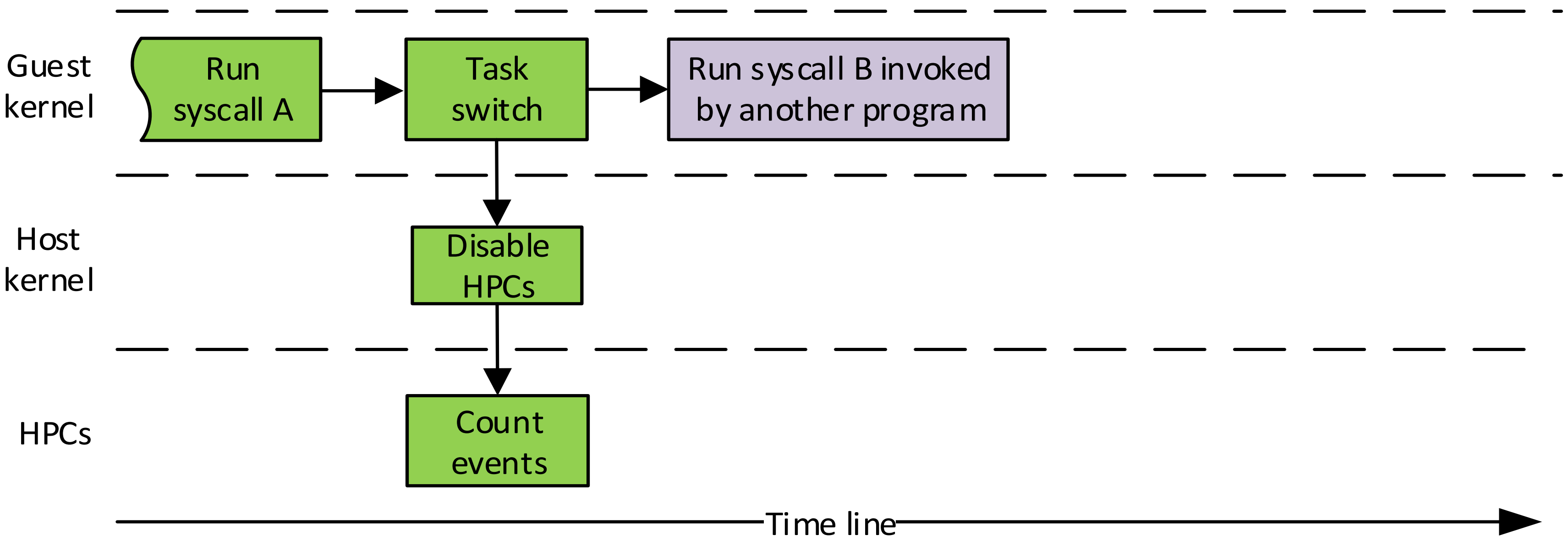
Syscall Measurement



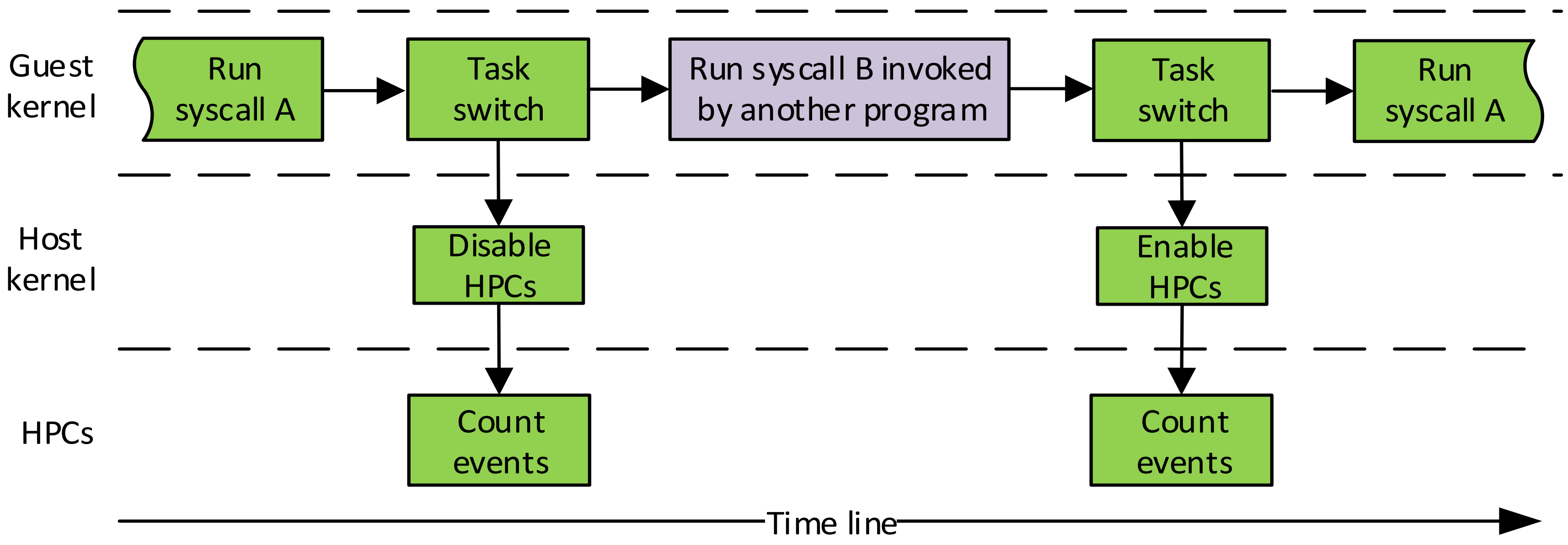
Kernel Preemption Handling



Kernel Preemption Handling



Kernel Preemption Handling



Detection: Test Programs

- Select preamble system calls to allow VMM to identify the process
- Ensures that we control the system call execution with selected arguments
- A sequence of selected system calls for measurement

Detection: Choosing Proper Events

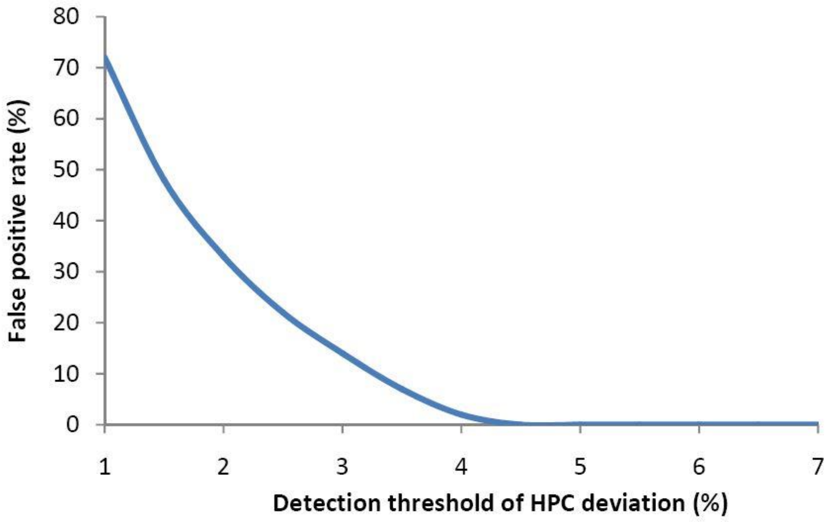
- Events that occur frequently during the syscall
- Events that are statistically more stable in the presence of noises
- Events selected
  - UOPS: retired micro-ops
  - INST: retired instructions
  - NRET: retired near returns
  - BRAN: retired branch instructions
  - BRNT: retired branch taken instructions

Detection: Deviation Threshold

- Deviation
  - Event: E_x, system call: S_y
  - Count: $C(E_x, S_y)$

$$D_{test}(x, y) = \frac{C_{test}(E_x, S_y) - C_{ref}(E_x, S_y)}{C_{ref}(E_x, S_y)}$$

- Deviation threshold
  - Pick the threshold with the least false positive rate
  - HPC deviations is smaller than 5%
  - If the deviation exceeds 5%, malicious modifications are suggested



Detection: Kernel Rootkits Detected

Detection capabilities. The numbers are deviations (%) from uninfected executions. Any deviation of more than 5% suggests a malicious modification.

Guest OS	Rootkit	Events counted	System calls monitored: sys_open	sys_close	sys_read	sys_getdents64	sys_stat64	Detected?
Linux 3.0	Matias	UOPS	0.0	-0.1	-0.1	25.9	0.0	Yes
		INST	0.0	-0.1	0.0	27.5	0.0	
		NRET	0.0	0.0	0.0	24.7	0.0	
		BRAN	0.0	-0.7	0.0	25.0	0.0	
		BRNT	0.6	0.0	0.0	36.1	0.0	
	Suterusu	UOPS	0.0	-0.1	0.6	139.8	0.0	Yes
		INST	0.0	-0.1	0.0	155.7	0.0	
		NRET	0.0	2.4	0.0	64.9	0.0	
		BRAN	0.2	0.0	0.0	219.6	0.0	
		BRNT	0.6	0.0	0.0	308.8	0.9	
	KBeast	UOPS	24.8	-0.1	129.8	107.7	-0.1	Yes
		INST	13.5	0.0	72.3	59.0	-0.1	
		NRET	9.9	0.0	56.7	24.7	0.0	
		BRAN	12.7	0.0	86.7	67.5	0.0	
		BRNT	12.0	0.0	82.1	60.0	0.9	

Detection: Kernel Rootkits Detected

Guest OS	Rootkit	Events counted	System calls monitored: sys_open	sys_close	sys_read	sys_getdents64	sys_stat64	Detected?
Linux 2.6	Enyelkm 1.1	UOPS	0.2	1.2	61.2	95.3	0.4	Yes
		INST	0.8	2.3	41.0	62.0	-2.3	
		NRET	4.0	12.5	28.1	54.9	4.0	
		BRAN	1.7	2.6	55.7	76.7	1.1	
		BRNT	0.8	2.1	38.3	74.8	0.8	
	Phalanx b6	UOPS	5.6	-0.1	132.0	24.7	0.0	Yes
		INST	8.5	0.0	201.5	35.0	-2.5	
		NRET	14.0	0.0	56.3	17.6	0.0	
		BRAN	19.5	-1.7	165.1	69.2	-0.5	
		BRNT	19.8	0.0	203.9	56.5	0.0	
	Sebek 3.2	UOPS	0.7	-0.1	1.8	0.0	-0.9	Yes
		INST	9.4	0.0	10.3	0.0	-0.7	
		NRET	8.0	0.0	18.8	0.0	0.0	
		BRAN	13.8	0.9	2.4	0.0	-0.5	
		BRNT	10.3	0.0	1.9	0.0	0.0	
	Adore-ng	UOPS	-10.7	4.1	40.0	228.6	0.0	Yes
		INST	0.0	0.0	0.0	289.0	-0.6	
		NRET	0.0	0.0	0.0	80.4	4.0	
		BRAN	0.0	2.6	2.4	524.4	-0.5	
		BRNT	-1.2	1.0	1.3	437.0	0.0	

Detection: Kernel Rootkits Detected

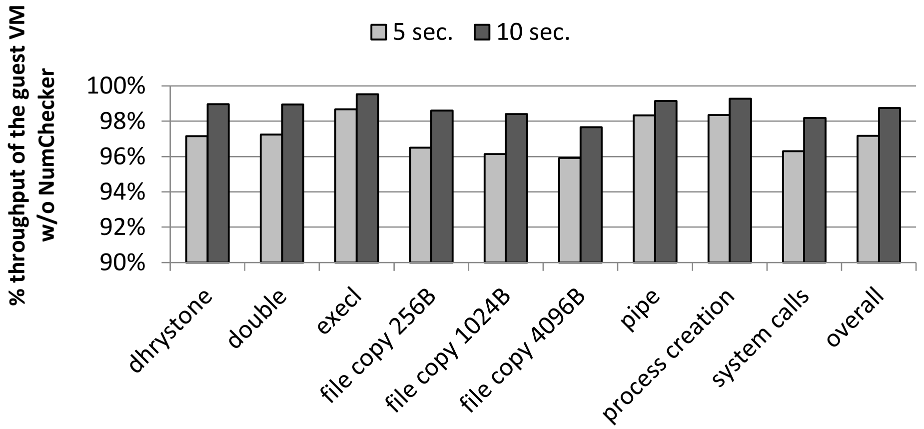
Guest OS	Rootkit	Events counted	System calls monitored					Detected?
			sys_open	sys_close	sys_read	sys_getde -nts64	sys_stat64	
Linux 2.4	SucKIT 1.3b	UOPS	923.9	13.3	42.7	212.4	276.5	Yes
		INST	836.1	8.6	59.5	242.9	284.3	
		NRET	676.5	50.0	150.0	483.3	383.3	
		BRAN	1294.2	72.0	33.3	1028.1	292.9	
		BRNT	1125.6	21.2	68.9	1227.2	301.4	
	Adore 0.42	UOPS	175.8	9.5	0.2	353.7	203.8	Yes
		INST	99.4	10.3	0.0	427.7	91.9	
		NRET	123.5	25.0	0.0	650.0	161.1	
		BRAN	119.9	24.0	0.0	1313.1	162.9	
		BRNT	119.2	9.1	0.0	1443.2	149.3	
	Sk2rc2	UOPS	384.2	22.1	73.4	36.5	121.8	Yes
		INST	363.4	52.4	79.5	39.8	63.1	
		NRET	488.2	50.0	166.7	95.8	166.7	
		BRAN	359.2	128.0	76.9	66.9	98.6	
		BRNT	365.6	27.3	75.6	123.5	36.1	
	Superkit	UOPS	955.4	13.3	42.7	215.8	284.4	Yes
		INST	827.8	10.8	59.5	244.4	283.1	
		NRET	535.3	50.0	233.3	483.3	383.3	
		BRAN	1399.5	28.0	61.5	1014.4	295.2	
		BRNT	1071.2	21.2	68.9	1235.8	298.6	

Detection: Performance Evaluation

- Test program execution time
  - Each test program contains 500 iterations to repeatedly invoke the corresponding system call

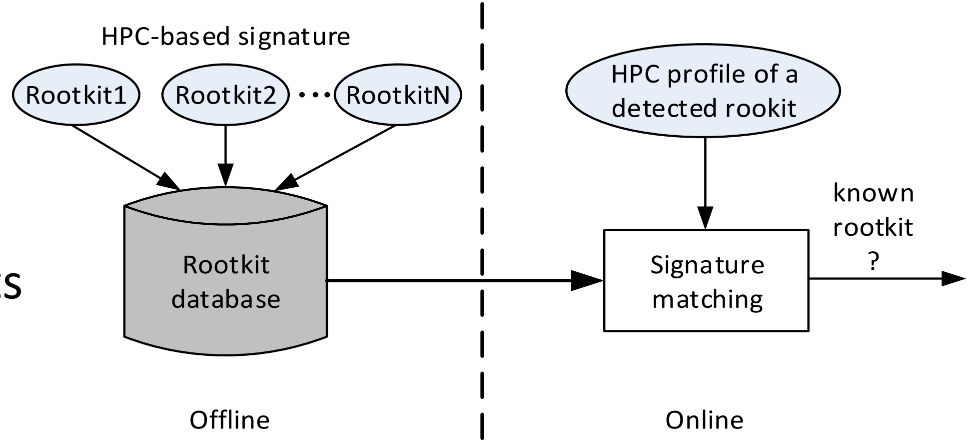
	Redhat 7.3	Fedora Core 4	Ubuntu 11.10
Test_open&close	44.9 ms	52.7 ms	50.9 ms
Test_read	50.5 ms	69.1 ms	65.5 ms
Test_getdents64	61.0 ms	75.7 ms	69.3 ms
Test_stat64	27.2 ms	40.5 ms	20.3 ms
Average	45.6 ms	59.5 ms	51.5 ms

- Guest performance overhead
  - Throughput degradation of the guest VM when NumChecker is invoked every 5 and 10 seconds



Identification: HPC-based Behavior Signature

- HPC-based behavior signature
  - Let $C(E_x, S_y)$ denote the count of event x from the execution of system call y.
  - m hardware events
  - n system calls
  - an vector V with $m * n$ elements can be obtained:



$$V=[C(E_1,S_1),C(E_2,S_1), \ldots C(E_m,S_1),C(E_1,S_2),C(E_2,S_2),\ldots C(E_m,S_n)]$$

Identification: HPC-based Signature

The deviation of the element in the tested vector from the one in the reference vector is:

$$D_{test}(x, y) = \left| \frac{C_{test}(E_x, S_y) - C_{ref}(E_x, S_y)}{C_{ref}(E_x, S_y)} \right|$$

D_{test} is calculated for each element in the tested vector and the largest one D_{test_max} is determined:

$$D_{test\_max} = \max_{1 \le x \le m, 1 \le y \le n} D_{test}(x, y)$$

Average deviation from the rootkit reference denoted as D_{test_avg} and the Fitting Rate (FR) on the rootkit reference, which is defined as follows:

$$\text{FR} = \frac{\text{no. of elements fitted to the targeted reference}}{\text{no. of elements in the tested vector.}}$$

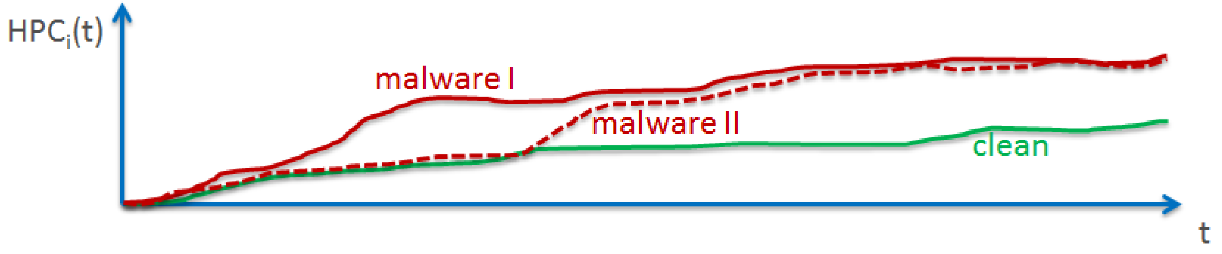
Identification: Kernel Rootkits Identified

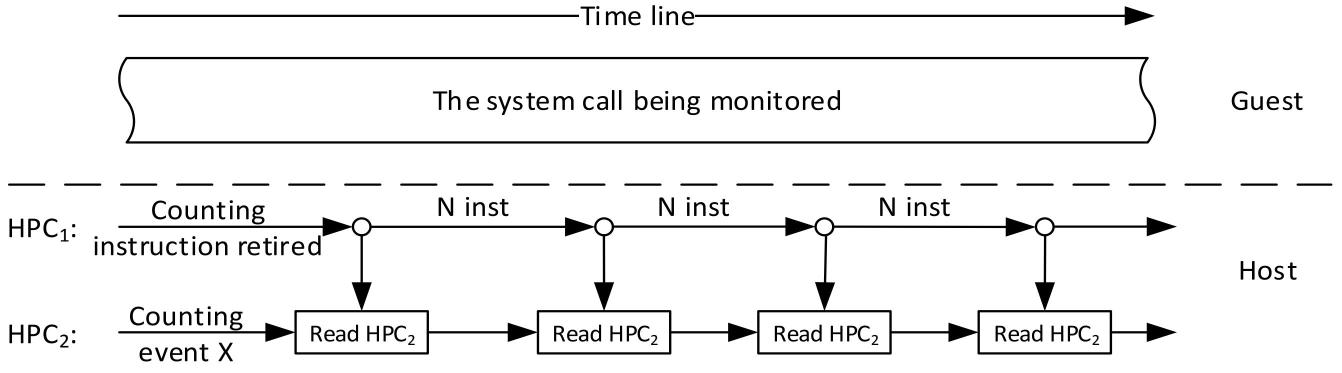
Rootkit under test		SucKIT 1.3b	Adore 0.42	Sk2rc2	Superkit	Identified?
SucKIT 1.3b	D_{test_max}	**3.80**	538.49	592.73	38.28	Yes
	D_{test_avg}	1.70	111.40	115.65	4.95*	
	FR	100	8	12	84*	
Adore 0.42	D_{test_max}	84.79	**3.77**	762.92	85.86	Yes
	D_{test_avg}	40.32	2.10	118.00	40.06	
	FR	8	100	4	12	
Sk2rc2	D_{test_max}	710.34	168.00	**3.71**	85.46	Yes
	D_{test_avg}	127.78	69.32	1.91	42.76	
	FR	0	0	100	8	
Superkit	D_{test_max}	30.00	569.41	572.96	**3.65**	Yes
	D_{test_avg}	5.51*	111.39	114.88	1.85	
	FR	84*	12	12	100	

Identification: Kernel Rootkits Identified

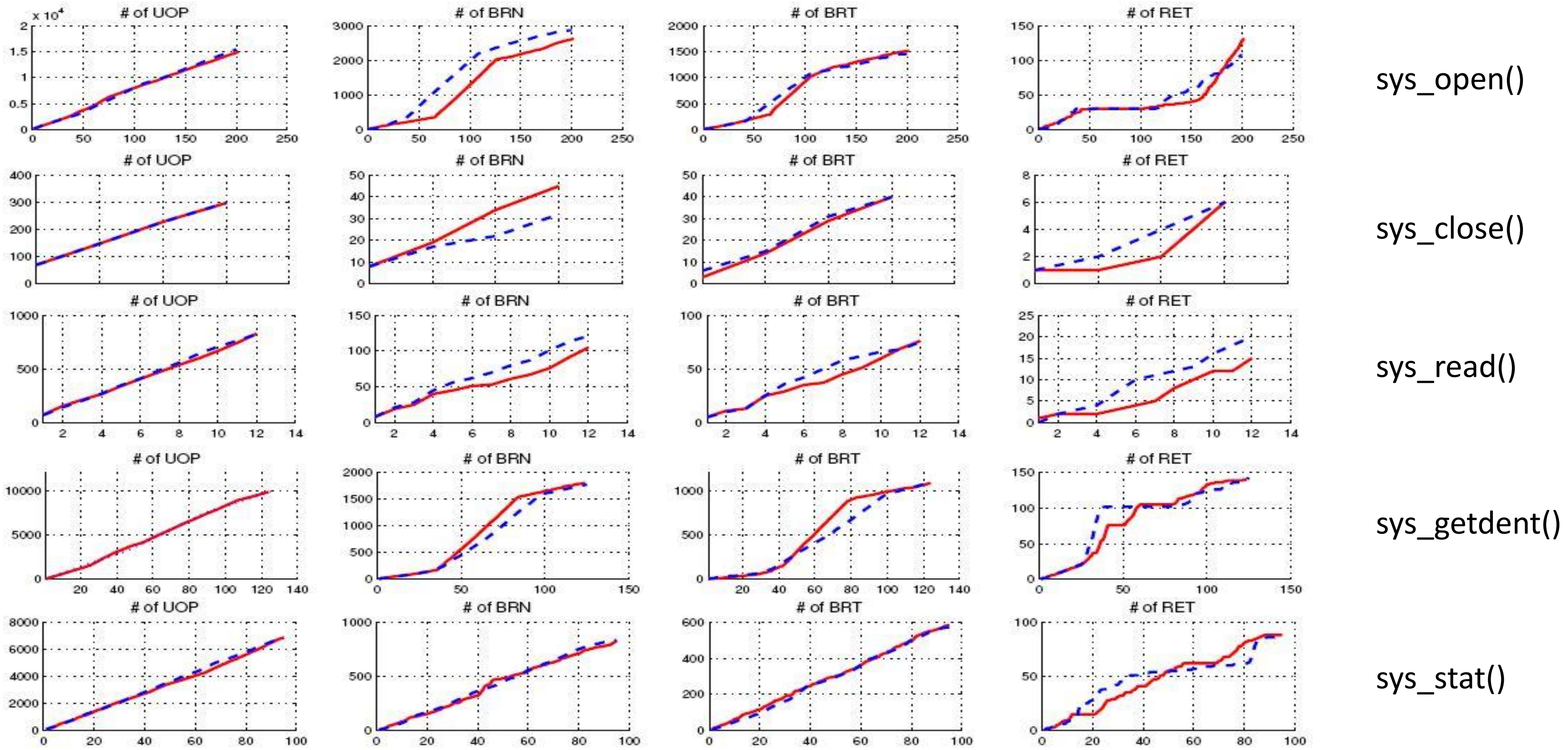
Rootkit under test		SucKIT 1.3b	Adore 0.42	Sk2rc2	Superkit	Identified?
SucKIT 1.3b	D_{test_max}	**3.80**	538.49	592.73	38.28	Yes
	D_{test_avg}	1.70	111.40	115.65	4.95*	
	FR	100	8	12	84*	
Adore 0.42	D_{test_max}	84.79	**3.77**	762.92	85.86	Yes
	D_{test_avg}	40.32	2.10	118.00	40.06	
	FR	8	100	4	12	
Sk2rc2	D_{test_max}	710.34	168.00	**3.71**	85.46	Yes
	D_{test_avg}	127.78	69.32	1.91	42.76	
	FR	0	0	100	8	
Superkit	D_{test_max}	30.00	569.41	572.96	**3.65**	Yes
	D_{test_avg}	5.51*	111.39	114.88	1.85	
	FR	84*	12	12	100	

Identification: Periodic Sampling





Identification: Periodic Sampling



Identification: Periodic Sampling

- W/O periodic sampling

Rootkit under test		SucKIT 1.3b	Superkit
SucKIT 1.3b	D_{test_max}	3.80	38.28
	D_{test_avg}	1.7	4.95
	FR	100	84
Superkit	D_{test_max}	30.00	3.65
	D_{test_avg}	5.51	1.85
	FR	84	100

- With periodic sampling

Rootkit under test		SucKIT 1.3b	Superkit
SucKIT 1.3b	D_{test_max}	3.90	54.67
	D_{test_avg}	1.35	12.04
	FR	100	45
Superkit	D_{test_max}	75.00	3.15
	D_{test_avg}	14.19	1.08
	FR	45	100

Security Analysis

- Rootkit may try to tamper with the HPCs
  - HPCs are controlled by host (VMM)
- Rootkit may tamper with the analysis process
  - Analysis process is done by host (VMM)
- Rootkit may try to predict the “good” number
  - The test program can be considered as a “secret key” and can be updated
  - The number of system call, system call argument, and hardware events are huge.

Security Analysis

- Rootkit may undo modifications
  - Rootkit is not aware of the test program
    - Not knowing the monitor time
  - Rootkit tries to identify the test program
    - VMM updates test program
  - Rootkit detects the test program and tries to undo the modification
    - Do or undo dilemma
    - Randomized sampling period
  - Strong rootkit detects the test program accurately and undo all modifications
    - Remove the test program and use machine learning approach

Conclusion

- NumChecker effectively detects and identifies kernel rootkits
  - VMM-based framework (can be applied to different types of virtualizations)
  - Validating the execution of guest system calls (can be changed to work with other software flows)
  - Based on hardware events (free to choose from hundreds of events)
- Using Hardware Performance Counters
  - Feature supported by hardware (Intel, AMD, etc.)
  - Very low performance overhead
  - Tamper-resistant from guest OS
  - Can be applied to other malware

Acknowledgement

- Rodrigo Branco
- Alexander Matrosov
- Nam Nguyen
- Jason Fung
- Mickey Shkatov