black hat ASIA 2017

MARCH 28-31, 2017

MARINA BAY SANDS / SINGAPORE

Myth and Truth about Hypervisor-Based Kernel Protector:

The Reason Why You Need Shadow-Box

Seunghun Han, Jungwhan Kang

(hanseunghun || ultract)@nsr.re.kr

Who are we ?

- Senior security researcher at NSR (National Security Research Institute of South Korea)
- Speaker at HITBSecConf 2016
- Author of the book series titled “64-bit multi-core OS principles and structure, Vol.1&2”
- a.k.a kkamagui, @kkamagui1

- Researcher at NSR
- Participated final round of some CTFs (Codegate, ISEC… held in South Korea)
- Interested in OS security and reading write-up of CTFs
- Got married last year ☺
- a.k.a ultract, @ultractt

Goal of This Presentation

- **We present lightweight hypervisor-based kernel protector, "Shadow-box"**
  - Shadow-box defends kernel from security threats efficiently, and we made it from scratch
- **We share lessons learned from deploying and operating Shadow-box in real world systems**
  - We have been operating Shadow-box since last year and share lessons learned!

Background

Design

Implementation

Lessons Learned

(and the Truth Previous Researches Did Not Tell You)

Demo. and Conclusion

(Black Hat Sound Bytes)

Linux Kernel Is Everywhere!

Security Threats of Linux Kernel

- **The Linux kernel suffers from rootkits and security vulnerabilities**
  - Rootkits: EnyeLKM, Adore-ng, Sebek, suckit, kbeast, and so many descendants
  - Vulnerabilities: CVE-2014-3153, CVE-2015-3636, CVE-2016-4557, CVE-2017-6074, etc.

Devices which use Linux kernel share security threats

Melee Combats at the Kernel-level

- **Kernel-level (Ring 0) protections are not enough**
  - Lots of rootkits and exploits work in the Ring 0 level
  - Protections against them are often easily bypassed and neutralized
    - Kernel Object Hooking (KOH)
    - Direct Kernel Object Manipulation (DKOM)

Protections need

an even lower level (Ring -1)

Well-known Rootkits

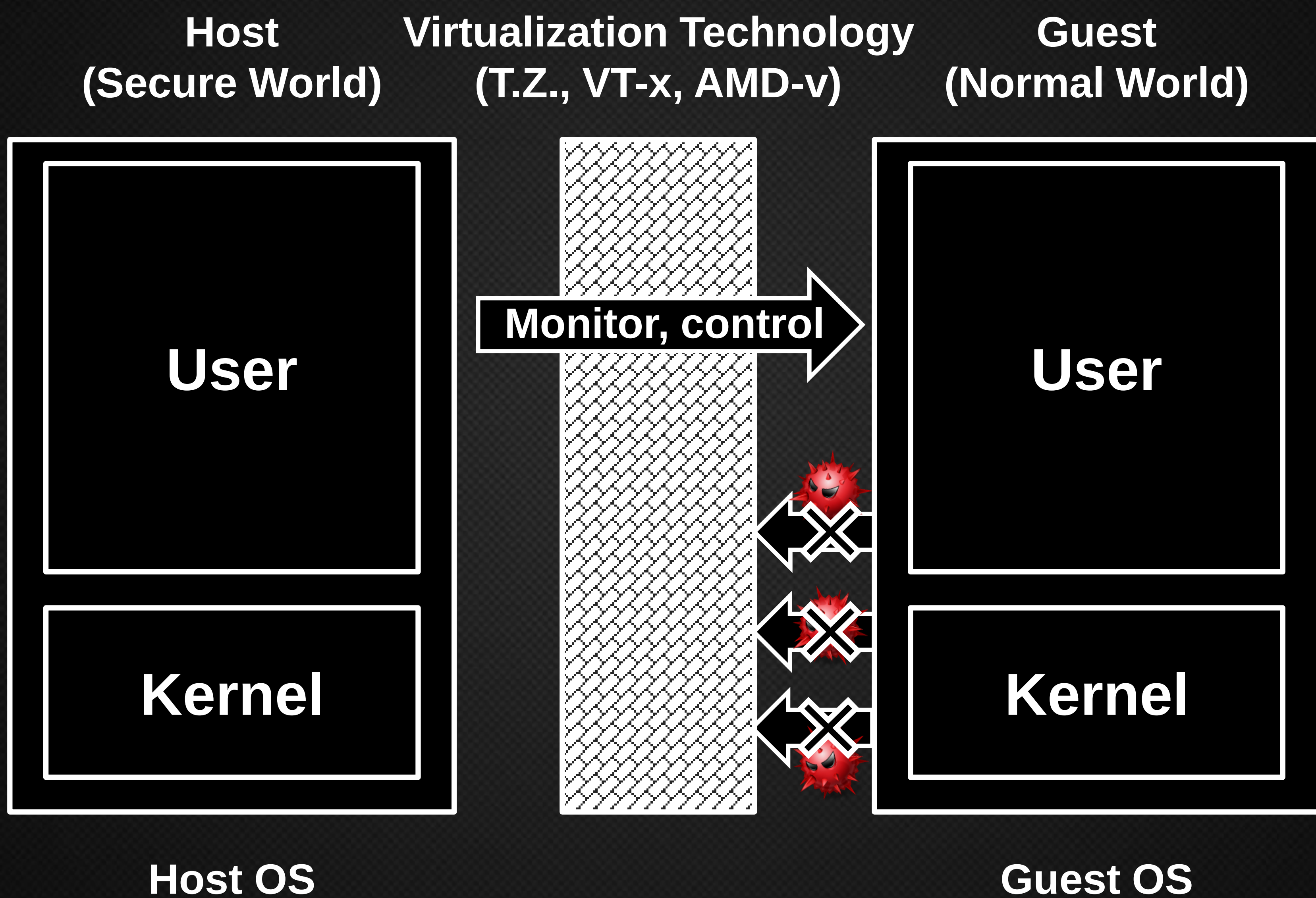
Name	Modified Kernel Object	Type	Attribute	Note
EnyeLKM 1.3	syscall_trace_entry	Code	Static	code change,
	sysenter_entry	Code	Static	syscall hook,
	module->list	Data	Dynamic	direct kernel object
	init_net->proc_net->subdir->tcp_data->tcp4_seq_show	Function pointer	Dynamic	manipulation (DKOM)
Adore-ng 0.56	vfs_root->f_op->write	Function pointer	Dynamic	function pointer hook
	vfs_root->f_op->readdir	Function pointer	Dynamic	
	vfs_proc->f_dentry->d_inode->i_op->lookup	Function pointer	Dynamic	
	socket_udp->ops->recvmsg	Function pointer	Dynamic	
Sebek 2.0	sys_call_table	System table	Static	syscall hook,
	vfs_proc_net_dev->get_info	Function pointer	Dynamic	function pointer hook,
	vfs_proc_net_packet->proc_fops	Function pointer	Dynamic	DKOM
	module->list	Data	Dynamic	
Suckit 2.0	idt_table	System table	Static	idt hook,
	sys_call_table	System table	Static	syscall hook
kbeast v1	sys_call_table	System table	Static	syscall hook,
	init_net->proc_net->subdir->tcp_data->tcp4_seq_show	Function pointer	Dynamic	function pointer hook,
	module->list	Data	Dynamic	DKOM

Other rootkits also have similar patterns

Taking the Higher Ground

- **Leveraging virtualization technology (VT)**
  - VT separates a machine into a host (secure world) and a guest (normal world)
  - The host in Ring -1 can freely access/control the guest in Ring 0 (the converse doesn’t hold)
  - VT-equipped HW: Intel VT-x, AMD AMD-v, ARM TrustZone

Trends of Introducing Ring -1



Previous Researches…

SecVisor: A Tiny Hypervisor to Provide Lifetime Kernel Code Integrity for Commodity OSes

Arvind Seshadri, CyLab/CMU, Pittsburgh, PA, USA, arvinds@cs.cmu.edu
Mark Luk, CyLab/CMU, Pittsburgh, PA, USA, mluk@ece.cmu.edu
Ning Qu, CyLab/CMU, Pittsburgh, PA, USA, quning@cmu.edu
Adrian Perrig, CyLab/CMU, Pittsburgh, PA, USA, perrig@cmu.edu

ABSTRACT

We propose SecVisor, a tiny hypervisor that ensures code integrity for commodity OS kernels. In particular, SecVisor ensures that only user-approved code can execute in kernel mode over the entire system lifetime. This protects the kernel against code injection attacks, such as kernel rootkits. SecVisor can achieve this property even against an attacker who controls everything but the CPU, the memory controller, and system memory chips. Further, SecVisor can even defend against attackers with knowledge of zero-day kernel exploits.

Our goal is to make SecVisor amenable to formal verification

1. INTRODUCTION

Computing platforms are steadily increasing in complexity, incorporating an ever-growing range of hardware and supporting an ever-growing range of applications. Consequently, the complexity of OS kernels is steadily increasing. The increased complexity of OS kernels also increases the number of security vulnerabilities. The effect of these vulnerabilities is compounded by the fact that, despite many efforts to make kernels modular, most kernels in common use today are monolithic in their design. A compromise of any part of a monolithic kernel could compromise the entire kernel. Since the kernel occupies a privileged position in the software stack

Guest-Transparent Prevention of Kernel Rootkits with VMM-based Memory Shadowing

Ryan Riley, *Purdue University*, *rileyrd@cs.purdue.edu*
Xuxian Jiang, *George Mason University*, *xjiang@gmu.edu*
Dongyan Xu, *Purdue University*, *dxu@cs.purdue.edu*

Abstract

Kernel rootkits pose a significant threat to computer systems as they run at the highest privilege level and have unrestricted access to the resources of their victims. Many current efforts in kernel rootkit defense focus on the *detection* of kernel rootkits – after a rootkit attack has taken place, while the smaller number of efforts in kernel rootkit *prevention* exhibit limitations in their capability or deployability. In this paper we present a kernel rootkit prevention system called NICKLE which addresses a common, fundamental characteristic of most kernel rootkits: the need for executing their own kernel code. NICKLE is a lightweight, virtual machine monitor (VMM) based system that transparently prevents unauthorized kernel code execution for unmodified commodity (guest) OSes. NICKLE is based on a new scheme

Lares: An Architecture for Secure Active Monitoring Using Virtualization

Bryan D. Payne Martim Carbone Monirul Sharif Wenke Lee
School of Computer Science
Georgia Institute of Technology
Atlanta, Georgia 30332–0765
{bdpayne,mcarbone,msharif,wenke}@cc.gatech.edu

Ab

Host-based security tool
sion detection systems are
day's computers. Malware
ately disable any security t
ing them useless. While cu
moving these vulnerable sec
tual machine, this approach

Ensuring Operating System Kernel Integrity with OSck

Owen S. Hofmann Alan M. Dunn Sangman Kim Indrajit Roy* Emmett Witchel
The Uni
{osh,adunn,sangma

Abstract

modify operating system state to a
threat to system security. This pa
it discovers kernel rootkits by de
is to operating system data. OSc
techniques for detecting rootkits,
large portions of the kernel heap
educe type information for verifica
kernel source code and in-memory

e integrity checks that execute c
ating system create data races, and
c solution for ensuring kernel men
introduce two new classes of ker

black hat ASIA 2016

NumChecker: A System Approach for Kernel Rootkit Detection and Identification

Xueyang Wang, Ph.D.
Xiaofei (Rex) Guo, Ph.D.
(xueyang.wang || xiaofei.rex.guo) *noSPAM* intel.com

Researches Are Excellent, But They Look …

I heard and knew about them

But, I can not find in real world!

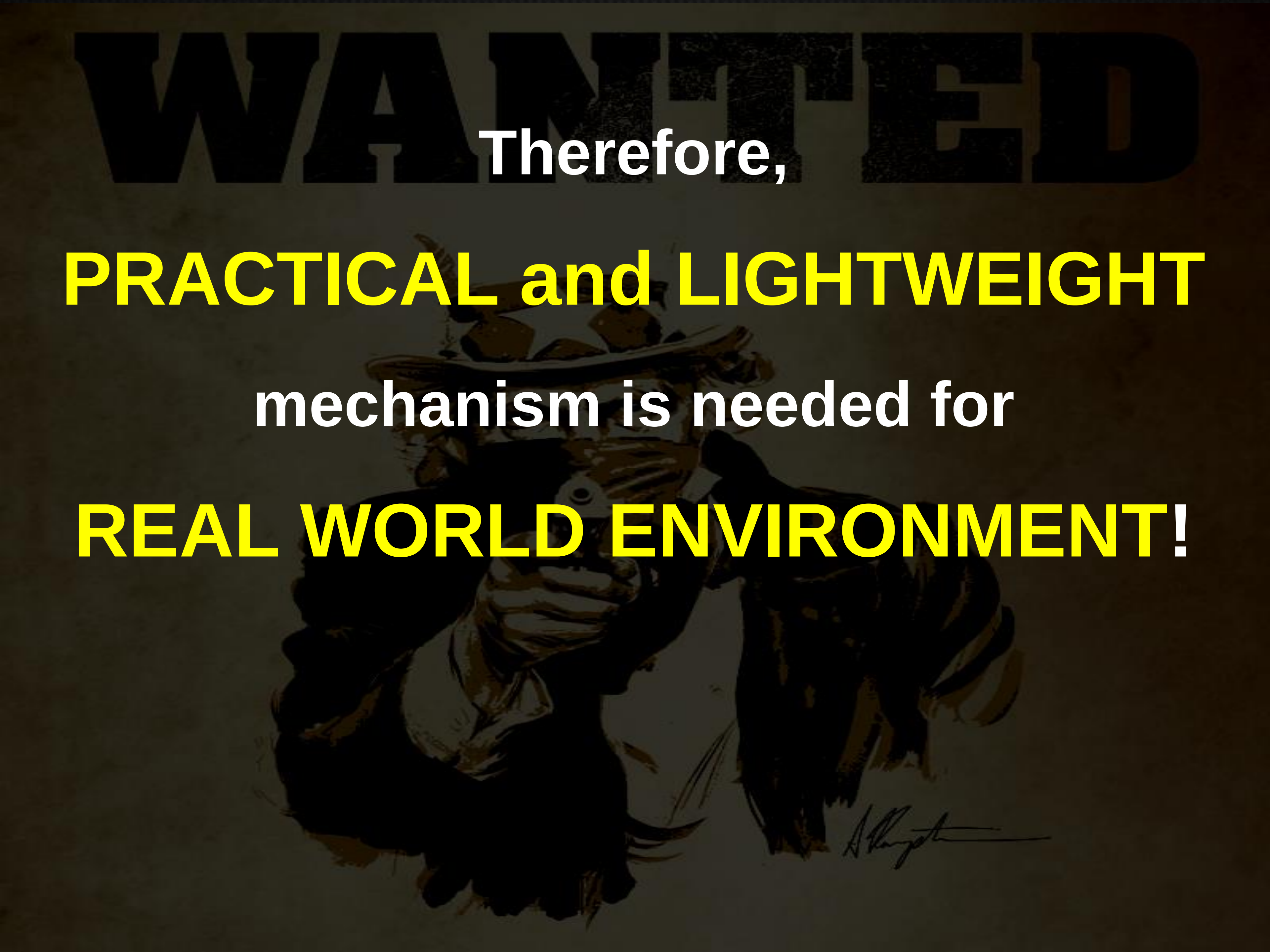
Restrictions on Previous Researches (1)

- **Many researches have preconditions**
  - They usually change kernel code or hypervisor
  - They also need well-known hashes of LKM, well-known value of kernel data, secure VM for analyzing target VM, etc.

- **Many researches consume much resource**
  - The host and the guest run each OS
    - They allocate resources independently!
  - The host consumes many CPU cycles to introspect the guest because of semantic gap

Restrictions on Previous Researches (2)

- **In conclusion, previous researches are considered for laboratory environment only**
  - They assume they can control environment!
  - But, real world environment is totally different from laboratory environment!
  - You even don't know the actual environment before the software is installed!



Therefore,

PRACTICAL and LIGHTWEIGHT

mechanism is needed for

REAL WORLD ENVIRONMENT!

Design Goals of Kernel Protector

- **Lightweight**
  - Focus on rootkit detection and protection
    - Simple and extensible architecture
  - Small memory footprint
    - No secure VMs and no multiple OSes
- **Practical**
  - Out-of-box approach
    - No modification of kernel code and data
  - Dynamic injection
    - Load any time from boot to runtime

Background

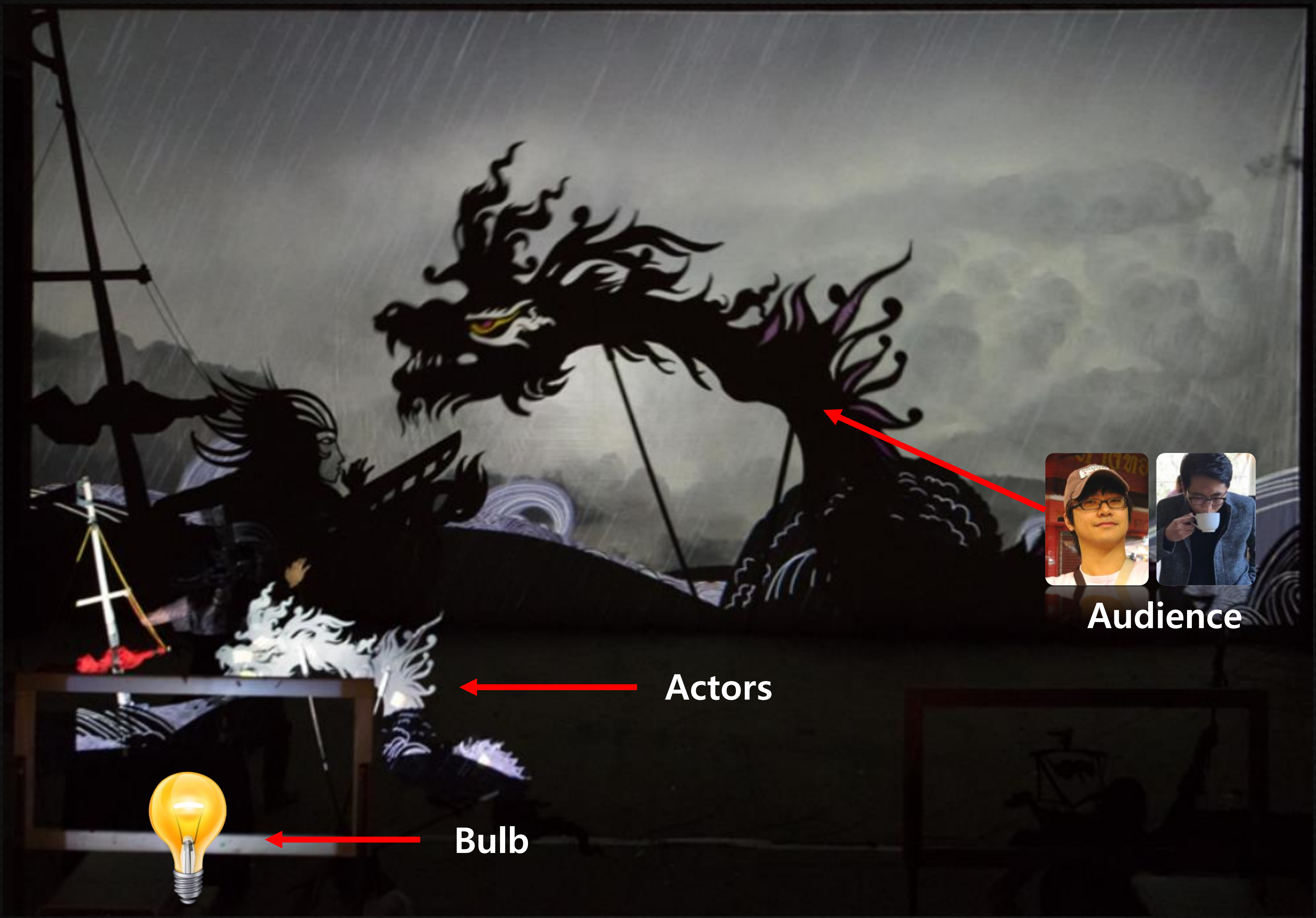
Design

Implementation

Lessons Learned
(and the Truth Previous Researches Did Not Tell You)

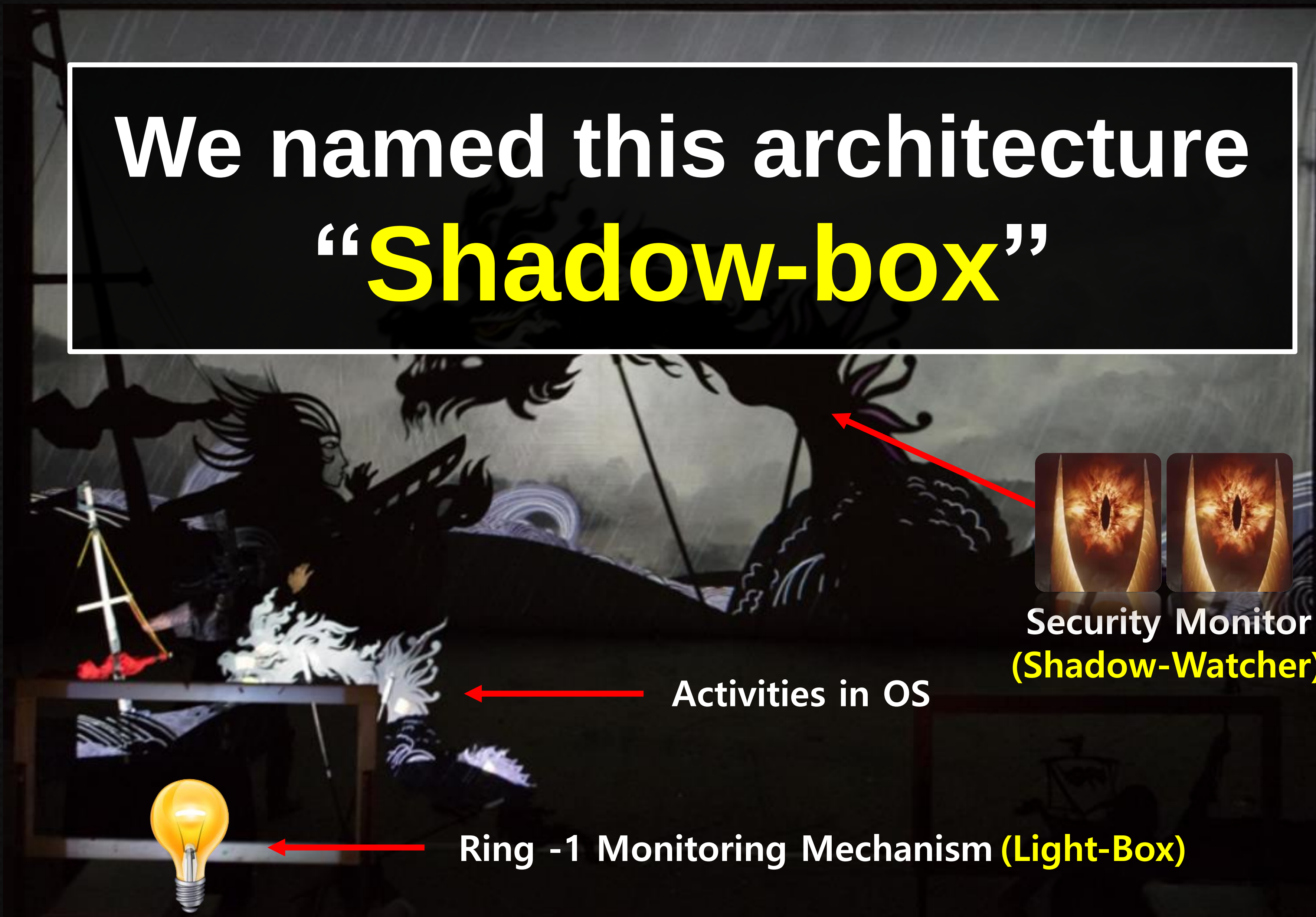
Demo. and Conclusion
(Black Hat Sound Bytes)

Security Architecture in Shadow Play

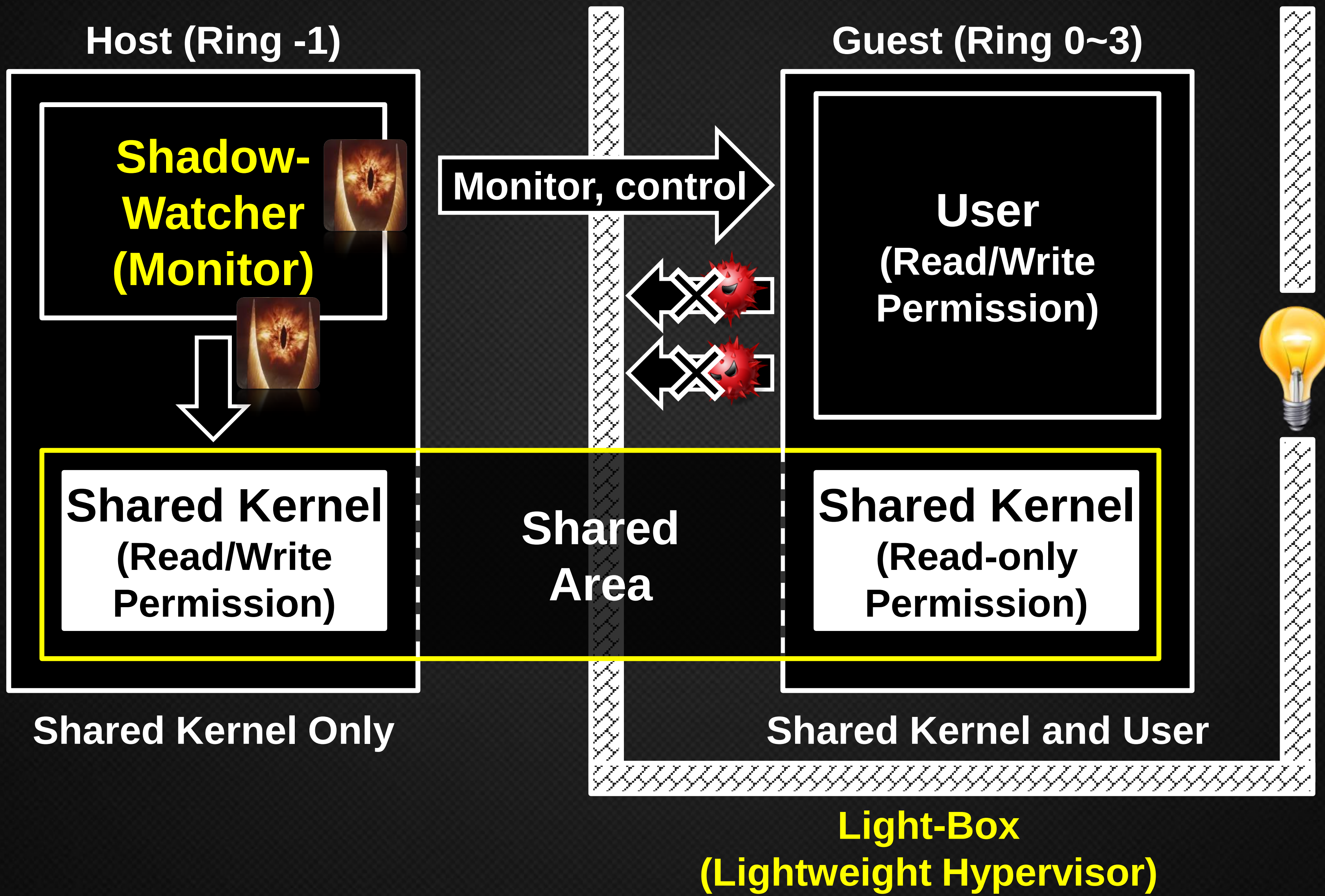


Security Architecture in Shadow Play

We named this architecture "Shadow-box"



Architecture of Shadow-Box



Architecture of Light-Box

- **Light-box, lightweight hypervisor,**
  - Isolates worlds by using memory protection technique in VT
  - Shares the kernel area between the host (Ring -1) and the guest (Ring 0 ~ 3)
    - Does not run each OS in two worlds
  - Uses smaller resources than existing mechanisms and has narrow semantic gap
  - Can be loaded any time (loadable kernel module)

Architecture of Shadow-Watcher

- **Shadow-watcher**
  - Monitors the guest by using Light-box
  - Checks if applications of the guest modify kernel objects or not by event-driven way
    - Code, system table, IDT table, etc.
  - Checks the integrity of the guest by introspecting kernel object by periodic way
    - Process list, loadable kernel module (LKM) list, function pointers of file system and socket

What can Shadow-Box do?

- **Shadow-box protects Linux kernel from**
  - Static kernel object attacks
    - Static kernel object = immutable in runtime
    - Code modification and system table modification attacks
  - Dynamic kernel object attacks
    - Dynamic kernel object = mutable in runtime
    - Process hiding and module hiding
    - Function pointer modification attacks

Background

Design

Implementation

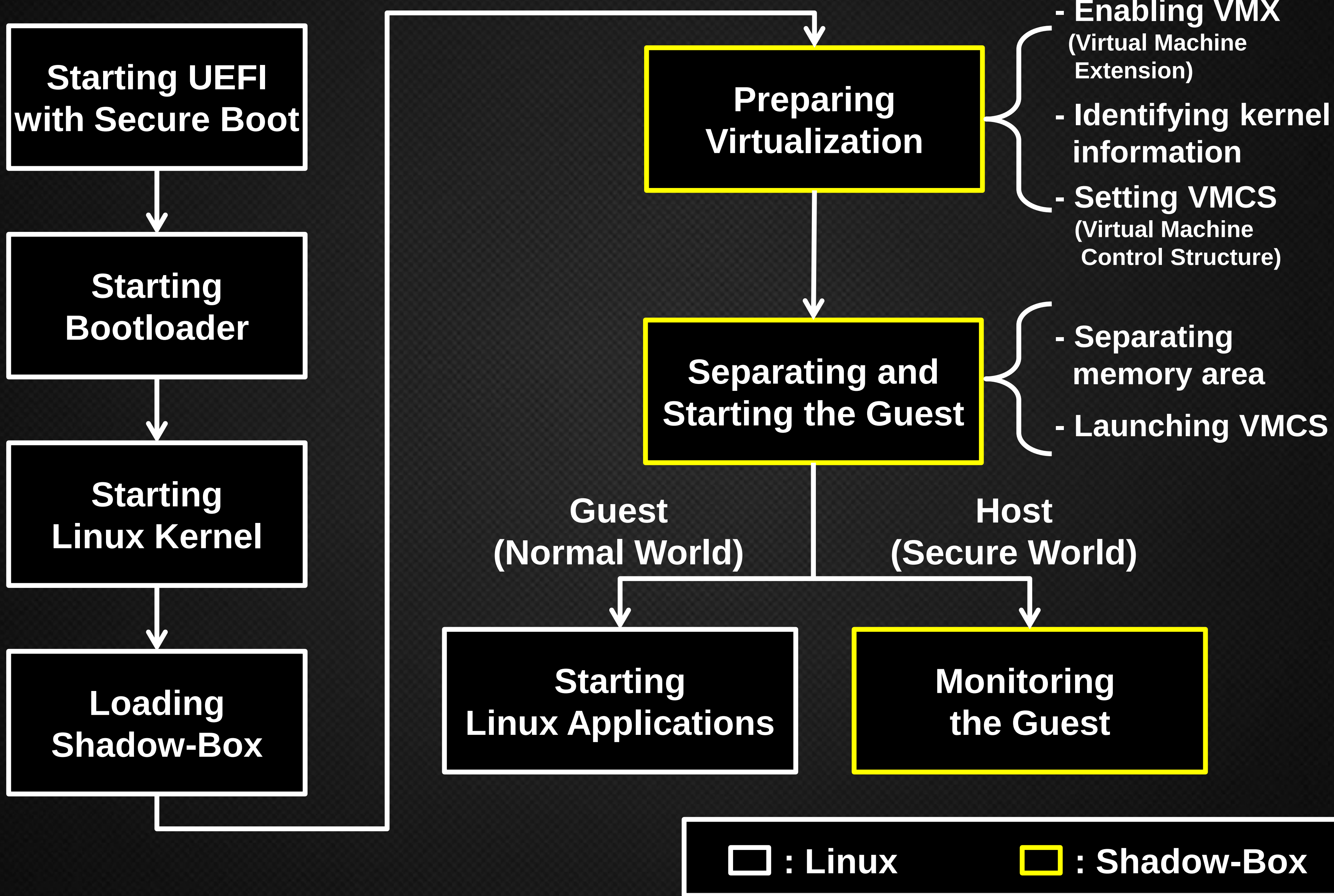
Lessons Learned

(and the Truth Previous Researches Did Not Tell You)

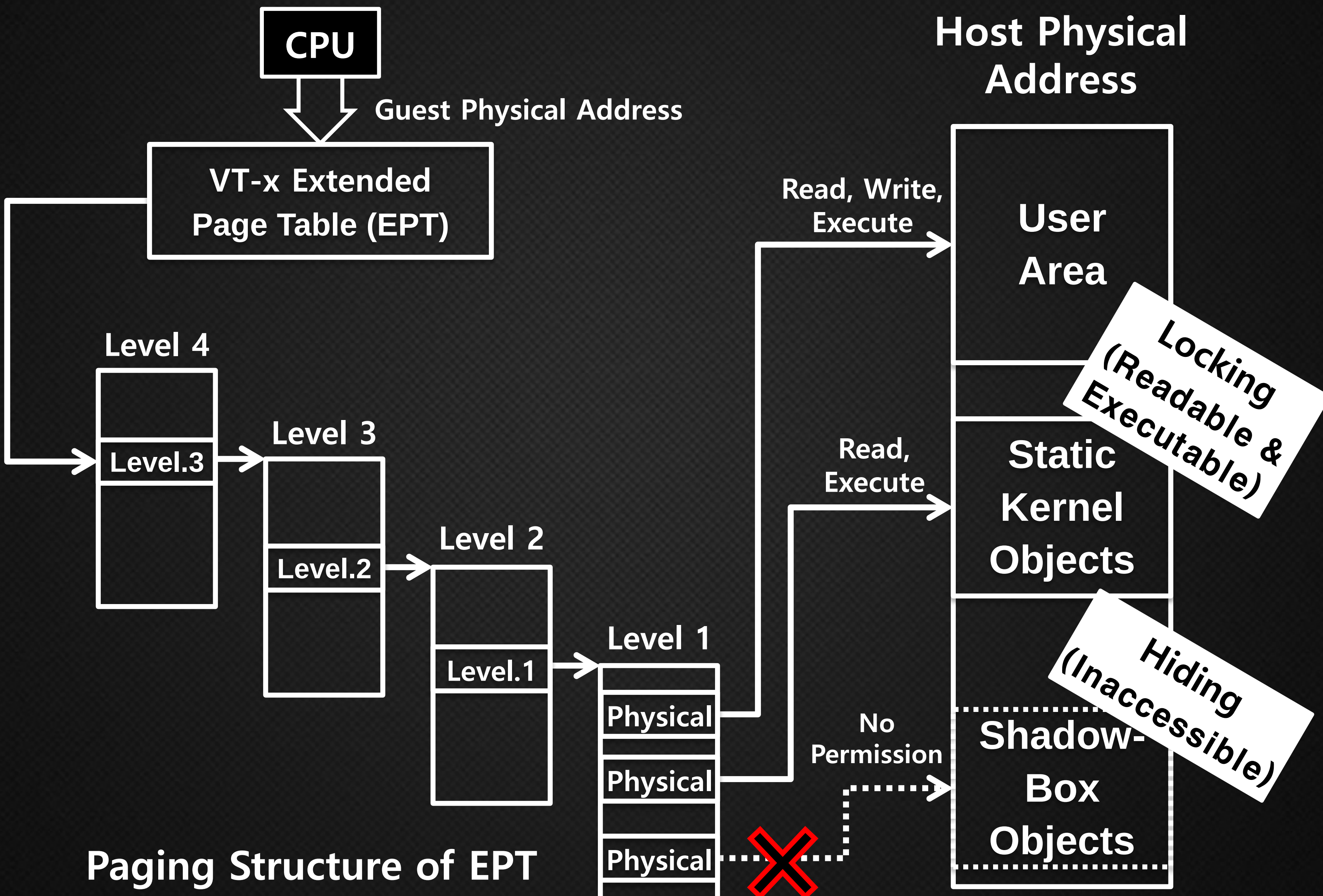
Demo. and Conclusion

(Black Hat Sound Bytes)

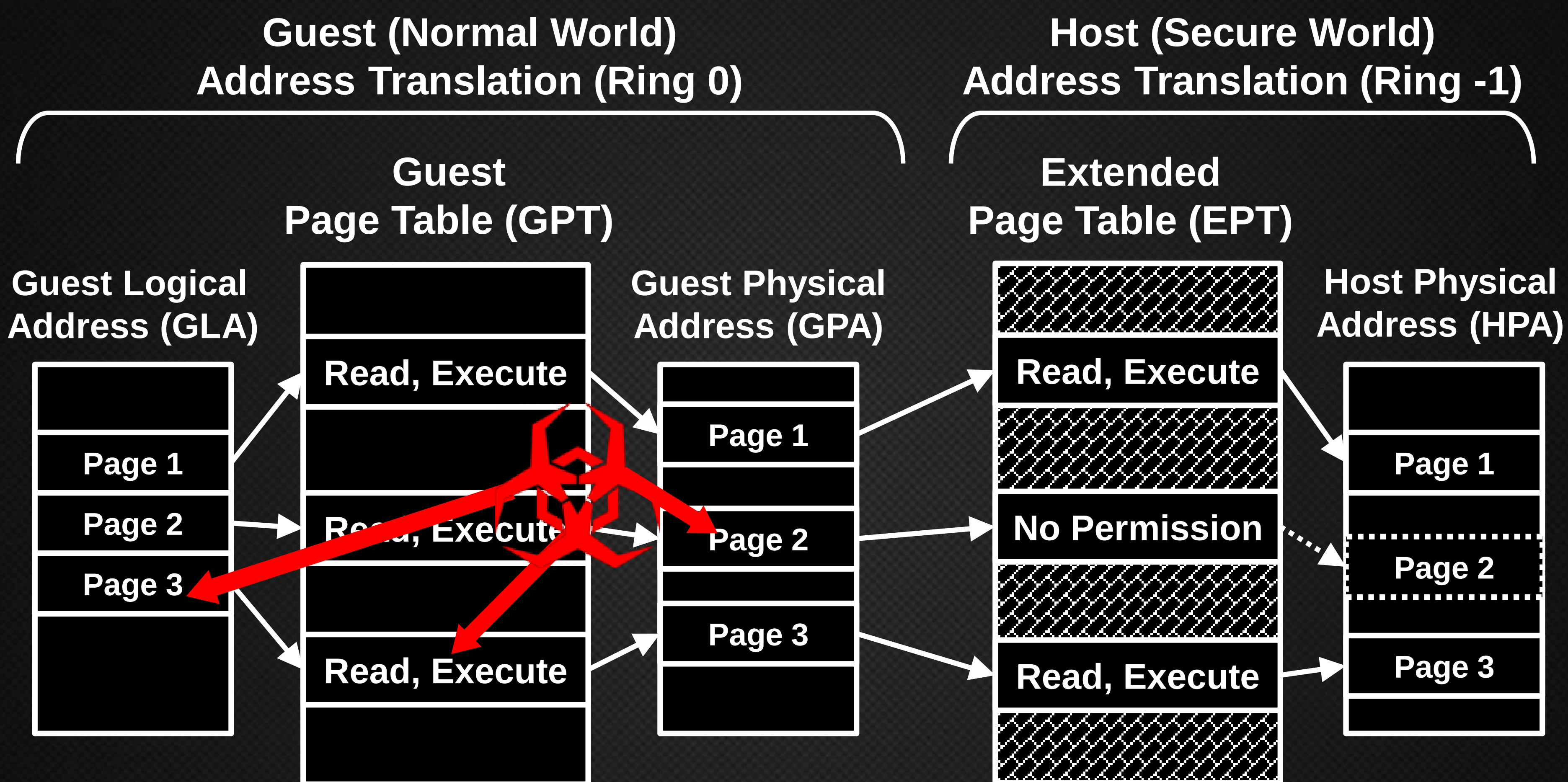
Boot Process using Shadow-Box



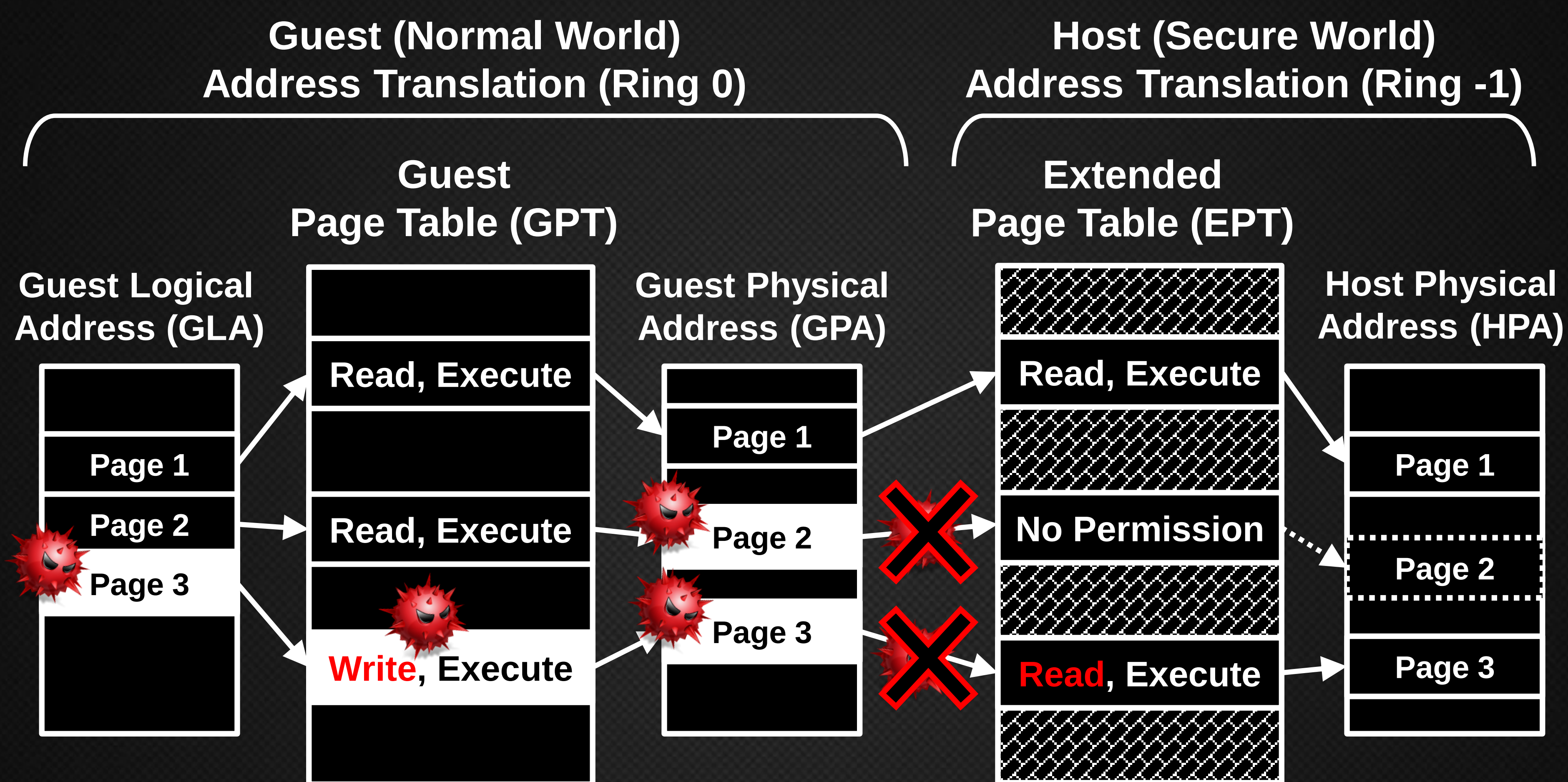
Static Kernel Object Protection (1)



Static Kernel Object Protection (2)

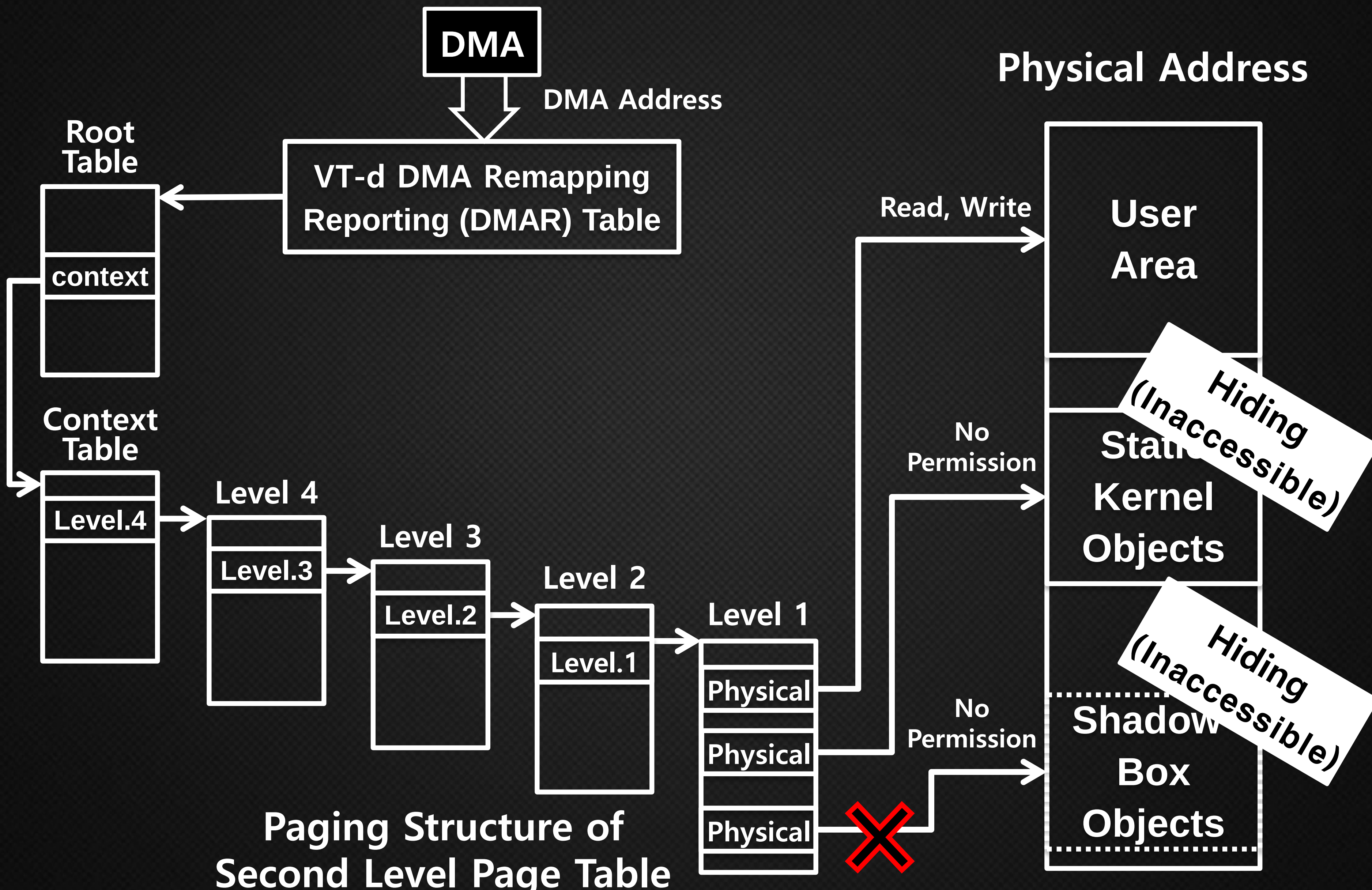


Static Kernel Object Protection (3)

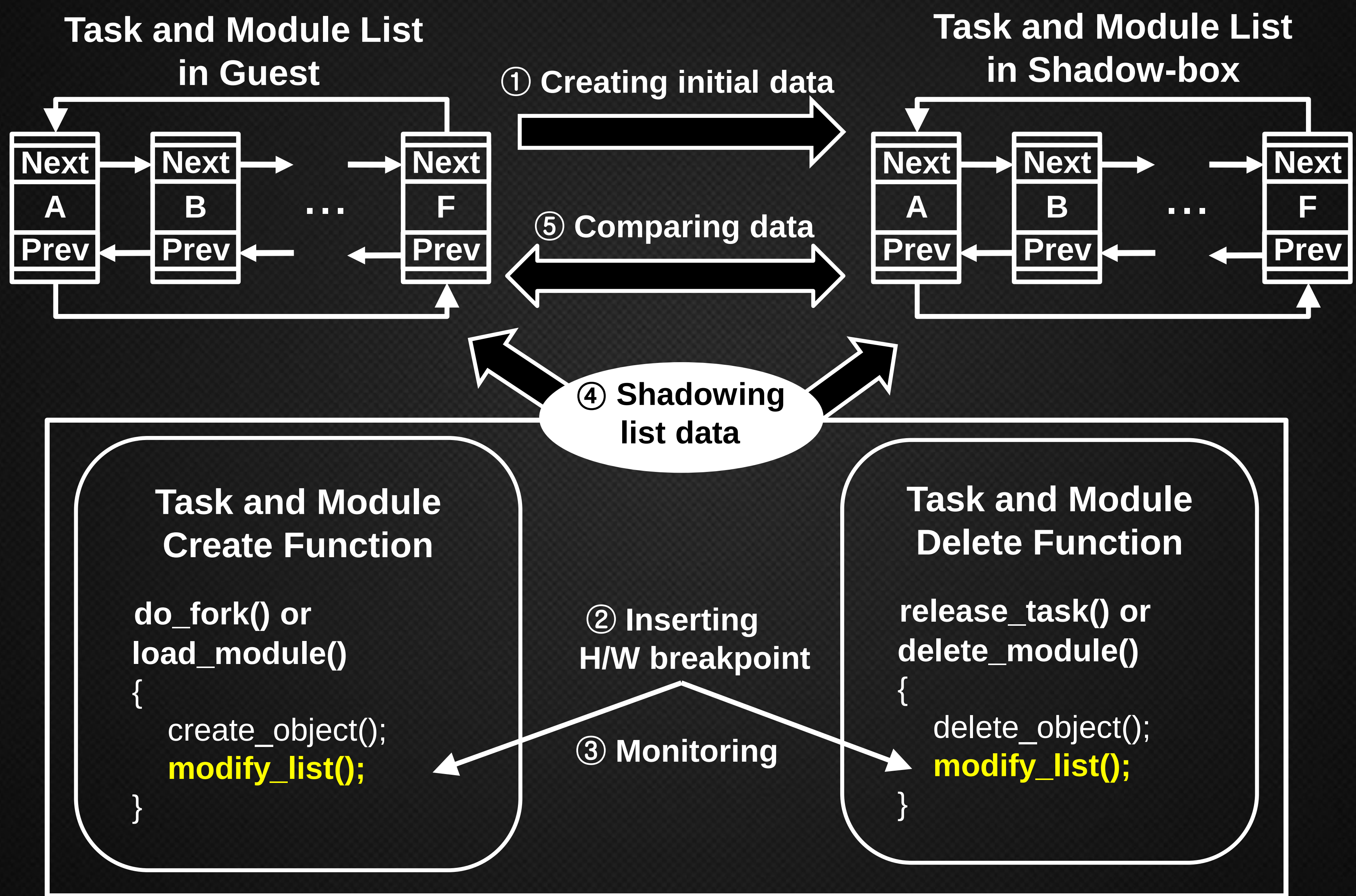


EPT protects the host from attack propagation of the guest

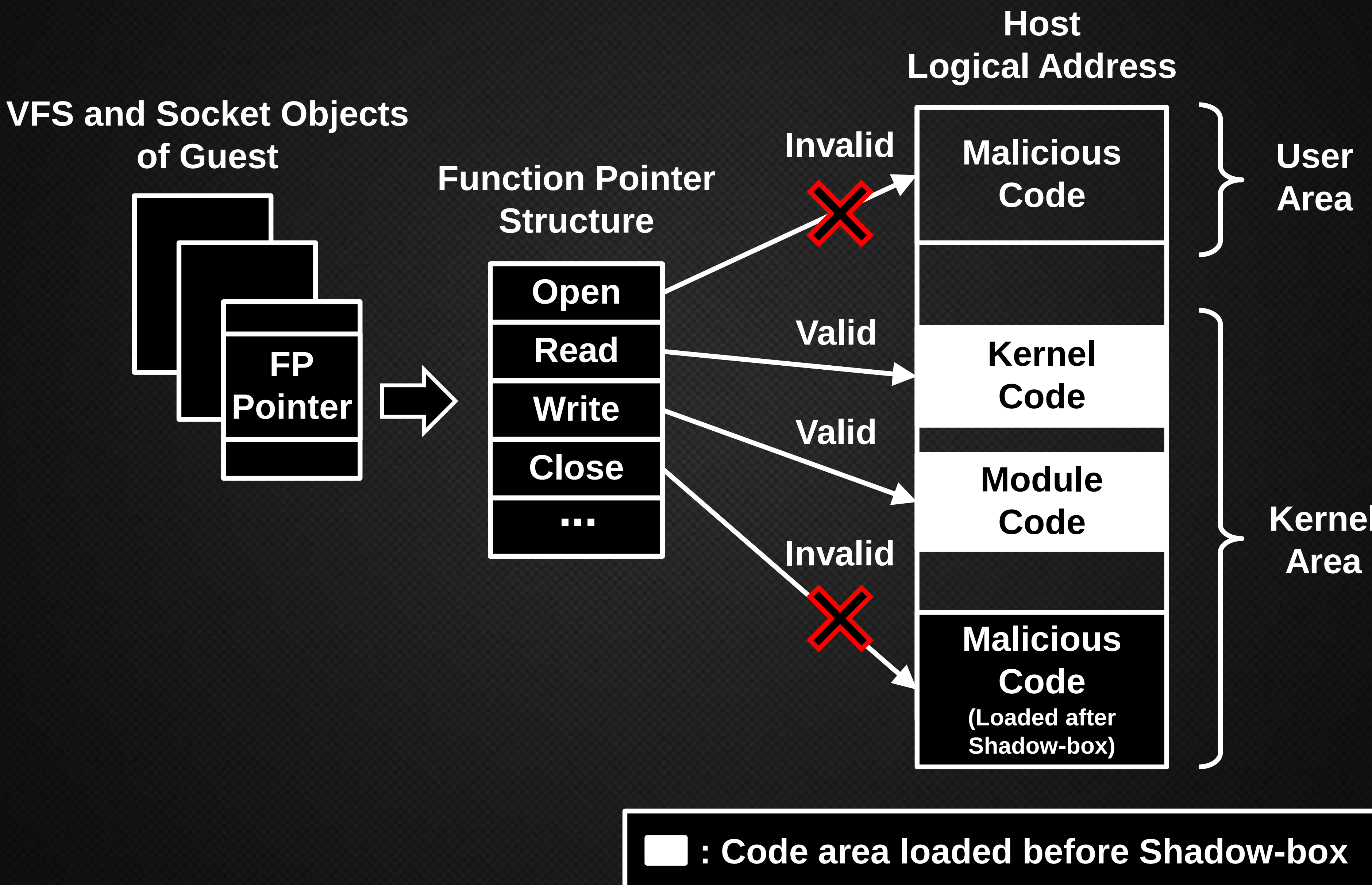
Static Kernel Object Protection (4)



Dynamic Kernel Object Protection (1)



Dynamic Kernel Object Protection (2)



Privileged Register Protection

- GDTR, LDTR and IDTR change interactions between kernel and user level
- IA32_SYSENTER_CS, IA32_SYSENTER_ESP, IA32_STAR, IA32-LSTAR and IA32_FMASK MSR also change them
- These privileged registers are rarely changed after boot!
- **So, Shadow-box**
  - Locks the privileged registers
  - Locks and Monitors GDT, LDT, and IDT table

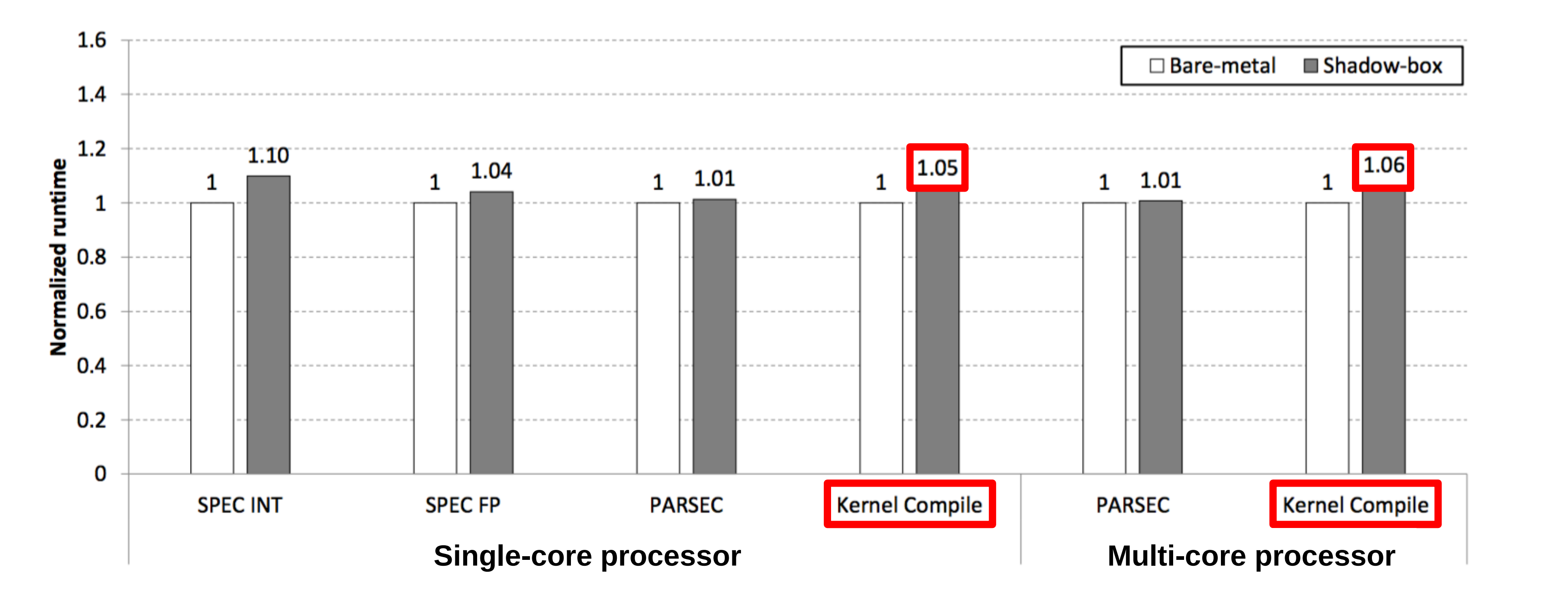
Rootkit Detection

- All rootkits are detected

Name	Detected?	Detected Point
EnyeLKM	√	code change, module hide
Adore-ng 0.56	√	function pointer change, module hide
Sebek 2.0	√	system table change, module hide
Suckit 2.0	√	system table change
kbeast	√	system table change, module hide

Performance Measurements of Prototype

- Application benchmarks show 1% ~ 10% performance overhead
  - 5.3% at kernel compile in single-core processor
  - 6.2% at kernel compile in multi-core processor



Results of Application Benchmark. Lower is better.
(Intel i7-4790 4core 8thread 3.6GHz, 32GB RAM, 512GB SSD)

Background

Design

Implementation

Lessons Learned

(and the Truth Previous Researches Did Not Tell You)

Demo. and Conclusion

(Black Hat Sound Bytes)

Ready to launch!

We deployed Shadow-box in REAL WORLD!

and …

We met

BEASTS of REAL WORLD!

(false positive, slow-down, system hang, etc.)

Previous researches did not tell us something important!

Lessons Learned - 1

- **Code is not immutable!**
  - Linux kernel has a CONFIG_JUMP_LABEL option!
  - If this option is set, Linux kernel patches itself on runtime!
  - Unfortunately, this option is set by default!

- **Solution**
  - Option 1: Add exceptional cases for mutable code pages
  - Option 2: If you can build kernel, Turn Off CONFIG_JUMP_LABEL option NOW!

Lessons Learned - 2

- **Cache type in EPT is very important!**
  - Linux system has some memory mapped I/O area
    - BIOS area, APIC area, PCI area, etc.
  - Misconfiguration makes various problems such as system hang, slow down, video mode change error, etc.

- **Solution**
  - Set uncacheable type by default
  - Set write-back type to "System RAM" area only!

Lessons Learned - 2

```
user$ cat /proc/iomem
00000000-00000fff : reserved
00001000-0009dbff : System RAM
0009dc00-0009ffff : reserved
000a0000-000bffff : PCI Bus 0000:00
000c0000-000ce7ff : Video ROM
  000c4000-000cbfff : PCI Bus 0000:00
000ce800-000cefff : Adapter ROM
000cf000-000cf7ff : Adapter ROM
000cf800-000d53ff : Adapter ROM
000d5800-000d67ff : Adapter ROM
000e0000-000fffff : reserved
  000f0000-000fffff : System ROM
00100000-ca336fff : System RAM
  01000000-01519400 : Kernel code
  01519401-018ecdff : Kernel data
  01a21000-01af2fff : Kernel bss
ca337000-cb68bfff : reserved
cb68c000-cbefefff : ACPI Non-volatile Storage
cbeff000-cbfcefff : ACPI Tables
cbfcf000-cbffffff : System RAM
d0000000-dfffffff : PCI MMCONFIG 0000 [bus 00-ff]
  d0000000-dfffffff : reserved
e0000000-f7ffbfff : PCI Bus 0000:00
  e0000000-f1ffffff : PCI Bus 0000:04
    e0000000-efffffff : 0000:04:00.0
    f0000000-f1ffffff : 0000:04:00.0
```

Uncacheable Cache Type by Default

Write-back Cache Type

Lessons Learned - 3

- **Multi-core environment is more complicated than you think!**
  - Each core modifies process list and module list concurrently
    - When H/W breakpoint exception occurred, other cores could be changing the lists already!
  - So, we need a mechanism for synchronizing lists

- **Solution**
  - Lock tasklist_lock and module_mutex of the guest while Shadow-box is checking the lists!


Now,
We have been operating
Shadow-box in REAL WORLD
SUCCESSFULLY !

Background

Design

Implementation

Lessons Learned

(and the Truth Previous Researches Did Not Tell You)

Demo. and Conclusion

(Black Hat Sound Bytes)

DEMO


[8.499767] intel_rapl: Found RAPL domain package
[8.499771] intel_rapl: Found RAPL domain core
[8.499773] intel_rapl: Found RAPL domain uncore
[8.499785] intel_rapl: RAPL package 0 domain package locked by BIOS
[8.505858] input: HDA Intel PCH Mic as /devices/pci0000:00/0000:00:1b.0/sound/card0/input12
[8.506342] input: HDA Intel PCH Headphone as /devices/pci0000:00/0000:00:1b.0/sound/card0/input13
[8.506783] input: HDA Intel PCH HDMI/DP,pcm=3 as /devices/pci0000:00/0000:00:1b.0/sound/card0/input14
[8.728906] iwlwifi 0000:03:00.0: CONFIG_IWLWIFI_DEBUG disabled
[8.728909] iwlwifi 0000:03:00.0: CONFIG_IWLWIFI_DEBUGFS enabled
[8.728910] iwlwifi 0000:03:00.0: CONFIG_IWLWIFI_DEVICE_TRACING enabled
[10.161003] Bluetooth: RFCOMM socket layer initialized
[10.161008] Bluetooth: RFCOMM ver 1.11
[11.796833] psmouse serio2: trackpoint: IBM TrackPoint firmware: 0x0e, buttons: 3/3
[11.993577] input: TPPS/2 IBM TrackPoint as /devices/platform/i8042/serio1/serio2/input/input9
[486.787670] shadow_box: module verification failed: signature and/or required key missing - tainting kernel
SHADOW-BOX
Lightweight Hypervisor-Based Kernel Protector
[486.826181] Shadow-box: CPU Count 4
[486.826185] Shadow-box: Booting CPU ID 0
[486.878993] Shadow-box: Protect Kernel Code Area
[486.904495] Shadow-box: [*] Complete
[486.904497] Shadow-box: Protect Module Code Area
[486.910824] Shadow-box: [*] Complete
[486.911132] Shadow-box: Framework Preinitialize
[486.912296] Shadow-box: [*] Complete
[486.947846] Shadow-box: Framework Initailize
[486.947881] Shadow-box: [*] Task count 216
[486.947882] Shadow-box: [*] Module count 88
[486.947883] Shadow-box: [*] Complete
[488.039134] Shadow-box: Lock IOMMU
[488.039732] Shadow-box: [*] Lock IOMMU complete
[488.063655] Shadow-box: Execution Complete
[488.063658] Shadow-box: ErrorCode: 0
user@Shadow-Box:~/Demo$

Future Work

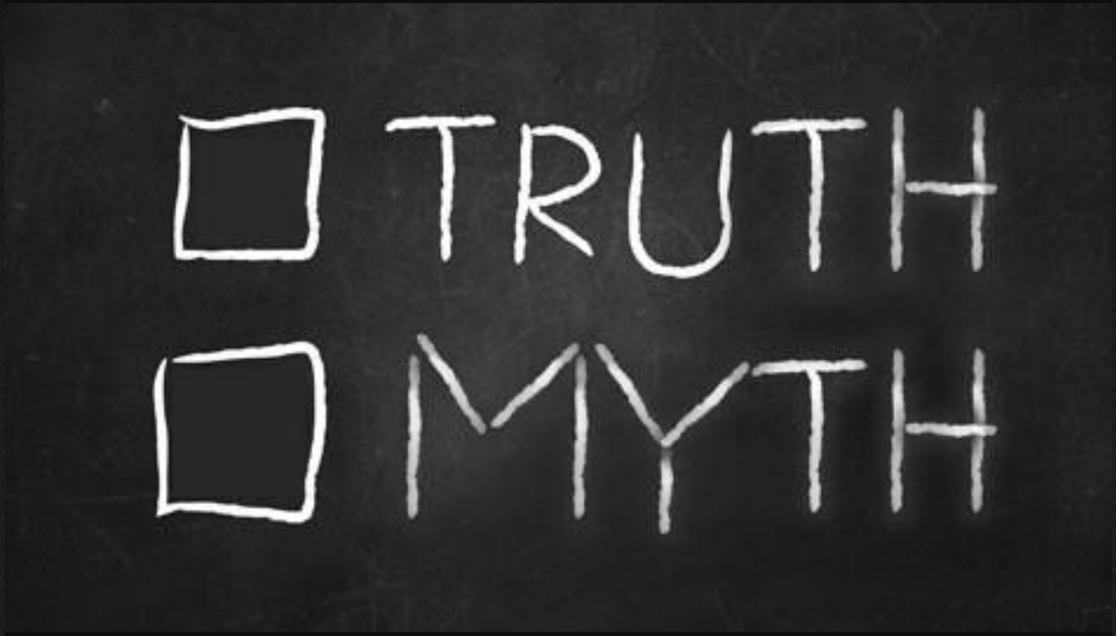
Linux	Linux
Shadow-Box	Shadow-Box
VT-x, VT-d (Virtualization Technology)	TrustZone (Virtualization Technology)
intel	ARM

Multi-platform Support!

Conclusion and Black Hat Sound Bytes

- **Kernel-level (Ring 0) threats should be protected in a more privileged level (Ring -1)**
  - We create Ring -1 level by using VT from scratch
- **Shadow-box is lightweight and practical**
  - Shadow-box uses less resource than existing mechanisms and protects kernel from rootkits
- **Real world is Serengeti!**
  - Real world is different from laboratory environment
  - You should have a strong mentality for defeating beasts of real world! or use Shadow-box instead!

THE CHOICE IS YOURS !



THANK YOU

Project : github.com/kkamagui/shadow-box-for-x86

Contact: hanseunghun@nsr.re.kr, @kkamagui1
ultract@nsr.re.kr, @ultractt