The Last Man Standing:

The Only Practical, Lightweight and Hypervisor-Based Kernel Protector Struggling with the Real World Alone

Seunghun Han

hanseunghun@nsr.re.kr

Who Am I?

- Senior security researcher at NSR (National Security Research Institute of South Korea)
- Speaker at
  - USENIX Security 2018
  - Black Hat Asia 2017, 2018
  - HITBSecConf 2016, 2017
  - KIMCHICON 2018
- Author of the book series titled “64-bit multi-core OS principles and structure, Vol.1&2”
- a.k.a kkamagui, @kkamagui1

Who Am I?

Offensive Researches

- IRON-HID: Create Your Own Bad USB Device
- I Don’t Want to Sleep Tonight: Subverting Intel TXT with S3 Sleep
- A Bad Dream: Subverting Trusted Platform Module while You Are Sleeping

Defensive Researches

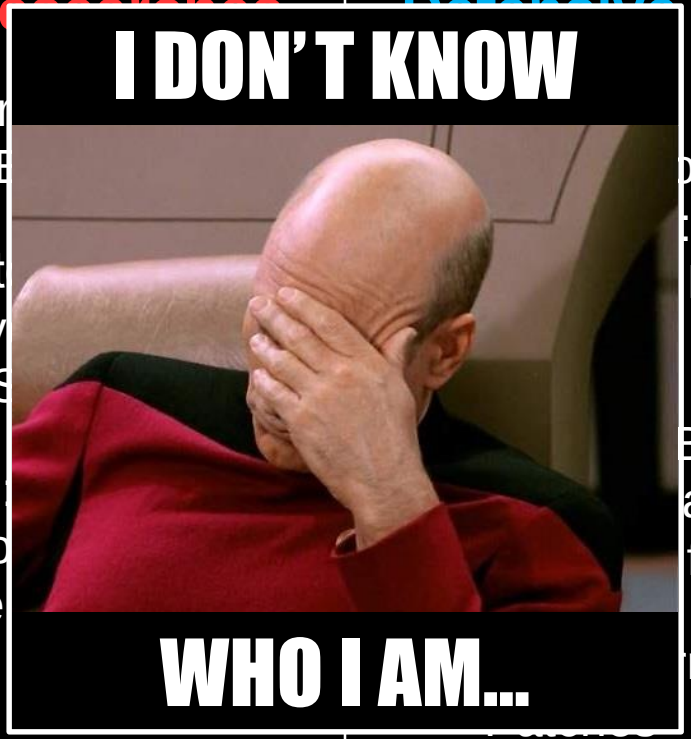
- Myth and Truth about Hypervisor-Based Kernel Protector: The Reason Why You Need Shadow-Box
- Shadow-Box v2: The Practical and Omnipotent Sandbox for ARM
- Linux Kernel Vulnerability Patches

Who Am I?

Offensive Researches | **Defensive Researches**

- IRON-HID: Cr… Truth about Own Bad USE…r-Based Kernel
- I Don't Want t… The Reason Tonight: Subv… Need Shadow-TXT with S3 S…
- A Bad Dream:… Box v2: The Trusted Platfo… and Omnipotent while You Are… for ARM

…rnel Vulnerability



Background

Analysis of Privilege Escalation Exploits

Design and Implementation of Gatekeeper
(with Shadow-Box)

Evaluation, Demo., Hunting the Beasts
(and Why I am Struggling with the Real World Beasts Alone)

Limitations and Conclusion

Background

Analysis of Privilege Escalation Exploits

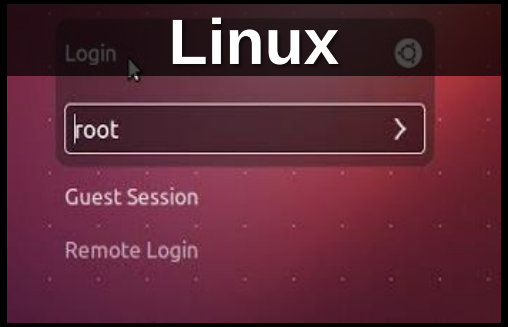
Design and Implementation of Gatekeeper
(with Shadow-Box)

Evaluation, Demo., Hunting the Beasts
(and Why I am Struggling with the Real World Beasts Alone)

Limitations and Conclusion

Linux Kernel Is Everywhere!

We Love Root!


Linux
Login
root
Guest Session
Remote Login


Android
Installing update...
Rooting with SuperSU 0.95
Installing temporary busybox
Mounting system...
Removing old superuser
Installing Superuser...
Setting Permission...
Symlinking...
Unmounting system...
Deleting temporary busybox
Root complete!
SuperSU 0.95


Smart TV


IP Camera
motion detected in region 00000002
md process was done!


T-Shirts!!
whoami
root
sudo rm -rf /
Don't Drink and Root

Exploit Vulnerabilities for Root!

Write Once, Run Anywhere!

- Java

Write Once, Run Anywhere and Pray!

- You and I

But, Our Prayers Went to /dev/null

- Kernel Address Space Layout Randomization (KASLR)
  - Makes exploits difficult to run Return-Oriented Programing (ROP) gadgets!
- Supervisor Mode Access Prevention (SMAP)
  - Makes **OUR** exploits hard to access user-level data!
- Supervisor Mode Execution Protection (SMEP)
  - Makes **OUR PRECIOUS** exploits super hard to execute user-level code!

But, Our Prayers Went to /dev/null

- Kernel Address Space Layout Randomization (KASLR)
  - Makes exploits difficult to run Return-Oriented Programing (ROP) gadgets!
- Supervisor Mode Access Pre
  - Makes **OUR** exploits hard t
    data!
- Supervisor Mode Execution
  (SMEP)
  - Makes **OUR PRECIOUS** ex
    to execute user-level code!

But, We Finally Got Root!

- KASLR
  - Is neutralized by **information leak vulnerabilities**!
- SMAP
  - Is defeated by **turning off SMAP bit of the CR4 register**!
- SMEP
  - Is also broken by **turning off SMEP bit of the CR4 register!**

But, We Finally Got Root!

- KASLR
  - Is neutralized by **information leak vulnerabilities**!
- SMAP
  - Is defeated by **turning off the CR4 register**!
- SMEP
  - Is also broken by **turning o the CR4 register**!

Background

Analysis of Privilege Escalation Exploits

Design and Implementation of Gatekeeper
(with Shadow-Box)

Evaluation, Demo., Hunting the Beasts
(and Why I am Struggling with the Real World Beasts Alone)

Limitations and Conclusion

Disclaimer (1)

- ***This presentation contains harmful information*** for offensive security researchers
  - Techniques for detecting abnormal privilege escalations are included!
- ***Evening party might be canceled*** if you listen to this presentation until the end
  - Because you could be busy to subvert my detection techniques or upgrade your exploits!

Disclaimer (2)

- If ***you want to go to the party*** ***after the presentation***
  - Check the GitHub repository and see it!

https://github.com/**kkamagui**/
shadow-box-for-x86

Analysis of (Our) Privilege Escalation Exploits (1)

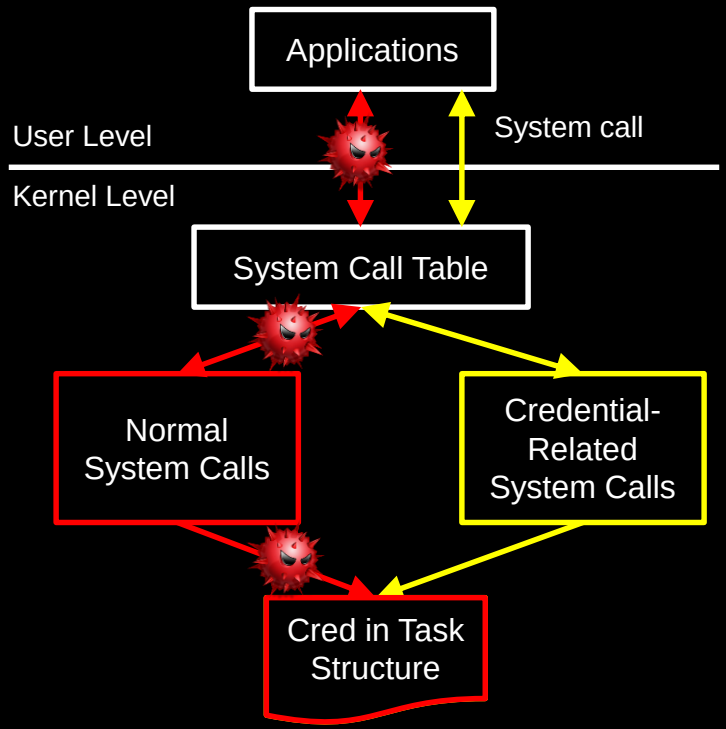
- (1) If vulnerabilities can read/write the kernel memory,
  - We overwrite **UID** in cred field of task structure with **ZERO**
  - ex) CVE-2017-16995: eBPF vulnerability
- (2) If vulnerabilities can hijack the control flow,
  - We make a **ROP chain** to execute below "**commit_creds(prepare_kernel_cred(0))**"!
  - ex) CVE-2017-1000112: UDP fragmentation offload (UFO) vulnerability

Analysis of (Our) Privilege Escalation Exploits (2)

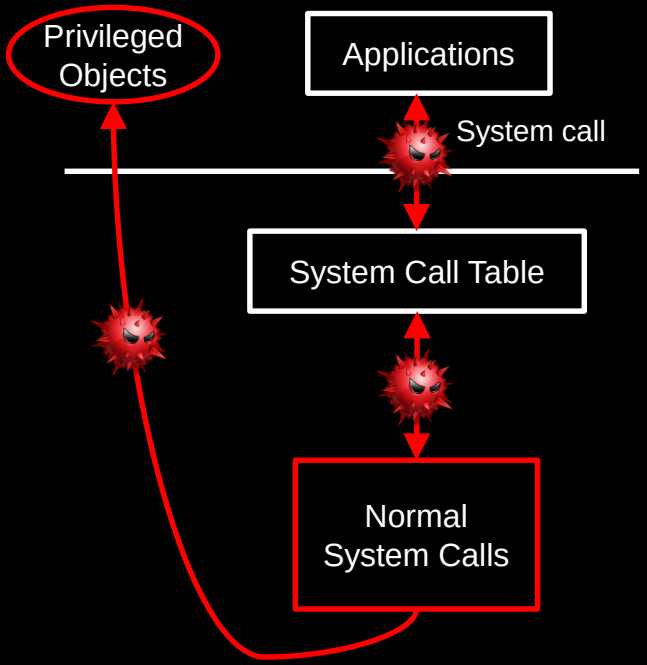
- (3) If vulnerabilities can read/write the privileged objects in user level,
  - We overwrite **/etc/crontab** or **/usr/bin/passwd** to get a root shell
  - ex) CVE-2016-4557: File descriptor Use-After-Free (UAF) vulnerability
  - ex) CVE-2016-5195: Copy-On-Write (COW) vulnerability a.k.a Dirty COW

Call Flows of (Our) Privilege Escalation Exploits

Call Flow of Case (1) and (2)



Call Flow of Case (3)



Detection and Prevention Techniques

- Case (1) and (2) change a cred of a task structure directly during normal system calls
  - Normal system calls don’t change the credential
  - If someone monitors system calls and changes o creds, he can detect and prevent abnormal privilege escalations!
- Case (3) does not change a cred directly
  - It is hard to be detected and prevented!
  - If someone monitors all processes created with UID = 0, he can trace and detect it (not perfect, but possible)

Detection and Prevention Techniques

- Case (1) and (2) change a cred of a task structure directly during normal system calls
  - Normal system calls don't change the credential
  - If someone monitors system calls and changes of creds, he can detect and prevent abnormal privilege escalations!
- Case (3) does not change a cred directly
  - It is hard to be detected and prevented!
  - If someone monitors all processes created with UID = 0, he can trace and detect it (not perfect, but possible)

I made a tool and named "Gatekeeper"

Background

Analysis of Privilege Escalation Exploits

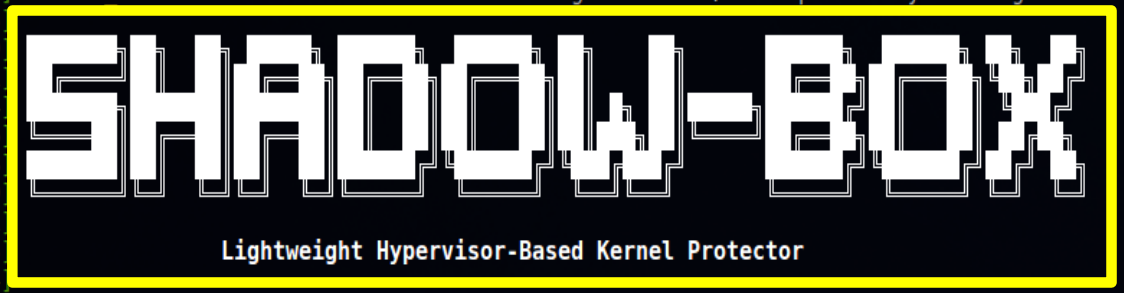
Design and Implementation of Gatekeeper
(with Shadow-Box)

Evaluation, Demo., Hunting the Beasts
(and Why I am Struggling with the Real World Beasts Alone)

Limitations and Conclusion


WANTED
Simple Design
≠
Simple Implementation
Let’s find something
to save us

```
[   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
[  486.810413]
[  486.810419]
[  486.810421]
[  486.810424]
[  486.810426]
[  486.810428]
[  486.810430]
[  486.810431]
[  486.810433]
[  486.810434]
```

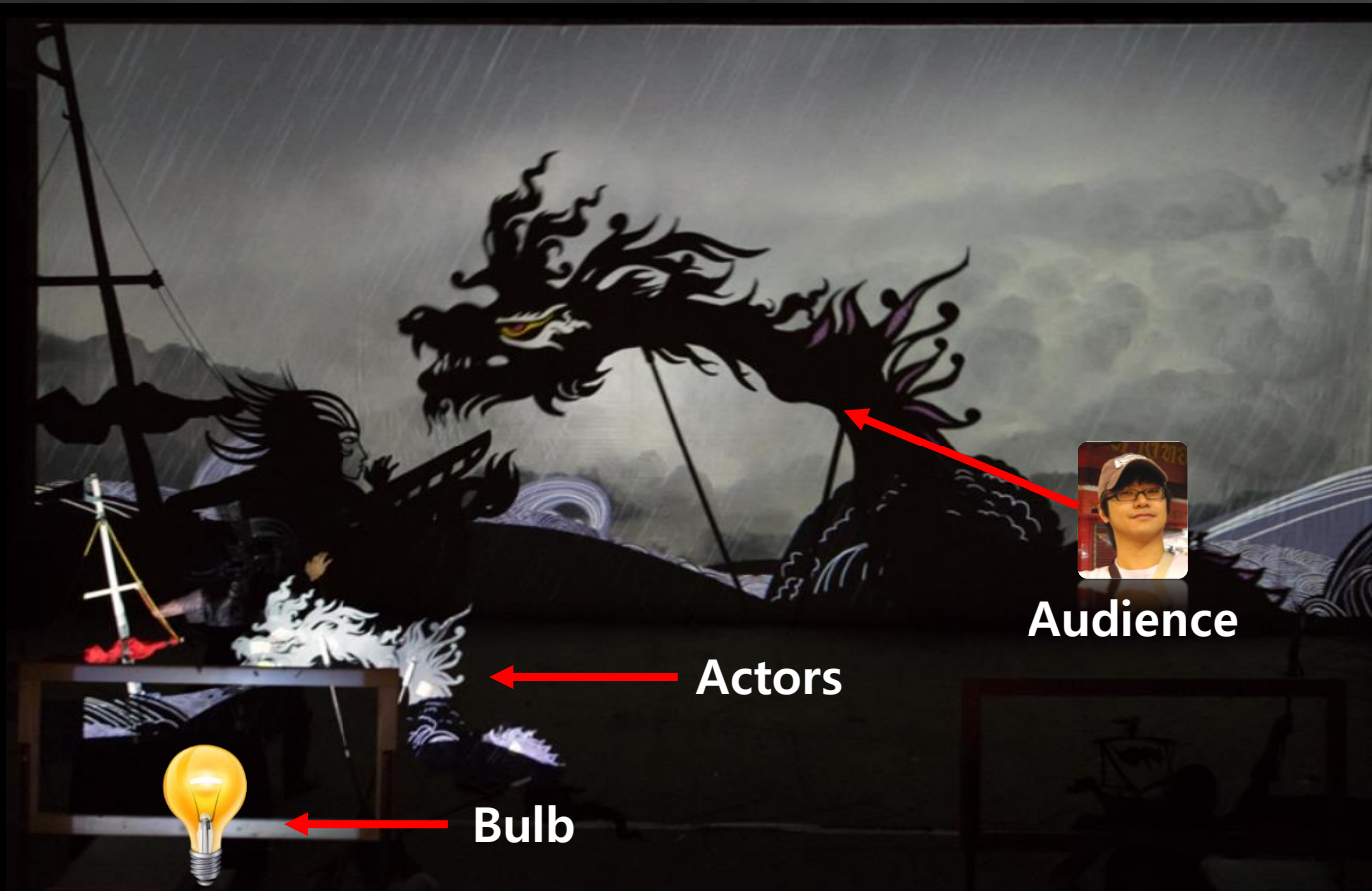



```
[  486.826181] Shadow-box: CPU Count 4
[  486.826185] Shadow-box: Booting CPU ID 0
[  486.878993] Shadow-box: Protect Kernel Code Area
[  486.904495] Shadow-box:     [*] Complete
[  486.904497] Shadow-box: Protect Module Code Area
[  486.910824] Shadow-box:     [*] Complete
[  486.911132] Shadow-box: 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$
```

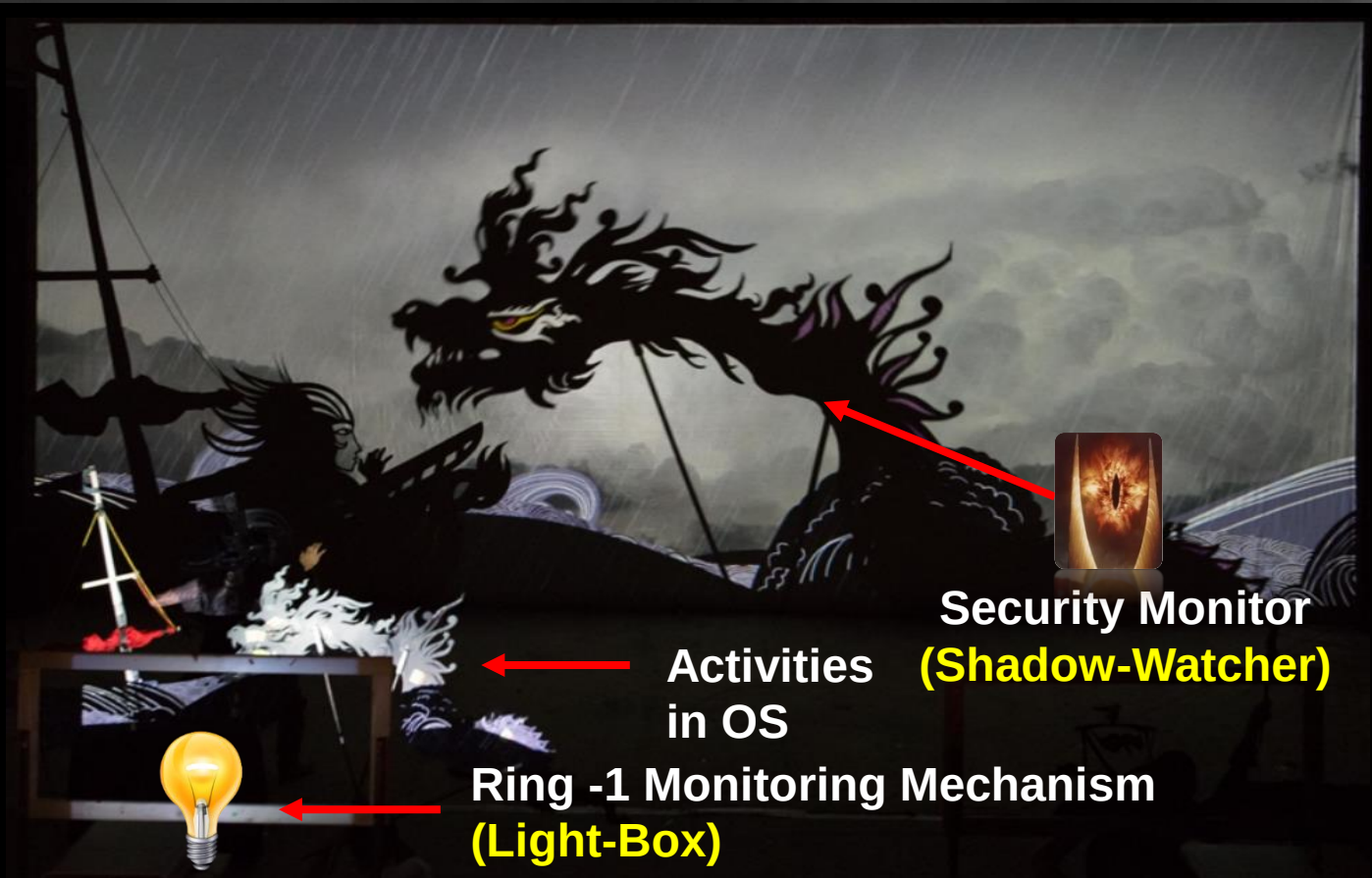
What Is Shadow-Box?

- Lightweight hypervisor-based kernel protector
  - Focuses on **ROOTKIT** detection and protection
  - has a small memory footprint
    - It shares kernel area and no multiple operating systems
- Practical and Portable
  - Out-of-box approach
    - No modification of the kernel code and data
  - Dynamic injection (Loadable Kernel Module, LKM)

Security Architecture in Shadow Play

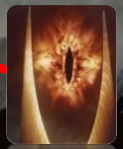


Security Architecture in Shadow Play



Security Architecture in Shadow Play

I named this architecture
"**Shadow-box**"

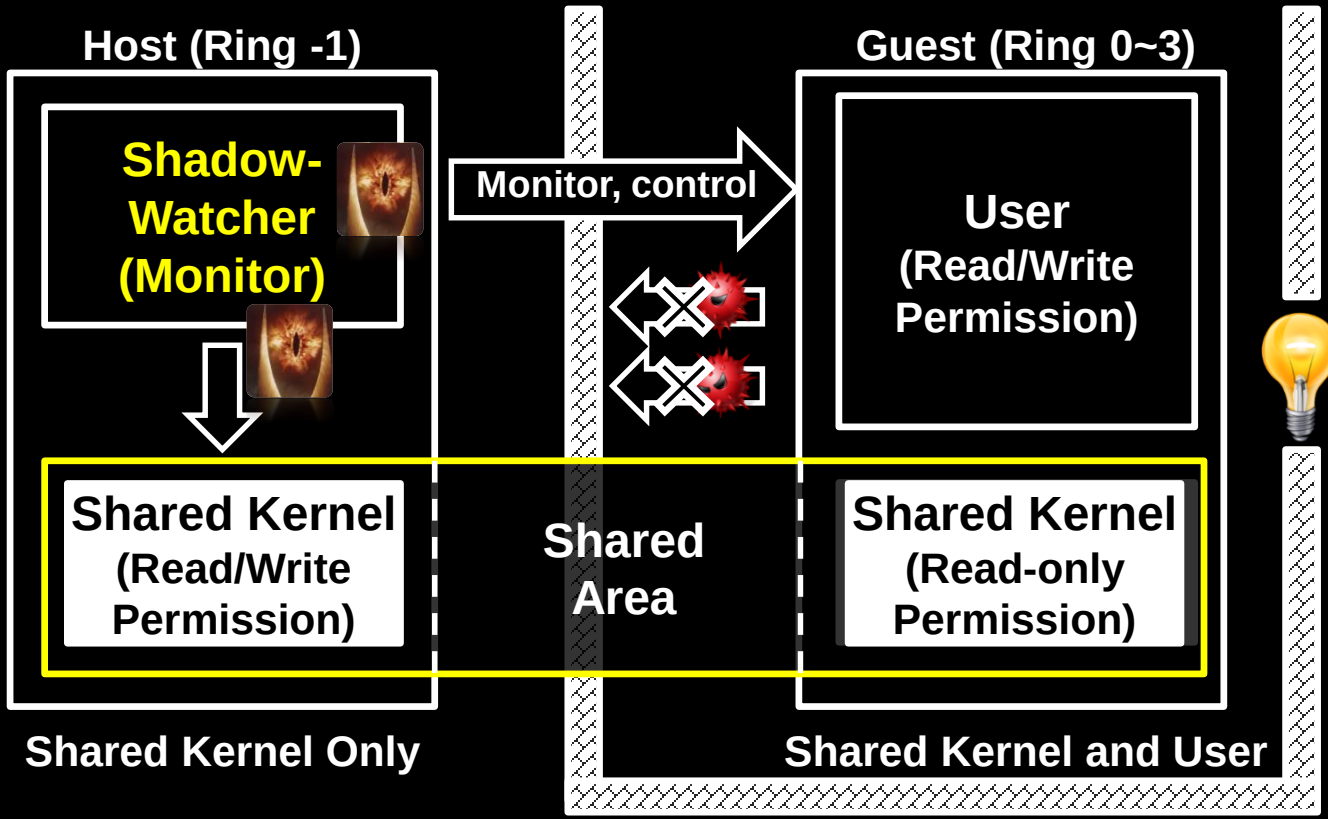


Security Monitor
(Shadow-Watcher)

Activities
in OS

Ring -1 Monitoring Mechanism
(Light-Box)

Architecture of Shadow-Box



Architecture of Light-Box

- Light-box, lightweight hypervisor,
  - Isolates worlds with memory protection technique in Intel Virtualization Technology (VT-x and VT-d)
  - **Shares the kernel area** between the host (Ring -1) and the guest (Ring 0~3)
  - Consumes fewer resources than existing mechanisms
  - Can be loaded at runtime

Architecture of Shadow-Watcher

- Shadow-watcher
  - Monitors the guest with Light-box
  - Checks if applications of the guest **modify static kernel objects or not** by an event-driven way
    - Code, system table, IDT table, etc
  - Checks the integrity of the guest by **introspecting dynamic kernel objects** by a periodic way
    - Process list, LKM list, function pointers of file system and socket

Detailed Information of Shadow-Box

- If you want to know more about Shadow-box, check the materials below

Design and Implementation of Gatekeeper Prototype

- Goal is to replace Shadow-watcher functions to Gatekeeper functions!
- Shadow-watcher uses four hardware breakpoints to monitor processes and LKMs
- For Gatekeeper, process and system call information is needed!
  - So, two breakpoints for LKMs are used to monitor system calls

Key Functions of Linux kernel

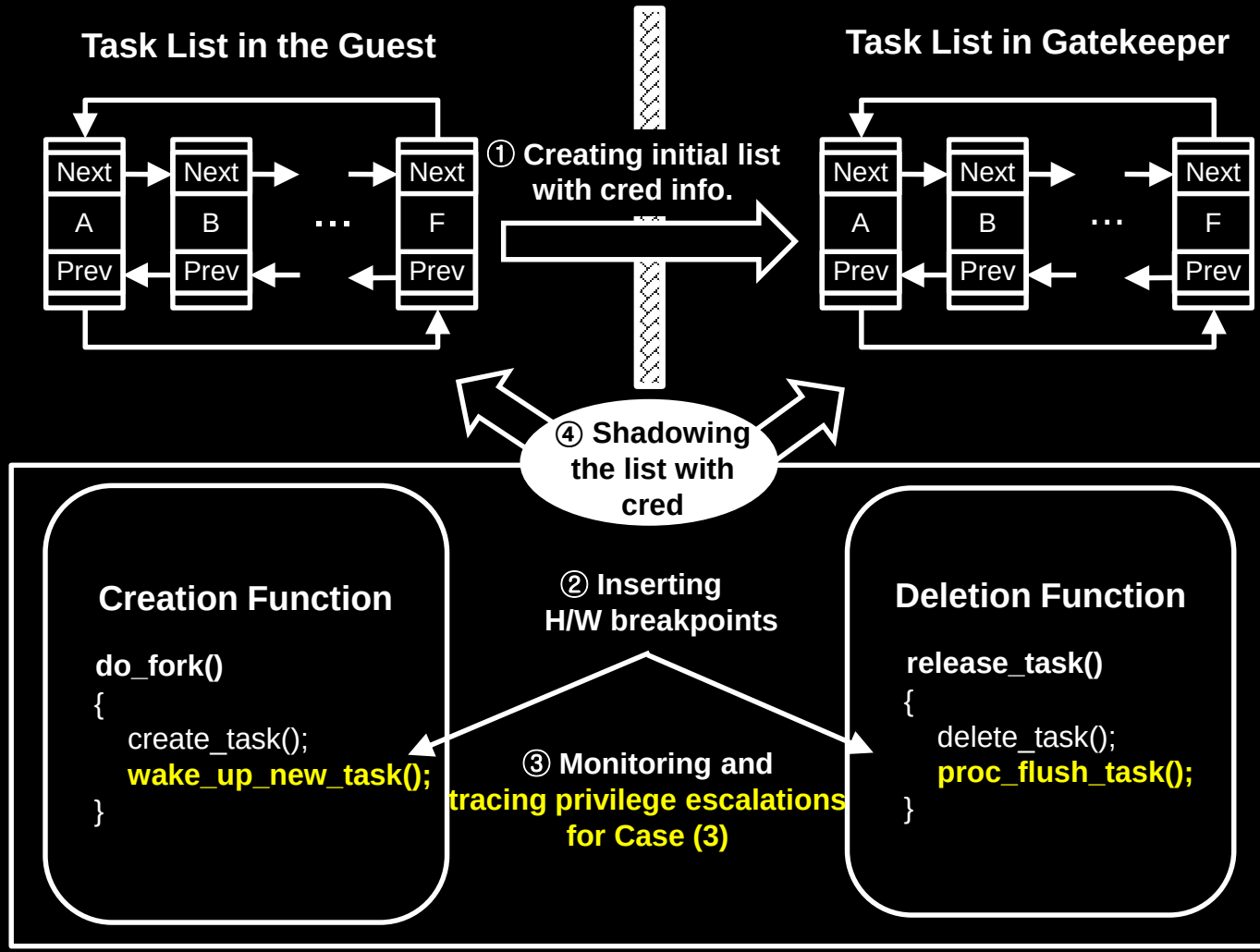
- wake_up_new_task()
  - Wakes up a new
- proc_flush_task()
  - Removes task from /proc
- entry_SYSCALL_64
  - Is an entry point of system calls
- commit_creds():
  - Installs new credentials

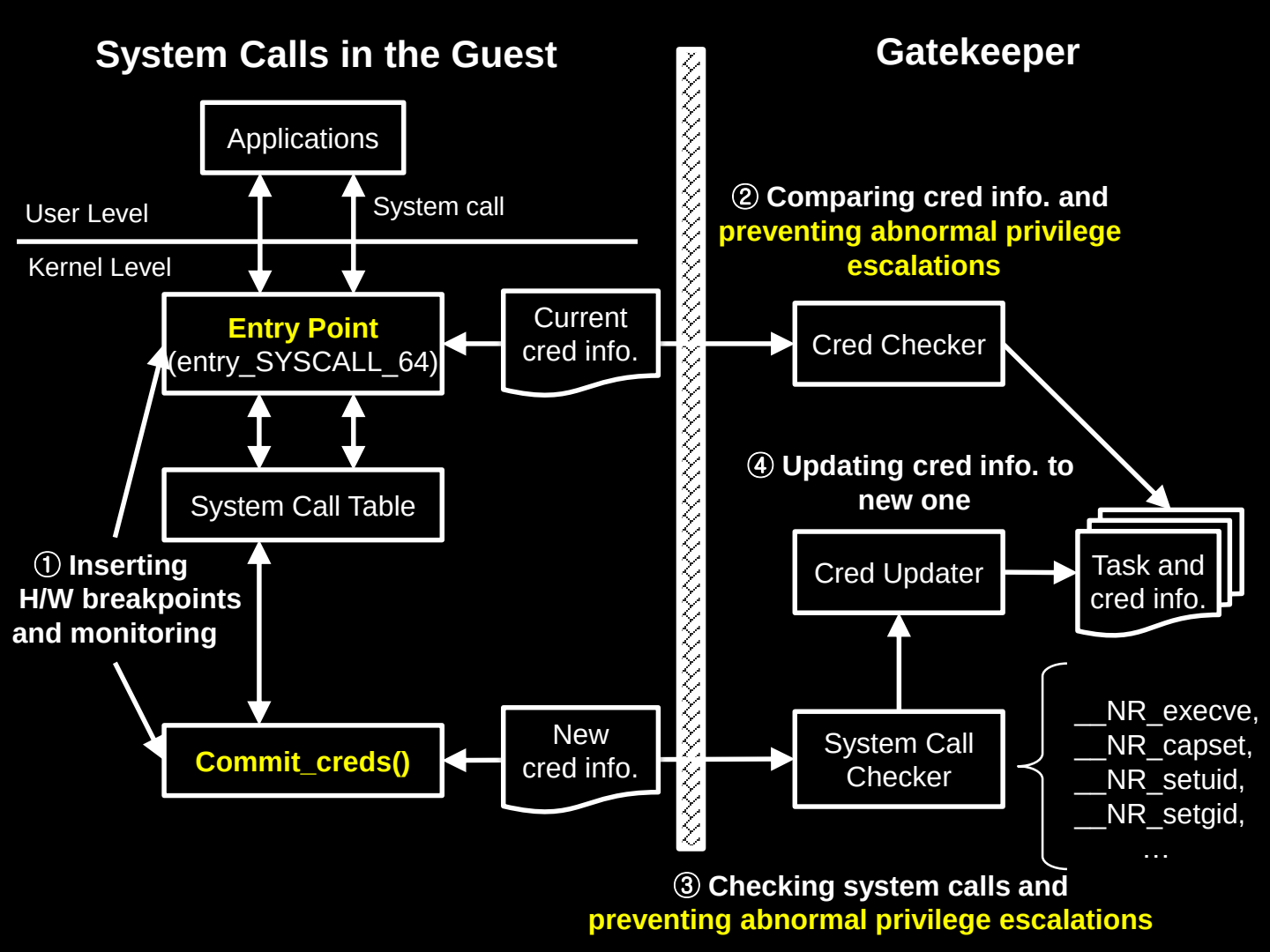
```
ffffffff81aab670 T commit_creds
ffffffff81ab5a20 T wake_up_new_task
ffffffff81cbf9e0 T proc_flush_task
ffffffff822d3e50 T entry_SYSCALL_64
```

Cred-Related System Calls

```
/* Check cred-related system calls. */
((syscall_number == __NR_capset)    || (syscall_number == __NR_execve)    ||
 (syscall_number == __NR_keyctl)    || (syscall_number == __NR_prctl)     ||
 (syscall_number == __NR_setfsgid)  || (syscall_number == __NR_setfsuid)  ||
 (syscall_number == __NR_setgid)    || (syscall_number == __NR_setuid)    ||
 (syscall_number == __NR_setregid)  || (syscall_number == __NR_setreuid)  ||
 (syscall_number == __NR_setresgid) || (syscall_number == __NR_setresuid) ||
 (syscall_number == __NR_setgroups) || (syscall_number == __NR_unshare))
```

- Only these system calls execute commit_creds()
  - Credential information is in the cred field of task structure
  - If the applications do not call them, cred **should be unchanged**!


Task List in the Guest
Task List in Gatekeeper
Next
A
Prev
Next
B
Prev
Next
F
Prev
① Creating initial list with cred info.
④ Shadowing the list with cred
Creation Function
do_fork()
{
create_task();
wake_up_new_task();
}
② Inserting H/W breakpoints
③ Monitoring and tracing privilege escalations for Case (3)
Deletion Function
release_task()
{
delete_task();
proc_flush_task();
}


System Calls in the Guest
Gatekeeper
Applications
User Level
System call
Kernel Level
Entry Point
(entry_SYSCALL_64)
Current
cred info.
Cred Checker
② Comparing cred info. and
preventing abnormal privilege
escalations
System Call Table
④ Updating cred info. to
new one
Cred Updater
Task and
cred info.
① Inserting
H/W breakpoints
and monitoring
Commit_creds()
New
cred info.
System Call
Checker
__NR_execve,
__NR_capset,
__NR_setuid,
__NR_setgid,
…
③ Checking system calls and
preventing abnormal privilege escalations

Background

Analysis of Privilege Escalation Exploits

Design and Implementation of Gatekeeper
(with Shadow-Box)

Evaluation, Demo., Hunting the Beasts
(and Why I am Struggling with the Real World Beasts Alone)

Limitations and Conclusion

Evaluation

- Ubuntu 16.04.01 with kernel 4.4.0-21-generic, 4.8.0-41-generic, 4.8.0-58-generic, and 4.10.0-28-generic

NO.	Privilege Escalation CVEs	Detection or Prevention?
1	CVE-2016-4557	**Detection**
2	CVE-2016-5195	**Detection**
3	CVE-2016-8655	**Prevention**
4	CVE-2017-6074	**Prevention**
5	CVE-2017-7308	**Prevention**
6	CVE-2017-1000112	**Prevention**
7	CVE-2017-16995	**Prevention**

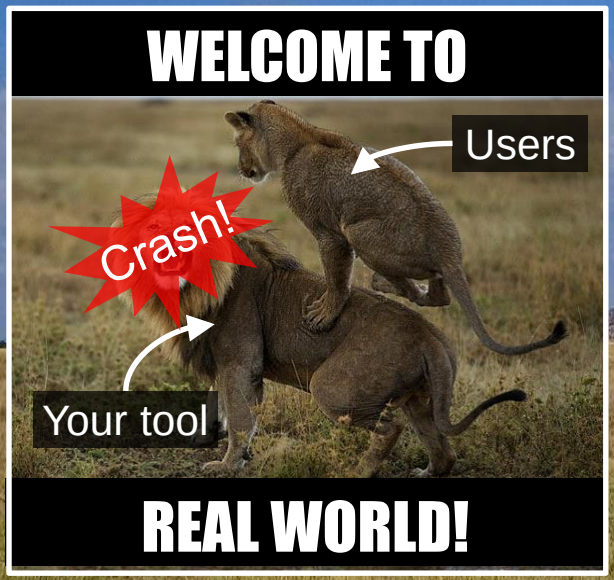
https://github.com/**kkamagui**/**linux-kernel-exploits**

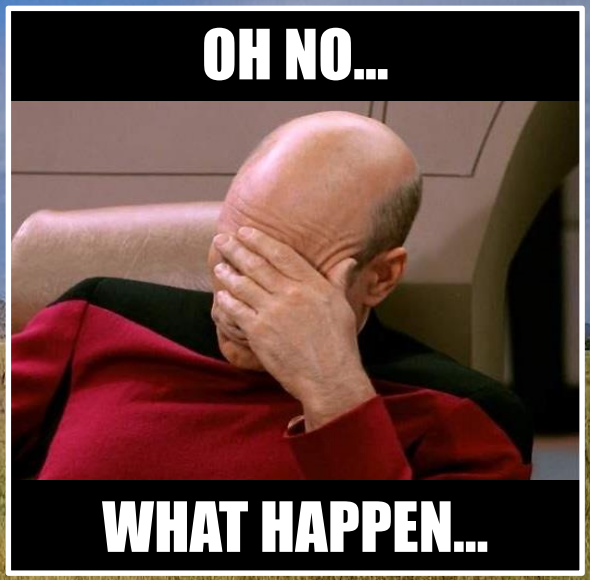

GATEKEEPER
Powered by
SHADOW-BOX
Lightweight Hypervisor-Based Kernel Protector v2.0.0
DEMO
[63.077716]
[63.077717]
[63.077718]
[63.077718]
[63.077719]
[63.077719]
[63.077720]
[63.077720]
[63.077721]
[63.077721]
[63.077722]
[63.077723]
[63.077723]
[63.077724]
[63.077725]
[63.077725]
[63.077726]
[63.077726]
[63.077727]
[63.088633] shadow-box: CPU C
[63.088634] shadow-box: Booti CPU
[63.102729] shadow-box: Prote Ker de Area
[63.131388] shadow-box:
[63.139345] shadow-box: Framework Preinitialize
[63.284596] shadow-box: [*] Complete
[63.305411] shadow-box: Protect Module Code Area
[63.316548] shadow-box: [*] Complete
[63.337863] shadow-box: Framework Initailize
[63.338027] shadow-box: [*] Task count 425
[63.338028] shadow-box: [*] Module count 118
[63.338029] shadow-box: [*] Complete
[63.348943] shadow-box: Lock IOMMU
[63.358808] shadow-box: [*] Intel Integrated Graphics is detected
[63.359222] shadow-box: [*] Lock IOMMU complete
[63.398196] shadow-box: Execution Complete
[63.398197] shadow-box: errorcode=0

Deployment, Real World, and the Beasts

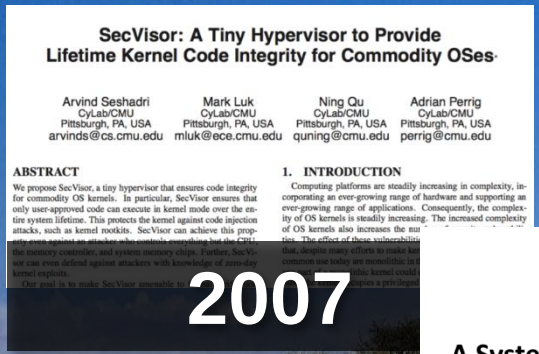
Real world is Serengeti and…

BEASTS live there!!!





There were many friends…



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

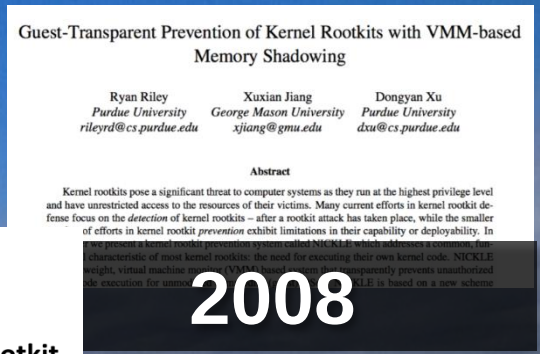
ABSTRACT

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.

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

2007



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 of efforts in kernel rootkit *prevention* exhibit limitations in their capability or deployability. In

characteristic of most kernel rootkits: the need for executing their own kernel code. NICKLE

weight, virtual machine

2008

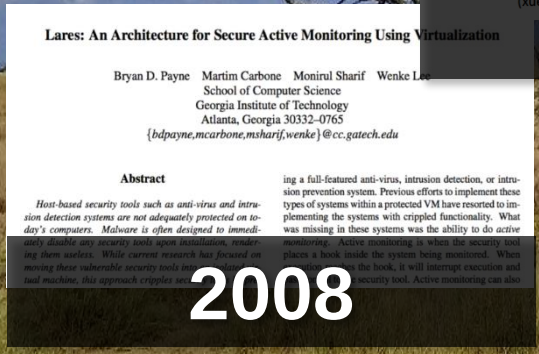
black hat ASIA 2016

NumChecker:

A System Approach for Kernel Rootkit Detection and Identification

Xueyang Wang, Ph.D.
Xiaofei (Rex) Guo, Ph.D.
{xueyang.wang, [illegible]}@[illegible].com *SPAM* intel.com

2016



Lares: An Architecture for Secure Active Monitoring Using Virtualization

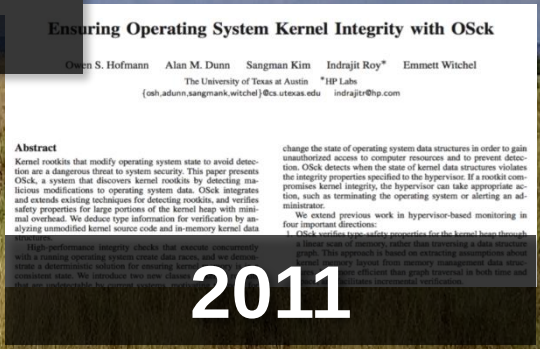
Bryan D. Payne, Martim Carbone, Monirul Sharif, Wenke Lee
School of Computer Science
Georgia Institute of Technology
Atlanta, Georgia 30332–0765
{bdpayne,mcarbone,msharif,wenke}@cc.gatech.edu

Abstract

Host-based security tools such as anti-virus and intrusion detection systems are not adequately protected on today's computers. Malware is often designed to immediately disable any security tools upon installation, rendering them useless. While current research has focused on moving these vulnerable security tools into a

ing a full-featured anti-virus, intrusion detection, or intrusion prevention system. Previous efforts to implement these types of systems within a protected VM have resorted to implementing the systems with crippled functionality. What was missing in these systems was the ability to do *active monitoring*. Active monitoring is when the security tool places a hook inside the system being monitored. When the hook, it will interrupt execution and security tool. Active monitoring can also

2008



Ensuring Operating System Kernel Integrity with OSck

Owen S. Hofmann, Alan M. Dunn, Sangman Kim, Indrajit Roy*, Emmett Witchel
The University of Texas at Austin, *HP Labs
{osh,adunn,sangmank,witchel}@cs.utexas.edu, indrajitr@hp.com

Abstract

Kernel rootkits that modify operating system state to avoid detection are a dangerous threat to system security. This paper presents OSck, a system that discovers kernel rootkits by detecting malicious modifications to operating system data. OSck integrates and extends existing techniques for detecting rootkits, and verifies safety properties for large portions of the kernel heap with minimal overhead. We deduce type information for verification by analyzing unmodified kernel source code and in-memory kernel data

change the state of operating system data structures in order to gain unauthorized access to computer resources and to prevent detection. OSck detects when the state of kernel data structures violates the integrity properties specified to the hypervisor. If a rootkit compromises kernel integrity, the hypervisor can take appropriate action, such as terminating the operating system or alerting an administrator.

We extend previous work in hypervisor-based monitoring in four important directions:

1. OSck verifies type safety properties for the kernel heap through

High-performance integrity checks that execute concurrently with a running operating system create data races, and we demonstrate a deterministic solution for ensuring kernel consistent state. We introduce two new classes

2011

And… Gone with the Laboratory!

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

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 prop-

1. INTRODUCTION

Computing platforms are

2007

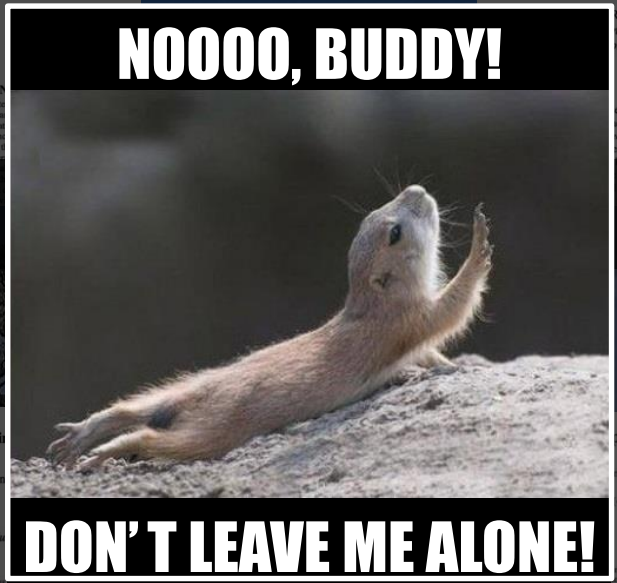
Guest-Transparent Prevention of Kernel Rootkits with VMM-based Memory Shadowing

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

Abstract

a significant threat to computer systems as they run at the highest privilege level access to the resources of their victims. Many current efforts in kernel rootkit de- *tion* of kernel rootkits – after a rootkit attack has taken place, while the smaller rnel rootkit *prevention* exhibit limitations in their capability or deployability. In

2008



Lares: An Architecture for Secure Active Monitoring Usi

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

Abstract

Host-based security tools such as anti-virus and intrusion detection systems are not adequately protected on today's computers. Malware is often designed to immedi-

ing a full-featured anti-virus sion prevention system. Previous efforts to implement these types of systems within a protected VM have resorted to implementing the systems with crippled functionality. What was missing in these systems was the ability to do *active*

2008

perating System Kernel Integrity with OSck

Alan M. Dunn Sangman Kim Indrajit Roy* Emmett Witchel
The University of Texas at Austin *HP Labs
{dunna,sangmank,witchel}@cs.utexas.edu indrajit@hp.com

system state to avoid detec- security. This paper presents OSck, a system that discovers kernel rootkits by detecting malicious modifications to operating system data. OSck integrates and extends existing techniques for detecting rootkits, and verifies safety properties for large portions of the kernel heap with minimal overhead. We deduce type information for verification by analyzing unmodified kernel source code and in-memory kernel data

change the state of operating system data structures in order to gain unauthorized access to computer resources and to prevent detection. OSck detects when the state of kernel data structures violates the integrity properties specified to the hypervisor. If a rootkit compromises kernel integrity, the hypervisor can take appropriate action, such as terminating the operating system or alerting an administrator.

We extend previous work in hypervisor-based monitoring in four important directions:

2011

Beasts and Hunting – Easy Level

- **(1) Code area was not immutable!**
  - I turned CONFIG_JUMP_LABEL off in kernel config
- **(2) Cache types of memory areas were misconfigured!**
  - I set uncacheable type by default and write-back type to “System RAM” area according to /proc/iomem

Beasts and Hunting – Normal Level

- **(1) Machine check error (MCE) exception could not handled in the host (Ring -1)!**
  - I enabled MCE polling feature in the kernel!
  - **I found a kernel bug (?!)** and contributed a patch to kernel
- **(2) New instructions such as XSAVES and XRSTORS were added to new CPUs!**
  - I added new instruction handlers according to Intel manuals

Beasts and Hunting – Hardcore Level

- **MELTDOWN** was revealed!
  - And Page Table Isolation (PTI) feature was added to the kernel!
    - PTI flushes kernel-level page tables every privilege-level transition! (ring 0 ← → ring 3)
  - I read the meltdown paper and kernel source code **AGAIN** and **AGAIN**!
  - Finally, I could change memory mapping of Shadow-box according to PTI patches

https://github.com/**kkamagui**/**shadow-box-for-x86**/**issues**

Background

Analysis of Privilege Escalation Exploits

Design and Implementation of Gatekeeper
(with Shadow-Box)

Evaluation, Demo., Hunting the Beasts
(and Why I am Struggling with the Real World Beasts Alone)

Limitations and Conclusion

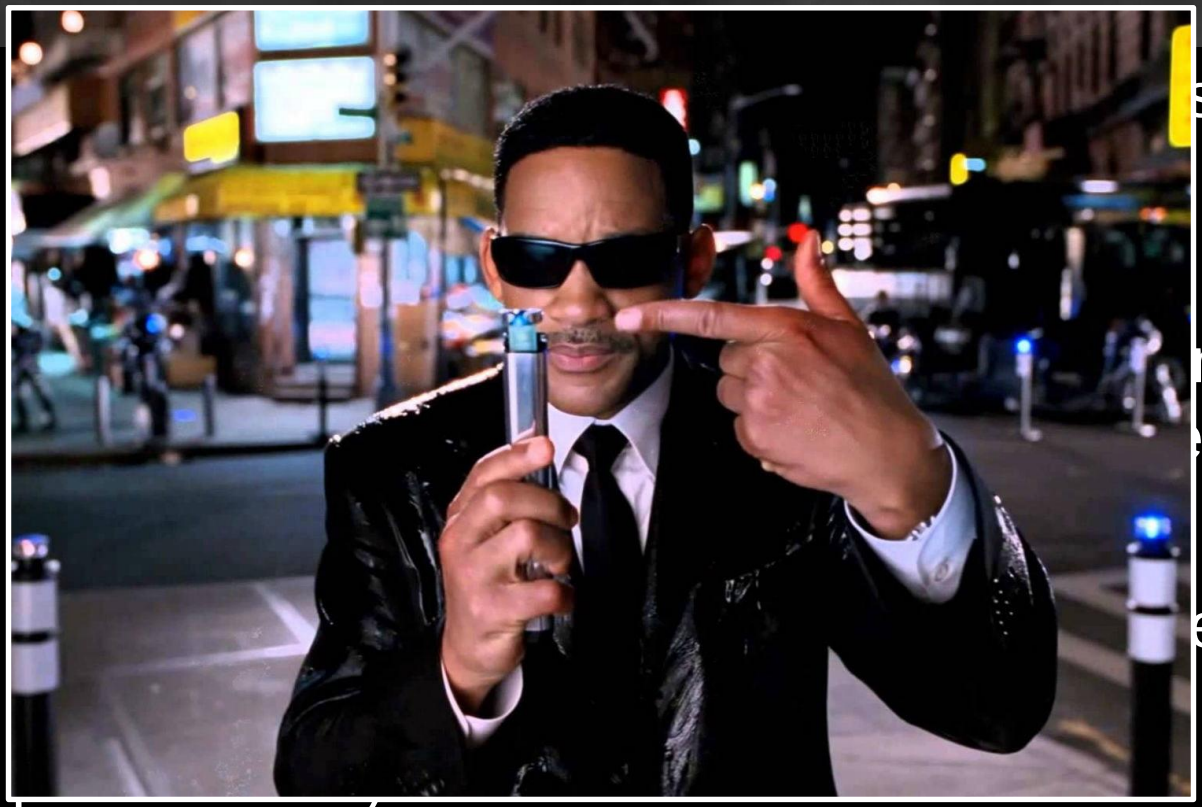
Limitations of Gatekeeper Prototype

- Gatekeeper only works in 64bit mode
  - It cannot monitor system calls with 32bit mode (compat) and interrupt mode
- Gatekeeper's prevention feature can be avoided
  - If you make complicate ROP chain to change privilege objects (not cred) in user level
- Gatekeeper can be neutralized
  - If cred-related system calls have vulnerabilities

Conclusion

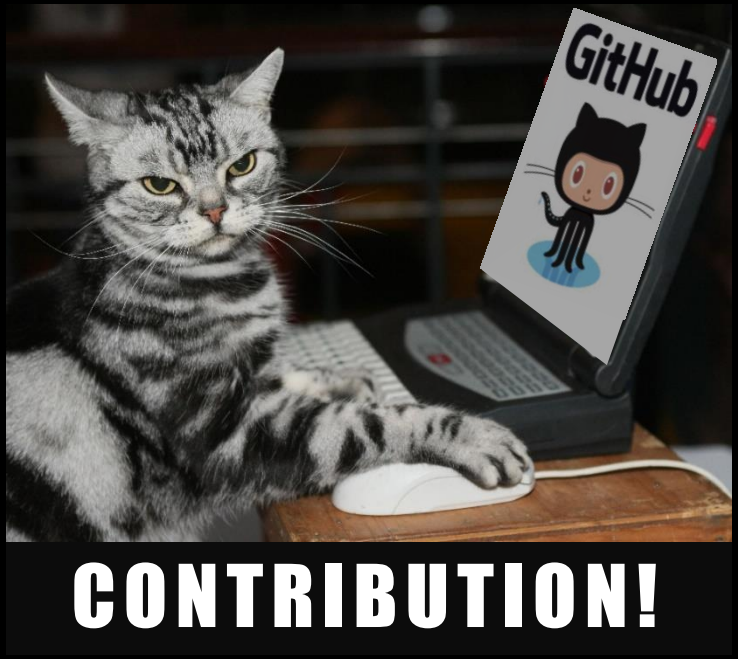
- Three types of privilege escalation exploits can be detected and prevented by **Gatekeeper (based on Shadow-box)**
- **Shadow-box is the last man standing** and struggling with the real world beasts alone (unfortunately)
- **Exploits might evolve** to change privileged objects, not cred (because of this presentation)

Conclusion



- s
- nd
- ed

Questions?



Project : https://github.com/kkamagui/shadow-box-for-x86
Contact: hanseunghun@nsr.re.kr, @kkamagui1

Reference

- “Myth and Truth about Hypervisor-Based Kernel Protector: The Reason Why You Need Shadow-Box”, Black Hat Asia 2017
- “Shadow-Box: The Practical and Omnipotent Sandbox”, HITBSecConf 2017
- “Shadow-Box v2: The Practical and Omnipotent Sandbox for ARM”, Black Hat Asia 2018
- “Proposal of a Method to Prevent Privilege Escalation Attacks for Linux Kernel”, Linux Security Summit 2017
- Shadow-box and Gatekeeper, https://github.com/kkamagui/shadow-box-for-x86
- Linux Kernel Exploits, https://github.com/kkamagui/linux-kernel-exploits