Follow the WhiteRabbit: Towards Consolidation of On-the-Fly Virtualization and Virtual Machine Introspection

IFIP SEC 2018

Sergej Proskurin, [1] Julian Kirsch, [1] and Apostolis Zarras [2]
[1]Technical University of Munich
[2]Maastricht University

19.09.2018

“Follow the white rabbit.”

— The Matrix

“Follow the white rabbit.”

— The Matrix

Introduction & Background

Technical University of Munich

Modern Operating Systems (OSes) provide a large attack surface

- 334 system calls in Linux kernel v4.18 (excluding compatibility system calls for 32-bit)
- Malware can gain the same privileges as OSes

→ Bypass or disable security mechanisms

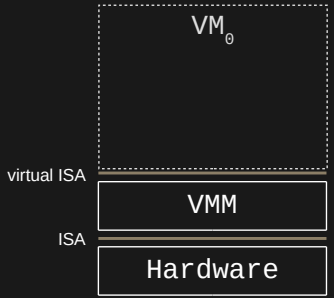
Move security applications out of the OS [1]:

- Place security applications into a small environment with higher privileges

→ Employ system virtualization

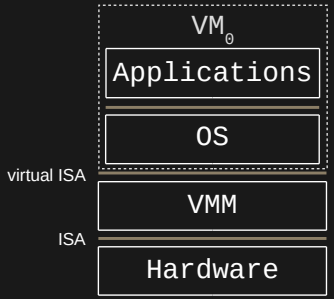
Some background on system virtualization:

> "x86 virtualization is about basically placing another nearly full kernel, full of new bugs, on top of a nasty x86 architecture which barely has correct page protection. Then running your operating system on the other side of this brand new pile of s****. [...] That's all x86 virtualization is."
>
> — An openbsd-misc email by Theo de Raadt



Some background on system virtualization:

> "x86 virtualization is about basically placing another nearly full kernel, full of new bugs, on top of a nasty x86 architecture which barely has correct page protection. Then running your operating system on the other side of this brand new pile of s****. [...] That's all x86 virtualization is."
>
> — An openbsd-misc email by Theo de Raadt



Some background on system virtualization:

> "x86 virtualization is about basically placing another nearly full kernel, full of new bugs, on top of a nasty x86 architecture which barely has correct page protection. Then running your operating system on the other side of this brand new pile of s****. [...] That's all x86 virtualization is."
>
> — An openbsd-misc email by Theo de Raadt

System virtualization employed for different purposes

- Malware detection [4] and analysis [2, 5]
- System integrity validation [6]

Benefits of system virtualization

- Narrow attack surface
- Strong isolation capabilities
- Complete view over the VM's state

System virtualization employed for different purposes

- Malware detection [4] and analysis [2, 5]
- System integrity validation [6]

Benefits of system virtualization

- Narrow attack surface
- Strong isolation capabilities
- Complete view over the VM's state

How can we analyze OS internals from the outside?

Employ Virtual Machine Introspection (VMI) [4] techniques

Virtual Machine Introspection [4]

- Analyze and manipulate the guest OS state from the outside of the VM
- Binary state of guest OSes needs interpretation
  - Map the binary state to OS data structures
  - → Lack of semantic information

Excerpt of the OS binary state

```
1 20 00 00 00 00 00 00 00 FF FF FF FF FF FF 00 00
2 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
3 00 80 54 0C 00 00 FF FF 02 00 00 00 00 01 40 00
4 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
5 01 00 00 00 01 00 00 00 10 00 00 00 00 00 00 00
6 BA EC FE FF 00 00 00 00 80 CF 66 28 00 80 FF FF
7 [...]
```

The Semantic Gap problem [1]

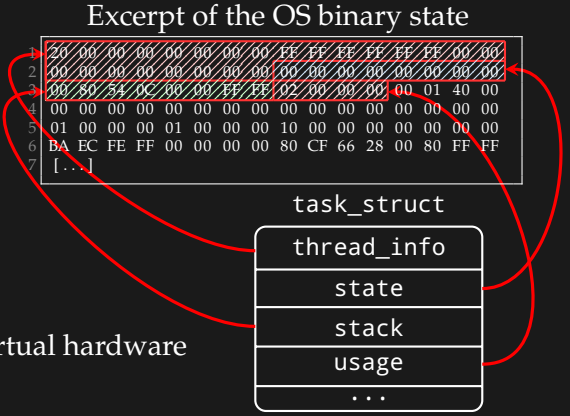
- Use semantic information of the guest OS and virtual hardware to bridge the Semantic Gap [7]

Virtual Machine Introspection [4]

- Analyze and manipulate the guest OS state from the outside of the VM
- Binary state of guest OSes needs interpretation
  - Map the binary state to OS data structures
  - → Lack of semantic information

The Semantic Gap problem [1]

- Use semantic information of the guest OS and virtual hardware to bridge the Semantic Gap [7]



Motivation

Technical University of Munich

Conventional VMI frameworks

- Employ VMI-aware VMMs (Xen, KVM, etc.)

Issue: Target systems must be explicitely set up for VMI before operation

- Increases the administrative overhead
- Constraints employment of VMI

Motivation

Technical University of Munich

Conventional VMI frameworks

- Employ VMI-aware VMMs (Xen, KVM, etc.)

Issue: Target systems must be explicitely set up for VMI before operation

- Increases the administrative overhead
- Constraints employment of VMI

Idea: Combine VMI along with on-the-fly virtualization

→ WhiteRabbit VMI framework for forensic analysis

- Based on the idea of the Blue Pill rootkit [9]
- Employs Intel VT-x and ARM virtualization extensions

Goals

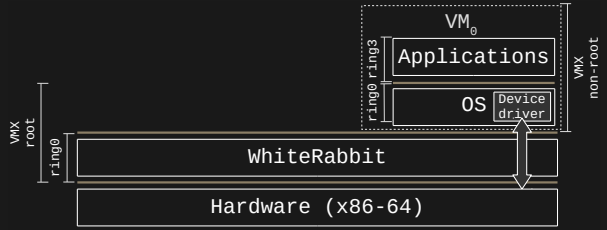
Technical University of Munich

(1) Virtual Machine Monitor
- Take over control of a running Linux (idea not limited to any OS)

(2) Hiding from (split-personality) malware in memory
- Employ Second Stage Address Translation

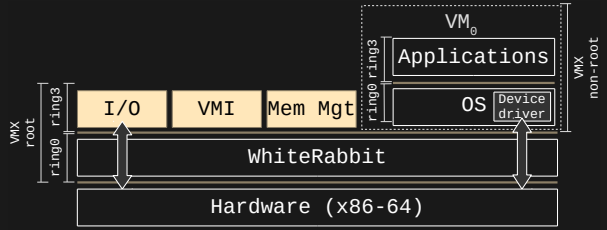
(3) (Remote) Virtual Machine Introspection
- Expose LibVMI interface to local or remote applications

Goal 1: WhiteRabbit VMM



Microkernel architecture designed for on-the-fly virtualization

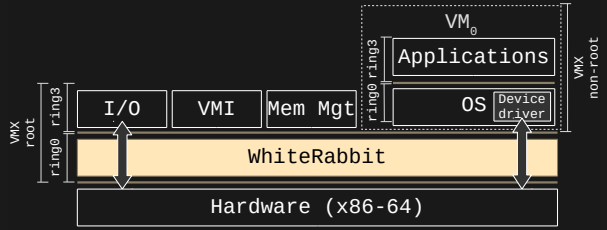
Goal 1: WhiteRabbit VMM



Microkernel architecture designed for on-the-fly virtualization

- Subsystems placed in user space (ring 3 on Intel; EL0 on ARM)
- I/O drivers isolated from the guest
  - Establish a secure communication channel
  - Leverage unused I/O devices or hardware multiplexing (e.g., Intel VT-d, ARM SMMU)

Goal 1: WhiteRabbit VMM



Microkernel architecture designed for on-the-fly virtualization

- Only essential functionality in ring 0 on Intel (VMX root) and EL2 on ARM
  - Reduced size and complexity of the VMM
- Can be deployed in an OS-dependent or OS-independent way

Goal 1: On-the-Fly Virtualization

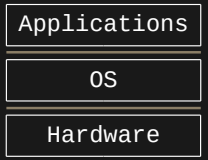
Technical University of Munich

A VMM distributes its tasks across [8] :

- Allocator
- Dispatcher
- Interpreter

Goal 1: On-the-Fly Virtualization

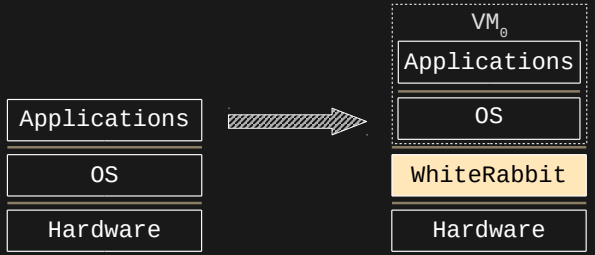
The Allocator



The allocator moves a running OS into a virtual environment

Goal 1: On-the-Fly Virtualization

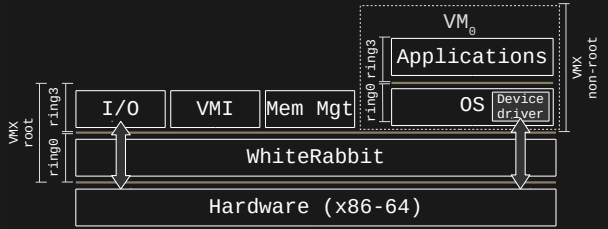
The Allocator



The allocator moves a running OS into a virtual environment

Goal 1: On-the-Fly Virtualization

The Allocator

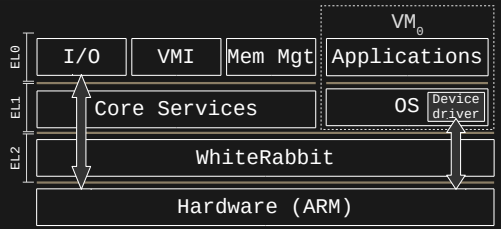


The allocator moves a running OS into a virtual environment

- Intel: Initializes the Intel VT-x defined VMCS data structure
  - Guest state initialized with the current system state
  - Set up host state representing the VMM
  - Initiates virtualization

Goal 1: On-the-Fly Virtualization

The Allocator



The allocator moves a running OS into a virtual environment

- ARM: Uses the hypervisor stub to take control of the exception levels in EL2
  - Boot loader launches OS kernel in EL2 before entering EL1
  - OS installs a general purpose hypervisor stub in EL2
  - Use the hypervisor stub to place running OS into a VM

Goal 1: WhiteRabbit VMM

The Dispatcher & The Interpreter

The dispatcher determines which task to perform next

- Can be regarded as a task scheduler
- Dispatcher triggered on every VM exit
- Relays tasks to subsystems or the interpreter

The interpreter simulates behavior of guest events

- Emulation routines for trapped instructions and events
- Is leveraged for VMI purposes

Goal 2: Hiding Techniques

Technical University of Munich

Split-personality malware

- Employ anti-virtualization to reveal a VMM (red pills)

Perfect VM transparency is unfeasible [3]

- Insufficient to reveal virtual environments alone!

More interesting to know whether the system is being analyzed

→ Hide analysis artifacts from the guest

Goal 2: Hiding Techniques

Second Level Address Translation

Technical University of Munich

Harden disclosure of the analysis framework in memory

- Leverage Second Level Address Translation
- Intercept and redirect access to memory used by WhiteRabbit

Goal 2: Hiding Techniques

Second Level Address Translation

Harden disclosure of the analysis framework in memory

- Leverage Second Level Address Translation
- Intercept and redirect access to memory used by WhiteRabbit

Issue: If WhiteRabbit deployed as a kernel module

- First instruction of the VM is one of the last instructions of module initialization
- The `struct task_struct` filled with remaining information after module loading

→ Both regions are masked within Second Level Address Translation (SLAT) tables

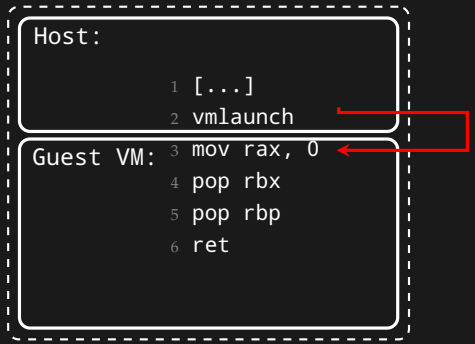
Goal 2: Hiding Techniques

Second Level Address Translation

Technical University of Munich

Harden disclosure of the analysis framework in memory

- Leverage Second Level Address Translation
- Intercept and redirect access to memory used by WhiteRabbit



Module Initialization Routine

Issue: If WhiteRabbit deployed as a kernel module

- First instruction of the VM is one of the last instructions of module initialization
- The `struct task_struct` filled with remaining information after module loading

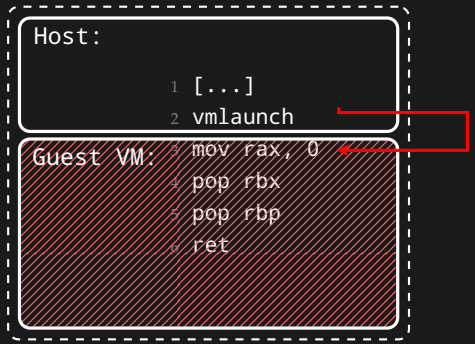
→ Both regions are masked within SLAT tables

Goal 2: Hiding Techniques

Second Level Address Translation

Harden disclosure of the analysis framework in memory

- Leverage Second Level Address Translation
- Intercept and redirect access to memory used by WhiteRabbit



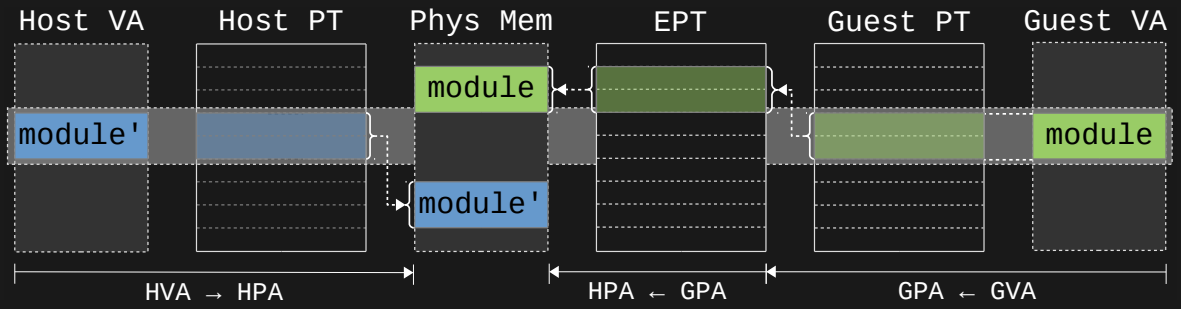
Issue: If WhiteRabbit deployed as a kernel module

- First instruction of the VM is one of the last instructions of module initialization
- The `struct task_struct` filled with remaining information after module loading

→ Both regions are masked within SLAT tables

Goal 2: Hiding Techniques

SLAT & Module Relocation



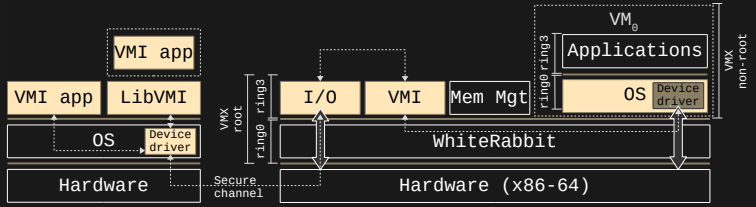
Relocate code and data segments of WhiteRabbit

- `module'` and `module` mapped to the same virtual address space (guest and host)

Remove the kernel module inside of the guest OS

- `module` returns a negative value (visible via `dmesg`)
- Better: Use a work-queue to initiate a clean module destruction

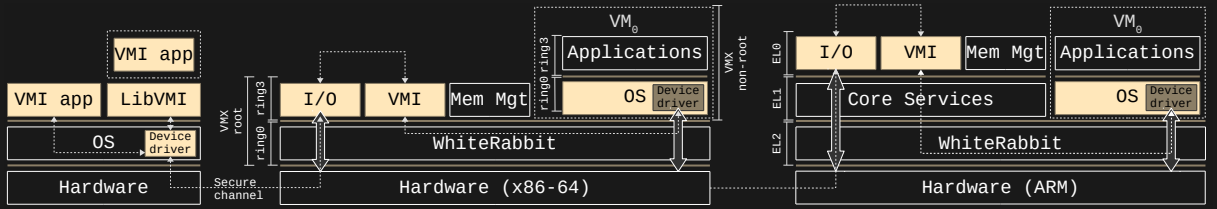
Goal 3: Virtual Machine Introspection



WhiteRabbit offers a LibVMI-compatible interface towards remote clients

- WhiteRabbit acts as a vehicle for generic VMI tools
- Remote VMI tools perform arbitrary VMI on the virtualized OS
  - Need for a secure communication channel

Goal 3: Virtual Machine Introspection



WhiteRabbit offers a LibVMI-compatible interface towards remote clients

- WhiteRabbit acts as a vehicle for generic VMI tools
- Remote VMI tools perform arbitrary VMI on the virtualized OS
  - Need for a secure communication channel

Evaluation

Performance: Virtualization Overhead

VMM	CPU	Memory
WhiteRabbit	0.74%	3.48%
Xen	2.66%	3.48%
Linux KVM	4.02%	4.92%

Table: Averaged results of CPU- and memory-intensive macro-benchmarks

System setup

- Linux kernel v4.13
- Skylake μarchitecture with Intel Core i7-6700
- CPU- and memory-intensive macro-benchmarks of the Phoronix Test Suite v7.6.0

Evaluation

Effectiveness: Anti-Virtualization Techniques

Technical University of Munich

- Virtualization artifacts:
  - → No system configuration changes and no guest-visible artifacts
- Hardware artifacts:
  - Guest can directly access the hardware without emulation
  - Timing can reveal the VMM
    - → Today, it is insufficient to reveal the virtual environment alone
- User behavior artifacts:
  - → Virtualize production systems with realistic wear-and-tear relics

Evaluation

Effectiveness: Anti-Virtualization Techniques

- Virtualization artifacts:
  - → No system configuration changes and no guest-visible artifacts
- Hardware artifacts:
  - Guest can directly access the hardware without emulation
  - Timing can reveal the VMM
    - → Today, it is insufficient to reveal the virtual environment alone
- User behavior artifacts:
  - → Virtualize production systems with realistic wear-and-tear relics
- Hardware deficiencies:
  - Foreshadow [10]!

Conclusion & Future Work

Technical University of Munich

- Design of WhiteRabbit VMI framework
  - Virtualizes Linux OSes upon Intel and ARM architectures
- Hardened WhiteRabbit exposure
  - Employs SLAT tables to delude memory carving
- Virtual Machine Introspection
  - Exposes a LibVMI-compatible interface to local (and remote) parties
- Future Work:
  - Explore the need for nested on-the-fly virtualization
  - Open source WhiteRabbit!

Evaluation cont.

Performance: Makro-benchmarks

Technical University of Munich

Table: Virtualization overhead (OHD) of WhiteRabbit, Xen, and KVM measured by the Phoronix Test Suite v7.6.0 on x86-64.

Benchmark (*unit*)	**w/o**	**KVM**	**(OHD)**	**Xen**	**(OHD)**	**WhiteRabbit**	**(OHD)**
Blake2 (*Cycles/Byte*)	5.94	5.94	(0.00%)	5.94	(0.00%)	5.94	(0.00%)
C-Ray (*s*)	107.08	108.56	(1.38%)	107.67	(0.55%)	107.09	(0.00%)
Gzip Compression (*s*)	11.50	12.06	(4.86%)	11.98	(4.17%)	11.74	(2.08%)
John-the-Ripper DES (*Real C/s*)	5,419,000	5,340,000	(1.45%)	5,394,000	(0.46%)	5,417,667	(0.02%)
John-the-Ripper MD5 (*Real C/s*)	16,844	16,583	(1.54%)	16,748	(0.56%)	16,822	(0.13%)
N-queens (*s*)	216.70	220.94	(1.95%)	217.77	(0.49%)	216.80	(0.04%)
OpenSSL (*Signs/s*)	145	141.83	(2.18%)	142.53	(1.70%)	144.70	(0.20%)
7-Zip Compression (*MIPS*)	4,603	3,736	(18.83%)	3,988	(13.36%)	4,443	(3.47%)
RAMspeed Integer (*MB/s*)	17,370.73	16,630.25	(4.26%)	16,942.71	(2.46%)	17,016.09	(2.04%)
RAMspeed Floating Point (*MB/s*)	17,734.84	16,744.56	(5.58%)	16,875.53	(4.84%)	16,861.26	(4.92%)

Evaluation cont.

Performance: Makro- and Micro-benchmarks

Technical University of Munich

Table: SPEC CPU2017, in sec.

Benchmark	w/a	WhiteRabbit	OHD
600.perlbench_s	282	286	(1.41%)
602.gcc_s	409	419	(2.44%)
605.mcf_s	624	641	(2.72%)
620.omnetpp_s	382	406	(6.28%)
623.xalancbmk_s	283	294	(3.88%)
625.x264_s	378	378	(0.00%)
631.deepsjeng_s	357	363	(1.68%)
641.leela_s	460	460	(0.00%)
648.exchange2_s	264	265	(0.37%)
657.xz_s	2220	2379	(7.16%)

Table: Lmbench 3.0, in μsec

Benchmark	w/a	WhiteRabbit	OHD
fork+execve	50.04	58.23	(16.36%)
fork+/bin/sh	254.36	289.84	(13.94%)
pipe	1.51	1.65	(9.27%)
protection fault	0.30	0.31	(3.33%)
read	0.09	0.09	(0.00%)
select 500 fd	2.46	2.52	(2.38%)
select 500 TCP fd	8.16	8.35	(2.32%)
signal handle	0.66	0.65	(1.51%)
sock	1.99	2.07	(4.02%)
write	0.05	0.06	(19.99%)

References I


P. M. Chen and B. D. Noble.
When Virtual Is Better Than Real.
In *USENIX Workshop on Hot Topics in Operating Systems (HotOS)*, 2001.


A. Dinaburg, P. Royal, M. Sharif, and W. Lee.
Ether: Malware Analysis via Hardware Virtualization Extensions.
In *ACM Conference on Computer and Communications Security (CCS)*, 2008.


T. Garfinkel, K. Adams, A. Warfield, and J. Franklin.
Compatibility Is Not Transparency: VMM Detection Myths and Realities.
In *USENIX Workshop on Hot Topics in Operating Systems (HotOS)*, 2007.


T. Garfinkel and M. Rosenblum.
A Virtual Machine Introspection Based Architecture for Intrusion Detection.
In *ISOC Network and Distributed System Security Symposium (NDSS)*, 2003.

References II


T. K. Lengyel, S. Maresca, B. D. Payne, G. D. Webster, S. Vogl, and A. Kiayias.
Scalability, Fidelity and Stealth in the DRAKVUF Dynamic Malware Analysis System.
In *Annual Computer Security Applications Conference (ACSAC)*, 2014.


L. Litty, H. A. Lagar-Cavilla, and D. Lie.
Hypervisor Support for Identifying Covertly Executing Binaries.
In *USENIX Security Symposium*, 2008.


J. Pfoh, C. Schneider, and C. Eckert.
A Formal Model for Virtual Machine Introspection.
In *Workshop on Virtual Machine Security (VMSec)*, 2009.


G. J. Popek and R. P. Goldberg.
Formal Requirements for Virtualizable Third Generation Architectures.
Communications of the ACM, 17(7):412–421, 1974.

References III

J. Rutkowska.
Subverting VistaTM Kernel for Fun and Profit.
Black Hat, USA, 2006.

J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx.
Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution.
In *USENIX Security Symposium*, 2018.