Hypervisor-Level Debugger Benefits & Challenges

Mathieu Tarral

Whoami

- Researcher at **F-Secure**
- Stealth sandboxing
- Virtual Machine Introspection

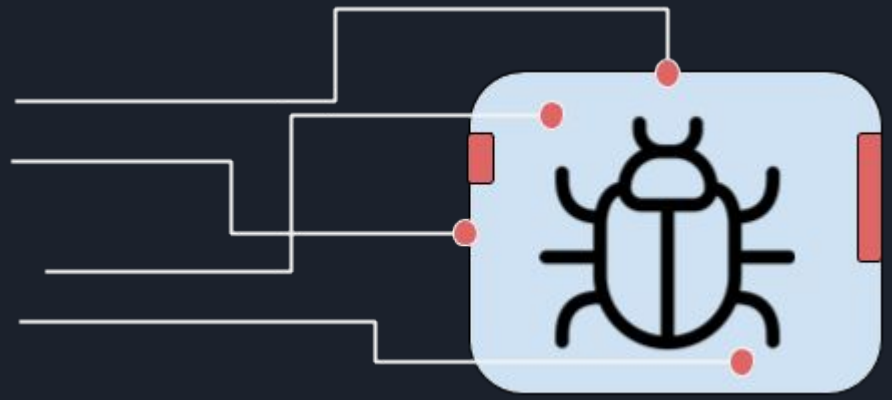
https://github.com/KVM-VMI/kvm-vmi

@mtarral

Why ?

Problem 1: Debuggers are noisy

- A debugger modifies the execution environment of a debuggee

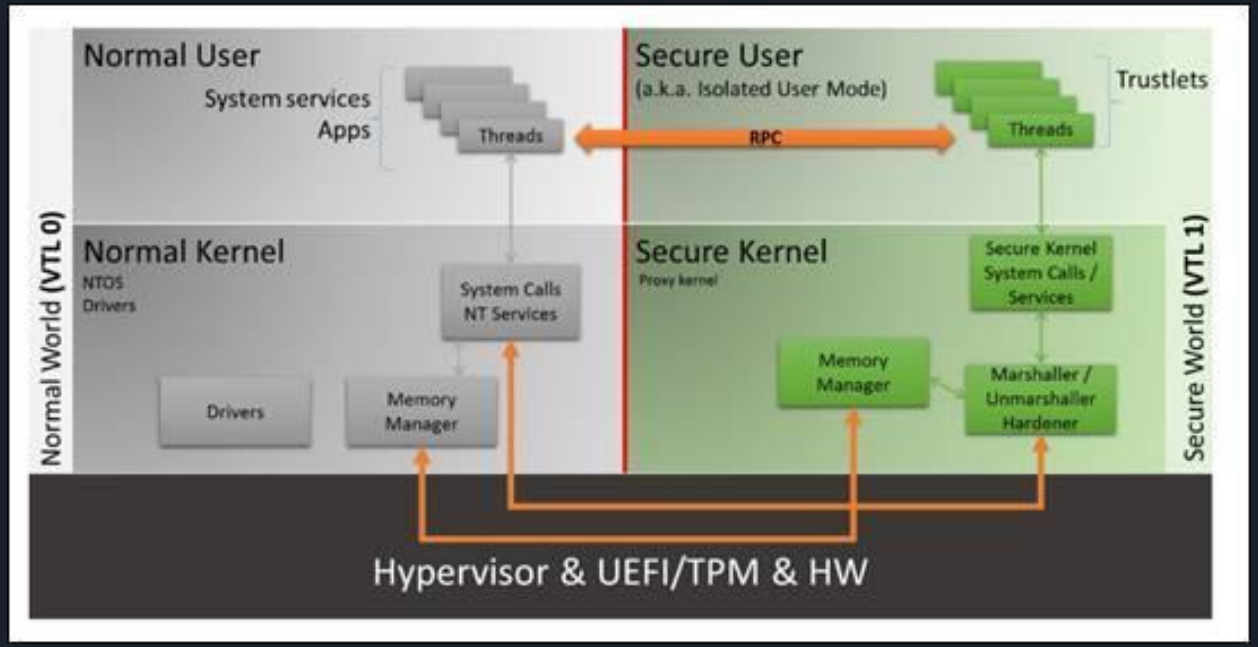


Problem 2: Protected OS features

- The observer effect might sometimes be intentional
- `bcdedit /debug on`
  - disables Patchguard
  - disables Protected Media Path
    - Used to enforce DRM

Problem 3: Incomplete system view

- Debuggers fighting against new OS security features



Solution: Moving to ring -1

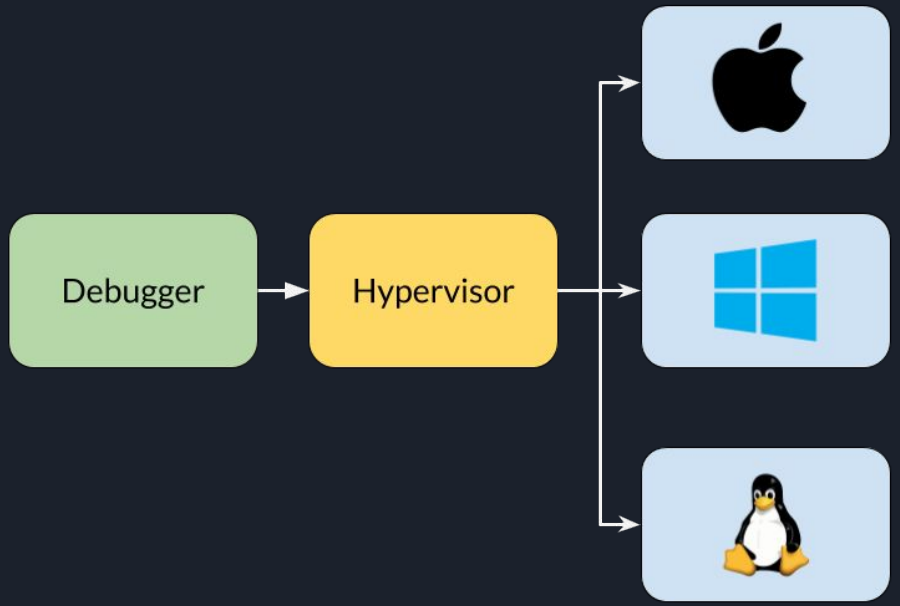
- Leverage the hypervisor as a debugging platform
- Stealth
  - do not use the operating system's debug API
  - bonus: invisible breakpoints with EPT violations
- Full system analysis
  - VMM's property: Resource control / Safety
  - access to the entire guest state
  - bonus: debug bootloaders

Benefit: Unmodified guests

- No remote debug agent/stub
- No custom VM setup
  - hardware
    - network card
    - serial cable
  - software
    - install Windows SDK
  - configuration
    - bcdedit /set debug on
    - bcdedit /dbgsettings serial debugport:1 baudrate:115200
- On-the-fly debugging

Benefit: Cross-platform debugger

- Build your knowledge/scripts on top of one tool



Projects ?

Bare-metal debuggers

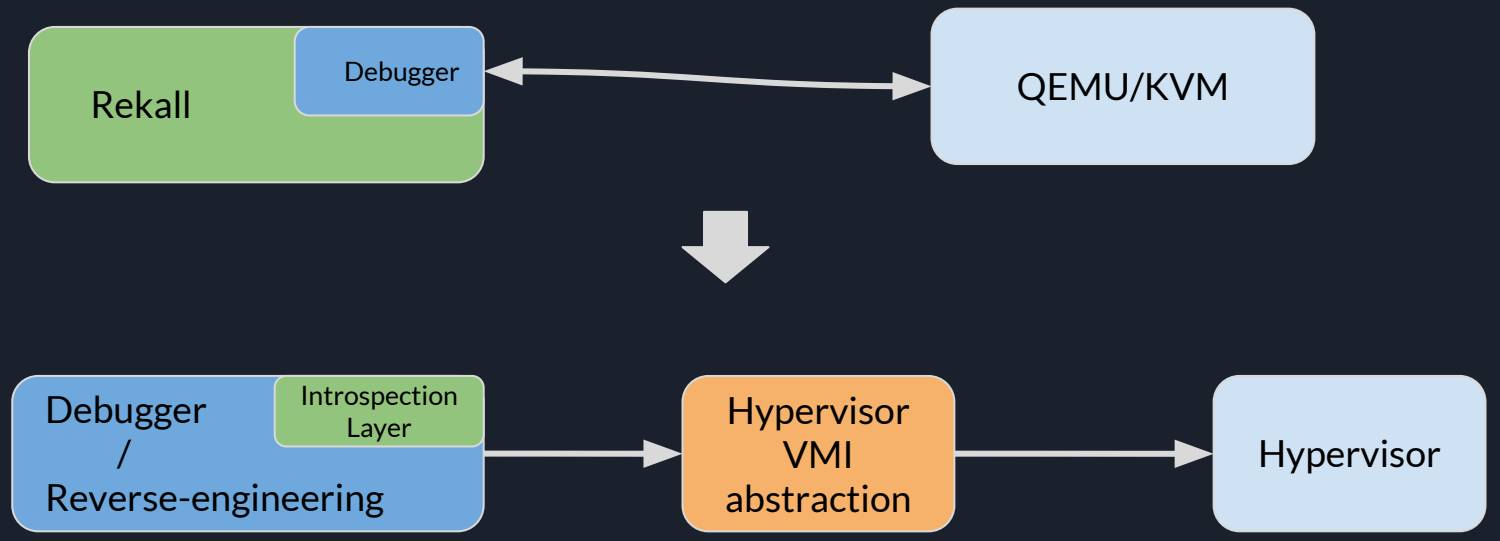
- HyperDBG (2010)
  - *"I want to take full control of a production system"*
  - Hyperjacking: driver is installed on the host
- virtdbg (2011)
  - *"I want to debug PatchGuard"*
  - Hyperjacking: driver is injected via DMA attack
- PulseDBG (2017)
  - "*I want a better WinDBG UI*"
  - Hypervisor is contained in an EFI bootloader (*bootx64.efi*)

Virtual machine debuggers

- Built-in debug stubs
  - QEMU (2003)
  - VMware Workstation 6.0 (2007)
- PyREBox - CISCO Talos (2017)
  - *"I want a scriptable dynamic instrumentation system"*
  - Instrumentation of QEMU (emulator)
- rVMI - FireEye (2017)
  - *"I want to understand why a malware sample didn't run"*
  - Instrumentation of KVM
  - Rekall as introspection layer / debugger interface

How ?

Design: improve rVMI



Hypervisor-agnostic: LibVMI

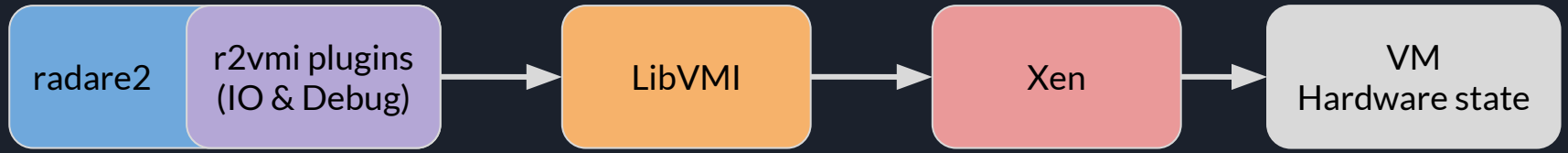
- VMI Abstraction layer
- Offers basic introspection
- Standard for VMI applications
- Future support ?
  - VMware, VirtualBox ?

	VCPU Registers	Physical memory	Hardware events
Xen	✓	✓	✓
KVM	✓	✓	✗

https://github.com/libvmi/libvmi

Architecture

- IO plugin (`io_vmi.c`)
  - initialize LibVMI, access memory and registers
- Debug plugin (`debug_vmi.c`)
  - attach process
  - singlestep
  - breakpoints
- `r2 -d vmi://vm_name:name|pid`



Status ?

Features

- Intercept an existing process by name/pid (*CR3 load*)
- Single-step process execution
- Set software/memory breakpoints
- Load kernel symbols into r2 flagspace (*from Rekall profile*)
- radare2 interface
  - powerful shell
  - graph view
  - structures
  - scripting

Demo

Interactive debugger

Scripting: Intercepting syscalls

```
# find NtOpenFile address
rekall = RekallVMI('win7', 'xen')
syscall_addr = rekall.find_syscall('NtOpenFile')
# open radare2 pipe
r2_url = "vmi://{}:{}".format('win7','firefox.exe')
r2 = r2pipe.open(r2_url, ['-d', '-2'])
# set breakpoint
r2.cmd('db {}'.format(hex(syscall_addr)))
while True:
    # continue
    r2.cmd('dc')
    # hit
    regs = r2.cmdj('drj')
    logging.info("At NtOpenFile: rax=%s", regs['rax'])
```

Scripting: Intercepting syscalls

Future ?

Challenges

- Attach existing process
  - `CR3` -> locate threads context, find RIP
- Break on addresses not mapped yet
  - pagefault injection
- Introspection
  - drop rekall profile
  - `rabin2` to parse PE in memory
  - `radare2` to download/load PDB symbols
- Attach new process
  - guest frozen, Xen development

https://github.com/Wenzel/vagrant-xen-r2vmi

Goals

- Malware analysis
  - stealth sandbox
  - highly interactive reverse-engineering framework
- Fuzzing
- Windows 10 VSM debugging
  - support of Hyper-V on Xen/KVM (?)
- Multi-purpose, cross-platform, full system debugger
  - hypervisor-agnostic by design

https://github.com/Wenzel/r2vmi

Thanks

- pancake (*radare2*)
- Maxime Morin (*radare2*)
- Tamas K lengyel (*LibVMI*)
- Michael Cohen (*Rekall*)
- Alexey Konovalov (*Windows Internals*)
- Lorenzo Martignoni & Aristide Fattori (*HyperDBG*)
- Damien Aumaître (*virtdbg*)
- Artem Shishkin (*PulseDBG*)
- Thais Moreira Hamasaki

Questions

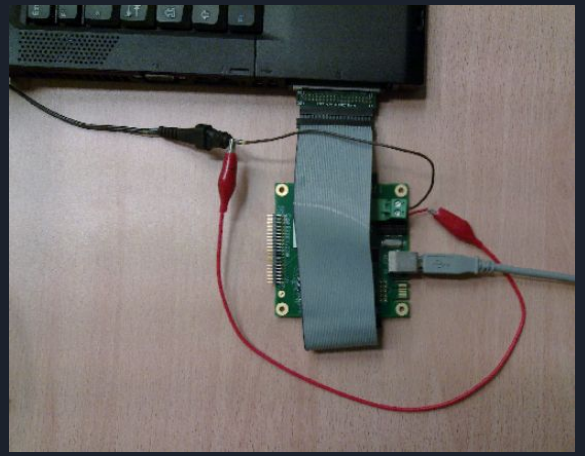
Annex

HyperDBG - 2010

- *“I want to debug production systems”*
- Hyperjacking
- Press *F12* to invoke the debugger UI
- Pros
  - “unmodified guest”: install a driver
  - on-the-fly debugging
- Cons
  - OS support ?
  - user interface
  - unmaintained

VirtDBG - 2011

- *“I want to debug PatchGuard”*
- Hyperjacking
  - hypervisor is silently injected via DMA attack !
- Pros:
  - “unmodified guest”: inject a driver
  - on-the-fly debugging
  - GDB protocol
- Cons
  - hardware requirements
  - unmaintained



PulseDbg - 2017

- *"I want a better WinDBG UI"*
- Hypervisor is contained in an EFI bootloader (*bootx64.efi*)
  - USB stick or network boot via PXE
- Pros:
  - "unmodified guest": boot sequence
  - BIOS and bootloader debugging
  - "can" work on top of another hypervisor (*VMware*)
  - OS-agnostic (hypervisor in EFI module)
- Cons:
  - custom client/server protocol
  - closed source

VMware Workstation GDB stub - 2007

- Since VMware Workstation 6.0+
- edit .vmx file
  - debugStub.listen.guest64 = "TRUE"
- Pros
  - unmodified guest
  - can debug bootloaders
    - monitor.debugOnStartGuest64 = "TRUE"
- Cons
  - VMWare-only
  - need a licence
  - not open-source

PyREBox (CISCO Talos) - 2017

- *"I want a scriptable sandbox environment"*
- Full instrumentation of QEMU (emulator)
- Pros:
  - fine grained control (instruction-level callbacks)
  - IPython shell, scripts
- Cons:
  - Emulation
  - QEMU-only

rVMI (FireEye) - 2017

- *"I want to understand why a malware sample didn't run"*
- VMI instrumentation of KVM
- Pros:
  - introspection layer, thanks to Rekall
  - support for snapshots
- Cons:
  - QEMU/KVM only
  - pushing a debugger into a forensic tool (?)
  - lots of custom code modifications
    - upstream integration (?)