Hypervisor-Level Debugger with Radare2

r2vmi

Mathieu Tarral

Whoami

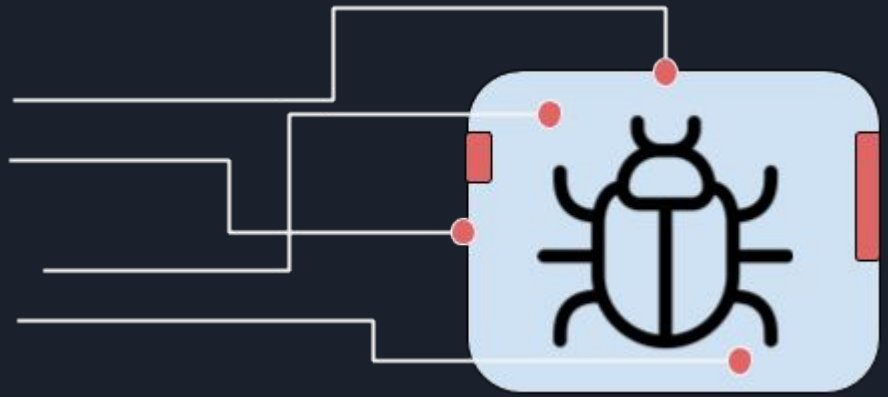
- Researcher at **F-Secure**
- Malware behavioral analysis
- Maintainer of Nitro framework
- Created the KVM-VMI organization on Github

@mtarral

Why ?

Debuggers are noisy

- A debugger modifies the execution environment of a debuggee

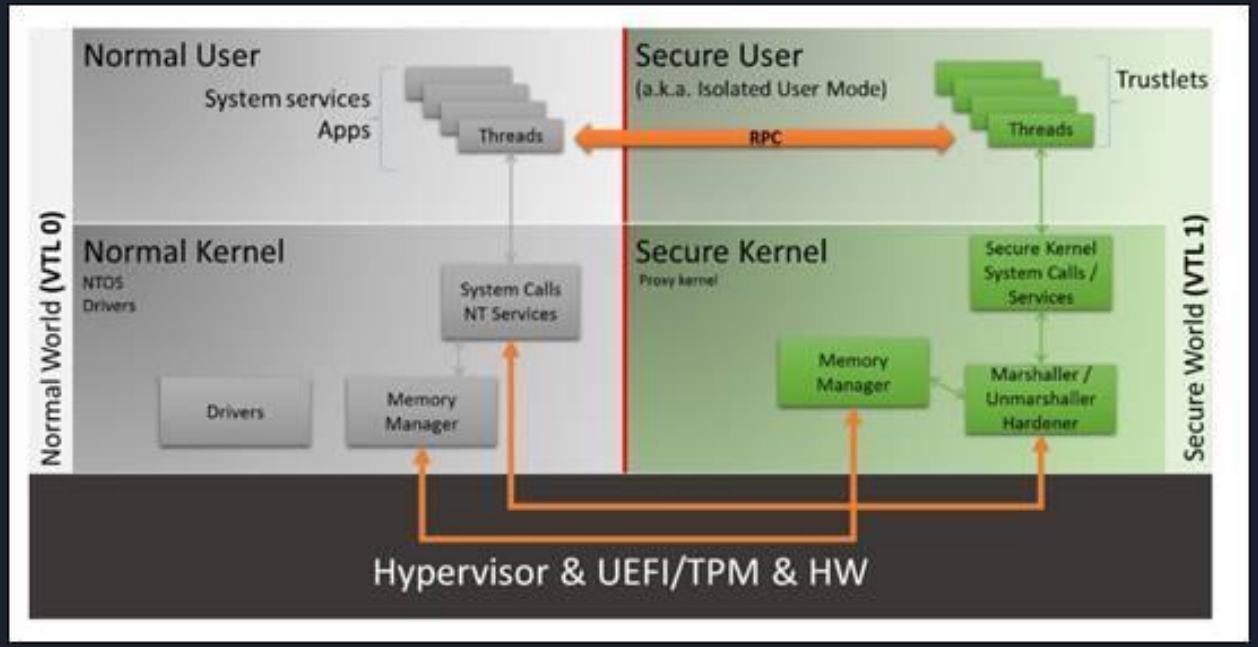


Debugger detection for dummies

- Windows API
  - `IsDebuggerPresent()`
  - `CheckRemoteDebuggerPresent()`
- Checking memory structures
  - `PEB.BeingDebugged`
  - `NtGlobalFlag`
  - `Process Heap Flags` and `ForceFlags`
  - scan for software breakpoints
- Side effects in system calls behavior
  - NtClose with invalid handle

Debugger's system view is incomplete

- Remote WinDBG is not enough



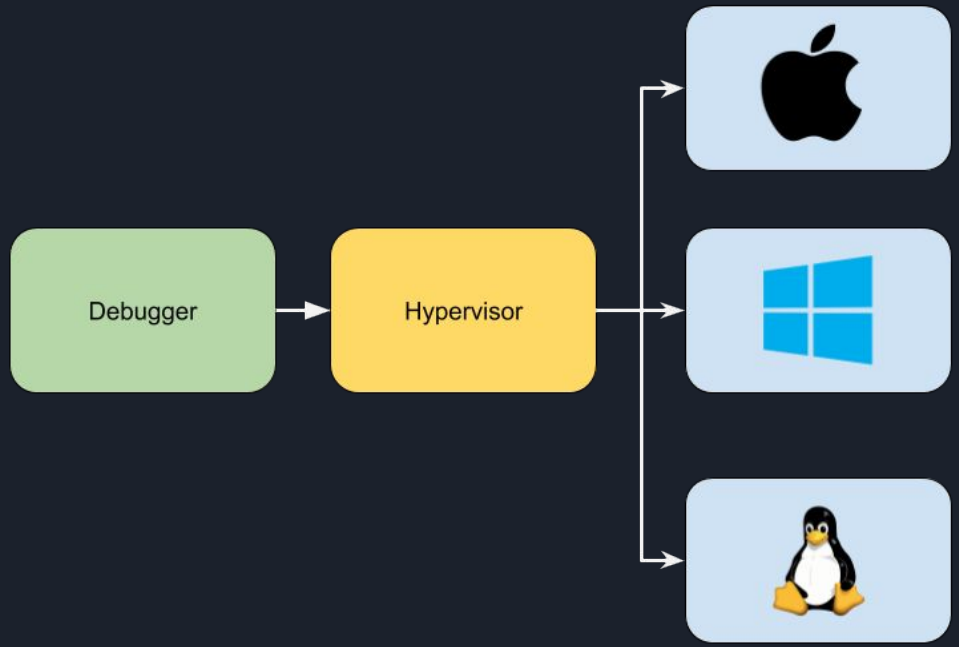
Moving to ring -1

- move the debugging process to the hypervisor
- 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

Zero configuration

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

Cross-platform debugger



Projects ?

Hypervisor-based debuggers

- Debugger is built inside a tiny hypervisor
- HyperDBG (2010)
  - virtualize your host on-the-fly
- PulseDBG (2017)
  - load EFI bootloader (bootx64.efi)
  - client/server, needs 2 physical machines
- Cons:
  - debug real hardware
  - not designed to work with virtual machines

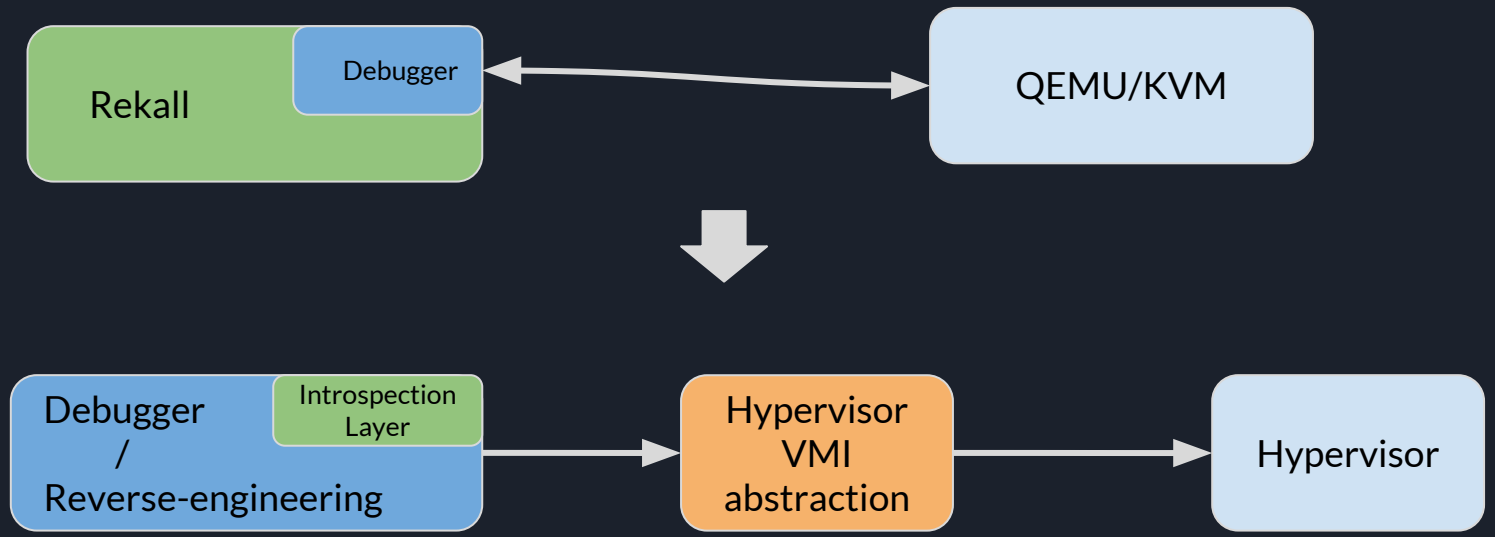
VMI-based debuggers

- Introspect the guest using VMI API's
- Talos PyREBox (2017)
  - dynamic instrumentation engine, based on QEMU
  - volatility for VMI
  - python callbacks
  - cons:
    - emulation
- FireEye rVMI (2017)
  - VMI on KVM
  - rekall as a debugger
  - cons:
    - KVM-only
    - maintainability ?

How ?

Design

- Use VMI approach
- Improve rVMI



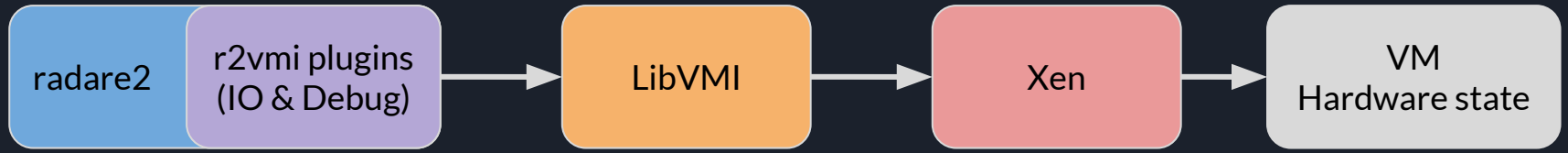
Hypervisor-agnostic: LibVMI

- VMI Abstraction layer
- Takes care of low-level details
- Offers basic introspection

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

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`



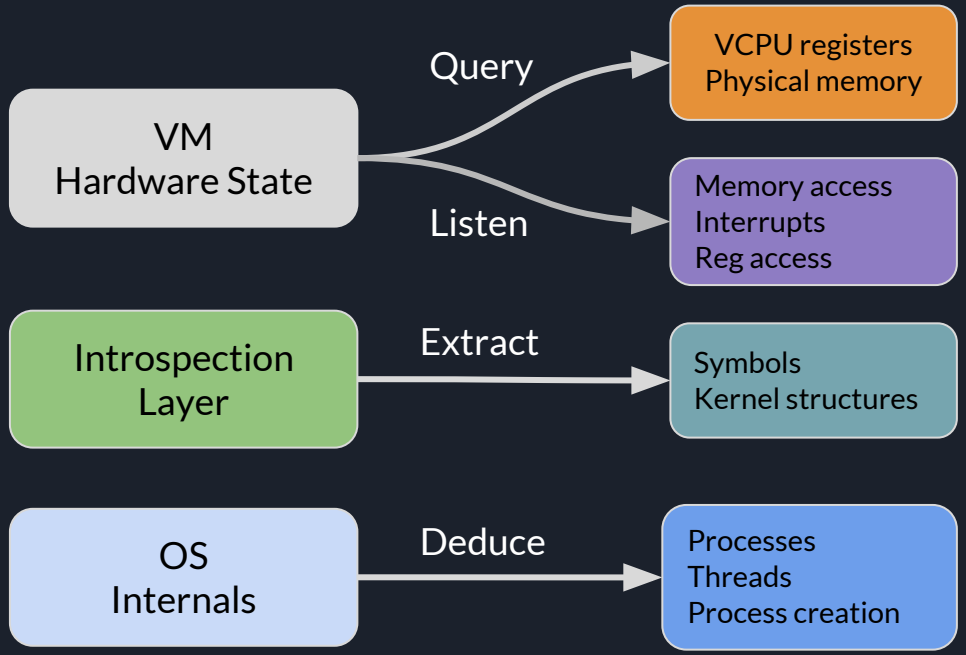
Implementation ?

Challenges

- Debugging is a set of API provided by the operating system
- We have to rebuild these APIs from the hypervisor

```
//
// Main User-Mode Debugger Loop //
CreateProcess("target.exe",..., DEBUG_PROCESS, ...);
while (1)
{
    WaitForDebugEvent(&event, ...);
    switch (event)
    {
        case ModuleLoad:
            Handle/Ignore;
            break;
        case TerminateProcess:
            Handle/Ignore;
            break;
        case Exception (code breakpoint, single step, etc...):
            Handle/Ignore;
            break;
    }
    ContinueDebugEvent(...);
}
```

Challenges - What we have



Attaching existing process

- How to implement __attach(RDebug *dbg, int pid) ?
- Listen on MOV-TO-CR3 events
- Check for your targeted pid

```
[0xfffff800026de1a9]> pd 10 @-3
            0xfffff800026de1a6      0f22da          mov cr3, rdx
            ;-- rip:
            0xfffff800026de1a9      4c8bbb88feff.   mov r15, qword [rbx - 0x178]
            0xfffff800026de1b0      488b6e28        mov rbp, qword [rsi + 0x28]
            0xfffff800026de1b4      49896f04        mov qword [r15 + 4], rbp
            0xfffff800026de1b8      48896b28        mov qword [rbx + 0x28], rbp
            0xfffff800026de1bc      f7057e741800.   test dword [0xfffff80002865644], 4
        ,=< 0xfffff800026de1c6      0f850b010000    jne 0xfffff800026de2d7
        |   0xfffff800026de1cc      c6474900        mov byte [rdi + 0x49], 0
        |   0xfffff800026de1d0      0fbe86650100.   movsx eax, byte [rsi + 0x165]
        |   0xfffff800026de1d7      84c0            test al, al
```

Introspection

- Use Rekall forensic framework
- Solution 1:
  - load a Python interpreter
  - call Rekall functions using Python C APIs (`PyObject_CallObject`)
- Solution 2:
  - run a Python RPC-endpoint, RESTful API, running Rekall
  - serialize objects and structs, send JSON

Introspection

- Solution 3:
  - Load kernel symbols from LibVMI (`r2>.\symbols`)
  - run a Rekall interactive session using VMIAddressSpace
  - `rekall -f vmi://xen/vm_name`

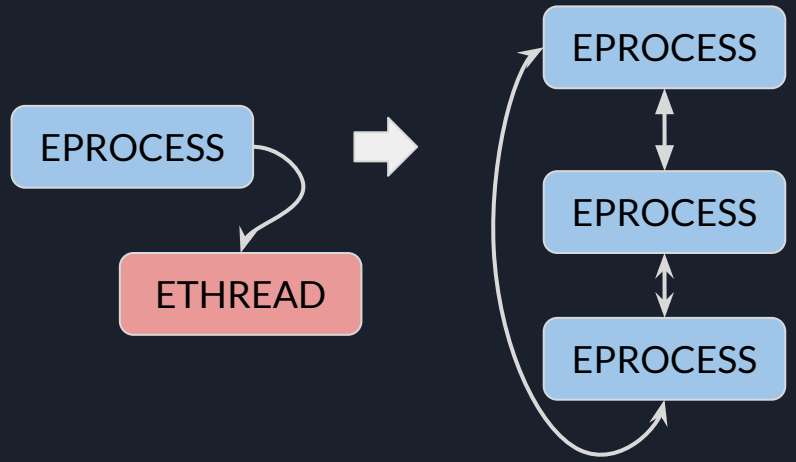
```
[1] xenwin7 21:38:31> pslist
--------------------> pslist()2800000 (0xa00000)
  _EPROCESS            name          pid   ppid  thread_count handle_count session_id wow64    process_create_time
------------- ------------------- ----- ------ ------------ ------------ ---------- ------ -----------------------
0xfa80033b3040 System                  4      0           76          456          - False  2018-08-22 01:18:51Z
0xfa80045b0b30 svchost.exe           192    464           17          312          0 False  2018-08-22 01:18:58Z
0xfa80044ef8a0 smss.exe              252      4            2           29          - False  2018-08-22 01:18:51Z
0xfa80042fd060 spoolsv.exe           296    464           12          264          0 False  2018-08-22 01:18:58Z
0xfa800469e420 csrss.exe             324    316            8          278          0 False  2018-08-22 01:18:53Z
0xfa800459e810 svchost.exe           340    464           16          401          0 False  2018-08-22 01:18:57Z
0xfa800470f060 wininit.exe           372    316            3           74          0 False  2018-08-22 01:18:54Z
0xfa8004839060 csrss.exe             380    364            7          136          1 False  2018-08-22 01:18:54Z
0xfa8004852460 winlogon.exe          420    364            3          107          1 False  2018-08-22 01:18:54Z
0xfa800488ab30 services.exe          464    372            5          178          0 False  2018-08-22 01:18:54Z
0xfa8004892b30 lsass.exe             472    372            6          527          0 False  2018-08-22 01:18:54Z
0xfa800465eac0 lsm.exe               480    372            9          135          0 False  2018-08-22 01:18:54Z
```

Attaching new process

- How to implement __attach(RDebug *dbg, int pid) ?
  - Find CR3 doesn't exist in kernel process list **yet**
- How to access the entrypoint ?
- Windows Internals

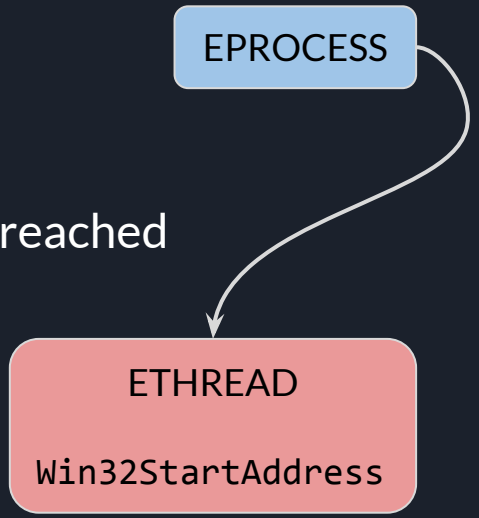
Process Creation (Windows 7)

- `nt!NtCreateUserProcess`
  - `pspAllocateProcess`
  - `pspsInsertProcess`
  - `PspAllocateThread`
  - `PspInsertThread`
- `nt!KiStartUserThread`
  - `ntdll!LdrInitializeThunk`
  - `ntdll!NtContinue`
  - `ntdll!RtlUserThreadStart`
    - `call entrypoint!`



To the entrypoint

- Read `Win32StartAddress`
- Populated during `nt!pspAllocateThread`
- Available when `nt!KiStartUserThread` is reached
- Solution:
  - break on `nt!KiStartUserThread`
  - read `Win32StartAddress`
  - break on entrypoint
    - not mapped yet !
  - Tests for the page to be mapped
    - `nt!MmAccessFault`
    - singlestep until ring3
    - watch for page tables modification



Status ?

Features

- Intercept a existing process by Name/pid
- Single-step process execution
- Set memory breakpoints
- Load Rekall symbols (kernel only)
- Radare2 interface !

Issues

- Attach existing process
  - Find threads context, find `rip`
- Attach new process
  - debug VMI state of guest
  - Xen development
- Break on addresses not mapped yet
  - Watch page tables for modification
- Introspection
  - Access Rekall from C ?
  - Python Rabin plugin + Rekall VMIAddressSpace ?

Demo

Future ?

Future

- Malware analysis
  - stealth sandboxes
  - highly interactive reverse-engineering frameworks
- Vulnerability research
- Multi-purpose, cross-plaform, full system debugger

https://github.com/Wenzel/r2vmi

Thanks

- pancake (*radare2*)
- Maxime Morin (*radare2*)
- Tamas K lengyel (*LibVMI*)
- Michael Cohen (*Rekall*)
- Alexey Konovalov (*Windows Internals*)

Questions