Leveraging KVM as a Debugging Platform

HACK.LU
2019 22-24 October

Wenzel

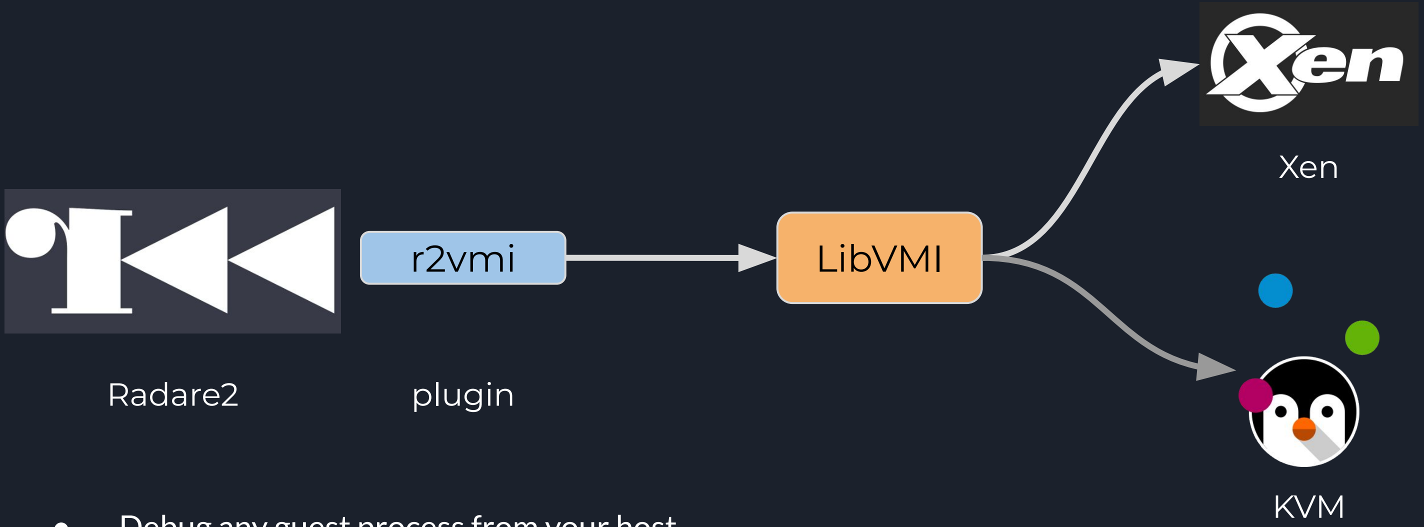
mtarral

mathieu.tarral

Agenda

1. VM introspection on KVM
2. KVM-VMI Setup
3. Integration in LibVMI (demo)
4. KVM as a debugging platform (demo)
5. Future

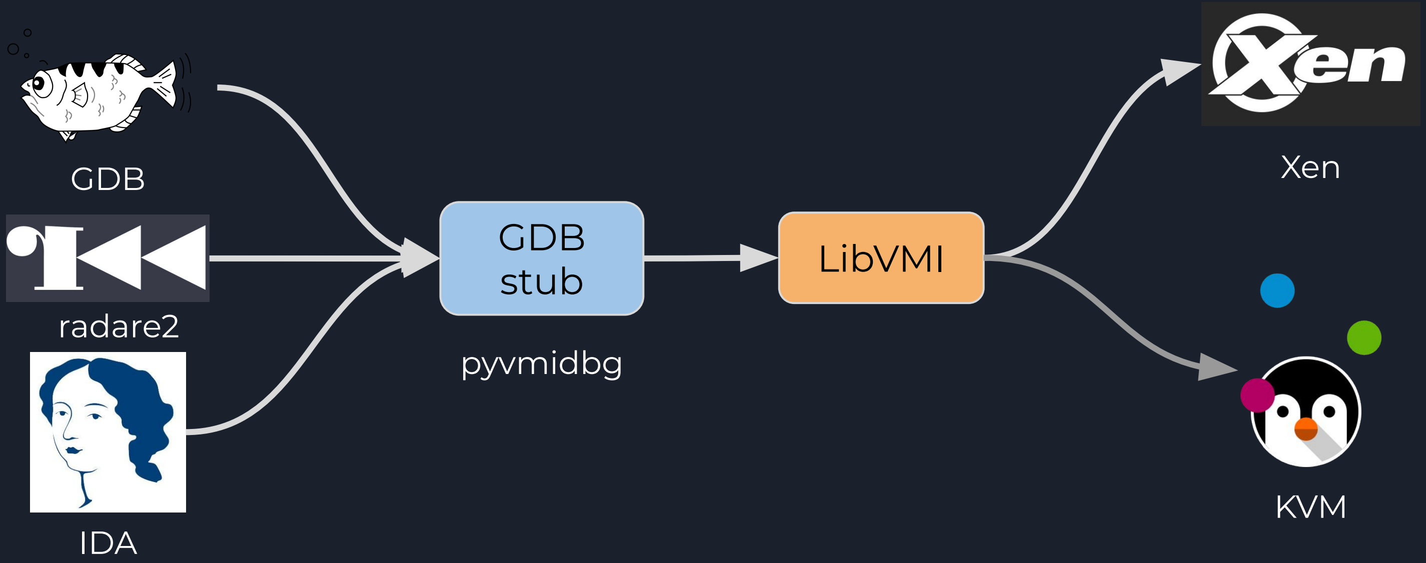
Hack.lu 2018: r2vmi on Xen



- Debug any guest process from your host
- Radare2 as main tool
- Hypervisor-agnostic by design

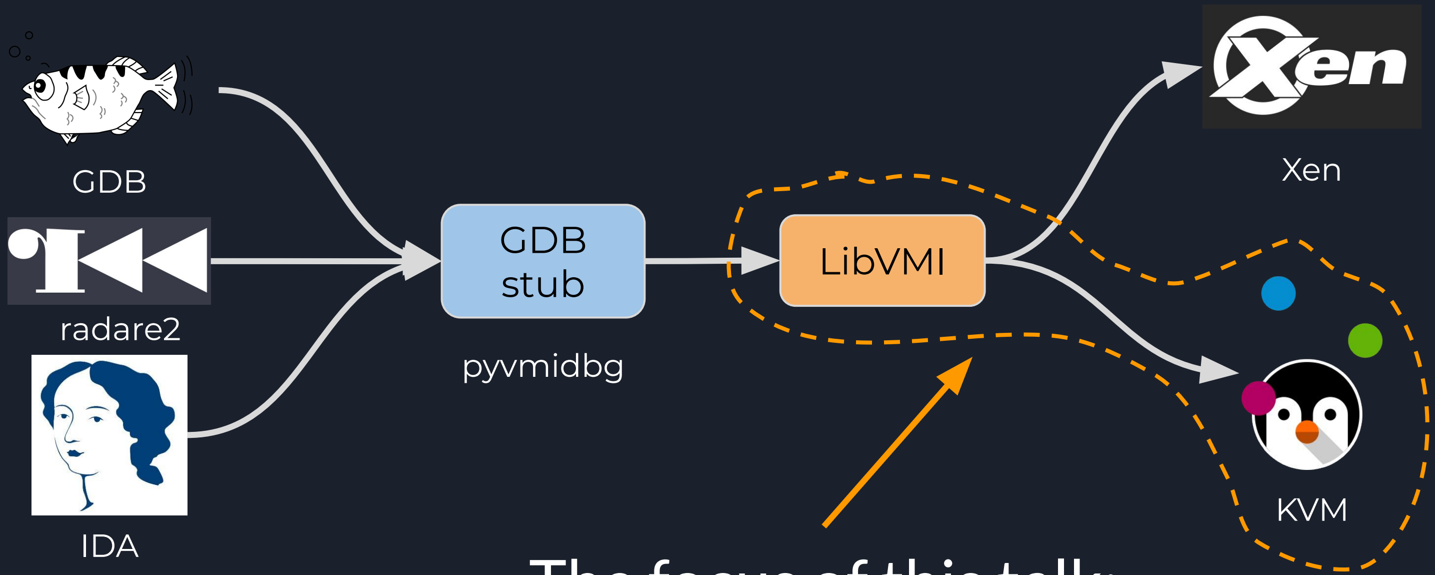
  - KVM was not available (missing APIs)

2019: a Python VMI-GDB stub



https://github.com/Wenzel/pyvmidbg

Hack.lu 2019: a VMI-GDB stub... on KVM ?!

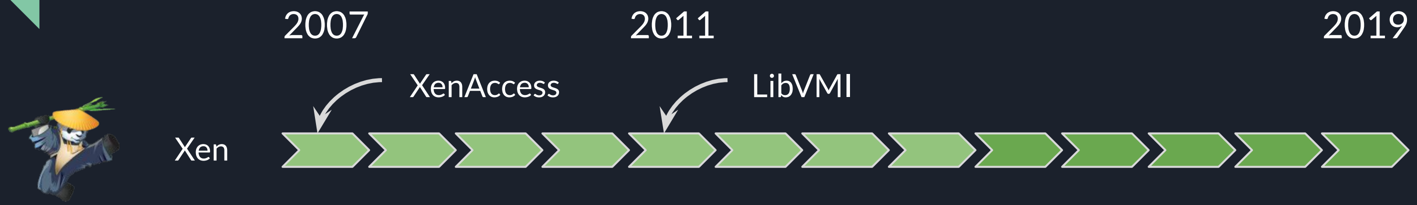


The focus of this talk:

a new LibVMI **KVM driver**

VM Introspection on KVM ?

VMI API: Hypervisor Support

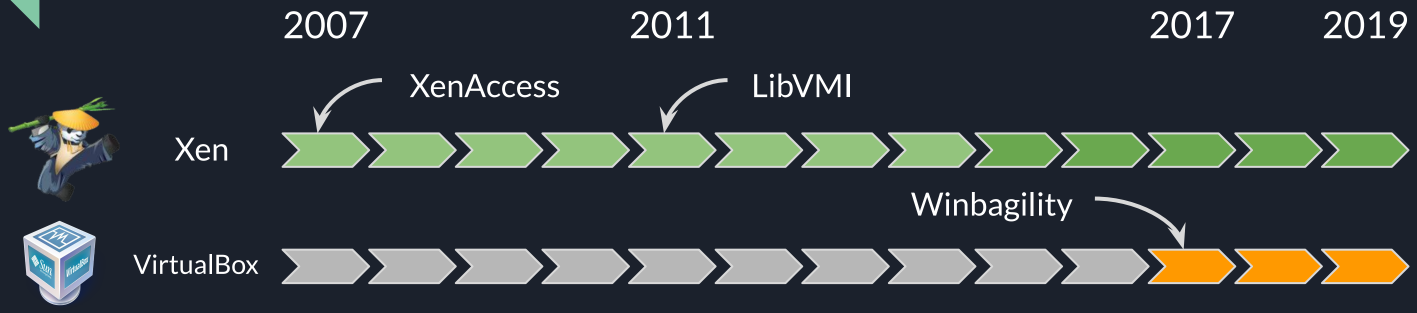


🟧 Patches available (community)

🟩 Upstream integration

🟩 Alternate EPT/RVI available

VMI API: Hypervisor Support



2007

2011

2017

2019

XenAccess

LibVMI

Xen

Winbagility

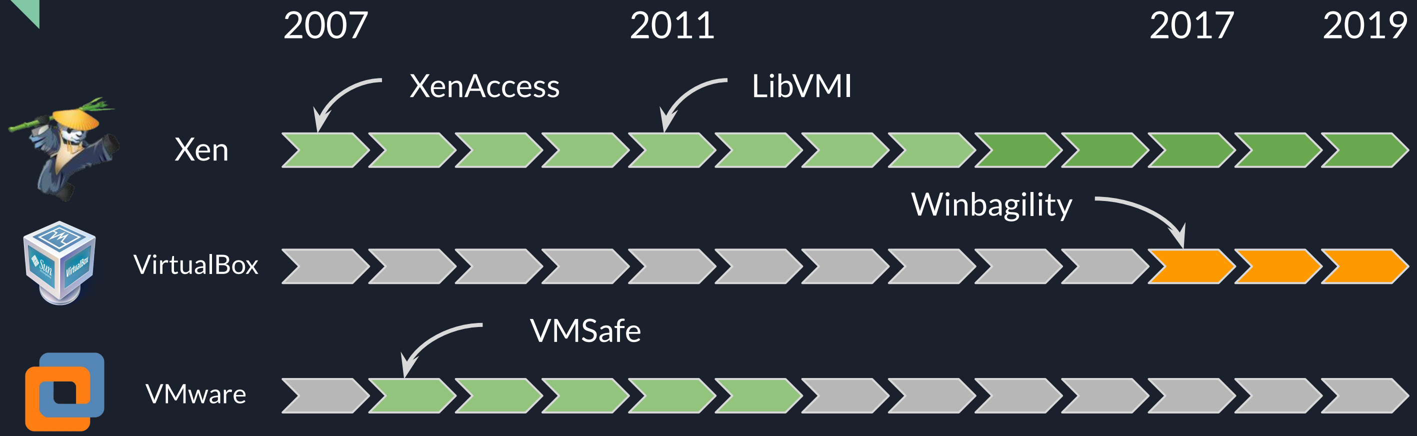
VirtualBox

🟧 Patches available (community)

🟩 Upstream integration

🟩 Alternate EPT/RVI available

VMI API: Hypervisor Support



2007 2011 2017 2019

XenAccess

LibVMI

Xen

Winbagility

VirtualBox

VMSafe

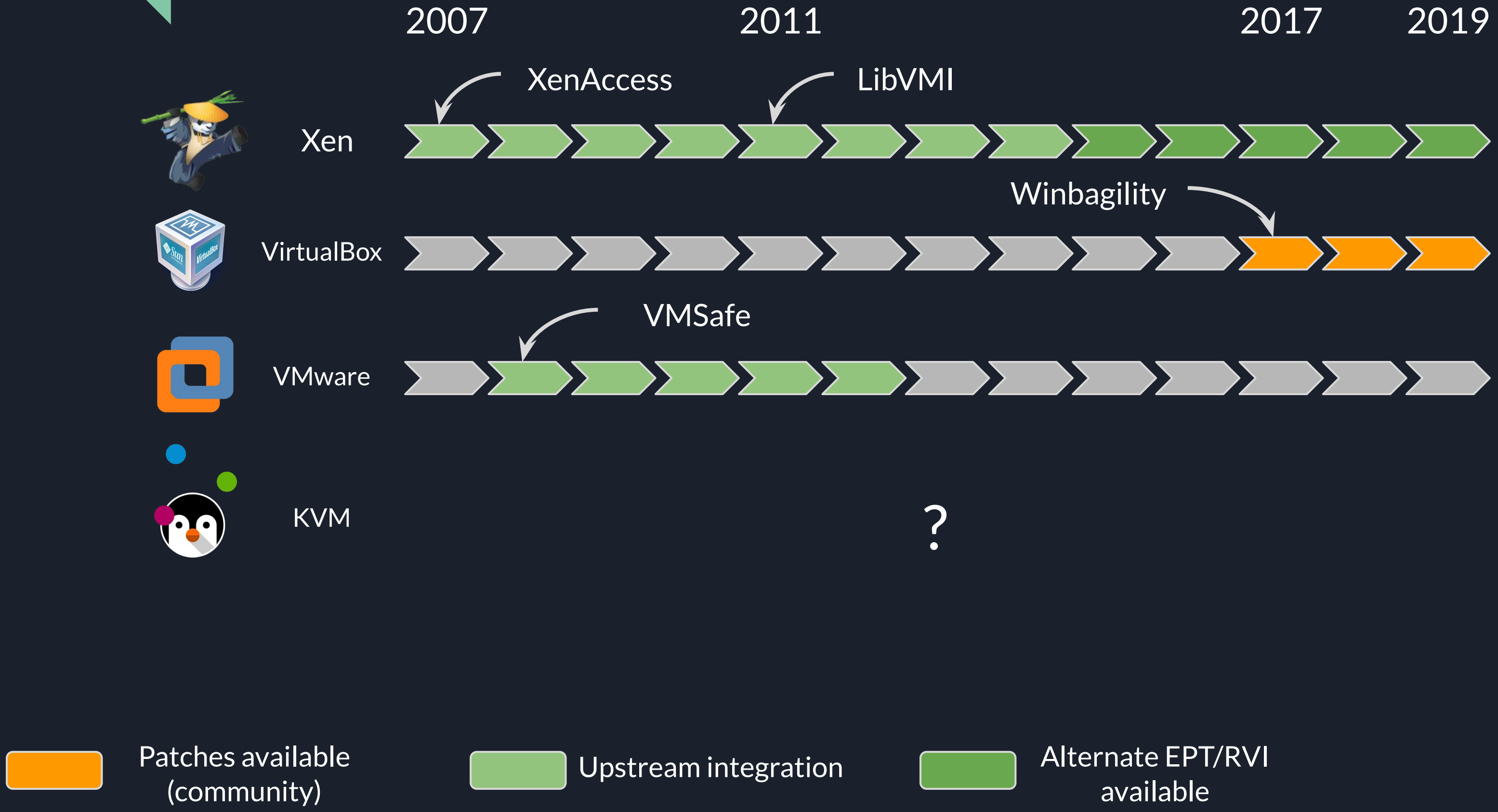
VMware

🟧 Patches available (community)

🟩 Upstream integration

🟩 Alternate EPT/RVI available

VMI API: Hypervisor Support



2007 2011 2017 2019

XenAccess

LibVMI

Xen

Winbagility

VirtualBox

VMSafe

VMware

KVM

?

Patches available (community)

Upstream integration

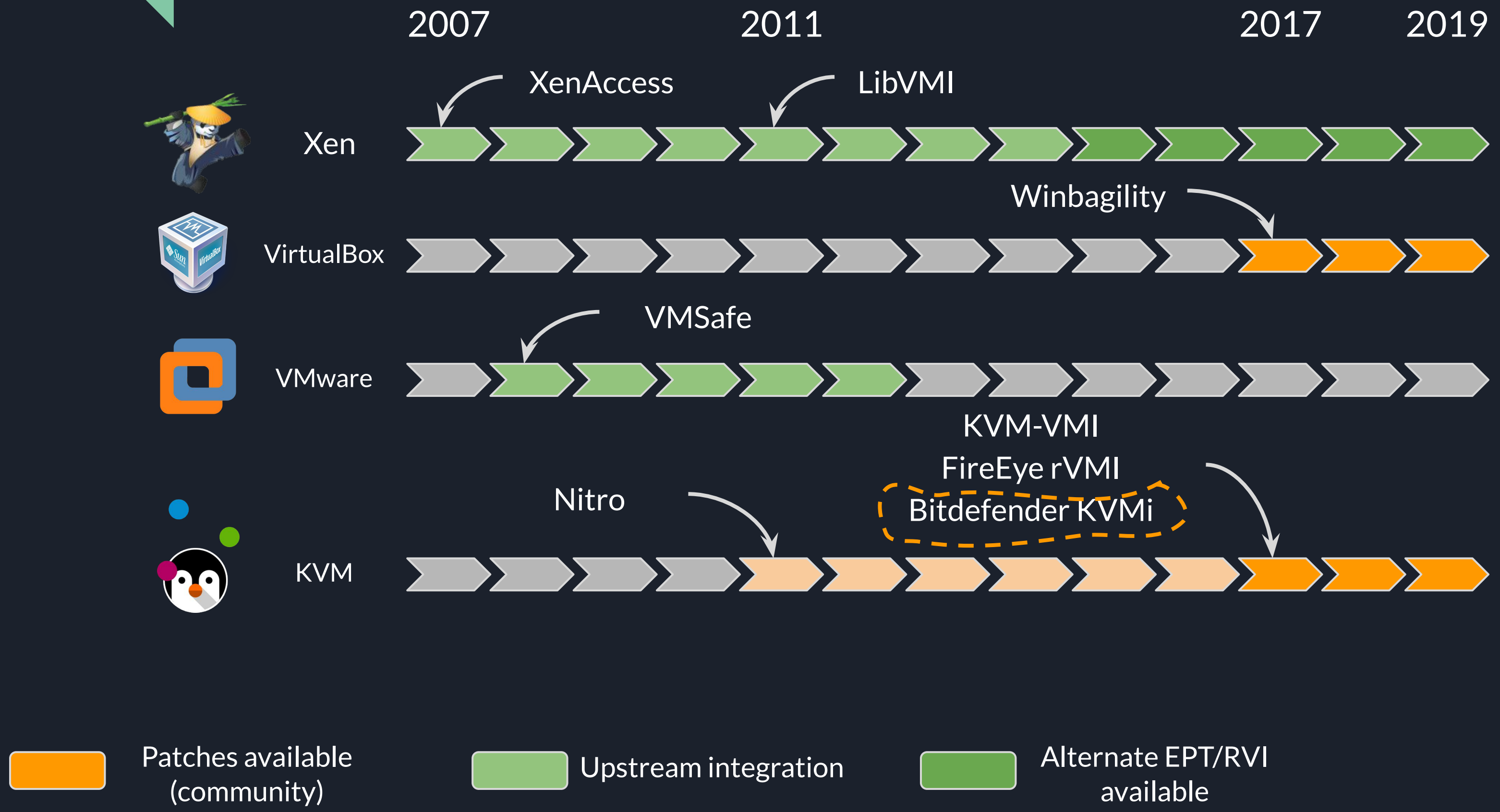
Alternate EPT/RVI available

VMI on KVM ?

- VMI-based frameworks are available for QEMU (**full-emulation**)
    - PANDA - 2011
    - DECAF - 2014
    - PyREBox - 2017

- What about **KVM** ?
    - Nitro - 2011
        - syscall interception
    - KVM-VMI community (Github) - Jan 2017
        - improved Nitro with Python framework
    - FireEye rVMI - July 2017
        - full system debugger
    - Bitdefender KVMi subsystem - June 2017
        - complete API for VMI
    - KVM-VMI integrates BitDefender KVMi patches - January 2018
        - + new LibVMI KVM driver !

VMI API: Hypervisor Support



2007

2011

2017

2019

XenAccess

LibVMI

Xen

Winbagility

VirtualBox

VMSafe

VMware

KVM-VMI

FireEye rVMI

Bitdefender KVMi

Nitro

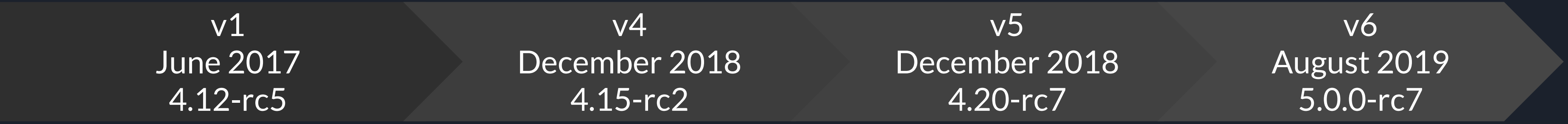
KVM

Patches available (community)

Upstream integration

Alternate EPT/RVI available

VMI on KVM: BitDefender KVMi Evolution



v1
June 2017
4.12-rc5

- initial API design
- RFC on KVM mailing list

v4
December 2018
4.15-rc2

- QEMU will connect to the introspection tool, and perform a handshake
- change mapping protocol

v5
December 2018
4.20-rc7

- improved remote-mapping
- single-step support as reply for #PageFault events
- new ioctl to remove hooks when a domain is suspended/live-migrated

v6
August 2019
5.0.0-rc7

- lots of fixes
- speed improvments
- guests are much more stable

Next: drop RFC and merge basic introspection in KVM !

Note: v2 and v3 were only documentation updates

KVMi API: Overview

- Get VM hardware state
  - `r/w physical memory`
  - `r/w VCPU registers`
  - `get domain info: { VPCU count, Max PFN, ... }`
  - `pause/resume domain`
- Listen for hardware events
  - `CR0/CR3/CR4`
  - `MSR`
  - `interrupts (int3)`
  - `memory access`
  - `descriptor`
  - `hypercall`
- Utilities
  - `guest remote memory mapping`
  - `exception injection`
    - `page fault injection`

https://github.com/KVM-VMI/kvm/blob/kvmi/tools/kvm/kvmi/include/kvmi/libkvmi.h

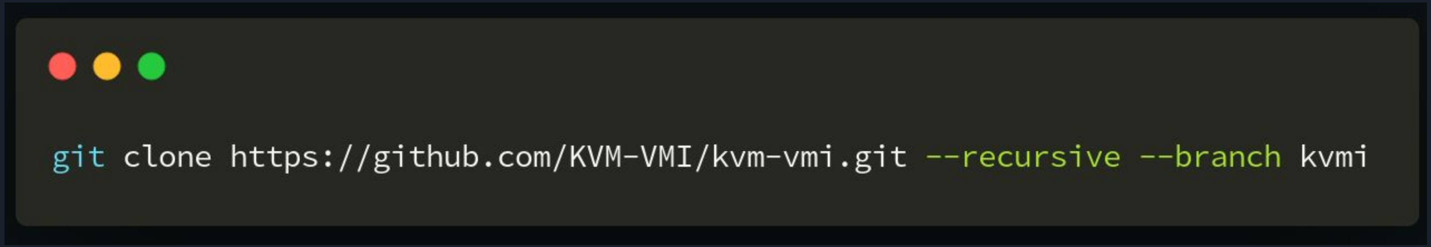
Why: BitDefender KVMi

“ At the moment, the target audience for KVMI are security software authors that wish to perform forensics on newly discovered threats (exploits) or to implement another layer of security like preventing a large set of kernel rootkits simply by "locking" the kernel image in the shadow page tables (ie. enforce .text r-x, .rodata rw- etc.). ”

KVM-VMI Setup

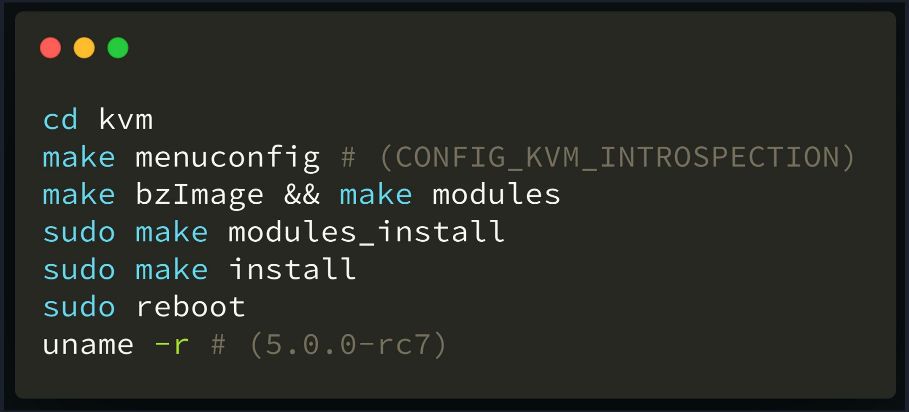
KVM-VMI Setup - 1

Setup Wiki page: https://github.com/KVM-VMI/kvm-vmi/wiki/KVM-VMI-setup



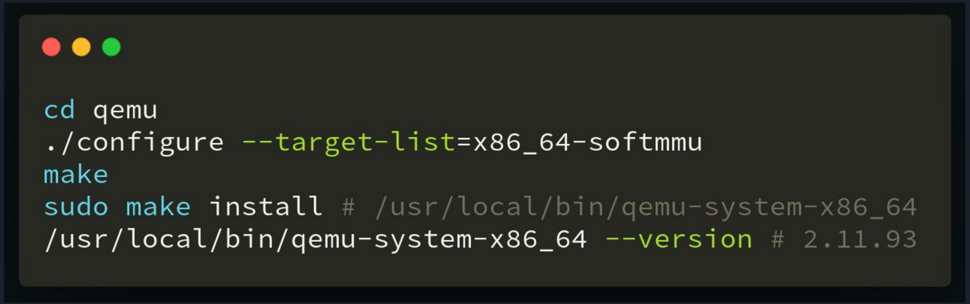
```
git clone https://github.com/KVM-VMI/kvm-vmi.git --recursive --branch kvmi
```

KVM



```
cd kvm
make menuconfig # (CONFIG_KVM_INTROSPECTION)
make bzImage && make modules
sudo make modules_install
sudo make install
sudo reboot
uname -r # (5.0.0-rc7)
```

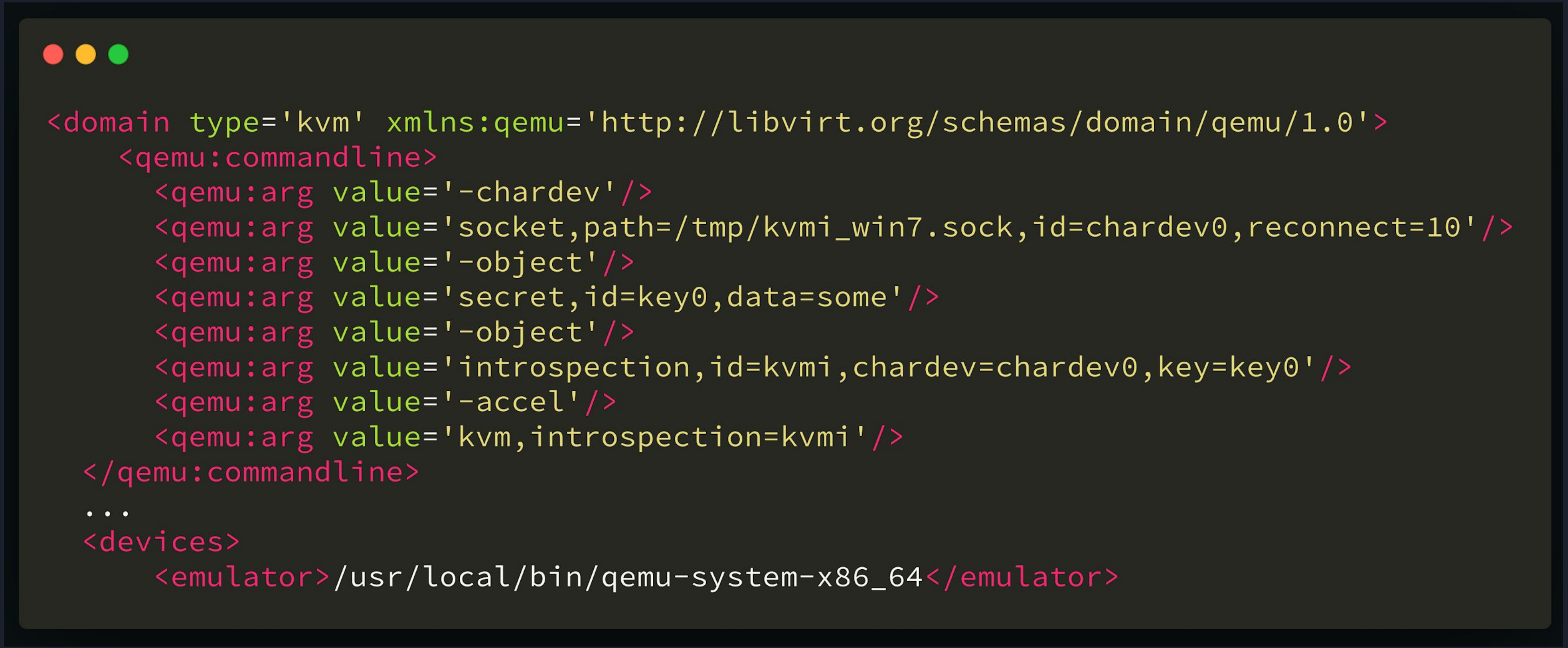
QEMU



```
cd qemu
./configure --target-list=x86_64-softmmu
make
sudo make install # /usr/local/bin/qemu-system-x86_64
/usr/local/bin/qemu-system-x86_64 --version # 2.11.93
```

KVM-VMI Setup - 2

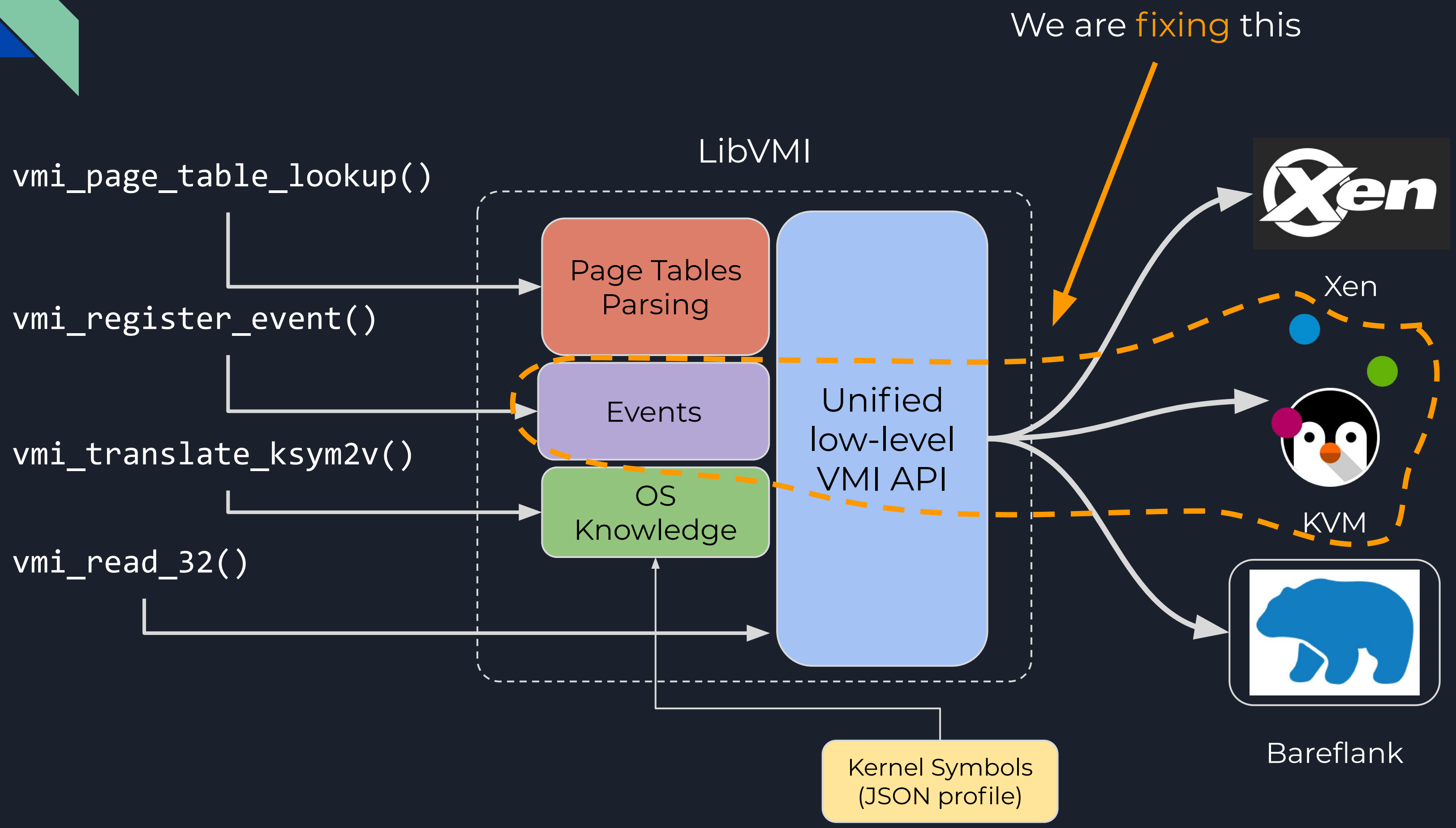
- QEMU has new command line parameters: a socket (ex: `/tmp/kvmi_win7.sock`)



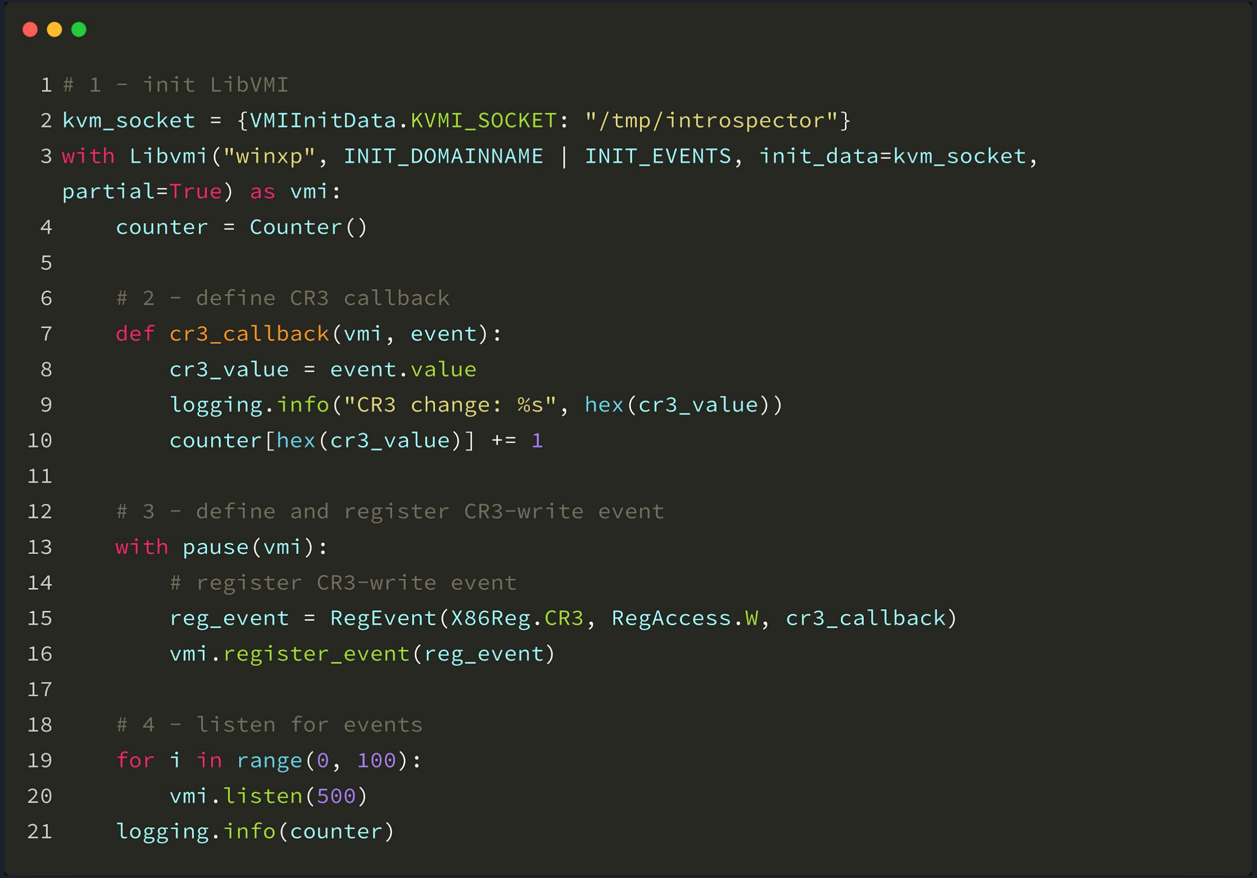
```
<domain type='kvm' xmlns:qemu='http://libvirt.org/schemas/domain/qemu/1.0'>
    <qemu:commandline>
      <qemu:arg value='-chardev'/>
      <qemu:arg value='socket,path=/tmp/kvmi_win7.sock,id=chardev0,reconnect=10'/>
      <qemu:arg value='-object'/>
      <qemu:arg value='secret,id=key0,data=some'/>
      <qemu:arg value='-object'/>
      <qemu:arg value='introspection,id=kvmi,chardev=chardev0,key=key0'/>
      <qemu:arg value='-accel'/>
      <qemu:arg value='kvm,introspection=kvmi'/>
  </qemu:commandline>
  ...
  <devices>
      <emulator>/usr/local/bin/qemu-system-x86_64</emulator>
```

Integration in LibVMI

LibVMI: Overview

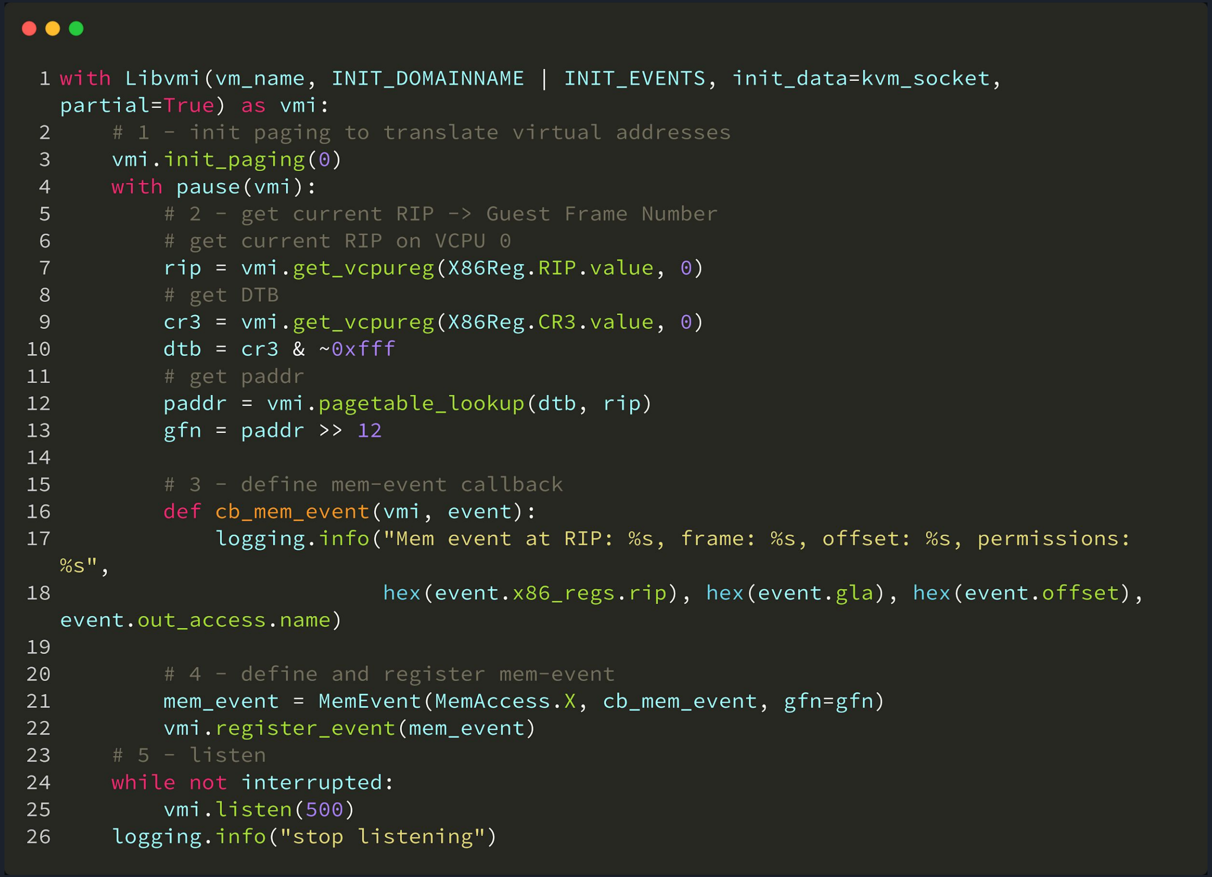


LibVMI: Python Bindings - CR3 Events



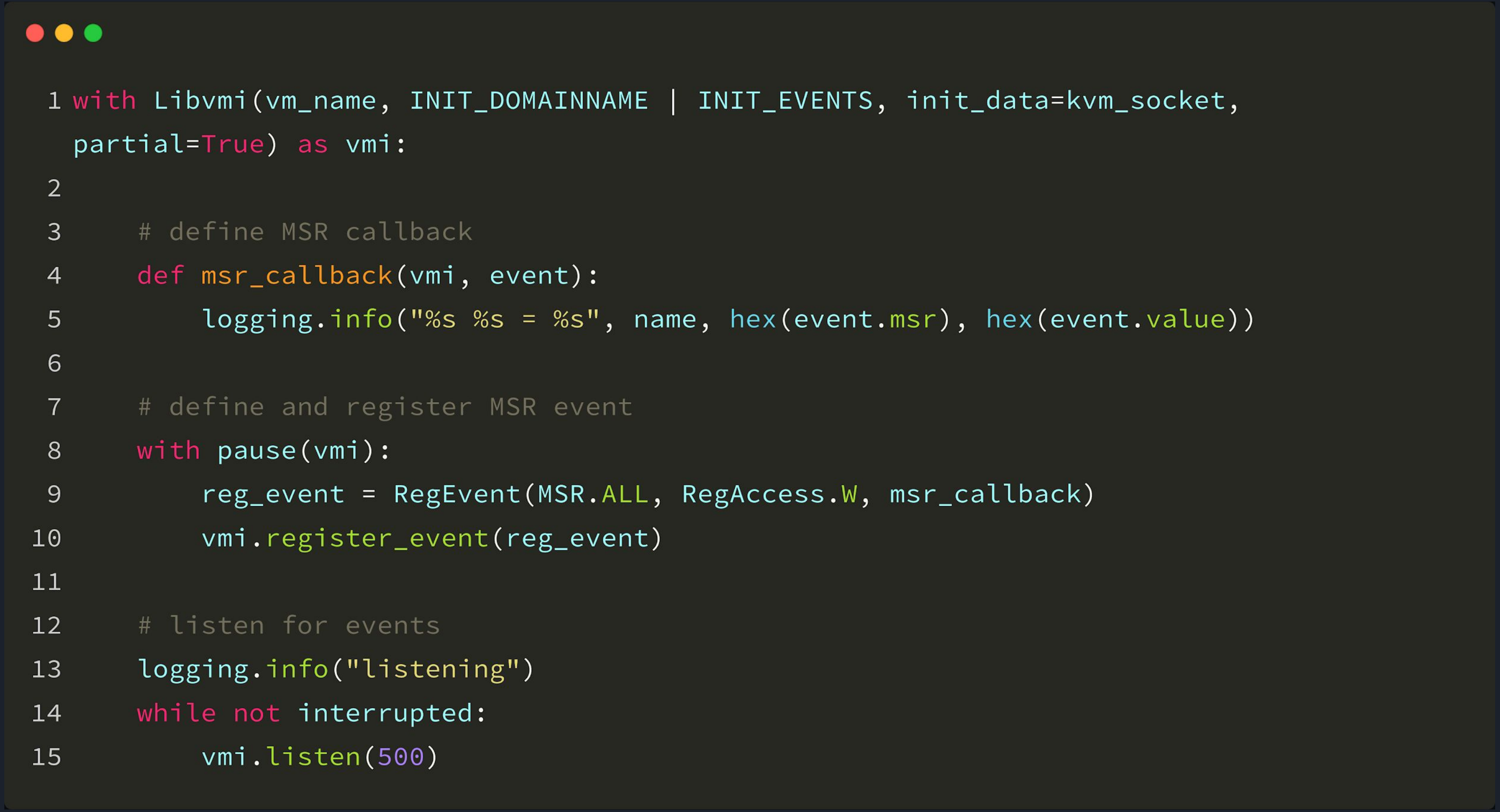
```python
# 1 - init LibVMI
kvm_socket = {VMIInitData.KVMI_SOCKET: "/tmp/introspector"}
with Libvmi("winxp", INIT_DOMAINNAME | INIT_EVENTS, init_data=kvm_socket, partial=True) as vmi:
    counter = Counter()

    # 2 - define CR3 callback
    def cr3_callback(vmi, event):
        cr3_value = event.value
        logging.info("CR3 change: %s", hex(cr3_value))
        counter[hex(cr3_value)] += 1

    # 3 - define and register CR3-write event
    with pause(vmi):
        # register CR3-write event
        reg_event = RegEvent(X86Reg.CR3, RegAccess.W, cr3_callback)
        vmi.register_event(reg_event)

    # 4 - listen for events
    for i in range(0, 100):
        vmi.listen(500)
    logging.info(counter)
```

LibVMI: Python Bindings - Memory Events



```python
with Libvmi(vm_name, INIT_DOMAINNAME | INIT_EVENTS, init_data=kvm_socket, partial=True) as vmi:
    # 1 - init paging to translate virtual addresses
    vmi.init_paging(0)
    with pause(vmi):
        # 2 - get current RIP -> Guest Frame Number
        # get current RIP on VCPU 0
        rip = vmi.get_vcpureg(X86Reg.RIP.value, 0)
        # get DTB
        cr3 = vmi.get_vcpureg(X86Reg.CR3.value, 0)
        dtb = cr3 & ~0xfff
        # get paddr
        paddr = vmi.pagetable_lookup(dtb, rip)
        gfn = paddr >> 12

        # 3 - define mem-event callback
        def cb_mem_event(vmi, event):
            logging.info("Mem event at RIP: %s, frame: %s, offset: %s, permissions: %s",
                         hex(event.x86_regs.rip), hex(event.gla), hex(event.offset), event.out_access.name)

        # 4 - define and register mem-event
        mem_event = MemEvent(MemAccess.X, cb_mem_event, gfn=gfn)
        vmi.register_event(mem_event)
    # 5 - listen
    while not interrupted:
        vmi.listen(500)
    logging.info("stop listening")
```

LibVMI: Python Bindings - MSR Events

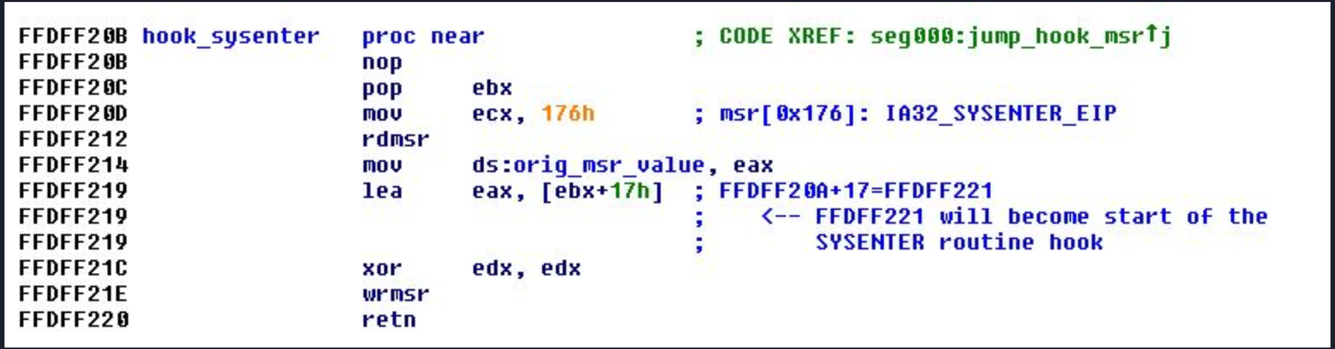


```python
with Libvmi(vm_name, INIT_DOMAINNAME | INIT_EVENTS, init_data=kvm_socket, partial=True) as vmi:

    # define MSR callback
    def msr_callback(vmi, event):
        logging.info("%s %s = %s", name, hex(event.msr), hex(event.value))

    # define and register MSR event
    with pause(vmi):
        reg_event = RegEvent(MSR.ALL, RegAccess.W, msr_callback)
        vmi.register_event(reg_event)

    # listen for events
    logging.info("listening")
    while not interrupted:
        vmi.listen(500)
```

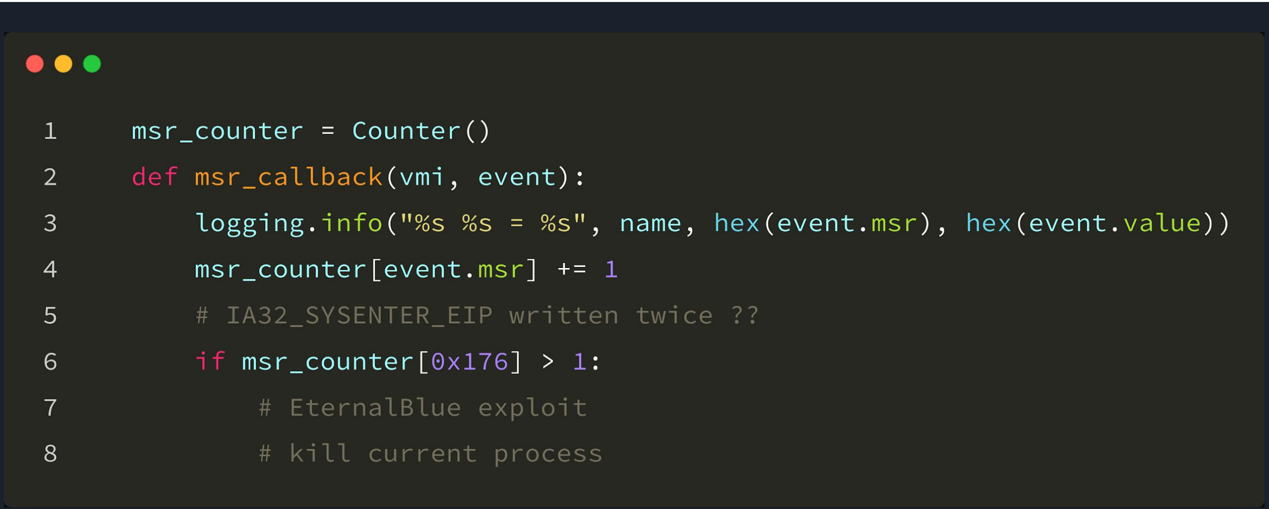
LibVMI: Python Bindings - Demo

Click Me,
I’m a video

LibVMI: Syscall Entrypoint OS Hardening



```
FFDFF20B hook_sysenter    proc near                 ; CODE XREF: seg000:jump_hook_msr↑j
FFDFF20B                  nop
FFDFF20C                  pop     ebx
FFDFF20D                  mov     ecx, 176h         ; msr[0x176]: IA32_SYSENTER_EIP
FFDFF212                  rdmsr
FFDFF214                  mov     ds:orig_msr_value, eax
FFDFF219                  lea     eax, [ebx+17h]    ; FFDFF20A+17=FFDFF221
FFDFF219                                            ;     <-- FFDFF221 will become start of the
FFDFF219                                            ;         SYSENTER routine hook
FFDFF21C                  xor     edx, edx
FFDFF21E                  wrmsr
FFDFF220                  retn
```



```
msr_counter = Counter()
def msr_callback(vmi, event):
    logging.info("%s %s = %s", name, hex(event.msr), hex(event.value))
    msr_counter[event.msr] += 1
    # IA32_SYSENTER_EIP written twice ??
    if msr_counter[0x176] > 1:
        # EternalBlue exploit
        # kill current process
```

KVM as a debugging platform

pyvmidbg on KVM: Windows 10 x64

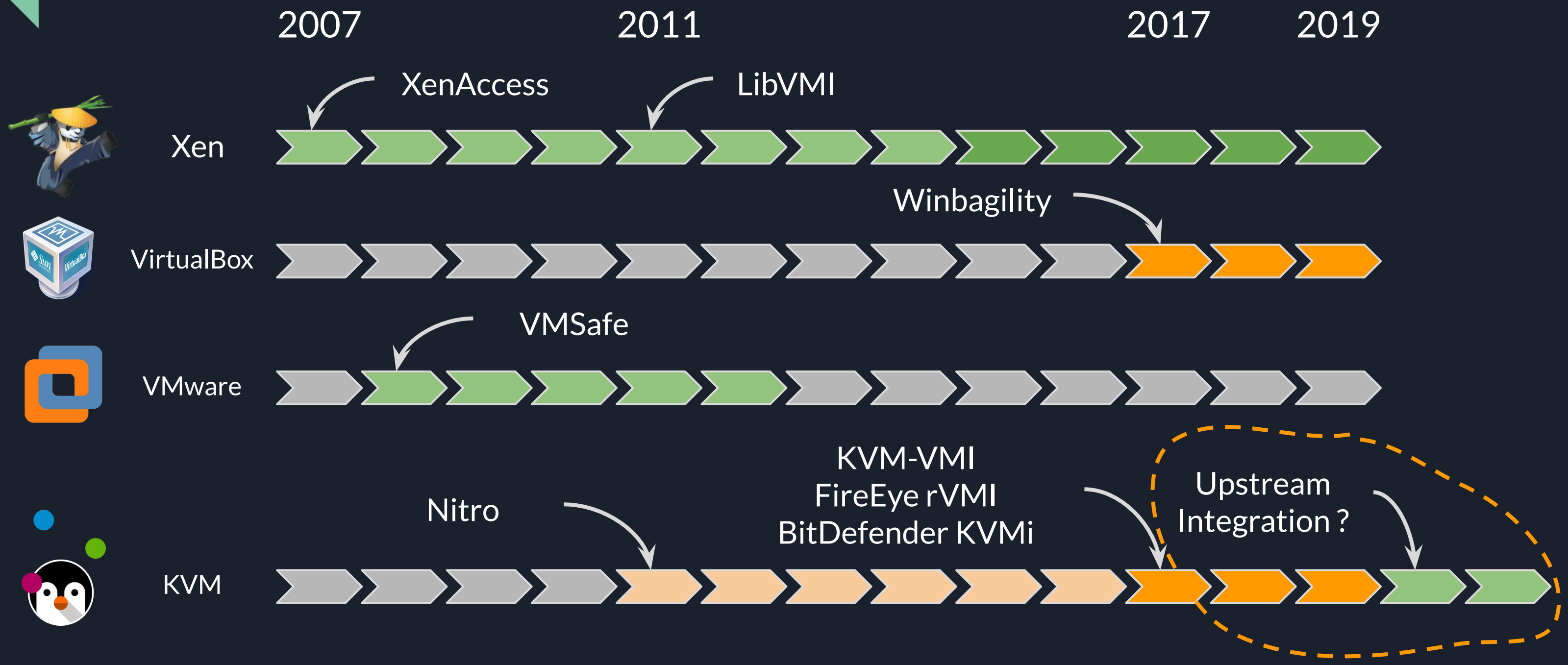
Click Me,
I’m a video

Room for Improvements

- Entrypoint
  - already implemented, but lacks proper pagefault injection to work
    - break on `KiUserThreadStart`
    - check process name
    - read and break on first `ETHREAD.StartAddress` (`ntdll!RtlUserThreadStart`)
    - read and break on entrypoint (`pfnStartAddr` parameter)
- Pagefaults
  - API is available in LibVMI/KVMi, used in pyvmidbg, but not working yet
- Symbols loading
  - enumerate loaded libraries
  - get full image path
  - run libguestfs instance to dynamically copy the binary on the host !
- Stealth breakpoints
  - guard int3 by memory access events, return fake data
  - alternate SLAT API in KVMi ?
- Add more Stubs
  - KD
  - LLDB

Future

VMI API: Hypervisor Support



2007 2011 2017 2019

Xen

XenAccess

LibVMI

VirtualBox

Winbagility

VMware

VMSafe

KVM

Nitro

KVM-VMI
FireEye rVMI
BitDefender KVMi

Upstream
Integration ?

🟧 Patches available (community)

🟩 Upstream integration

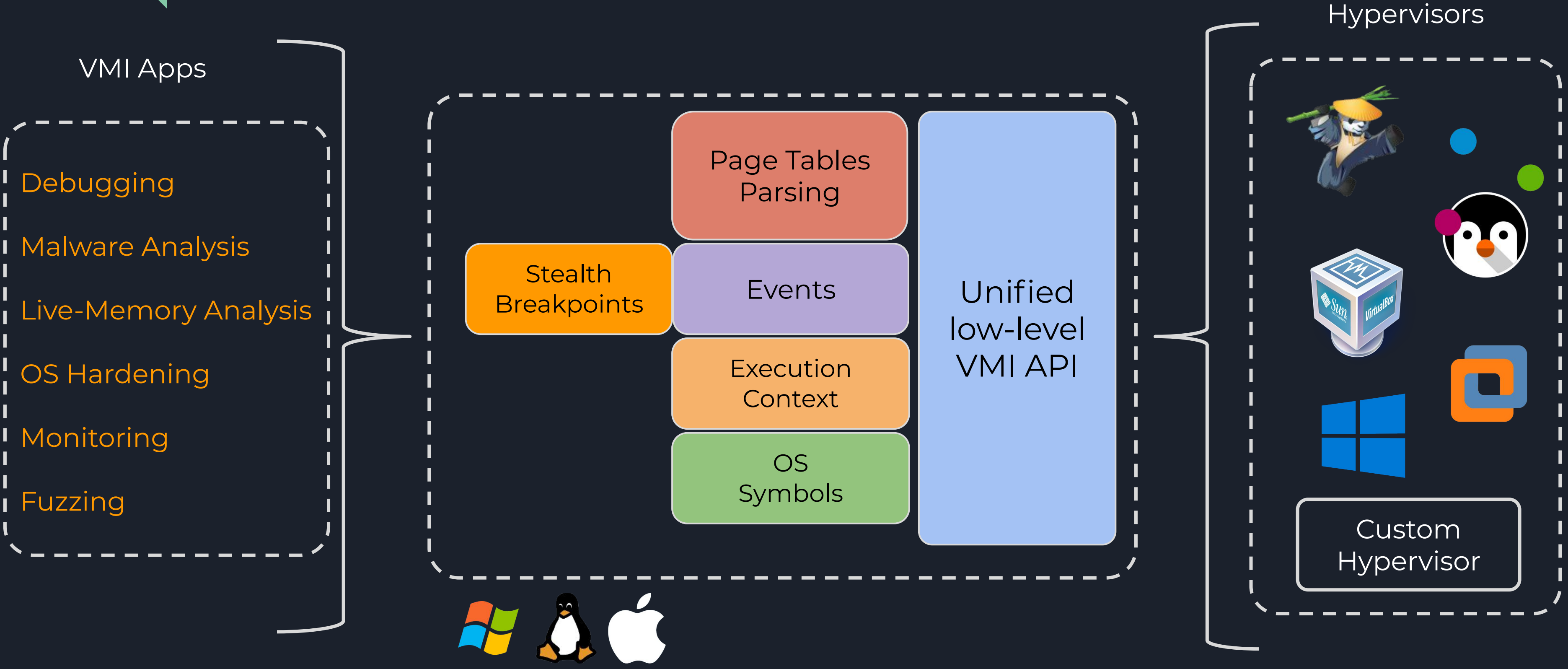
🟩 Alternate EPT/RVI available

Introspection on KVM: Use Cases

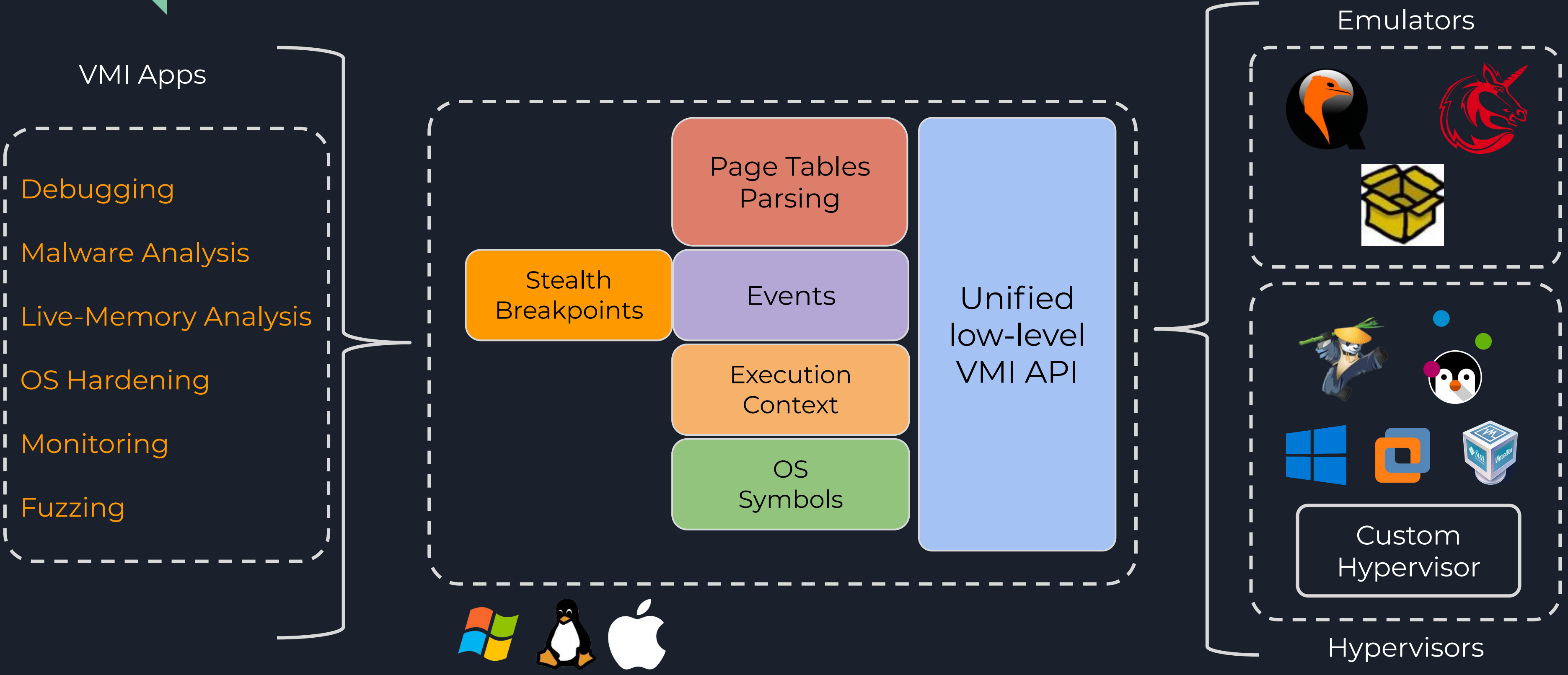
KVM-based

- Debugging
- Malware Analysis
- Live-Memory Analysis
- OS Hardening
- Monitoring
- Fuzzing

Towards a High-Level VMI Abstraction Library



Towards a High-Level VMI Abstraction Library



VMI Apps

- Debugging
- Malware Analysis
- Live-Memory Analysis
- OS Hardening
- Monitoring
- Fuzzing

Page Tables Parsing

Stealth Breakpoints

Events

Execution Context

OS Symbols

Unified low-level VMI API

Emulators

Custom Hypervisor

Hypervisors

Towards a High-Level VMI Abstraction Library

VMI Apps

Dynamic Analysis

- pyvmidbg
- icebox
- rVMI
- LiveCloudKd
- DECAF
- PANDA
- PyREBox
- Drakvuf

Live-Memory Analysis

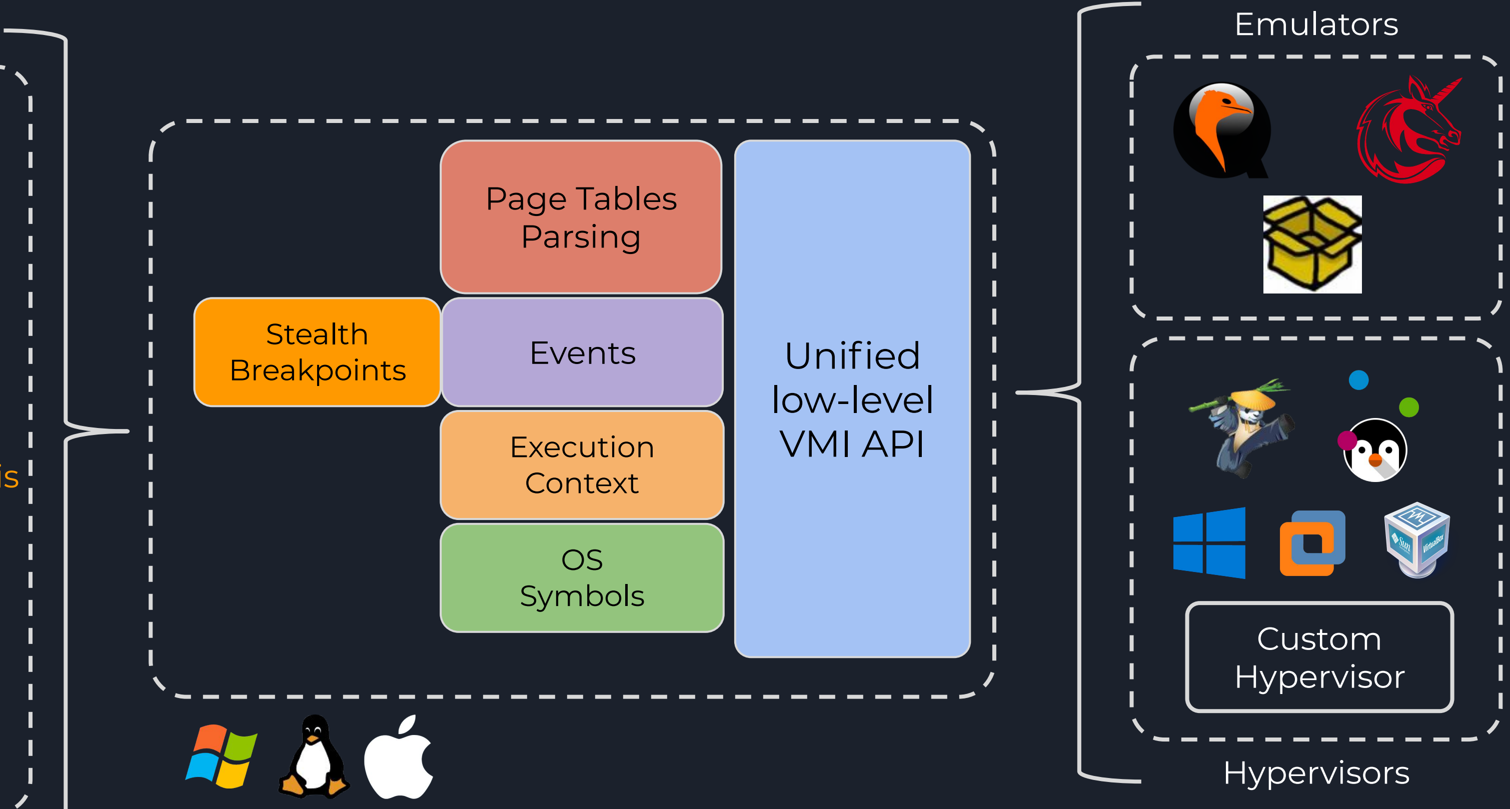
- Volatility
- Rekall

OS Hardening

Monitoring

Fuzzing

- ApplePie



High-Level VMI Library: What about Rust ?

https://github.com/Wenzel/libmicrovmi

VMI Apps

Dynamic Analysis

- pyvmidbg
- icebox
- rVMI
- LiveCloudKd
- DECAF
- PANDA
- PyREBox
- Drakvuf

Live-Memory Analysis

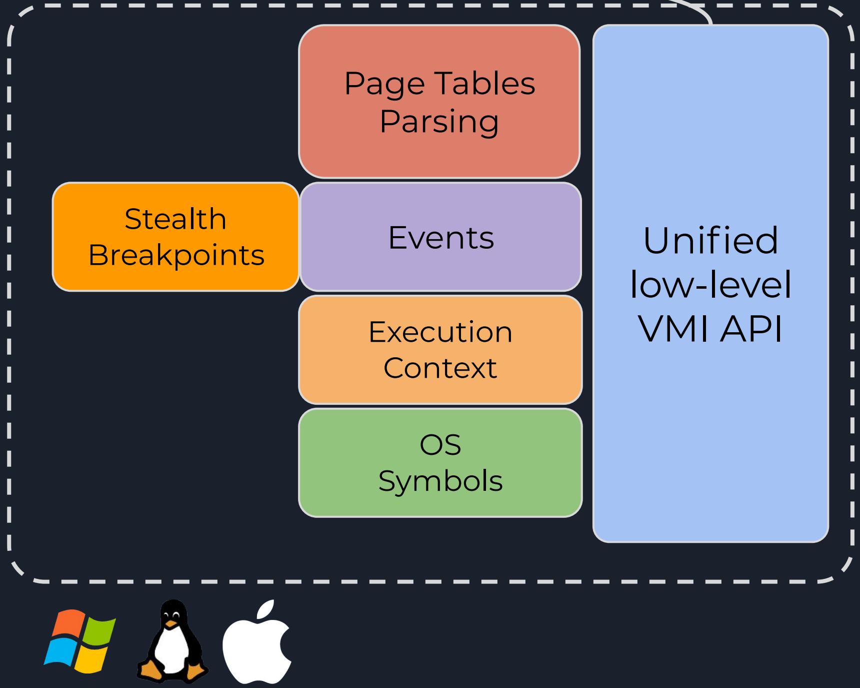
- Volatility
- Rekall

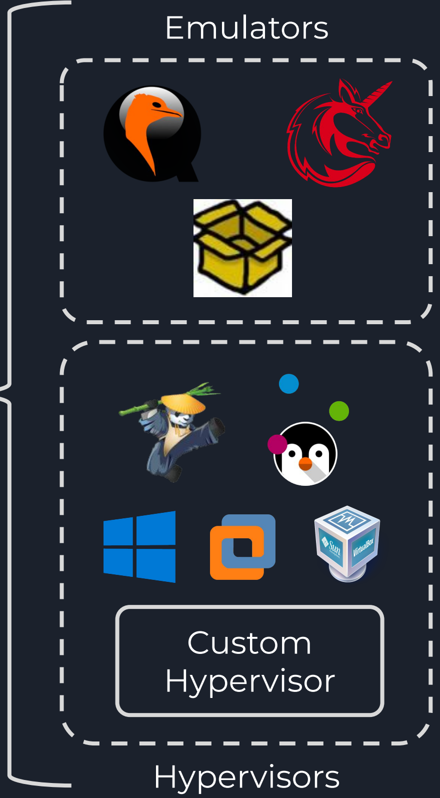
OS Hardening

Monitoring

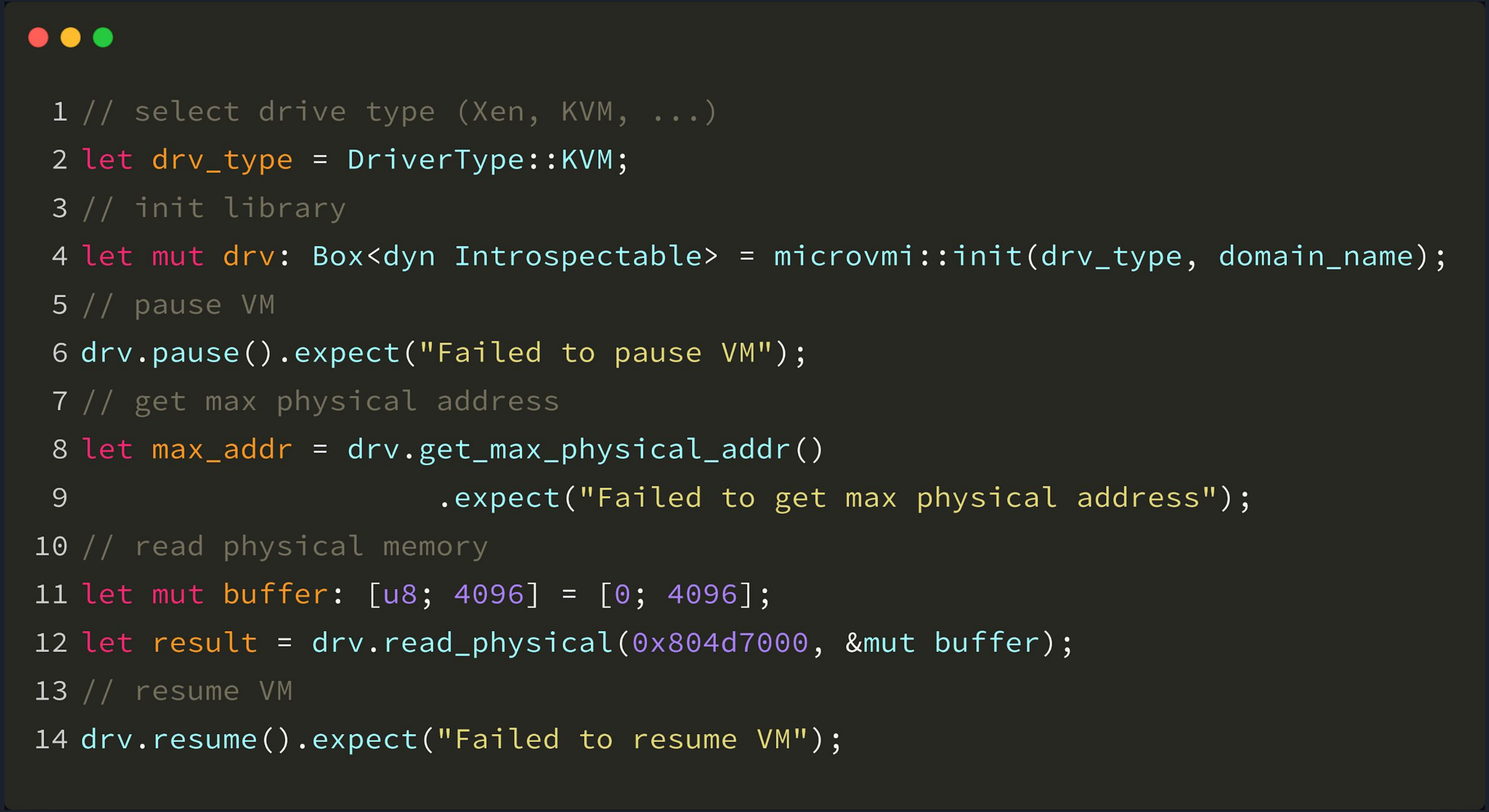
Fuzzing

- ApplePie





Libmicrovmi : API example



```rust
// select drive type (Xen, KVM, ...)
let drv_type = DriverType::KVM;
// init library
let mut drv: Box<dyn Introspectable> = microvmi::init(drv_type, domain_name);
// pause VM
drv.pause().expect("Failed to pause VM");
// get max physical address
let max_addr = drv.get_max_physical_addr()
                    .expect("Failed to get max physical address");
// read physical memory
let mut buffer: [u8; 4096] = [0; 4096];
let result = drv.read_physical(0x804d7000, &mut buffer);
// resume VM
drv.resume().expect("Failed to resume VM");
```

Conclusion

1. Native introspection on KVM is becoming a reality
2. A new LibVMI KVM driver is available to fully exploit its capabilities
3. KVM-based full-system debugging is just one of many possibilities
4. Building a high-level, cross-platform, multi-hypervisor VMI abstraction library is a condition to let the VMI ecosystem grow and mature

Thanks

- Mihai Dontu & Adalbert Lazar (Bitdefender)
  - adding singlestep support 35 days before the talk ;)
- Hady Azzam (@hady_azzam)
  - adding Linux support to pyvmidbg !
- Petr Beneš
- Tamas K. Lengyel
- Hack.lu team !

Leveraging KVM as a Debugging Platform

KVM-VMI/kvm-vmi

Wenzel/pyvmidbg

Wenzel/libmicrovmi

Wenzel

mtarral

mathieu.tarral

HACK.LU
2019 22-24 October

Annex: Experimental Build

- The demos in this presentation have been made on an experimental build
    - KVM-VMI/kvm: https://github.com/adlazar/kvm/tree/kvmi-v6-single-step
        - for singlestep events support
    - mtarral/libvmi: https://github.com/mtarral/libvmi/tree/kvmi_events_v6_sstep
        - for singlestep event support in LibVMI KVM
    - KVM-VMI/python:
        - init_data: https://github.com/KVM-VMI/python/tree/init_data
            - to init LibVMI Python with a kvmi socket parameter
        - msr: https://github.com/libvmi/python/pull/48
            - add support for MSR registers
    - Wenzel/pyvmidbg:
        - kvm_support: https://github.com/Wenzel/pyvmidbg/pull/40
            - add KVM support