Monitoring & Controlling Kernel-mode Events by HyperPlatform

Satoshi Tanda
Threat Researcher

SOPHOS

Friend me.

Takeaway

- If you want to have more ability to monitor and control Windows system activities in a lightweight manner, HyperPlatfrom is for you
- HyperPlatfrom is the hypervisor designed as a VM-exit filtering platform to utilize virtualization technology (VT) and write new types of tools on Windows quicker and easier

About Us

- Satoshi Tanda (@standa_t)
  - Reverse engineer interested in the Windows kernel
  - Implemented HyperPlatform
  - Threat Researcher at Sophos specializing in behaviour based detection on Windows
- Igor Korkin (@Igorkorkin)
  - An independent researcher focusing on cyber security science: memory forensics, rootkit detection & spy technologies
  - Co-researcher, focused on application of HyperPlatform

Background

- Issue: Lack of tools for kernel mode code analysis on Windows
  - Debugger and IDA are time consuming
  - Existing tools were not efficient
- Solution: Virtualization Technology (VT)
  - Plenty of analysis systems, and academic papers
  - VT is more than just sandbox

Challenges

- No suitable hypervisor to take advantage of VT only for system monitoring on Windows
- Existing lightweight hypervisors for Windows?
  - lacked modern platform support
- More comprehensive hypervisors?
  - Too large to understand and extend
  - Not straightforward to compile and run
  - Very slow (i.e., Bochs)

Challenges: Summary

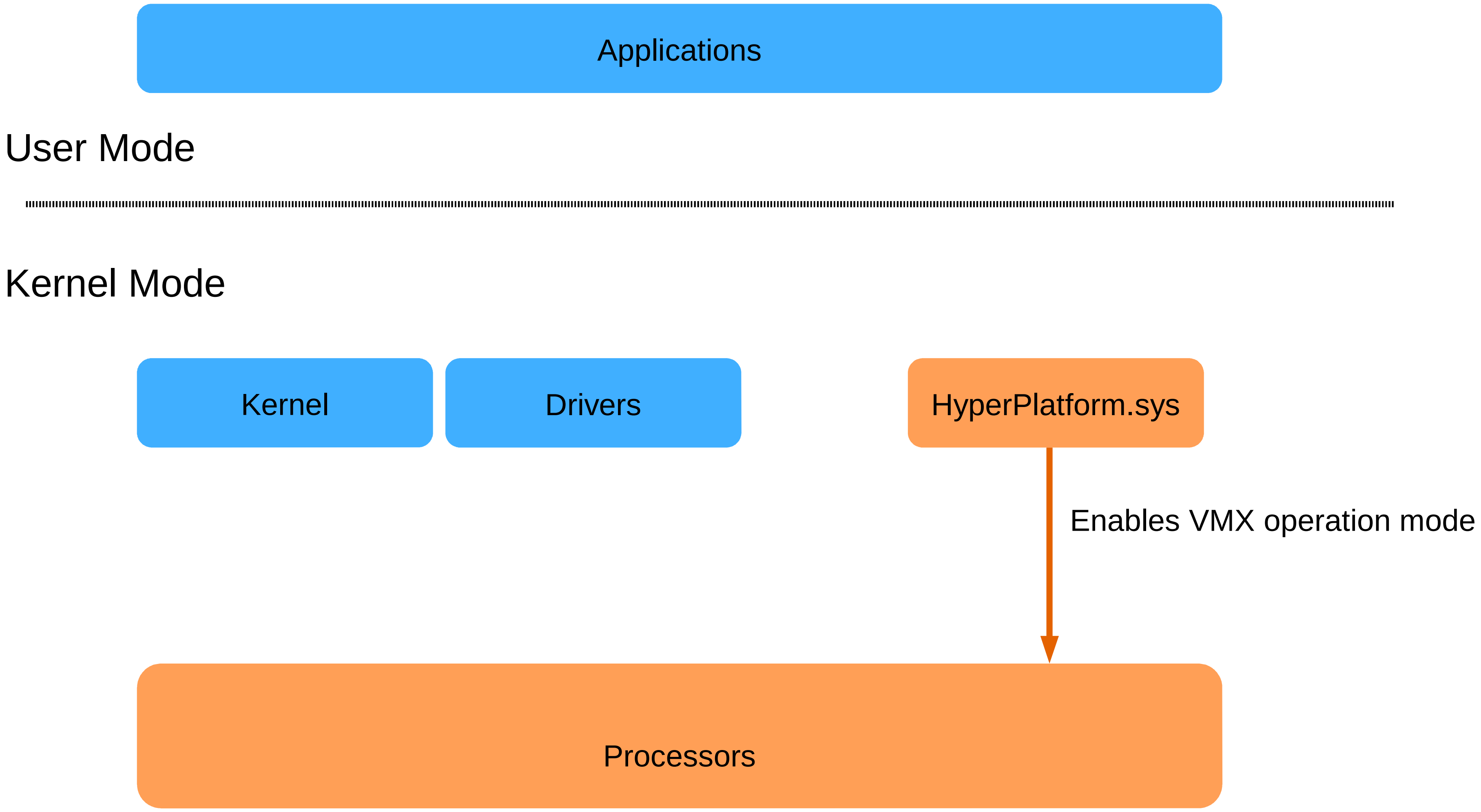
- Lack of tools to monitor kernel activities
- Commercial and proprietary
- Insufficient modern platform support
- Large to use VT just for system monitoring
- Not Windows researchers friendly
- Too slow

ACCEPTED

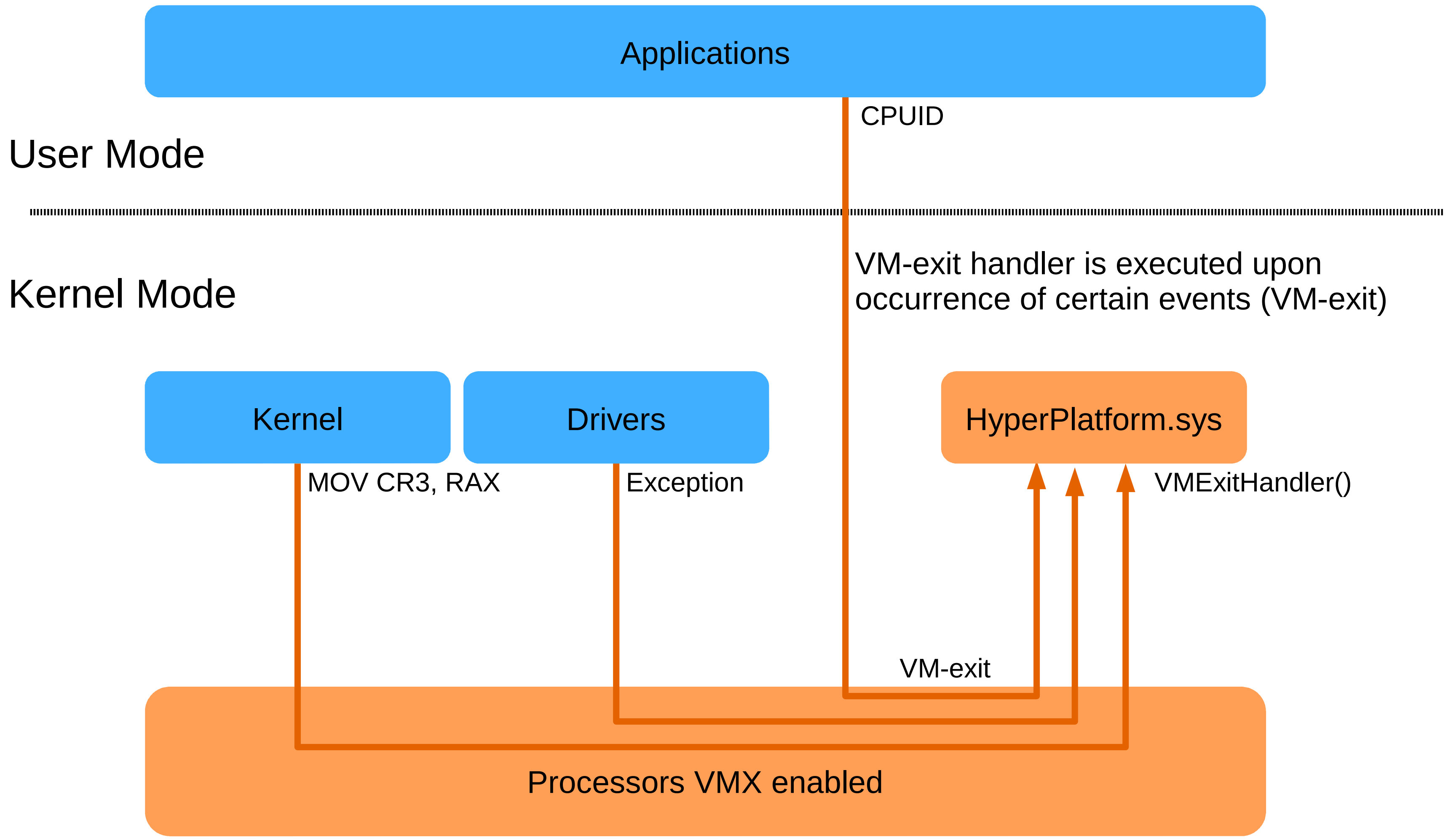
Answer: HyperPlatform

- Allows you to monitor system activities incl. kernel-mode
- Open source under the relaxed license (MIT License)
- Supports Windows 7-10 on x86/x64
- Small (7KLOC)
- Can be compiled on Visual Studio w/o any 3rd party libraries, and debugged just like ordinary Windows drivers
- Fast (about 10% of overhead)

How It Works: Overview



How It Works: Overview



How It Works: Implementation

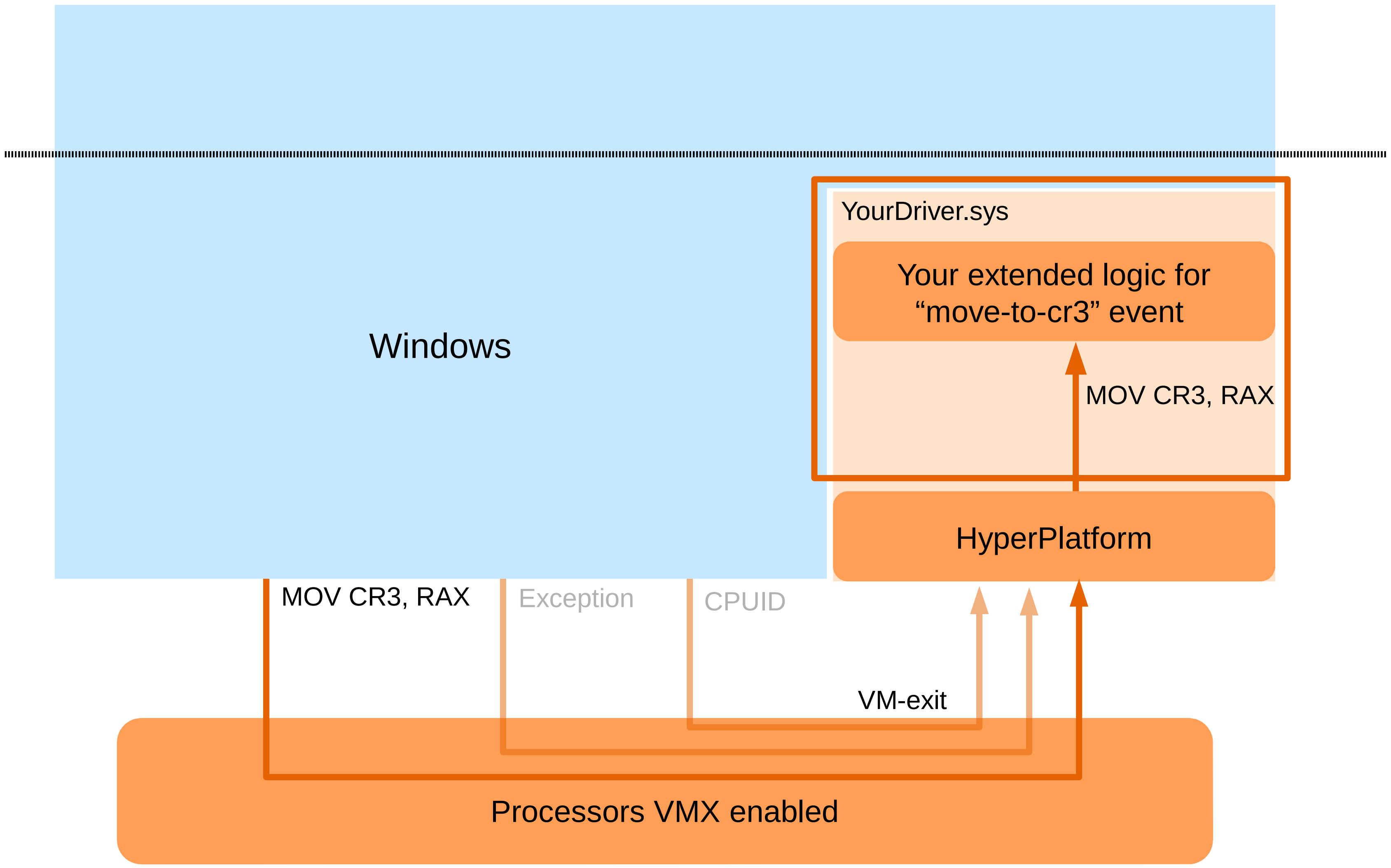
```
void VMExitHandler(
    GuestRegisters* context,
    int exit_reason)
{
    switch (exit_reason)
    {
        case VMEXIT_CPUID:
            CpuidHandler(context); break;
        case VMEXIT_EXCEPTION:
            ExceptionHandler(context); break;
        //...
    }
}
```

Invoked on VM-exit

Context of the system and VM-exit reason are given

Handle an event accordingly

As a VM-exit Filtering Platform

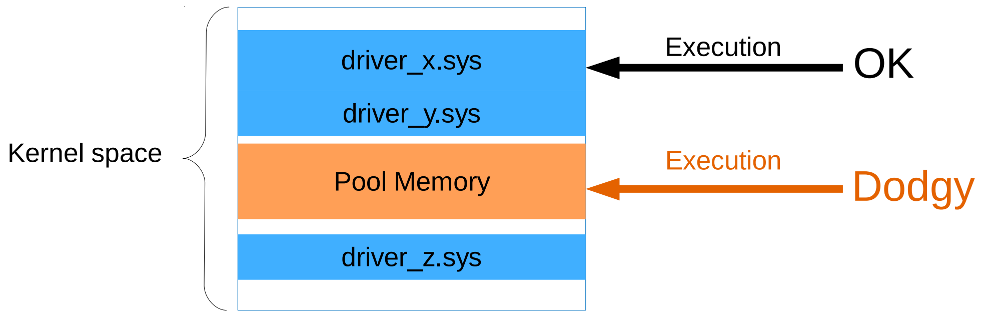


Advantage

- You can do what you cannot do without VT
- VM-exit is a new class of events
  - access to system registers
  - occurrence of exceptions and interruptions
  - execution of certain instructions
  - access to memory using extended page tables (EPT)
- VM-exit handler is flexible
  - returning different register values and/or memory contents
- None of them is easy to achieve without VT

Application (part 1)

- Kernel mode code analysis
  - Detection of dodgy instruction execution (e.g., modification of CR0.WP)
    - GuardMon – PatchGuard monitor
  - Detection of pool memory execution
    - MemoryMon – Memory execution monitor



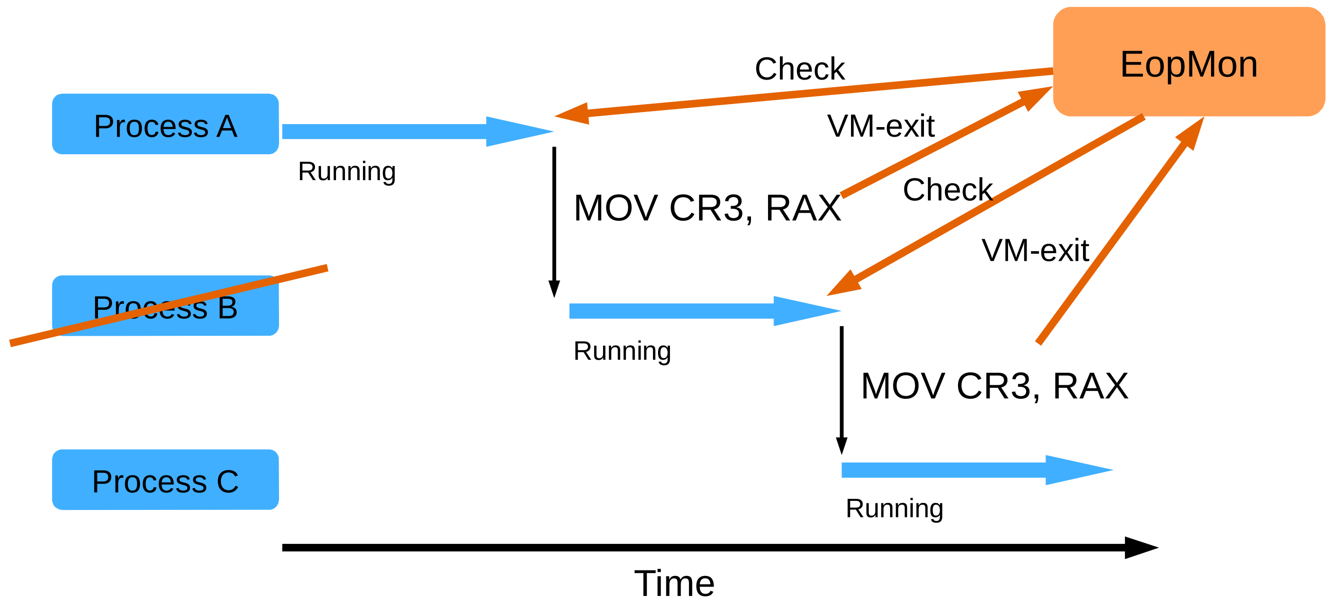
  - Invisible API hook
    - DdiMon – kernel-mode API monitor

Demo (part 1)

- MemoryMon against Turla (Uroburos)
  - getting unpacked code from memory

Application (part 2)

- Hypervisor based protection
  - Instead of monitoring, terminate a process upon dodgy events
  - Checking certain conditions on task switching
    - EopMon – elevation of privilege exploit (token stealing) monitor



Demo (part 2)

- EopMon against Gozi (Ursnif)
  - Detecting and killing elevated malware (stole a system token)

Limitations

- Cannot run inside VirtualBox by design
- No AMD processors support (#2, won't fix)
- Cannot run with other hypervisors simultaneously (#14)

Future

- Looking for more ideas on what we can do
  - Kernel code coverage with Intel Processor Trace for effective fuzzing
  - Memory access visualization and authorization
  - Race condition (TOCTOU) bug discovery with memory access monitoring

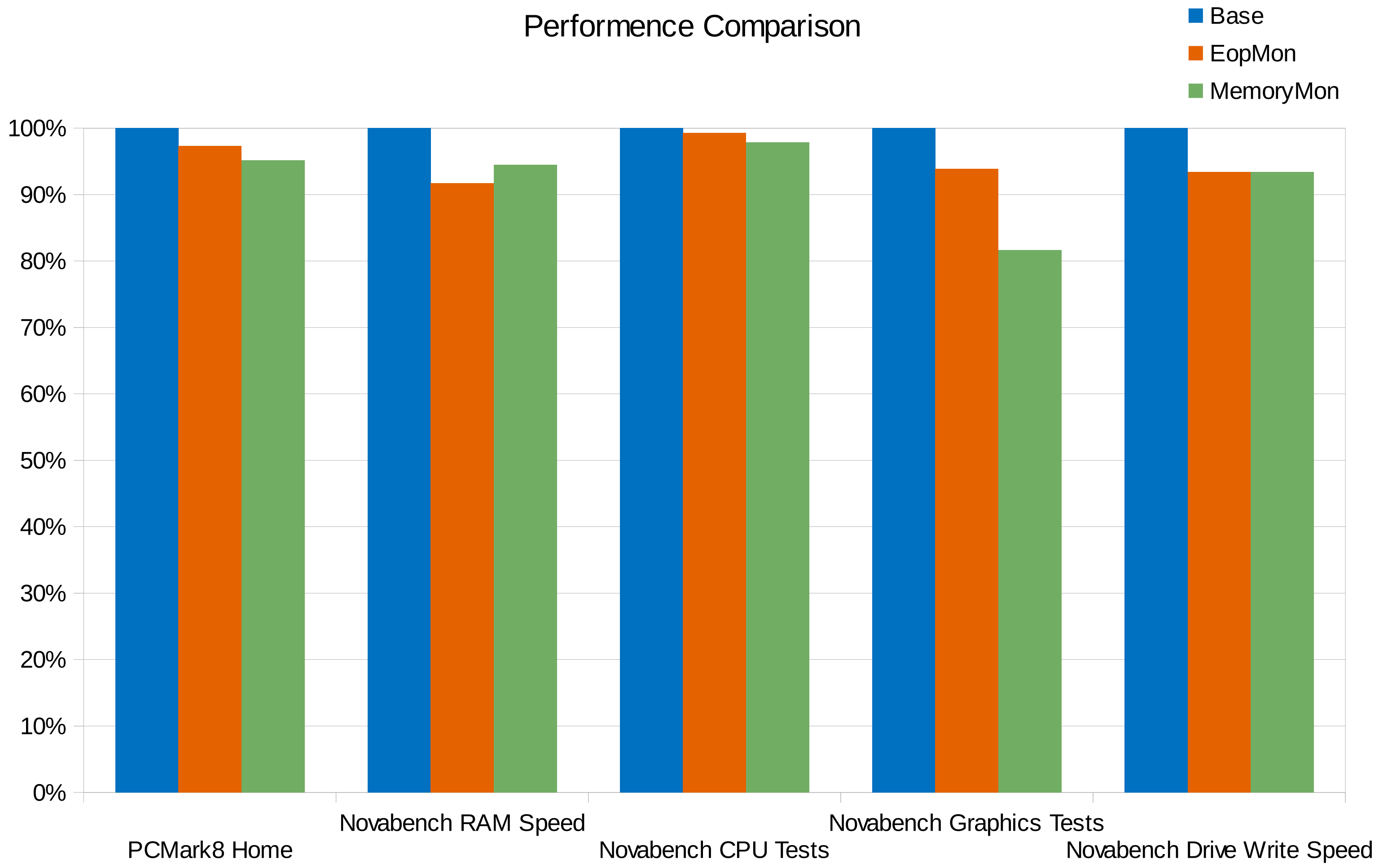
Conclusion

- Virtualization technology (VT) is powerful but underutilized in reverse engineering
- HyperPlatfrom is the hypervisor designed as a VM-exit filtering platform to utilize VT and write new types of tools on Windows quickly and easily
- Check out GitHub pages, develop your own unique ideas and solutions
    - github.com/tandasat/HyperPlatform

Thank You

- Contacts:
  - Satoshi Tanda (@standa_t)
    - tanda.sat@gmail.com
  - Igor Korkin (@Igorkorkin)
    - igor.korkin@gmail.com

Appendix 1: Performance Metrics



References 1

- VMRay
  - https://www.vmray.com/features/
- McAfee Deep Defender
  - http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/mcafee-deep-defender-deepsafe-rootkit-protection-paper.pdf
- SecVisor: A Tiny Hypervisor to Provide Lifetime Kernel Code Integrity for Commodity OSes
  - https://www.cs.cmu.edu/~arvinds/pubs/secvisor.pdf
- SPIDER: Stealthy Binary Program Instrumentation and Debugging via Hardware Virtualization
  - https://www.cerias.purdue.edu/assets/pdf/bibtex_archive/2013-5.pdf
- DRAKVUF
  - http://drakvuf.com/

References 2

- HyperDbg
  - https://github.com/rmusser01/hyperdbg
- Virtdbg
  - https://github.com/upring/virtdbg
- BluePill
  - http://invisiblethingslab.com/resources/bh07/nbp-0.32-public.zip
- MoRE
  - https://github.com/ainfosec/MoRE

References 3

- Bochs
  - https://github.com/svn2github/bochs
- Xen
  - http://xenbits.xen.org/gitweb/?p=xen.git
- QEMU
  - http://git.qemu.org/qemu.git
- VirtualBox
  - https://www.virtualbox.org/