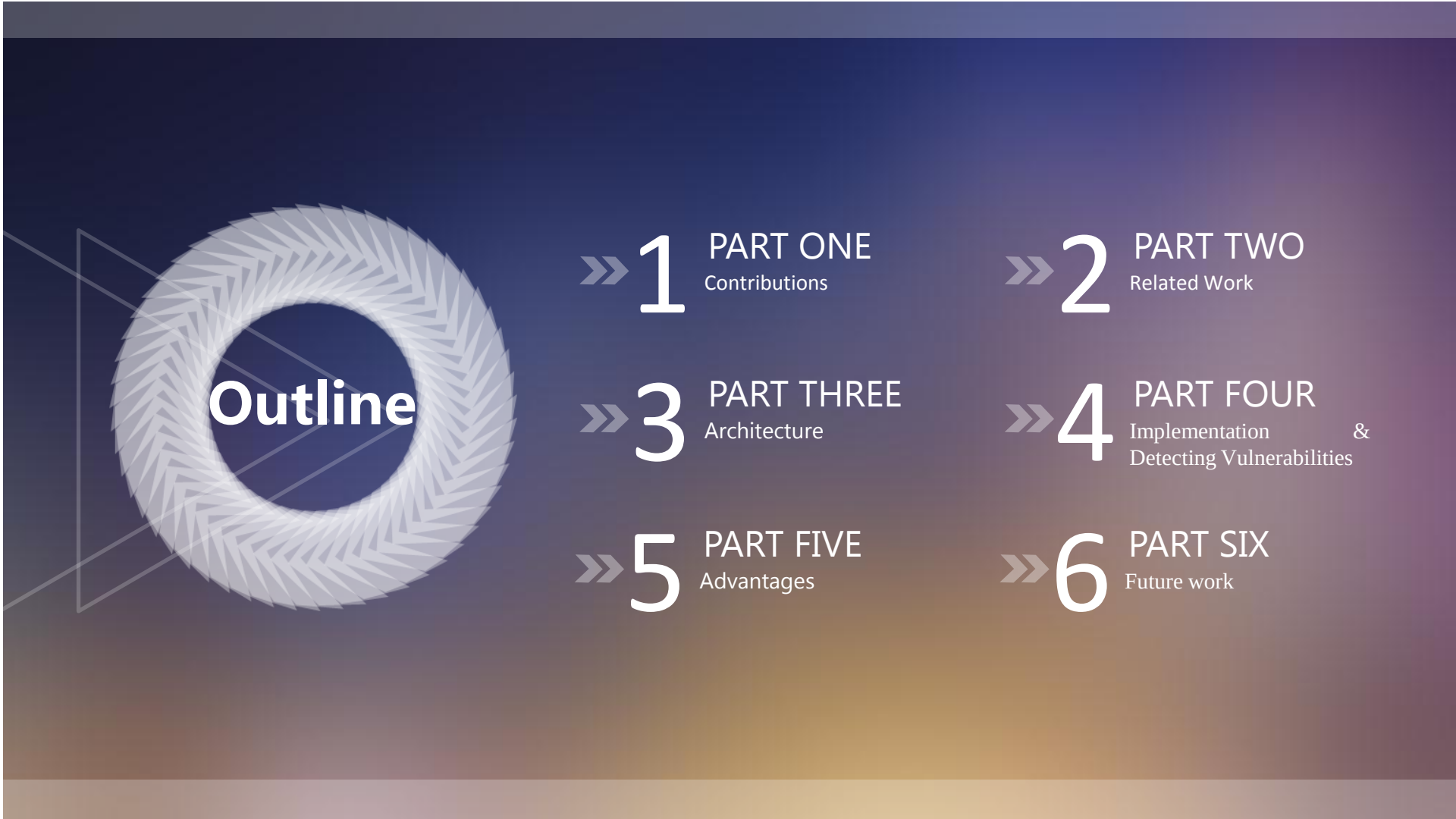
Digtool: A Virtualization-Based Framework for Detecting Kernel Vulnerabilities

Jianfeng Pan, Guanglu Yan, **Xiaocao Fan**

IceSword Lab, 360 Internet Security Center

USENIX Security 2017 - VANCOUVER, BC, CANADA


Outline
1
PART ONE
Contributions
2
PART TWO
Related Work
3
PART THREE
Architecture
4
PART FOUR
Implementation &
Detecting Vulnerabilities
5
PART FIVE
Advantages
6
PART SIX
Future work

Contributions

- Digtool is a bug checking framework
  - Based on virtualization
  - No compile-time requirement
  - For Windows kernel and device driver
  - Detecting UNPROBE, TOCTTOU, UAF, OOB and some other types of vulnerabilities

- 45 kernel-level zero-day vulnerabilities (four types) were found
  - MS16-090/CVE-2016-3252
  - MS16-123/CVE-2016-7211
  - ……
  - Some third-party's driver programs.

Related Work

Virtualization/emulator-based methods

Xenpwn - BlackHat 2016
Bochspwn - SyScan 2013

Kernel-Level Analysis Tools

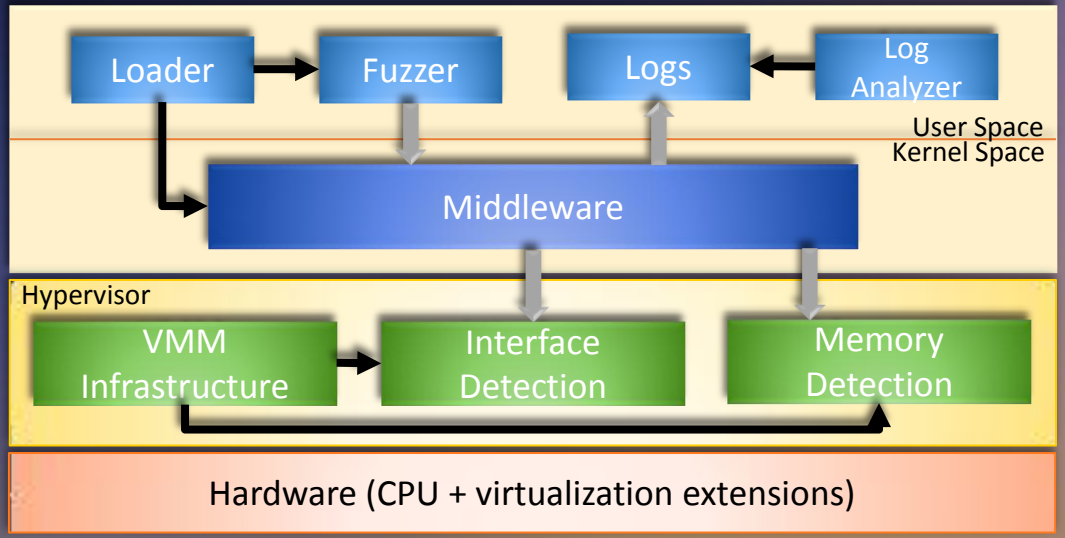
Driver verifier - Microsoft
Kmemcheck & Kmemleak
KEDR – ICST 2011

Other Tools

AddressSanitizer - Usenix ATC 2012
Valgrind - PLDI 2007
Dr. Memory - CGO 2011
DieHarder - CCS 2010

Overall Architecture

Architecture



Hypervisor

- VMM Infrastructure
  - Initializing hypervisor
  - Providing basic facilities
- Interface Detection
  - Detecting UNPROBE Vulnerabilities
  - Detecting TOCTTOU Vulnerabilities
- Memory Detection
  - Detecting UAF Vulnerabilities
  - Detecting OOB Vulnerabilities

Middleware

➢Connecting the hypervisor and user mode programs

- For Interface Detection
  - ✓ Recording behavior events into log files
  - ✓ Helping to limit the scope of system calls
  - ✓ Helping to set strategies & configuration information
- For Memory Detection
  - ✓ Calibrating monitored memory
  - ✓ Limiting monitored memory areas and kernel code
  - ✓ Interrupting guest OS

User-Space Components

- Loader
  - Loading target process
  - Distilling information from configuration file
- Fuzzer
  - Testing system calls in the detection scope
  - Exploring code branches
- Log Analyzer
  - Extracting valuable information from log files

Implementation Details & Detecting Vulnerabilities

Hypervisor

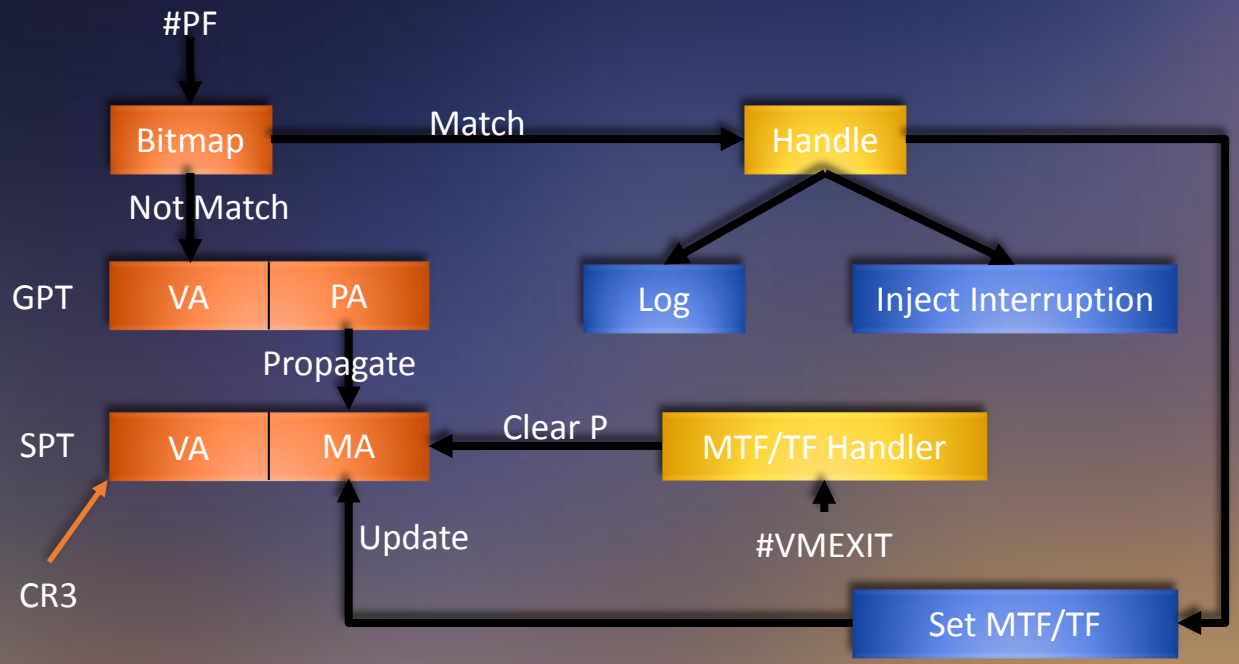
- VMM Infrastructure
  - Initializing hypervisor
  - Providing basic facilities
- Interface Detection
  - Detecting UNPROBE Vulnerabilities
  - Detecting TOCTTOU Vulnerabilities
- Memory Detection
  - Detecting UAF Vulnerabilities
  - Detecting OOB Vulnerabilities

VMM Infrastructure

- Initialization
  - Driver → hypervisor
  - OS → guest OS

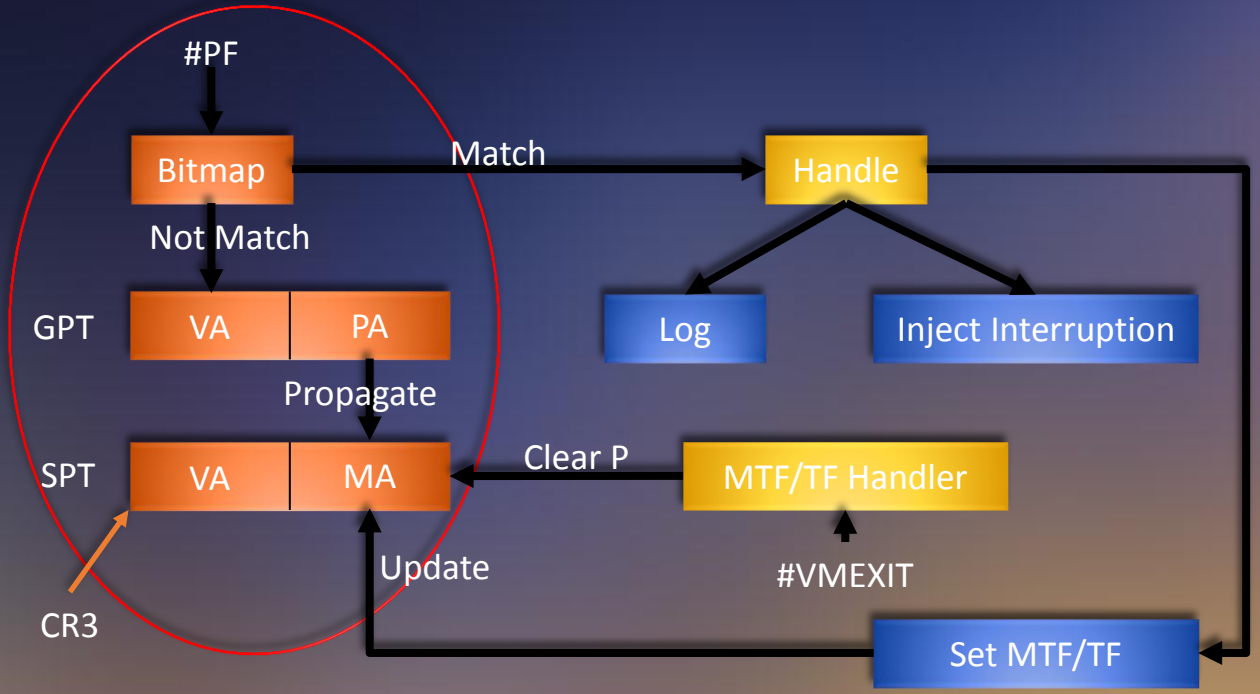
- Components
  - Virtual Pages Monitor
  - Thread Scheduling Monitor
  - Communication between Kernel and Hypervisor
  - CPU emulator
  - Events monitor

Virtual Page Monitor



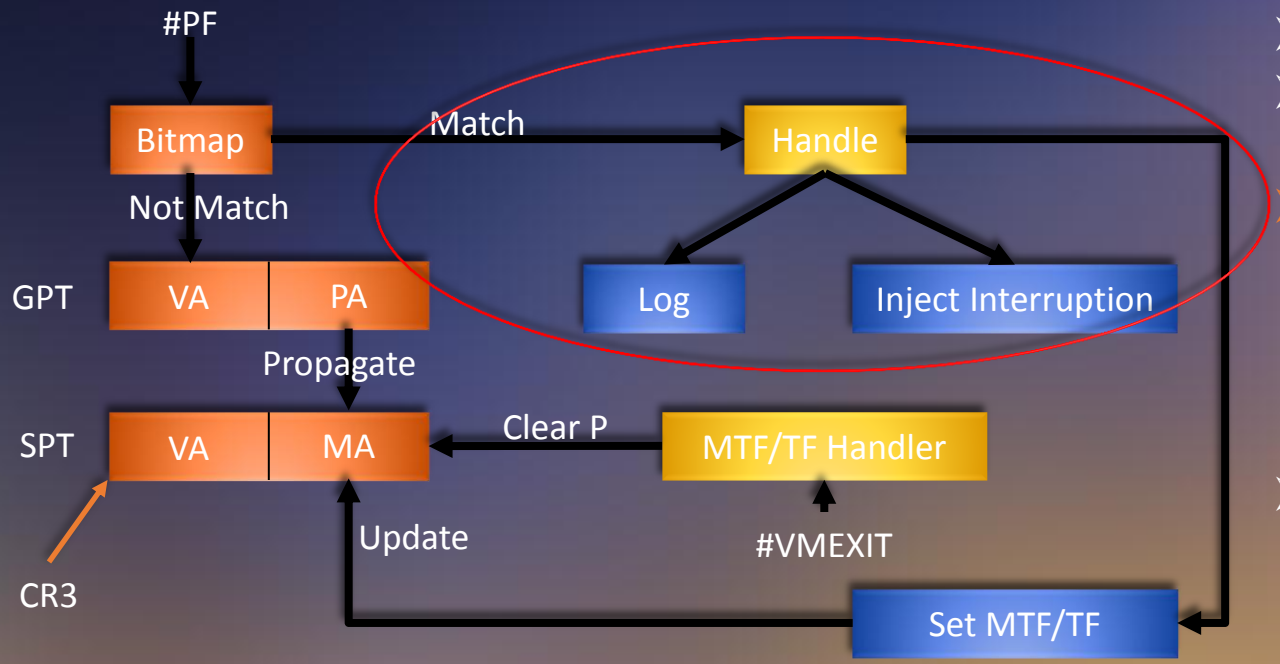
- Shadow Page Table
- BitMap
  - Recording pages
- #PF handler
  - Logging
  - Private interruption
  - Setting MTF/TF
  - Updating SPT
- MTF/TF handler
  - re-monitoring page

Virtual Page Monitor



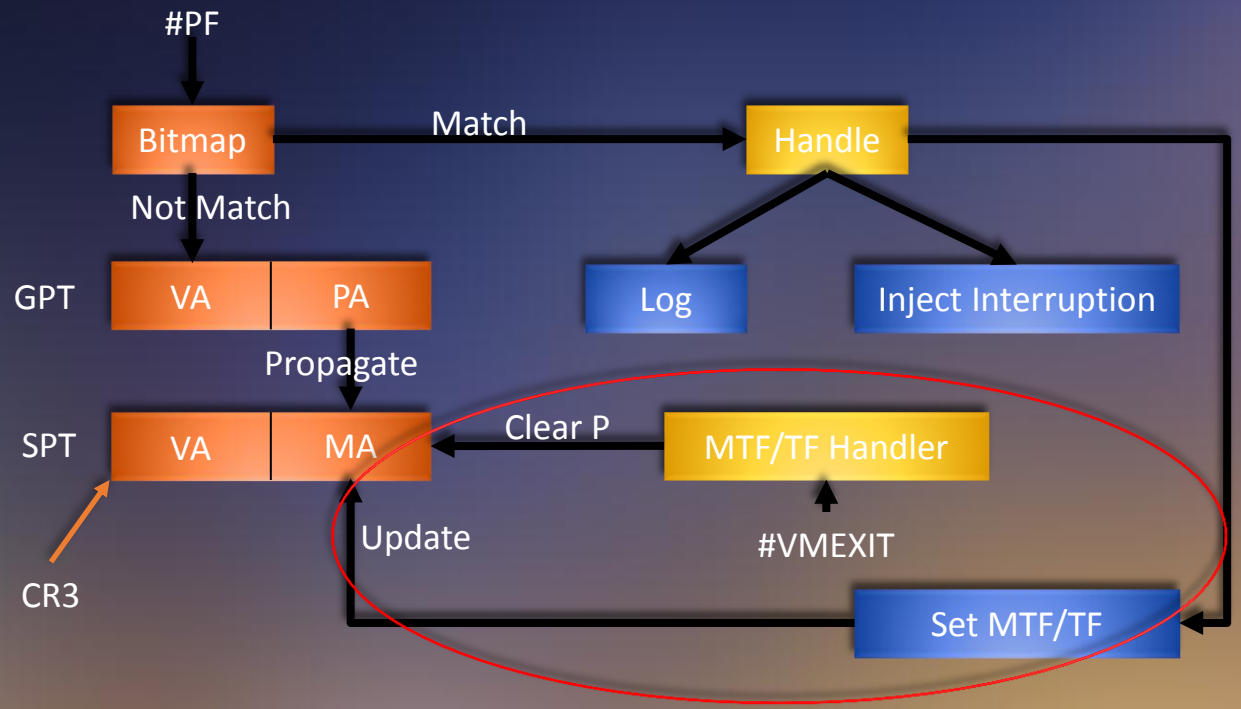
- Shadow Page Table
- BitMap
  - Recording pages
- #PF handler
  - Logging
  - Private interruption
  - Setting MTF/TF
  - Updating SPT
- MTF/TF handler
  - re-monitoring page

Virtual Page Monitor



- Shadow Page Table
- BitMap
  - Recording pages
- #PF handler
  - Logging
  - Private interruption
  - Setting MTF/TF
  - Updating SPT
- MTF/TF handler
  - re-monitoring page

Virtual Page Monitor



Memory region VS Memory page

- Shadow Page Table
- BitMap
  - Recording pages
- #PF handler
  - Logging
  - Private interruption
  - Setting MTF/TF
  - Updating SPT
- MTF/TF handler
  - re-monitoring page
  - Canceling MTF/TF

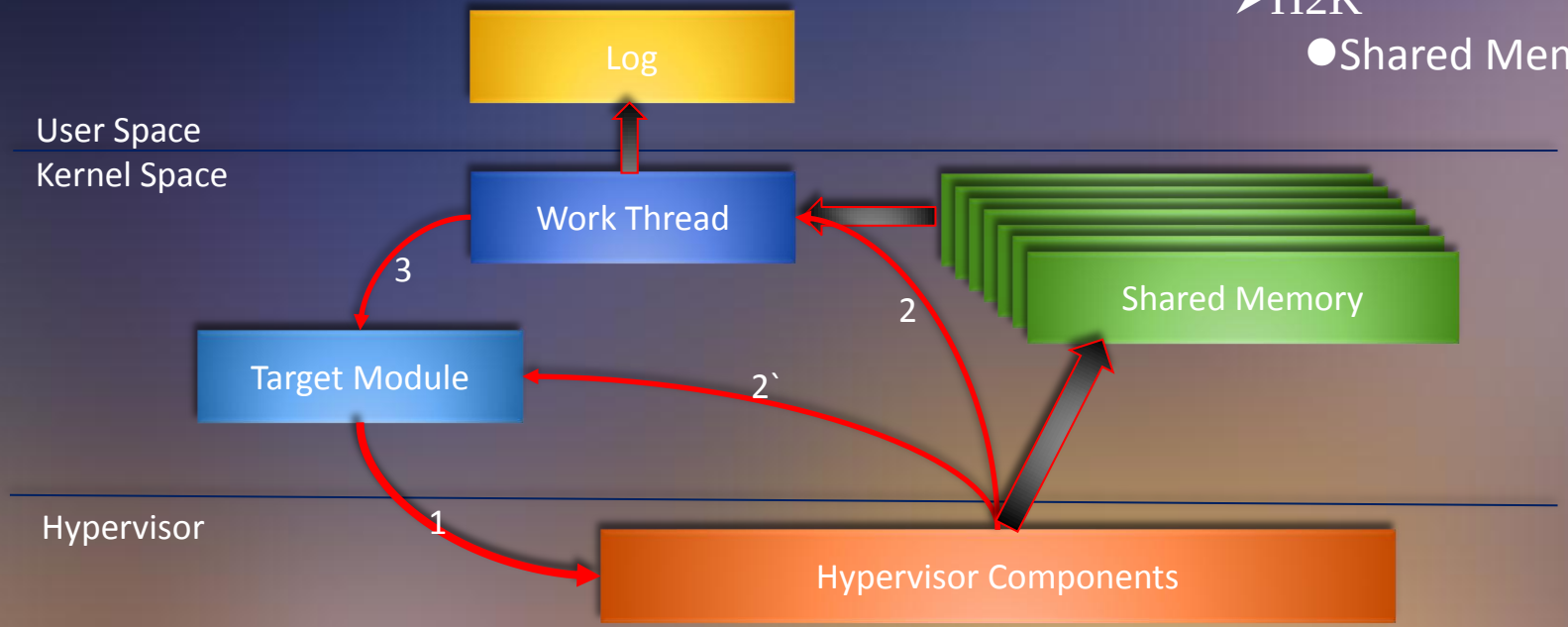
Thread Scheduling Monitor

- Target threads VS Non-monitored threads
  - SPT or GPT
  - Performance cost

- *FS->_KPCR->_KPRCB->CurrentThread*
- *Monitoring _KPRCB*

Communication between Kernel and Hypervisor

- K2H
  - Service Interfaces
- H2K
  - Shared Memory



Hypervisor

- VMM Infrastructure
  - Initializing hypervisor
  - Providing basic facilities
- Interface Detection
  - Detecting UNPROBE Vulnerabilities
  - Detecting TOCTTOU Vulnerabilities
- Memory Detection
  - Detecting UAF Vulnerabilities
  - Detecting OOB Vulnerabilities

Interface Detection —
Detecting Vulnerabilities at System Call Interface

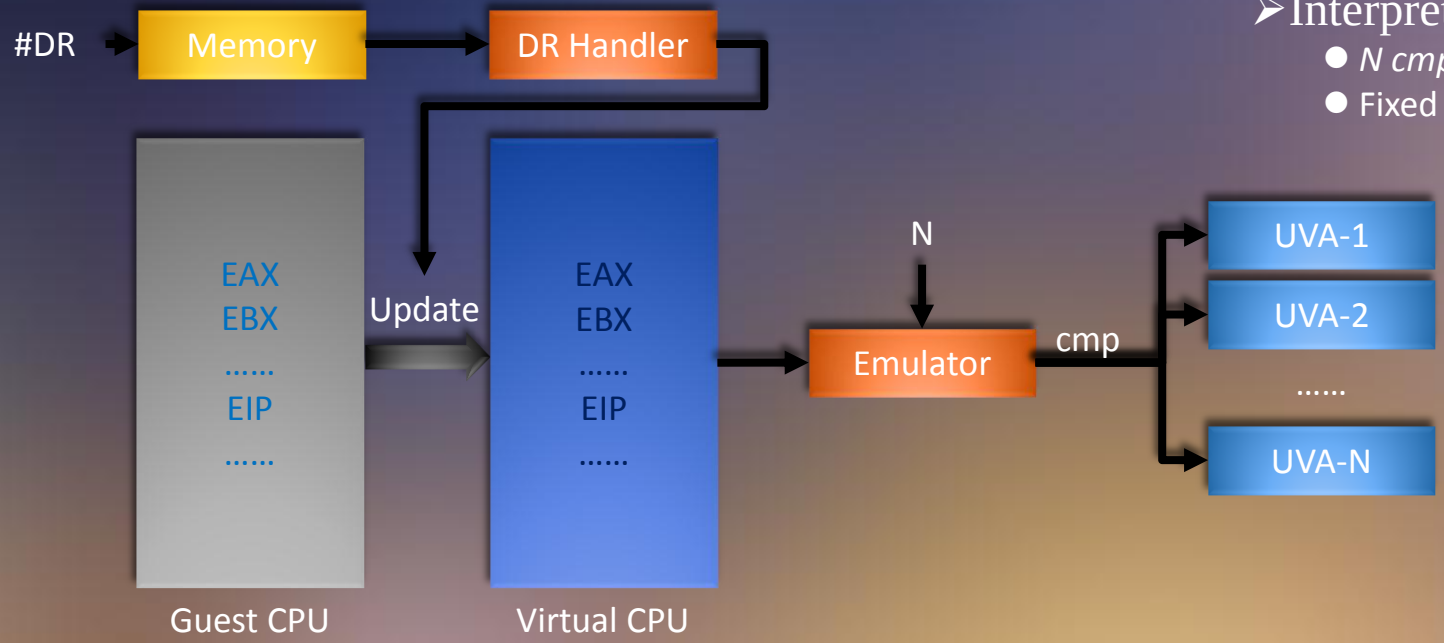
- Events monitor
  - Syscall/Trap2b/Trap2e
  - RetUser
  - MemAccess
  - ProbeRead/ProbeWrite/ProbeAccess
  - AllocVirtualMemory/GetPebTeb

CPU emulator

1) cmp esi, dword ptr [nt!MmUserProbeAddress]

2) mov eax, dword ptr [nt!MmUserProbeAddress]
   cmp eax,XXX

- ProbeAccess event
- Target memory
  - nt!MmUserProbeAddress
  - win32k!W32UserProbeAddress
- Interpreting and executing
  - *N cmp*
  - Fixed number of instructions



Detecting UNPROBE Vulnerabilities

Checking a user pointer:

- ProbeRead/ProbeWrite/ProbeAccess -> MemAccess

Accessing user memory deliberately:

- AllocVirtualMemory/GetPebTeb -> MemAccess

NtAllocateVirtualMemory :
Eip : 89993f3d, Address : 0023f304, rw: R
Eip : 84082ed9, Address : 0023f304, PROBE !
KiFastSystemCallRet

Detecting TOCTTOU Vulnerabilities

Fetching an input value from user mode memory only once

- No consecutive MemAccess events

```
NtCreateSection :
Count :3 ==============
……
Eip : 89370d54 Address :3b963c Sequence :399 rw: R
Eip : 89370d7b Address :3b963c Sequence :401 rw: R
KiFastSystemCallRet
```

Hypervisor

- VMM Infrastructure
  - Initializing hypervisor
  - Providing basic facilities
- Interface Detection
  - Detecting UNPROBE Vulnerabilities
  - Detecting TOCTTOU Vulnerabilities
- Memory Detection
  - Detecting UAF Vulnerabilities
  - Detecting OOB Vulnerabilities

Memory Detection —
Detecting Vulnerabilities via Memory Footprints

- Tracing memory allocation, release and access
  - Hooking allocation and free functions
    - ✓ Not wrapper functions
    - ✓ Memory pool & lookaside lists
  - Virtual Page Monitor → kernel memory
- Referencing to freed memory
- Accessing beyond the bounds of allocated heaps

Detecting UAF Vulnerabilities

- Tracing freed memory
- Capturing “use” instruction through virtual page monitor
- Recording “free” instruction when it invoked
- Delayed release

MS16-123/CVE-2016-7211:

```
Single step exception - code 80000004
win32k ! _ScrollDC+0x21 :
96b50f3e 83ff01 cmp edi ,1

ub 96b50f3e
96b50f3b 8b7e68 mov edi , dword ptr [esi+68h]
96b50f3e 83ff01   cmp edi ,1// win32k !_ScrollDC+0x21
```

Detecting OOB Vulnerabilities

- Tracing unallocated memory
  - Initialing unallocated memory areas
  - Adjusting the unallocated memory areas dynamically
- AVL tree
- Extra block

MS16-090/CVE-2016-3252:

Single step exception - code 80000004

win32kbase ! RGNMEMOBJ :: bFastFill +0x385 :

93e34bf9 895304 mov dword ptr [ebx +4] , edx

Advantages

- Crash resilient
  - No need of a BSOD.
- Providing an exact context
  - Stop the OS at the moment a program error occurs.
- More vulnerabilities
  - UNPROBE, TOCTTOU, UAF(MS16-123/CVE-2016-7211), OOB…
- Better performance
  - Only affect monitored threads and system calls.

Future work

- Performance optimization
  - Reduce switches between the hypervisor and guest OS
- Other detection algorithms
  - double-free, information leakage, race conditions, …
  - CVE-2017-8470, CVE-2017-8474, CVE-2017-8476, …
- Other platforms (MacOS…)

THANK YOU

IceSwordLab@360.cn