To Flexibly Tame Kernel Execution With Onsite Analysis

Xuhua Ding

Singapore Management University

Outline

- Review of existing dynamic kernel analysis techniques
- Introduction of the onsite analysis infrastructure (OASIS)
- Analysis primitives provided by OASIS
- Two examples of OASIS analyzers: function monitor and control flow tracer
- Discussions

Existing Approach 1: Code Instrumentation

Static code instrumentation:

- Linux kernel cooperates with GCC to add Kernel Coverage (KCOV) and Kernel Address SANitizer(KASAN) code into the kernel image at compilation time.
- KDB, KGDB

Dynamic Binary Instrumentation (DBI)

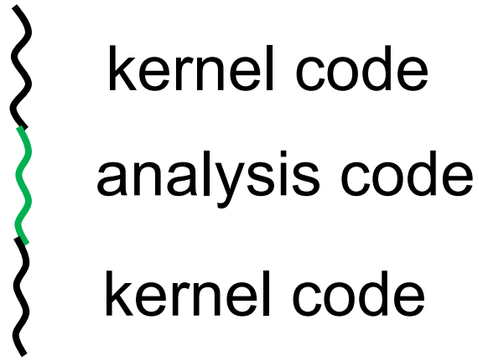
- DBI has been applied to kernel analysis as well: Cobra [S&P'06], PinOS [VEE'07], GILK [TOOLS'02], PEMU [VEE'15].

Code Instrumentation

The Idea: to *mix* the *analysis code* and the *kernel code* into one binary.

Share **execution flow** and **address space**

Pros: native control, introspection and modification
Cons: intrusive, no/weak transparency or security



Existing Approach 2: Hardware-assisted Analysis

Hypervisor based on Hardware Virtualization (VT-x)

- Ether [CCS'08], Gateway [NDSS'11], Spider [ACSAC'13]

Intel SMM + Performance Monitoring Unit (PMU)

- MALT [S&P'15]

TrustZone + ARM debugging facilities

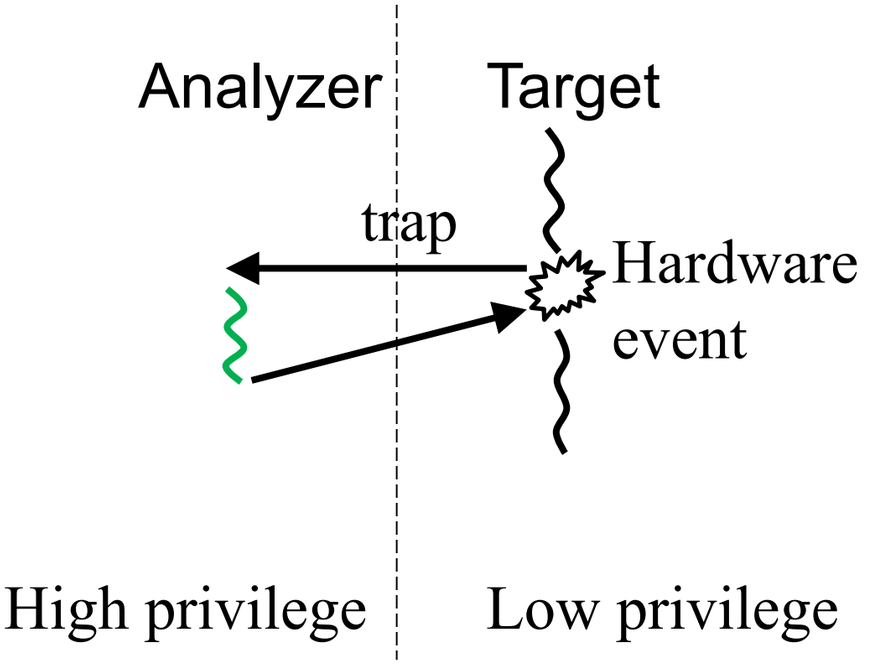
- Ninja [USENIX Security'17]

Hardware-assisted Analysis

The Idea: to *trap* the target to *an isolated and more privileged environment*, e.g., x86 VMX root mode, SMM mode, or ARM SecureWorld

Pros: transparency and security

Cons: inflexibility to control and introspect
- when/where to trigger the event
- introspection with semantic gap



Can we combine the best of the two approaches without their drawbacks?

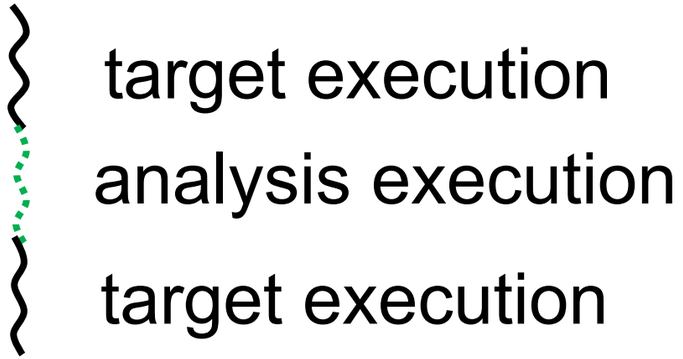
Transparency
Security

+

Native control, introspection,
& modification

What about this ...

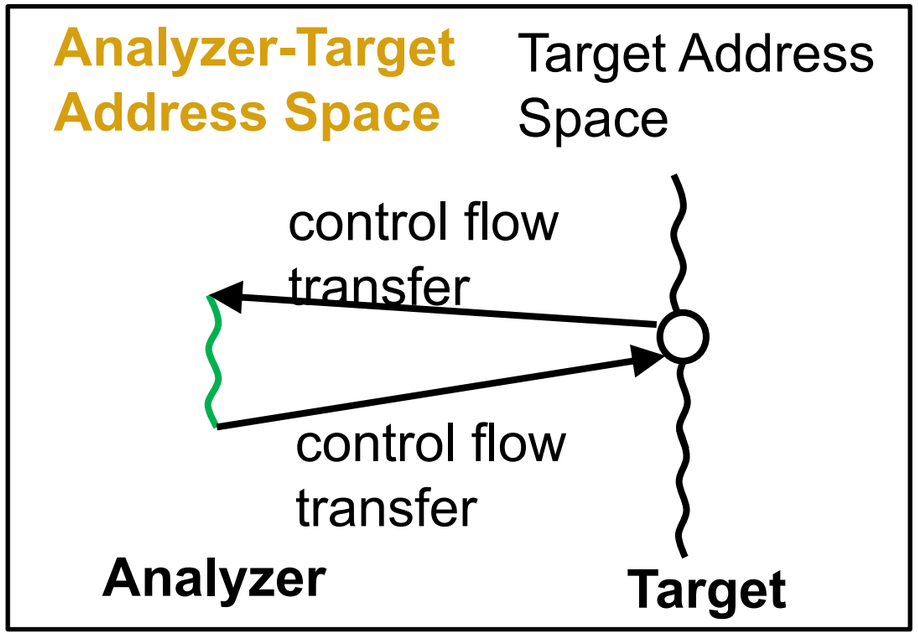
We interleave the target's instruction stream with the analyzer's without mingling their code.



Execution Flow Instrumentation (EFI)

Onsite Analysis: The analyzer analyzes the target "as if" it were one part of the target.

- The analyzer dynamically chooses the site(s) of instruction flow interleaving.
- **No CPU mode/privilege** switches between the target and the analyzer.
- **One-way address space isolation.** The target's address space is accessible to the analyzer, but **not** vice versa.



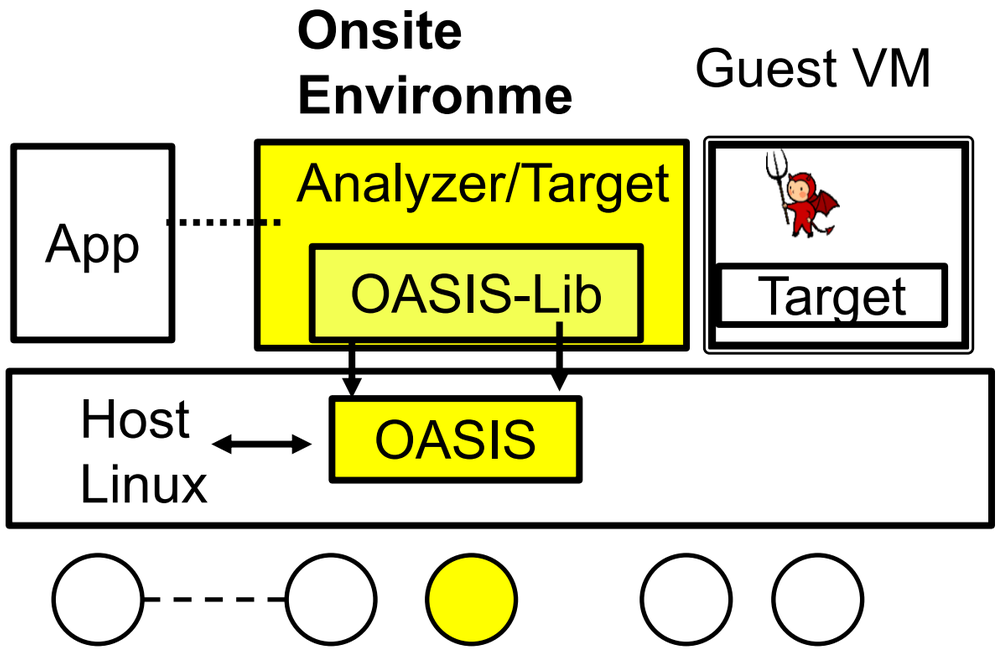
Secure | Native access

Transparent | Cross-space

System Overview

OASIS: Onsite AnalySis InfraStructure

- The target kernel runs in a guest virtual machine.
- OASIS empowers an onsite analysis application to read/write/control a captured live kernel thread.
- Most of OASIS is implemented as a host Linux kernel module running in tandem with KVM.



Onsite Environment.

- A dedicated CPU core
- a special paging hierarchy

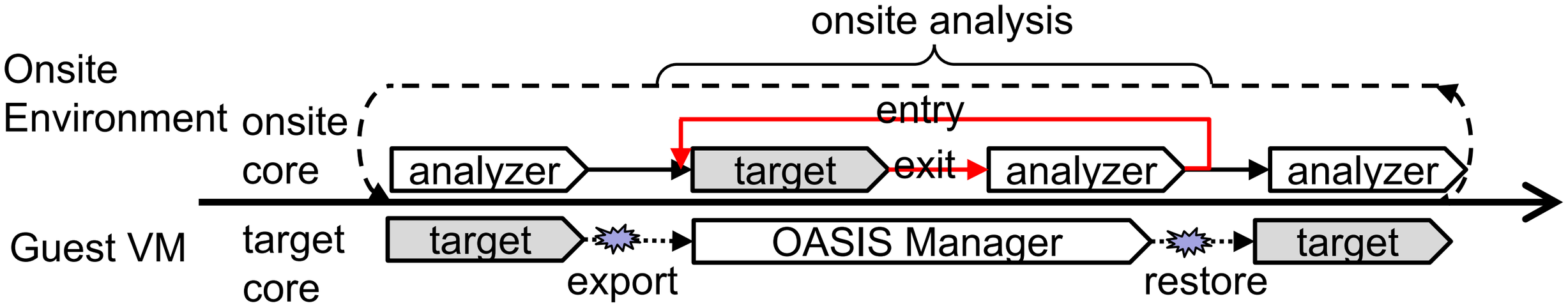
Workflow of an Onsite Analyzer

The top-level workflow

- Target thread export, onsite analysis, target thread restore.

Onsite Analysis

- Analyzer execution, target execution, analyzer execution, target execution, ...



Primitive 1: Read/Write Kernel Memory

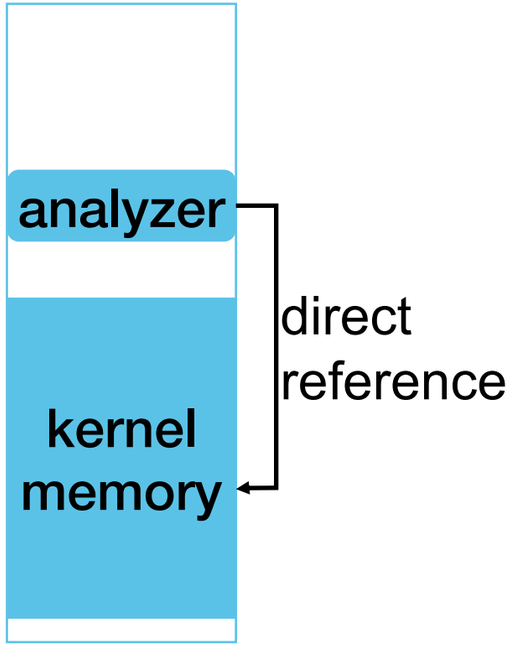
- Application developer treats the kernel memory as part of her analyzer's memory.
- Direct memory reference using kernel virtual addresses;
- Standard userspace APIs can be used.

```
void * target_addr = 0xffffffff816f3090;
struct file_security_struct obj;

memcpy(target_addr, &obj, sizeof(struct file_security_struct));

//memcpy(&obj, target_addr, sizeof(struct file_security_struct));
```

kernel VA

write to kernel memory

read kernel memory



Primitive 2: Hijack Target Execution

INT3 Probe for code breakpoint

- Replace one byte at the concerned kernel code with the int3 (0xCC) instruction.
- The interrupt handler transfers the control to OASIS Exit-Gate, a sequence of instructions that switch the underlying mapping so that the analyzer controls the CPU.

```
1. movq %rax, $rax_bak   ;save rax
2. movq %rcx,  $rcx_bak  ;save rcx
3. movq $0x0, %rax ; EPT switch
4. movq $0x9, %rcx ; 9 for A-EPT
5. vmfunc   ; switch to analyzer/target
6. jmpq *off_ana(%rip) ;to analyzer
```

paging hierarchy switch by an EPT switch

OASIS Exit-Gate

Primitive 2: Hijack Target Execution

JMP Probe for control flow tracing

- Replace 13 bytes at the concerned kernel code with: REX.W ljmp *offset(%rip)
- The long-jump instruction transfers the control to OASIS Exit-Gate via a call gate in the GDT.

Event interception

- A JMP probe is inserted to the entry of the corresponding handlers.

```
1. movq %rax, $rax_bak   ;save rax
2. movq %rcx,  $rcx_bak  ;save rcx
3. movq $0x0, %rax ; EPT switch
4. movq $0x9, %rcx ; 9 for A-EPT
5. vmfunc   ; switch to analyzer/target
6. jmpq *off_ana(%rip) ;to analyzer
```

paging hierarchy switch by an EPT switch

OASIS Exit-Gate

Primitive 3: Resume Target Execution

Resuming the target.

- Analyzer prepares the CPU context for the target execution (including RIP)
- It returns the control to the target by jumping to OASIS Entry-Gate, a sequence of instructions that switches the underlying mappings so that the target gets the control.

```
1.  movq $0x0, %rax ; EPT switch
2.  movq $0x0, %rcx ; 0 for T-EPT
3.  vmfunc   ; switch to target/lib
4.  lea 0x6(%rip), %rax ; rax points to line 7
5.  lea (%rax, %rcx, 4), %rax ;adjust rax
6.  jmpq *%rax ; jmp to Line7 if rcx=0;
7.  movq $rax_bak, %rax  ; restore rax
8.  movq $rcx_bak, %rcx  ;restore rcx
9.  nop    ;  nop slide (22 nops)
    ....
31.jmpq *off_tar(%rip) ; to target addr
```

switch to target's paging hierarchy

jump to the target destination

OASIS Entry-Gate

Example 1: Kmalloc() monitoring

- **To analyze how kmalloc() is called in a kernel thread**

```
void main ()
{
  //initialization
  ....

  OASIS_set_INT3(kmalloc_addr);
  OASIS_resume_targ(&CPU);
  return;
}
```

```
void int3_handler()
{
    //analysis workload
    ...

    if (end)
        OASIS_rm_INT3(&kmalloc_addr);

    OASIS_resume_targ(&CPU);
    return;
}
```

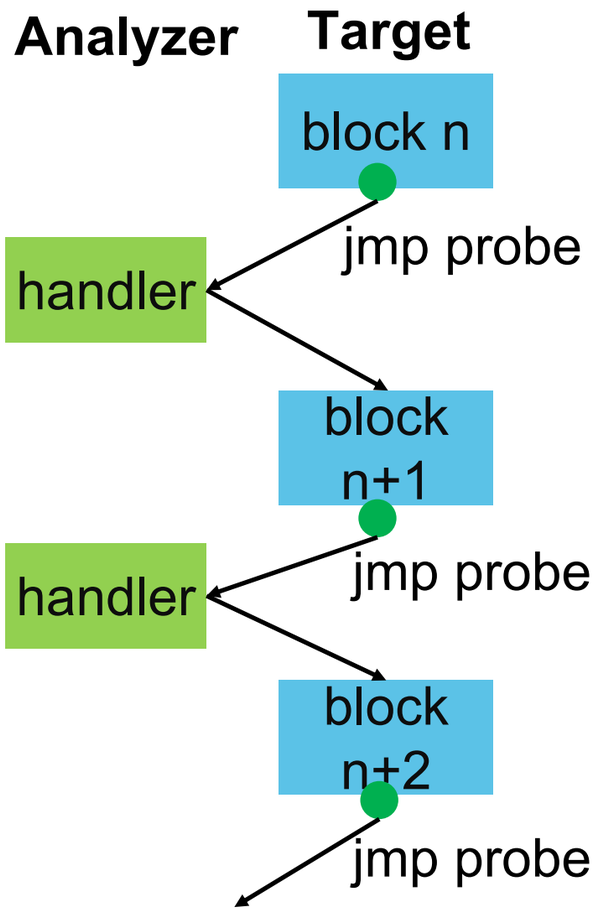
The handler function is called when the INT3-probe is encountered in the target kernel thread execution inside the onsite environment.

Example 2: Control Flow Tracing

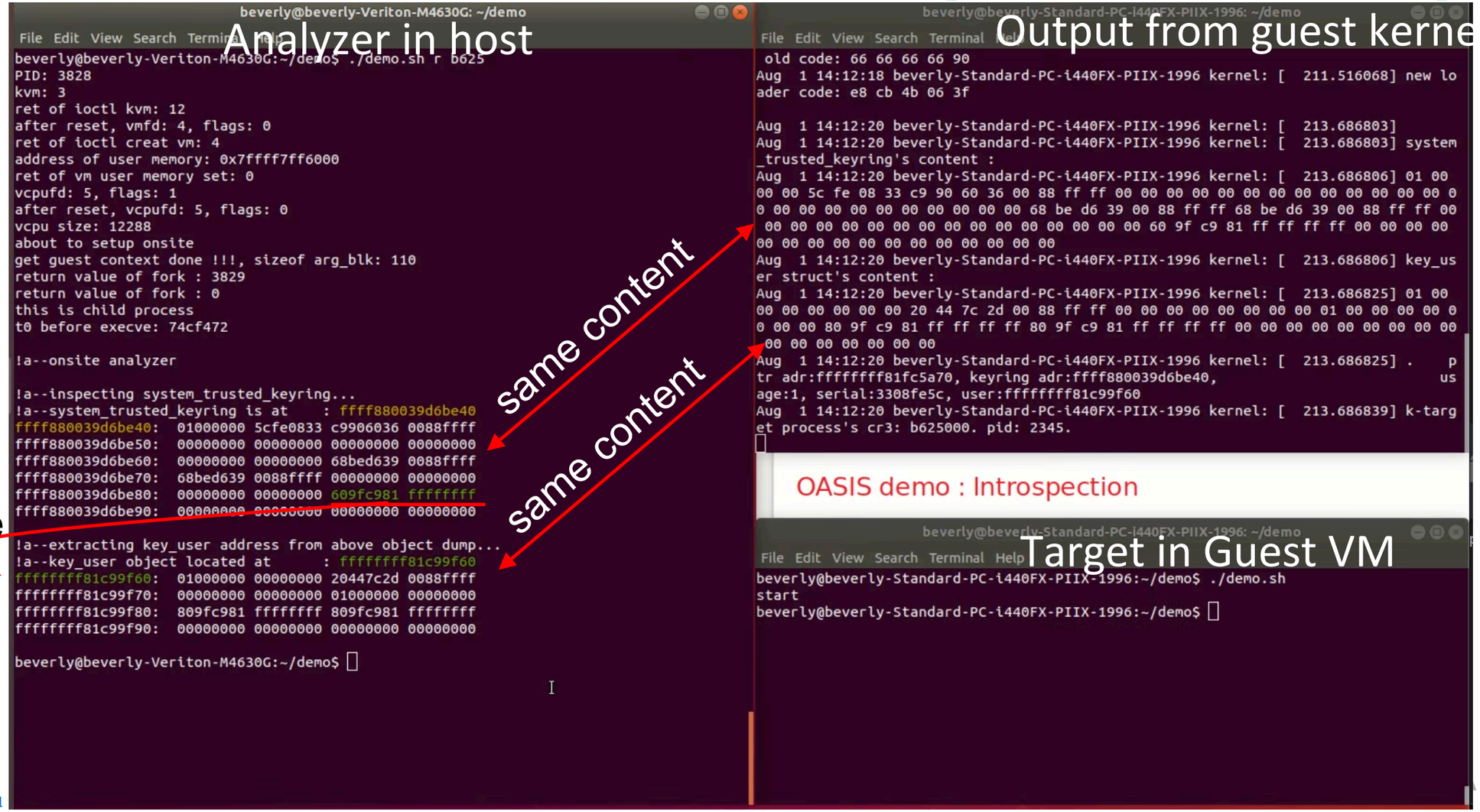
- **To track the control flow of the target from the capturing point**

```
void main ()
{
  //initialization
  ....

  OASIS_set_JMP(bb_exit);
  OASIS_resume_targ(&CPU);
  return;
}
```

```
void jmp_handler(){

    ...  //analysis workload
    ...  //find next block to run

    //remove the current one
    OASIS_rm_JMP(bb_exit);
    //set the new prob
    OASIS_set_JMP(next_bb_exit);
    // resume target from the next block.
    OASIS_resume_targ(&CPU);
    return;
}
```



Demo 1: Introspection (Screenshot)



Demo 2: Breakpoint + tracing (screenshot)

Analyzer in host

```
beverly@beverly-Veriton-M4630G: ~/demo
File  Edit  View  Search  Terminal  Help
beverly@beverly-Veriton-M4630G:~/demo$ ./demo.sh r 3a08c
PID: 4588
kvm: 3
ret of ioctl kvm: 12
after reset, vmfd: 4, flags: 0
ret of ioctl creat vm: 4
address of user memory: 0x7ffff7ff6000
ret of vm user memory set: 0
vcpufd: 5, flags: 1
after reset, vcpufd: 5, flags: 0
vcpu size: 12288
about to setup onsite
get guest context done !!!, sizeof arg_blk: 110
return value of fork : 4589
return value of fork : 0
this is child process
t0 before execve: 5ab5cc40

!a--onsite analyzer
redirected page start from :0xffffefff7e3d000. ends : 0xffffefff7fcd000.
!a--target thread exported to onsite : initAddr: 400620

!a--installing breakpoint at sys_getpriority syscall handler
!a--setting int3 breakpoint at ffffffff8108f740

!a--at int3 handler, bp1 trigger count 1, rip ffffffff8108f741
!a--k-function parameters : arg0 2, arg1 0
!a--resuming target in onsite...

!a--at int3 handler, bp1 trigger count 2, rip ffffffff8108f741
!a--starting bb trace : first jump probe placed at: ffffffff8108f774
!a--resuming target in onsite...

!a--at jmp handler      bb endAdr : ffffffff8108f774, bb# 001
!a--at jmp handler      bb endAdr : ffffffff8108f780, bb# 002
!a--at jmp handler      bb endAdr : ffffffff817f6c57, bb# 003
!a--at jmp handler      bb endAdr : ffffffff817f6c59, bb# 004
!a--at jmp handler      bb endAdr : ffffffff8108f788, bb# 005
!a--restoring target to guest...
beverly@beverly-Veriton-M4630G:~/demo$
```

Output from guest kernel

```
File  Edit  View  Search  Terminal  Help
 old code: 66 66 66 66 90
Aug  1 14:17:49 beverly-Standard-PC-i440FX-PIIX-1996 kernel: [   61.413948] new lo
ader code: e8 cb 6b 1b 3f

Aug  1 14:17:51 beverly-Standard-PC-i440FX-PIIX-1996 kernel: [   62.916757]
Aug  1 14:17:51 beverly-Standard-PC-i440FX-PIIX-1996 kernel: [   62.916757] system
_trusted_keyring's content :
Aug  1 14:17:51 beverly-Standard-PC-i440FX-PIIX-1996 kernel: [   62.916760] 01 00
00 00 34 bf 2d 13 c9 90 60 36 00 88 ff ff c8 e3 dd 39 00 88 ff ff 08 bf d6 39 00 8
8 ff ff 00 00 00 00 00 00 00 00 68 be d6 39 00 88 ff ff 68 be d6 39 00 88 ff ff 00
 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 60 9f c9 81 ff ff ff ff 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00
Aug  1 14:17:51 beverly-Standard-PC-i440FX-PIIX-1996 kernel: [   62.916760] key_us
er struct's content :
Aug  1 14:17:51 beverly-Standard-PC-i440FX-PIIX-1996 kernel: [   62.916780] 01 00
00 00 00 00 00 80 dd 81 1b 00 88 ff ff 00 00 00 00 00 00 00 01 00 00 00 00 00 0
0 00 00 80 9f c9 81 ff ff ff ff 80 9f c9 81 ff ff ff ff 00 00 00 00 00 00 00 00 00
 00 00 00 00 00 00 00
Aug  1 14:17:51 beverly-Standard-PC-i440FX-PIIX-1996 kernel: [   62.916780] .    p
tr adr:ffffffff81fc5a70, keyring adr:ffff880039d6be40,                        us
age:1, serial:132dbf34, user:ffffffff81c99f60
Aug  1 14:17:51 beverly-Standard-PC-i440FX-PIIX-1996 kernel: [   62.916793] k-targ
et process's cr3: 3a08c000. pid: 2275.
```

1st triggering

2nd triggering

5 bbs traced

OASIS demo : Breakpoints and control flow tracing

Target in Guest VM

```
File  Edit  View  Search  Terminal  Help
beverly@beverly-Standard-PC-i440FX-PIIX-1996:~/demo$ ./demo.sh
!t--first syscall : arg0 2, arg1 0
!t--secnd syscall : arg0 0, arg1 0
p1 -11, p2 0
beverly@beverly-Standard-PC-i440FX-PIIX-1996:~/demo$
```

Discussions

Features:

- Thread-centric, “surgical” analysis,
- Not for large-scale code-centric analysis such as profiling
- Strong security and transparency

Potential Applications:

- Virtual machine introspection
- Kernel debugger
- Cross-space malware analysis
- Attack scene forensics and response

Future Work

More primitives

- data breakpoint, multi-core

Migration to ARM Platform

- Feasible.
- Caveat: ARM does not have vmfunc instruction. A user space program cannot issue hypercalls.

Black Hat Sound Bytes

1. With OASIS, one can **easily** develop and run a **user-space** onsite analyzer to dynamically and **natively read, write and control** a user/**kernel** thread in a VM.

- No modification of the kernel is needed. No instrumentation.
- Strong security and transparency.

2. Suitable applications for onsite analyzers:

- VMI, kernel debugging, cross-space malware analysis, live kernel forensics, incident response etc.

Reference

- Jiaqi Hong, Xuhua Ding, "*A Novel Dynamic Analysis Infrastructure to Instrument Untrusted Execution Flow Across User-Kernel Spaces*", IEEE Symposium on Security and Privacy, 2021

- OASIS resources: https://github.com/OnsiteAnalysis/OASIS

Thank You

xhding@smu.edu.sg