HyperPill: Fuzzing for Hypervisor-bugs by Leveraging the Hardware Virtualization Interface

bob 14 25.08.27



HyperPill: Fuzzing for Hypervisor-bugs by Leveraging the Hardware Virtualization Interface

Alexander Bulekov, *EPFL, Boston University, and Amazon;* Qiang Liu, *EPFL and Zhejiang University;* Manuel Egele, *Boston University;* Mathias Payer, *EPFL*

https://www.usenix.org/conference/usenixsecurity24/presentation/bulekov

This paper is included in the Proceedings of the 33rd USENIX Security Symposium.

August 14–16, 2024 • Philadelphia, PA, USA

978-1-939133-44-1

Ref :

Morphuzz 2022 Usenix

ViDeZZo 2023 IEEE

V-Shuttle 2021 ACM

Hyper-Cube 2020 NDSS

MundoFuzz 2022 Usenix

Hyper-Cube
2020 / NDSS

hyper call interface fuzzing
MMIO, virtio 장치 등 I/O 경로는 배제
vm의 스냅샷/리셋 오버헤드

V-Shuttle
2021 / ACM

DMA fuzzing
복잡한 MMIO, virtio 장치 등 I/O 경로는 배제
hyper call fuzzing X
vm의 스냅샷/리셋 오버헤드 여전히 해결 X

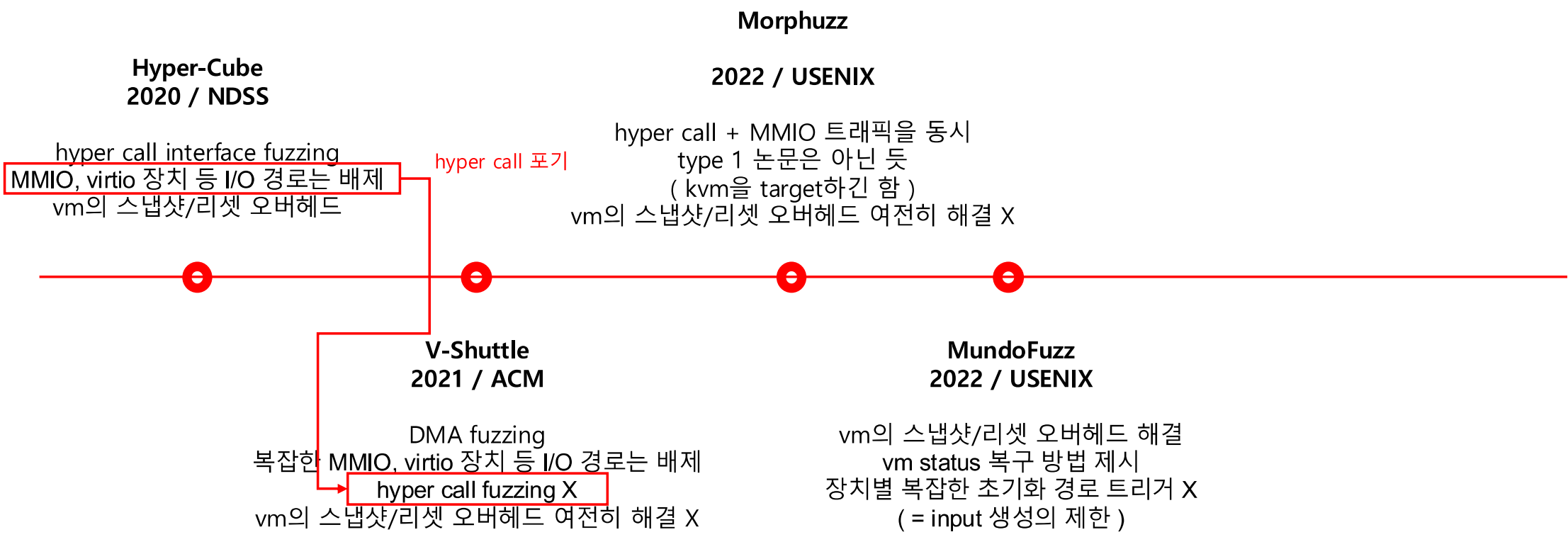
Morphuzz

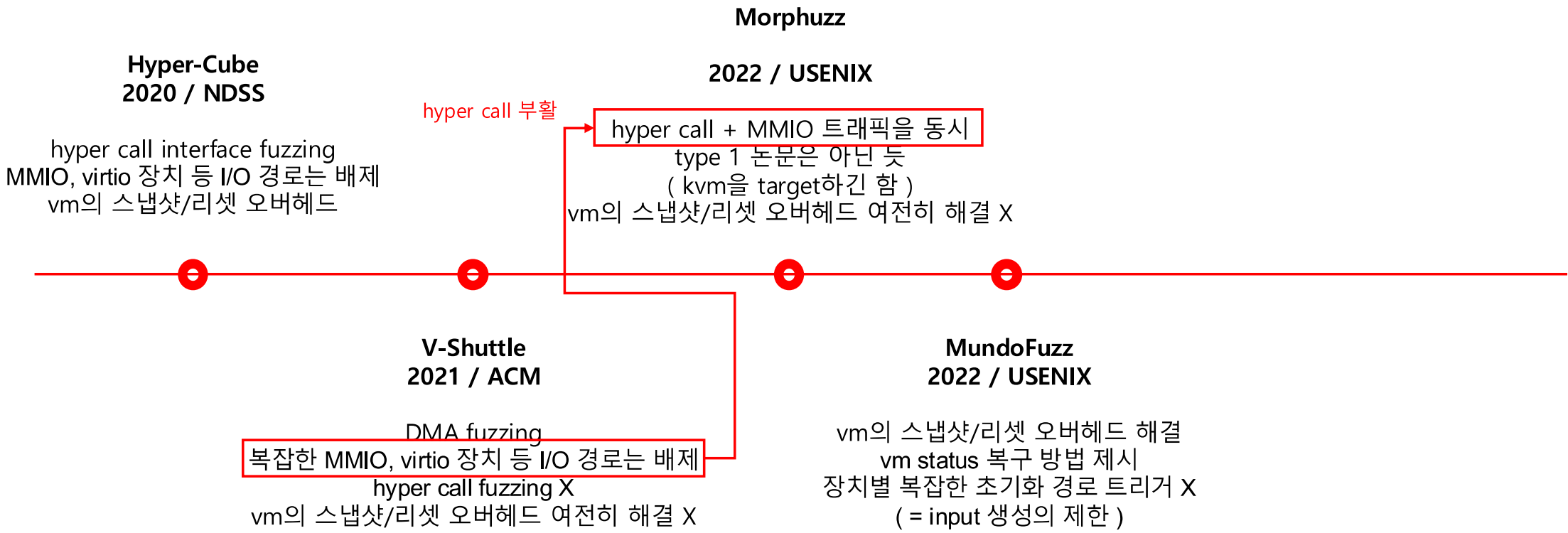
2022 / USENIX

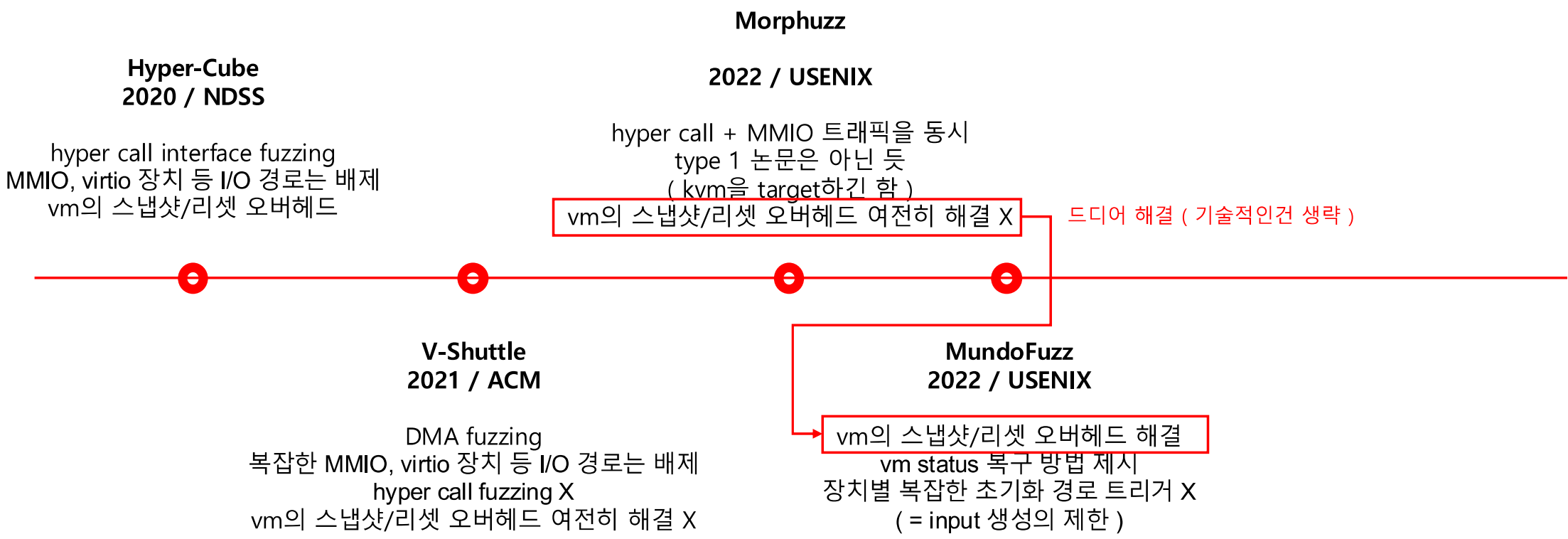
hyper call + MMIO 트래픽을 동시
type 1 논문은 아닌 듯
(kvm을 target하긴 함)
vm의 스냅샷/리셋 오버헤드 여전히 해결 X

MundoFuzz
2022 / USENIX

vm의 스냅샷/리셋 오버헤드 해결
vm status 복구 방법 제시
장치별 복잡한 초기화 경로 트리거 X
(= input 생성의 제한)


Hyper-Cube
2020 / NDSS
hyper call interface fuzzing
MMIO, virtio 장치 등 I/O 경로는 배제
vm의 스냅샷/리셋 오버헤드
hyper call 포기
V-Shuttle
2021 / ACM
DMA fuzzing
복잡한 MMIO, virtio 장치 등 I/O 경로는 배제
hyper call fuzzing X
vm의 스냅샷/리셋 오버헤드 여전히 해결 X
Morphuzz
2022 / USENIX
hyper call + MMIO 트래픽을 동시
type 1 논문은 아닌 듯
(kvm을 target하긴 함)
vm의 스냅샷/리셋 오버헤드 여전히 해결 X
MundoFuzz
2022 / USENIX
vm의 스냅샷/리셋 오버헤드 해결
vm status 복구 방법 제시
장치별 복잡한 초기화 경로 트리거 X
(= input 생성의 제한)


Hyper-Cube
2020 / NDSS
hyper call interface fuzzing
MMIO, virtio 장치 등 I/O 경로는 배제
vm의 스냅샷/리셋 오버헤드
hyper call 부활
Morphuzz
2022 / USENIX
hyper call + MMIO 트래픽을 동시
type 1 논문은 아닌 듯
(kvm을 target하긴 함)
vm의 스냅샷/리셋 오버헤드 여전히 해결 X
V-Shuttle
2021 / ACM
DMA fuzzing
복잡한 MMIO, virtio 장치 등 I/O 경로는 배제
hyper call fuzzing X
vm의 스냅샷/리셋 오버헤드 여전히 해결 X
MundoFuzz
2022 / USENIX
vm의 스냅샷/리셋 오버헤드 해결
vm status 복구 방법 제시
장치별 복잡한 초기화 경로 트리거 X
(= input 생성의 제한)


Hyper-Cube
2020 / NDSS
hyper call interface fuzzing
MMIO, virtio 장치 등 I/O 경로는 배제
vm의 스냅샷/리셋 오버헤드
Morphuzz
2022 / USENIX
hyper call + MMIO 트래픽을 동시
type 1 논문은 아닌 듯
(kvm을 target하긴 함)
vm의 스냅샷/리셋 오버헤드 여전히 해결 X
드디어 해결 (기술적인건 생략)
V-Shuttle
2021 / ACM
DMA fuzzing
복잡한 MMIO, virtio 장치 등 I/O 경로는 배제
hyper call fuzzing X
vm의 스냅샷/리셋 오버헤드 여전히 해결 X
MundoFuzz
2022 / USENIX
vm의 스냅샷/리셋 오버헤드 해결
vm status 복구 방법 제시
장치별 복잡한 초기화 경로 트리거 X
(= input 생성의 제한)

Hyper-Cube
2020 / NDSS

hyper call interface fuzzing
MMIO, virtio 장치 등 I/O 경로는 배제
vm의 스냅샷/리셋 오버헤드

V-Shuttle
2021 / ACM

DMA fuzzing
복잡한 MMIO, virtio 장치 등 I/O 경로는 배제
hyper call fuzzing X
vm의 스냅샷/리셋 오버헤드 여전히 해결 X

Morphuzz

2022 / USENIX

hyper call + MMIO 트래픽을 동시
type 1 논문은 아닌 듯
(kvm을 target하긴 함)
vm의 스냅샷/리셋 오버헤드 여전히 해결 X

MundoFuzz
2022 / USENIX

vm의 스냅샷/리셋 오버헤드 해결
vm status 복구 방법 제시
장치별 복잡한 초기화 경로 트리거 X
(= input 생성의 제한)

ViDeZZo
2023 / IEEE

Dependency-aware fuzzing
virtio 등 device message 내 필드 간 종속성 모델링

의미있는 시퀀스를 생성하여,
보다 유의미한 입력 생성

일부 장치(virtio-gpu, virtio-scsi 등)는 여전히 annotation 부족

특정 옵션의 경로는 cov 도달 불가능

HyperPill
2024 / USENIX

종류에 관계없이

소스코드 없이

하이퍼바이저의 시스템 전체 스냅샷

PIO/MMIO/Hypercalls/DMA 등 주요 공격 표면을 자동으로 탐색 가능

ViDeZZo
2023 / IEEE

Dependency-aware fuzzing
virtio 등 device message 내 필드 간 종속성 모델링

의미있는 시퀀스를 생성하여,
보다 유의미한 입력 생성

일부 장치(virtio-gpu, virtio-scsi 등)는 여전히 annotation 부족

특정 옵션의 경로는 cov 도달 불가능

HyperPill
2024 / USENIX

하이퍼바이저 종류에 관계없이

소스코드 없이

하이퍼바이저의 시스템 전체 스냅샷

PIO/MMIO/Hypercalls/DMA 등 주요 공격 표면을 자동으로 탐색 가능

Challenges

하이퍼바이저별 구현 다양성 및 클로즈드 소스로 인한 기존 퍼저를 사용하기 어렵다!

통일된 하이퍼콜 API가 없다!! 다 다름;

stateful하여 입력 간 정확한 상태 리셋이 필요함

선행연구는 대체로 소스 수정/훅킹에 의존.

DMA는 API 훅킹만으로는 누락되는 경우가 있다!

Insights

모든 하이퍼바이저는 동일한 하드웨어 가상화 인터페이스(VT-x / amd-v)에 의존

VM-Exit 직전 시점을 정확히 스냅샷해 trap-and-emulate 경로를 바로 타게 함

VMCS/게스트 메모리 조작으로 임의 IO 주입(PIO/MMIO/Hypercall)

DMA는 물리 메모리 단위에서 후킹하여 구조·API에 무관.

하이퍼콜은 비교 명령어 피드백으로 실사용 레지스터만 자동 추정.(???)

Contributions

최초의 **범용** 하이퍼바이저 퍼저

PIO/MMIO/Hypercall/DMA 전 범위를 자동 처리

소스/훅킹 없이 12개 QEMU 디바이스 중 10개에서 최고 커버리지. [비교군이 뭐지..?]

신규 버그 26개(QEMU 11, Hyper-V 9, macOS VF 6) 보고. [ㅈㄴ 꼬롬함… 조사 필요]

설계와 구현 개요 및 오픈소스 공개. [응 트랜드]

Method

그냥 해석이라 질문해도 모름

HYPERPILL-Snap:

L0(커스텀 KVM)에서 L1 타깃 하이퍼바이저와 L2 Fuzz-VM을 **중첩 가상화**로 구동·감시, 특수 하이퍼콜(0xdeadbeef)로 스냅샷 트리거.

HYPERPILL-Inspect:

Bochs 기반 MELTER로 스냅샷을 로드해 PIO/MMIO 활성 범위 동적 프로빙, EPT·페이지 테이블로 게스트 프레임 분류(DMA/IE).

HYPERPILL-Fuzz:

MELTER의 퍼징 모드에서 입력 해석·IO 주입·커버리지 수집, DMA atomic 후킹과 하이퍼콜 레지스터 자동 추정으로 복합 입력 생성.

Method

그냥 해석이라 질문해도 모름

HYPERPILL-Snap:

L0(커스텀 KVM)에서 L1 타깃 하이퍼바이저와 L2 Fuzz-VM을 **중첩 가상화**로 구동·감시, 특수 하이퍼콜(0xdeadbeef)로 스냅샷 트리거.

HYPERPILL-Inspect:

Bochs 기반 MELTER로 스냅샷을 로드해 PIO/MMIO 활성 범위 동적 프로빙, EPT·페이지 테이블로 게스트 프레임 분류(DMA/IE).

?!

HYPERPILL-Fuzz:

MELTER의 퍼징 모드에서 입력 해석·IO 주입·커버리지 수집, DMA atomic 후킹과 하이퍼콜 레지스터 자동 추정으로 복합 입력 생성.

잠만

MELTER는 HYPERPILL가 스냅샷한 하이퍼바이저를 **에뮬레이터(Bochs)** 안에 적재해 분석·퍼징을 수행하는 실행 환경

지들이 지은 이름임!

Bochs -> ㄹㅇ 있는 x86 애뮬레이터 (box 라고 읽음!)

시작 시 1단계에서 수집한 **메모리·레지스터 스냅샷**을 로드

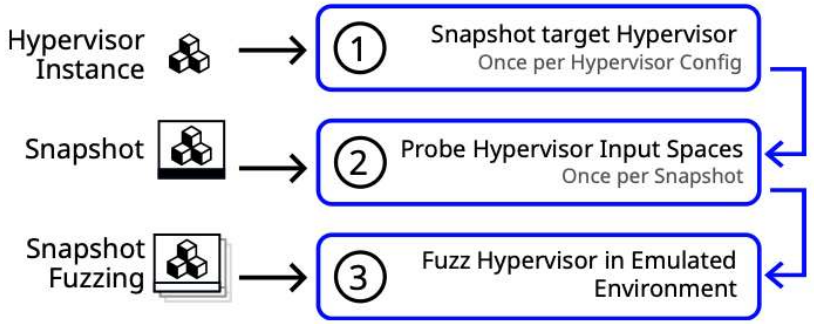
두 가지 모드(열거·퍼징)를 제공합니다.

HYPERPILL-Inspect는 열거 모드(**enumeration**)

HYPERPILL-Fuzz는 퍼징 모드(**fuzzing**)

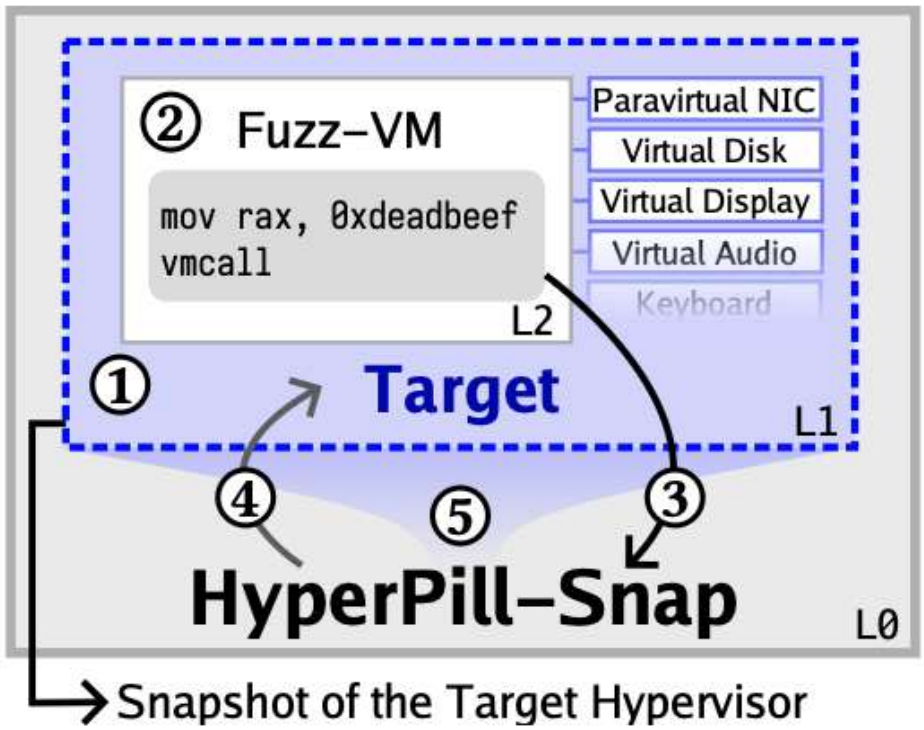
Bochs의 VT-x 지원과 계측을 활용해 하이퍼바이저 코드의 cov 를 수집

[overview]



전반적인 구조는 무난

HYPERPILL-Snap



*VMCS란? …　　어려움…

https://shhoya.github.io/hv_vmcsdata.html

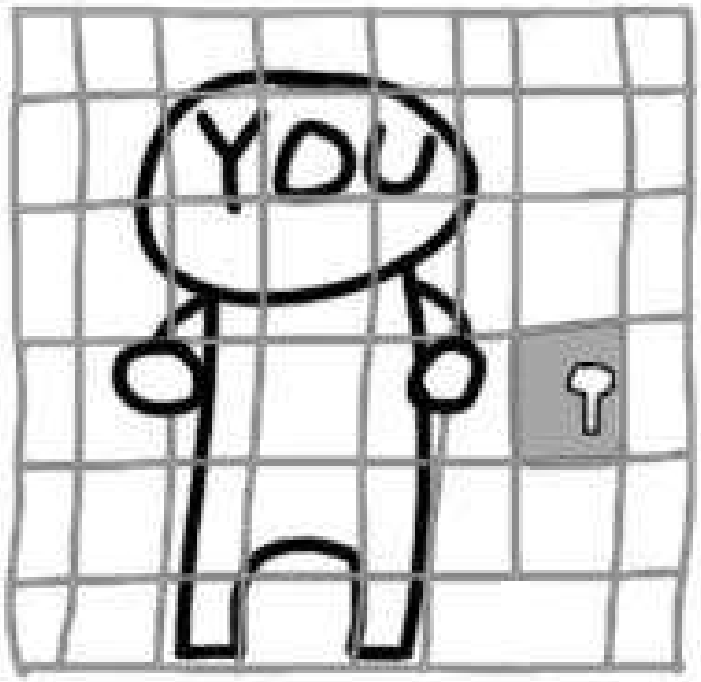
HYPERPILL-Snap

Snap은 L0 하이퍼바이저로 동작
타깃 하이퍼바이저를 L1 게스트로 실행
그 안에서 L2 Fuzz-VM을 구동하는 중첩 가상화 구성을 사용

x86 아키텍처 특성상 모든 L2 VM-Exit은 L0을 먼저 통과하므로, Snap은 L2에서 발생하는 이벤트(Exit Reason)를 미리 관찰할 수 있다.

스냅샷 트리거 : L2에서 특수 하이퍼콜 (RAX=0xdeadbeef)와 함께 vmcall을 실행
L0가 이를 감지해 *VMCS(하드웨어 VM 제어 구조)를 조정하고, Trap-and-Emulate 시작 직전의 L1 상태(타깃 하이퍼바이저)로 풀 시스템 스냅샷을 채집
이후 이 스냅샷이 MELTER로 이식됩니다.
-> 하이퍼바이저 구현(타입-1/2, 커널/유저 등)과 무관하게 정확한 재현 가능한 시작점을 고정하여, 이후 열거·퍼징 단계에서 상태 리셋 문제를 제거.
***열거 <- enumeration 로 쓰여있고, TTP recon 감성임

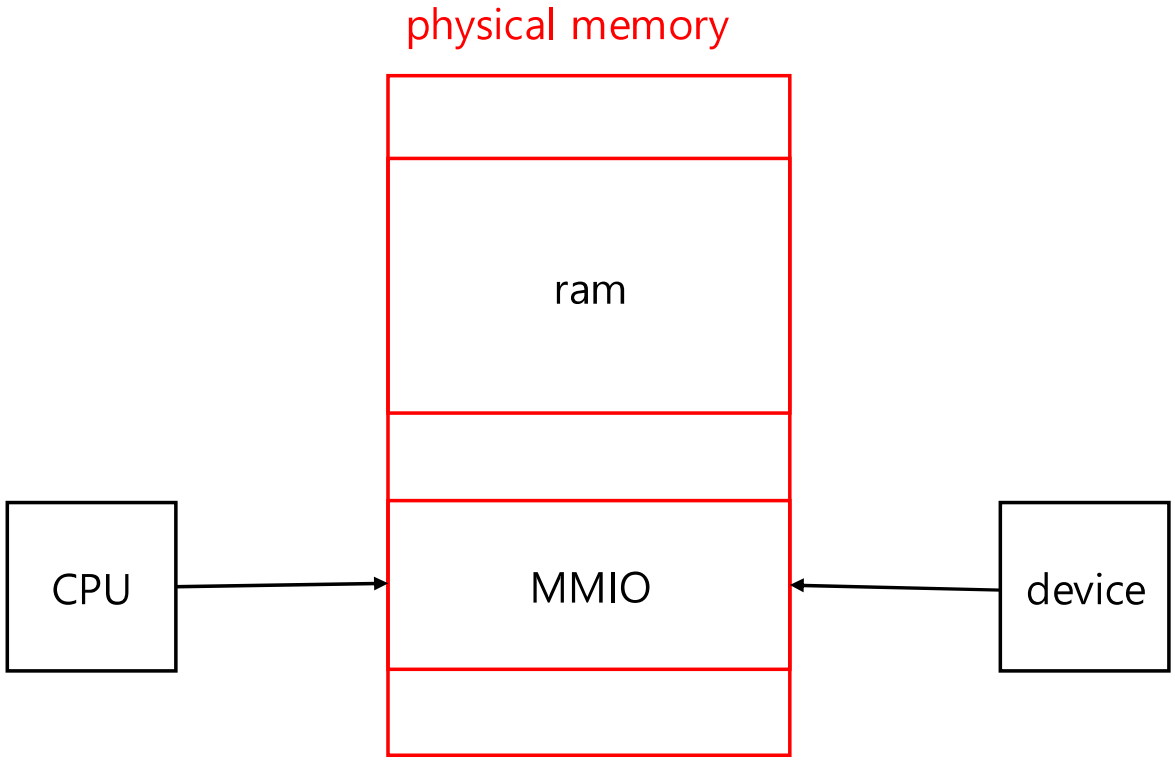
HYPERPILL-Inspect





논문 외 개념

MMIO



논문 외 개념

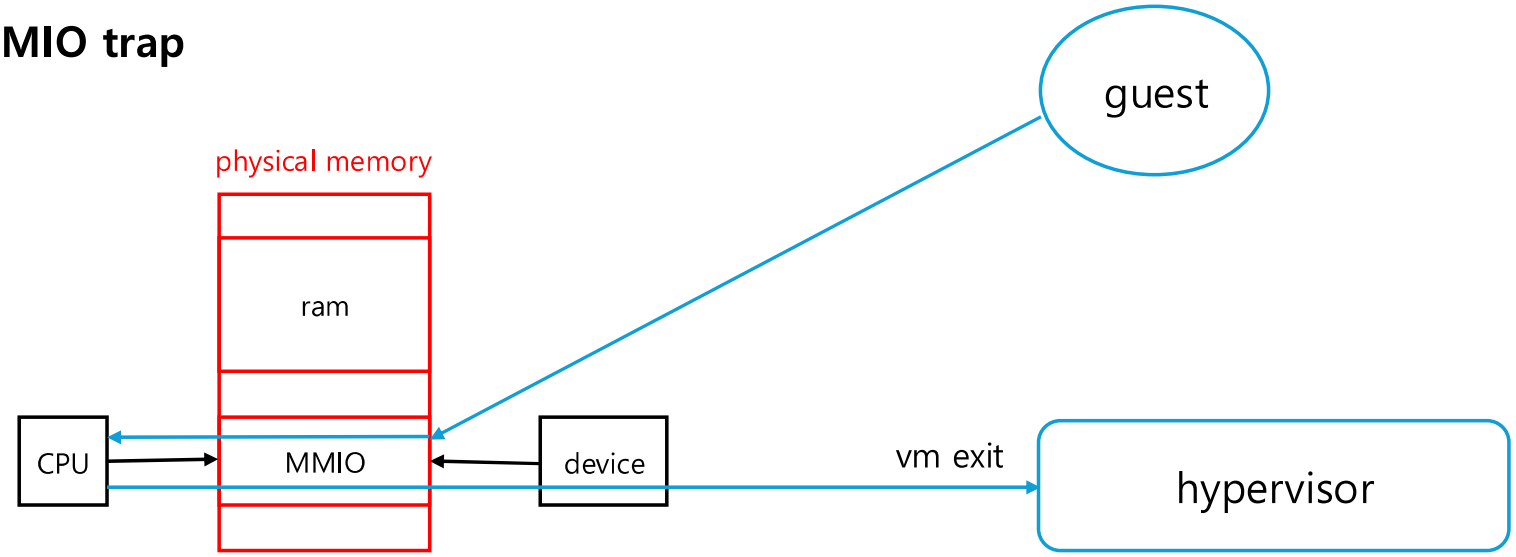
MMIO



guest 도 mov instruction 등으로 mmio 접근 가능

논문 외 개념

MMIO trap



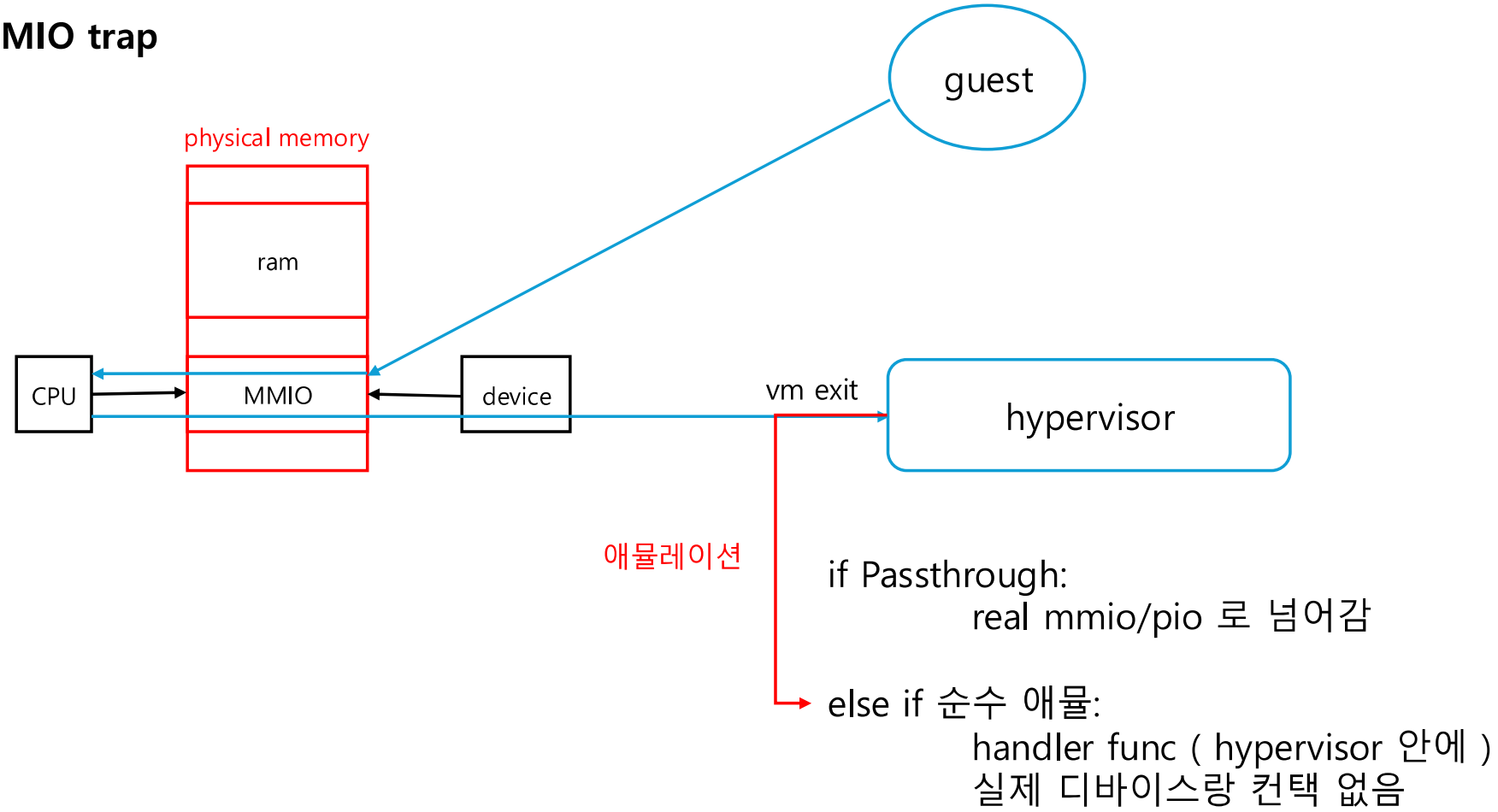
게스트가 MMIO 영역을 접근하면, 하드웨어(VT-x, AMD-V)가 **일반 메모리 접근이 아니라 장치 접근**으로 인식해, **VM-Exit 이벤트를 발생**.

→ CPU에서 게스트가 MMIO를 접근했다고 보면, 하이퍼바이저로 제어를 넘긴다.

→ 하이퍼바이저는 그 주소가 어떤 장치에 속하는지 확인하고, 해당 장치 에뮬레이션 코드를 실행

논문 외 개념

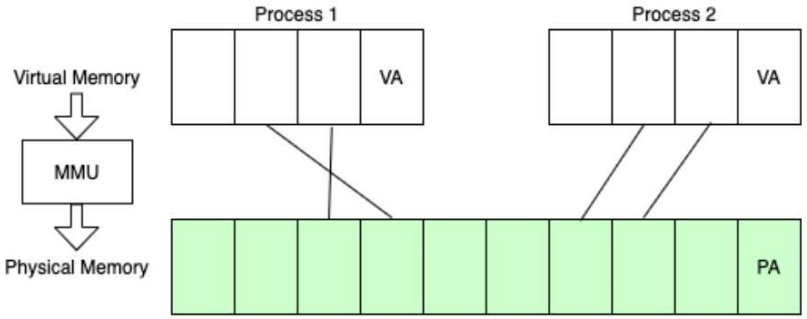
MMIO trap

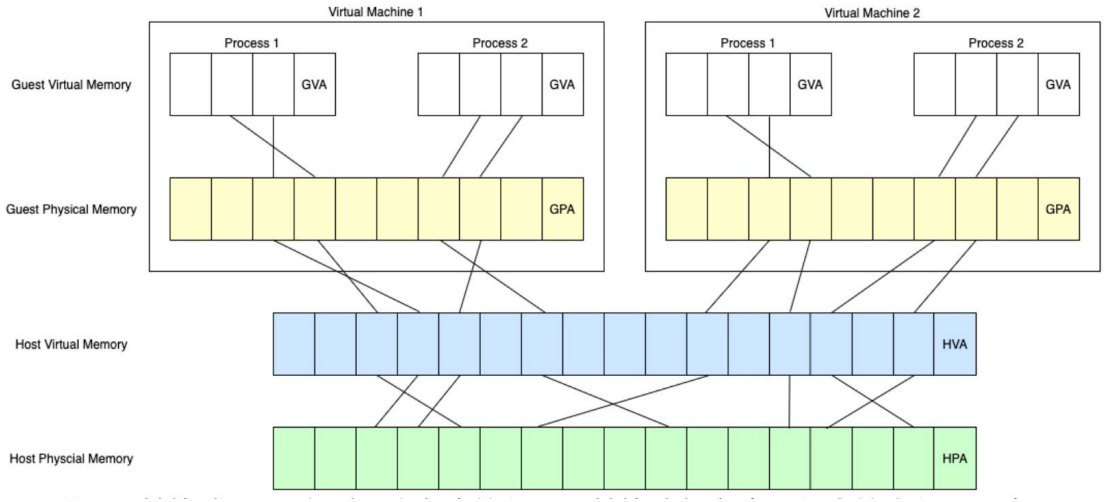


논문 외 개념

ref : https://www.alibabacloud.com/blog/599058

GPA / HPA / NPT / EPT





레거시 구조
-> 프로세스간 메모리 영역 침범의 위험이 있음

and new!
-> GVA : **g**uest 의 **v**irtual memory
-> GPA : **g**uest 의 **p**hysical memory
-> HVA : **h**ost 의 **v**irtual memory
-> HPA : **h**ost 의 **p**hysical memory

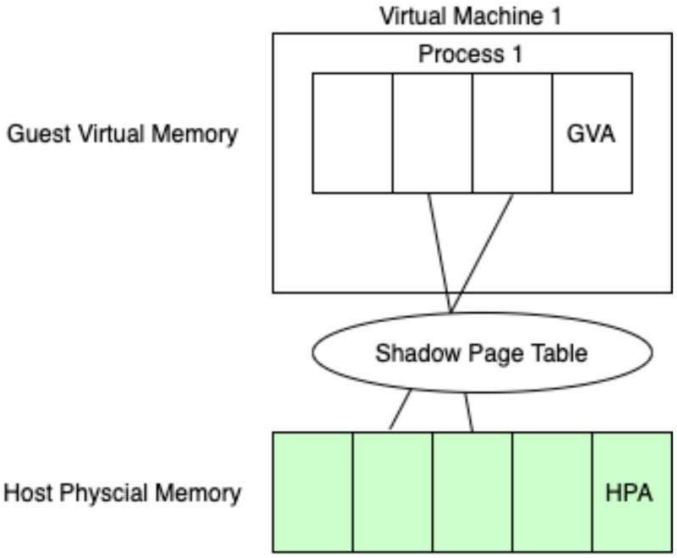
*a 가 address 였던 것 같음

논문 외 개념

GPA / HPA / NPT / EPT

ㅇㅇ 그래서 어케 구현함?

Shadow Page Table (SPT) and Extended Page Table (EPT) are two typical solutions

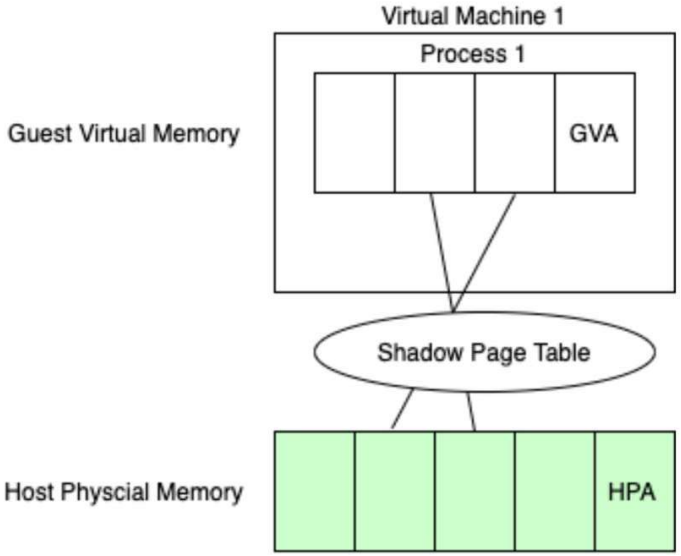


이건 SPT diagram

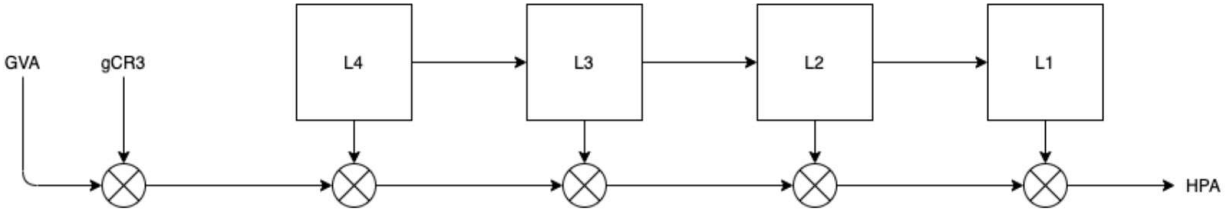
궁극적으론 GVA가 HPA로 맵핑 되어야 함

논문 외 개념

GPA / HPA / NPT / EPT



SPT는 GVA to HPA 가 한번에 맵핑되어 있음



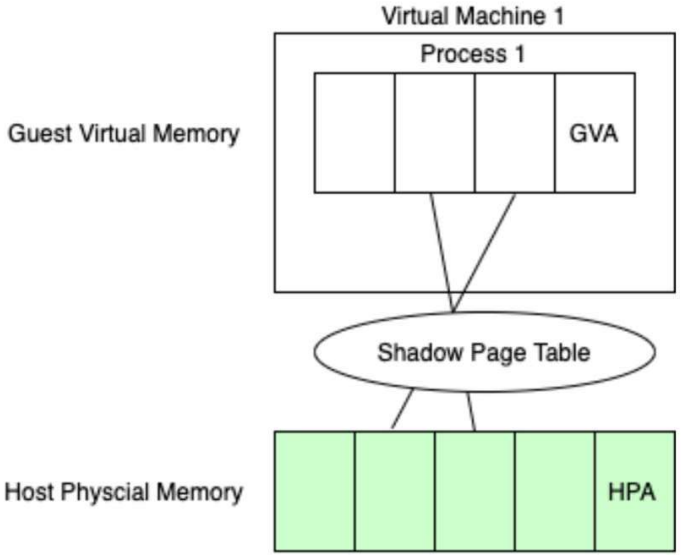
게스트 페이지 테이블을 그대로 쓰지 않고, 하이퍼바이저가 별도의 테이블을 만들어 관리해줌

프로세스가 메모리를 참조할 때, 하드웨어 MMU가 SPT만 참조하면 되기 때문에, 네이티브 급으로 빠름

host랑 메모리 접근 경로를 비교해도 오버헤드가 거의 없음
유사한 경로임

논문 외 개념

GPA / HPA / NPT / EPT



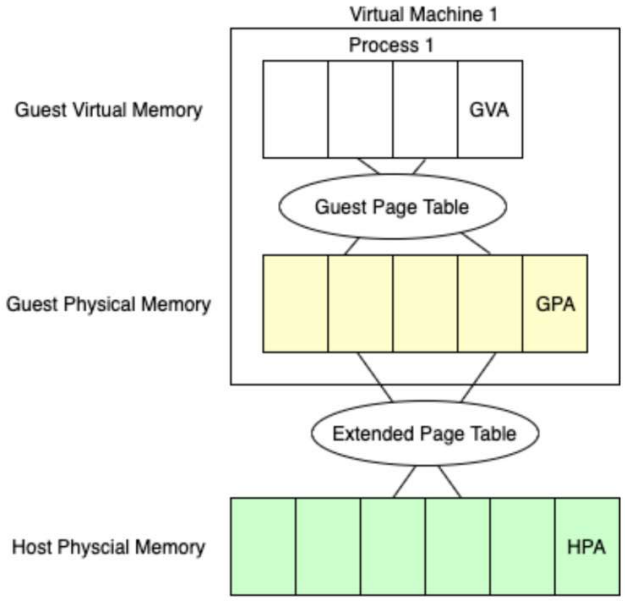
BUT!!!

하이퍼바이저가 이를 관리하니, guest가 페이지 테이블 수정하면 매 수정마다 trap이 발생하고 하이퍼바이저가 이를 핸들링 해줘야 한다.

guest 의 process 1개당 1개의 SPT를 가진다.
따라서 자원 낭비가 심해진다

논문 외 개념

GPA / HPA / NPT / EPT



EPT :

Later, hardware added support for nested page tables for virtualization, allowing the hardware to complete two-layer page table translation automatically.

2중 page table 을 하드웨어가 지원함
(nested page table 이라고 하는 듯)

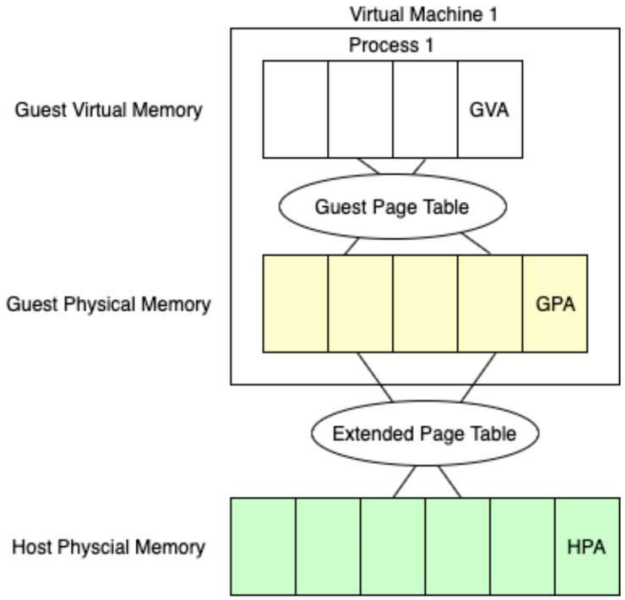
이걸 가지고 만든 기술이 EPT

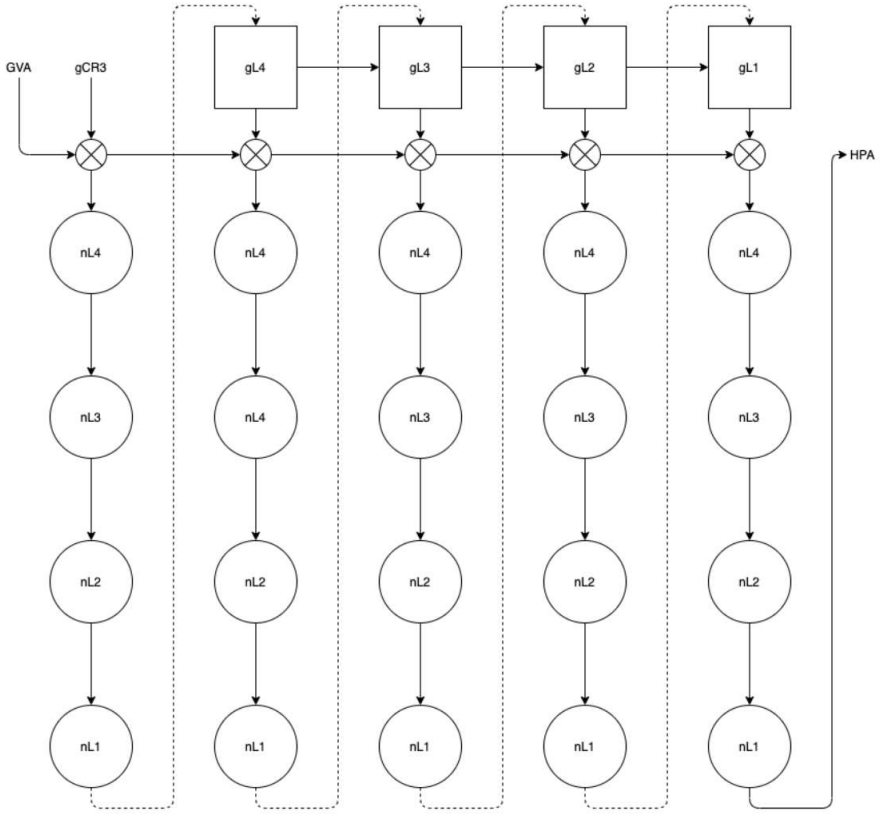
GVA to GPA 를 가상머신의 page table이 함
GPA to HPA 를 EPT가 함

암튼 하드웨어 기반 기술임 (CPU MMU)

논문 외 개념

GPA / HPA / NPT / EPT

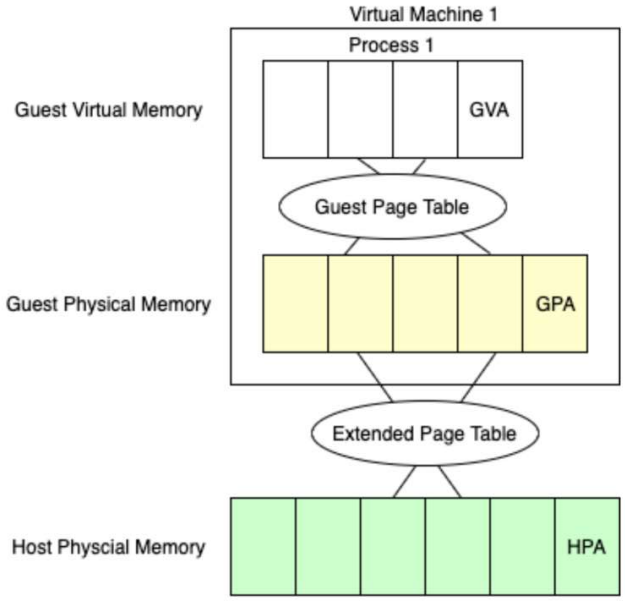




문제는 GPA가 어느 HPA 에 맵핑되는 지 찾아가려면 EPT를 참조해야하는데, 구조 상 이중으로 lookup을 해야해서 worst case에는 Path가 너무 길어진다는 단점이 있다
(고는 하는데, 이해가 잘 안되었음;;)

논문 외 개념

GPA / HPA / NPT / EPT



그래도 page table 수정마다 trap이 발생하지도 않고
h/w 지원으로 돌아가니까 overhead 도 엄청 낮아짐

논문 외 개념

GPA / HPA / NPT / EPT

이런 개념을 바탕으로 다양한 기법들이 등장하지만, 대부분이 앞서 발표된 방법론을 기반으로 함

One-Layer Page Table Solution

SPT(Shadow Page Table)와 유사한 방식.
하나의 페이지 테이블만 사용하여 GVA(게스트 가상 주소) → HPA(호스트 물리 주소) 매핑을 직접 관리.
하이퍼바이저가 할 일이 너무 많아서 비용이 큼(오버헤드 문제 ㅇㅇ)

논문 외 개념

GPA / HPA / NPT / EPT

이런 개념을 바탕으로 다양한 기법들이 등장하지만, 대부분이 앞서 발표된 방법론을 기반으로 함

Two-Layer Page Table Solution

EPT/NPT와 같은 구조.
두 개의 독립적인 테이블을 사용
게스트 페이지 테이블: GVA → GPA
EPT/NPT: GPA → HPA
하이퍼바이저 개입이 필요 없음.
단점은 주소 변환 경로가 길어지고 page walk 비용이 크다

논문 외 개념

GPA / HPA / NPT / EPT

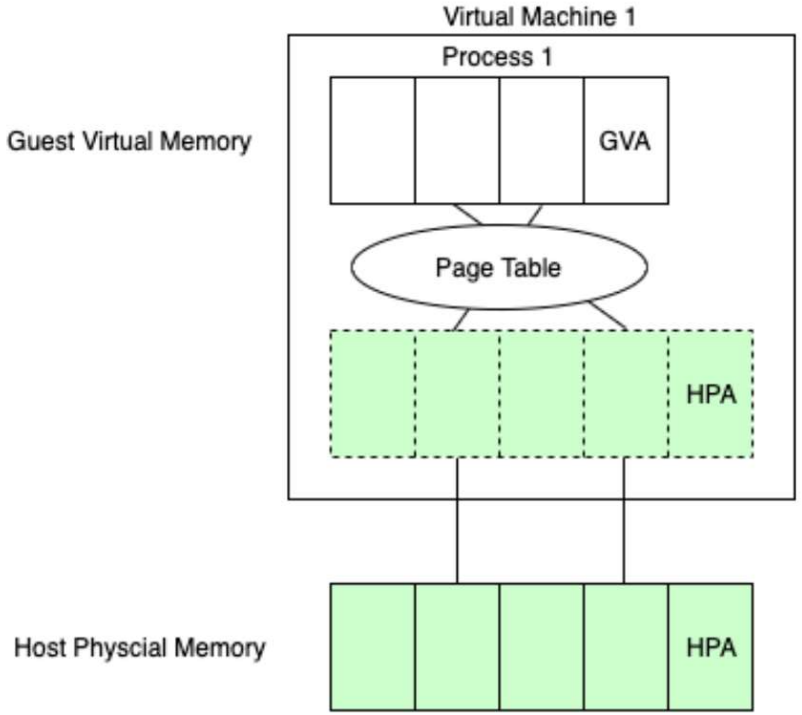
이런 개념을 바탕으로 다양한 기법들이 등장하지만, 대부분이 앞서 발표된 방법론을 기반으로 함

Hybrid Solution

예... 안나오면 이상하죠

논문 외 개념

GPA / HPA / NPT / EPT



Direct Paging

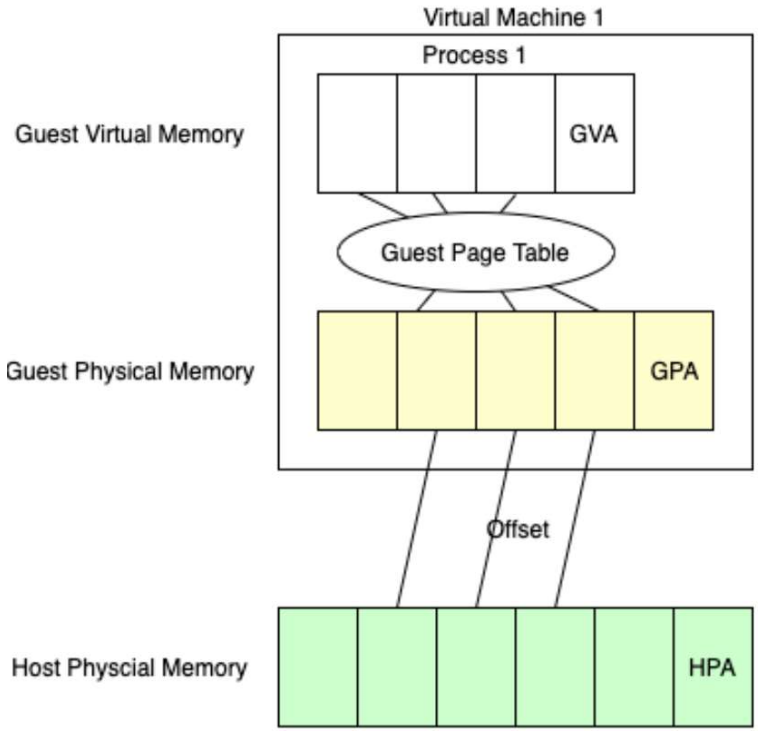
Xen 이 paravirtualization 에서 사용한 방법

one-layer page table

근데 이것도 이제 개소리인게,
EPT 등장으로 Xen도 cpu 의존적임
굳이 이거 안쓴다고 함

논문 외 개념

GPA / HPA / NPT / EPT



Direct Segment

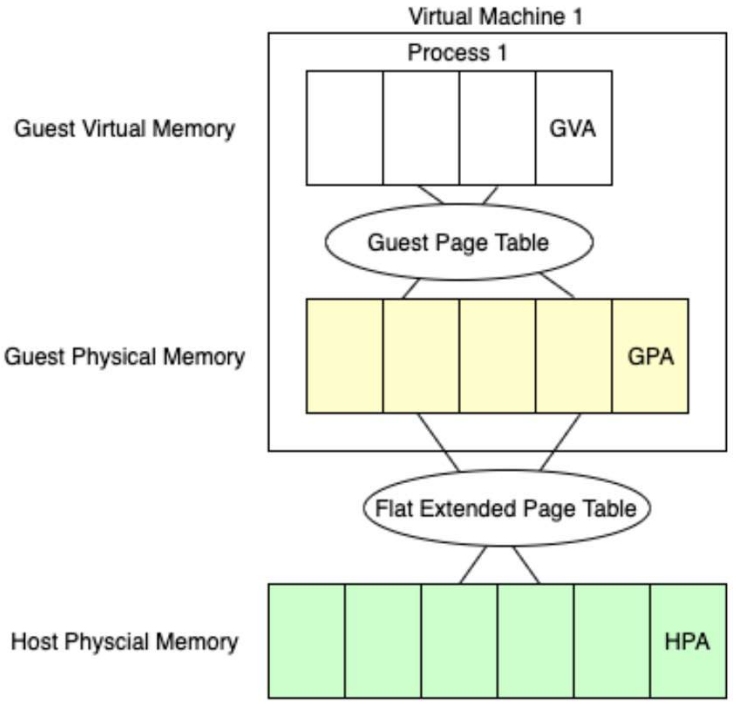
GVA → GPA 매핑은 EPT처럼 게스트 페이지 테이블을 그대로 사용.
그러나 GPA → HPA 매핑은 페이지 테이블 대신 세그먼트 기반 변환을 사용.

말이 좋아서 segment지 그냥 offset 더해서 매번 테이블 조회하지 않도록 만들었음

뭐 다 장점이고 단점은 딱히 없음
그러나!!!!!!!! ㅅㅂ 현재 상용 CPU에서 offset 연산으로 테이블 접근을 구현한 제품이 없다고 함 (= 이쁜 쓰레기)

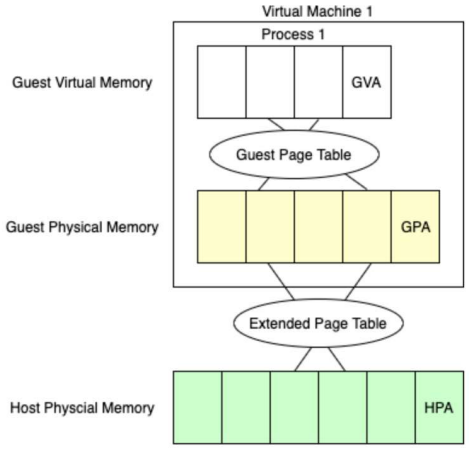
논문 외 개념

GPA / HPA / NPT / EPT



Flat EPT

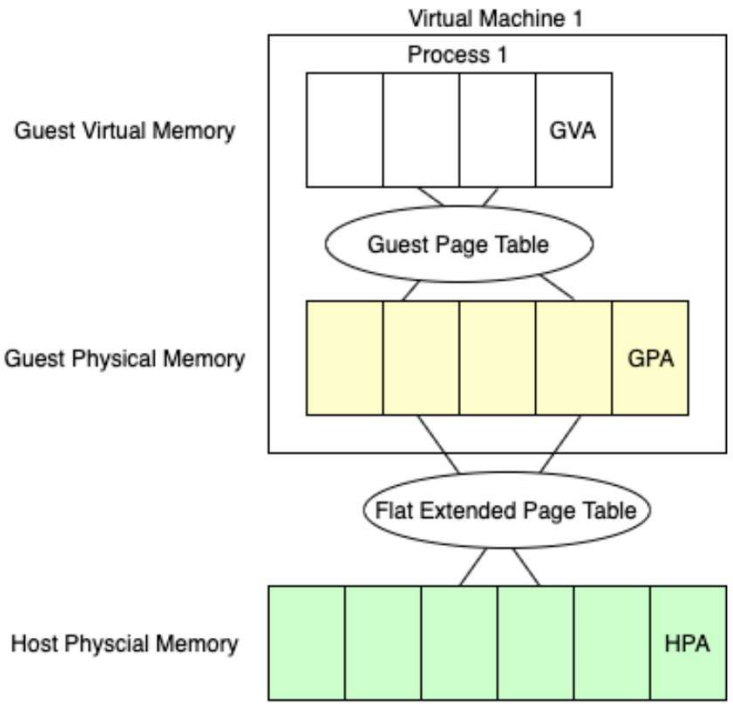
학계에서 제시된 새로운 하드웨어기반 EPT



EPT 랑 뭐가 다른건가 싶었는데

논문 외 개념

GPA / HPA / NPT / EPT



Flat EPT

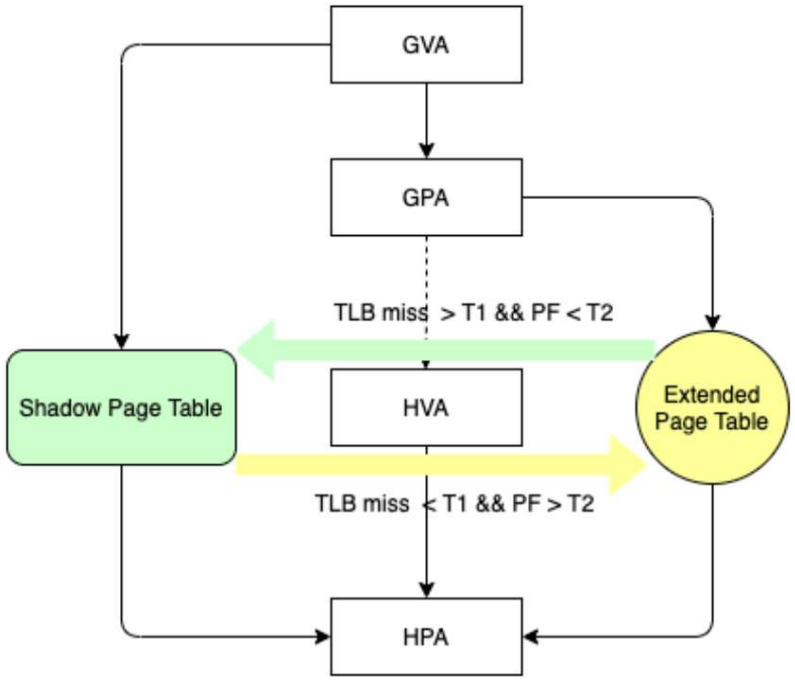
기존 EPT는
다단계 페이지 테이블(보통 4단계)를 씀

이걸 그냥 1개 테이블로 줄이고 사이즈 4배로 늘려서

worst case 에 path 불어나는걸 막겠다! 이거임

논문 외 개념

GPA / HPA / NPT / EPT



Mix SPT and EPT

이게 하이브리드

SPT : 변환 경로가 짧아 성능이 좋지만, 페이지 테이블 변경 시 VMM 개입 비용이 크다.

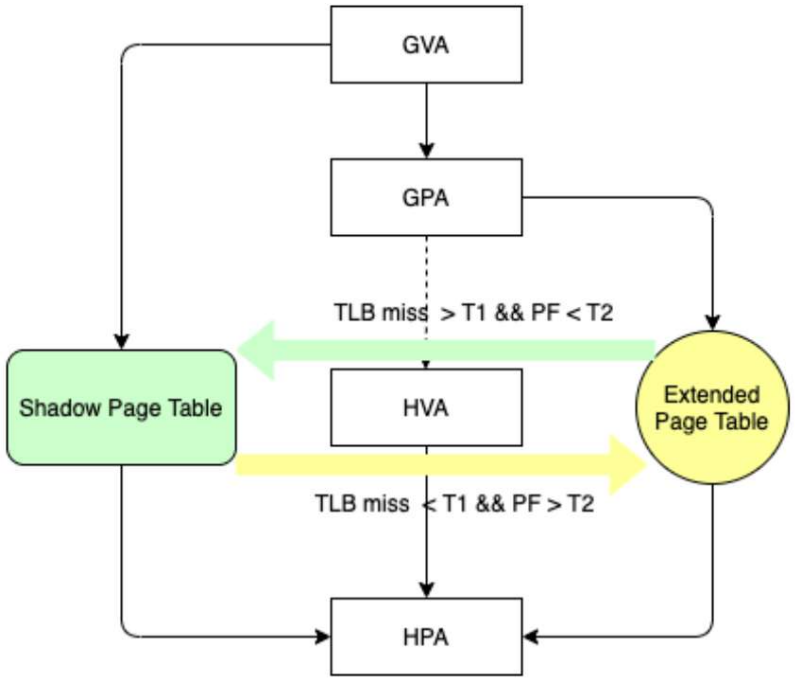
EPT : 게스트가 자유롭게 PT를 바꿀 수 있지만, 변환 경로가 길어 TLB miss 발생 시 성능이 나빠진다.

아 그러면 SPT <-> EPT를 동적으로 스위칭하자!!!

VM 실행 중에 **TLB miss율과 Page Fault 빈도**를 모니터링 해서, 상황에 따라 SPT/EPT를 선택

논문 외 개념

GPA / HPA / NPT / EPT



Mix SPT and EPT

EPT → SPT 전환
TLB miss rate > 임계값 T1
Page Fault 빈도 < 임계값 T2

SPT → EPT 전환
TLB miss rate < 임계값 T1
Page Fault 빈도 > 임계값 T2

<전제>

TLB miss가 많은 상황에서는 SPT가 더 효율적

Page Fault가 잦은 상황에서는 EPT가 더 효율적

논문 외 개념

GPA / HPA / NPT / EPT



NPT

NPT 안나와서 정독했는데

intel vt-x 는 EPT

AMD-v 는 NPT

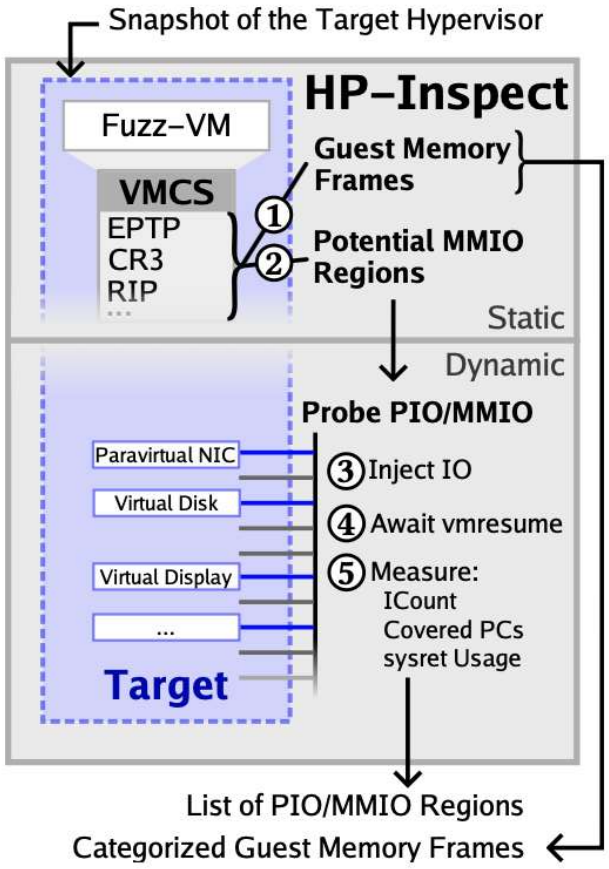
똑같다

HYPERPILL-Inspect

스냅샷을 Bochs 기반 실행환경(MELTER)에 복원한 뒤,

PIO/MMIO의 입력 범위와 게스트 메모리 중 DMA 표면을 자동 식별(=enumeration)하여,

이후 퍼징 단계가 의미 있는 서브공간만 때리도록 후보 집합을 만든다.


Snapshot of the Target Hypervisor
HP-Inspect
Fuzz-VM
VMCS
EPTP
CR3
RIP
...
Guest Memory Frames
Potential MMIO Regions
Static
Dynamic
Probe PIO/MMIO
Paravirtual NIC
Virtual Disk
Virtual Display
...
Target
3 Inject IO
4 Await vmresume
5 Measure:
ICount
Covered PCs
sysret Usage
List of PIO/MMIO Regions
Categorized Guest Memory Frames

*VMCS란? …　　어려움…

https://shhoya.github.io/hv_vmcsdata.html

아까도 나왔던건데, 그냥 ㅈㄴ 복잡한 구조체임
Virtual Machine Control Structure

HYPERPILL-Inspect

[1단계:정적] – GPT 직역

EPT 워크로 MMIO 후보 추출

EPT 포인터를 *VMCS에서 읽고 EPT를 워크하여, MMIO 트랩을 유발하도록 구성된 GPA 범위(Violation/Misconfig)를 후보로 담는다. 단, PCI hole 등 RAM/디바이스와 무관한 간극도 포함될 수 있으므로 이후 동적 필터링이 필요하다.

게스트 메모리 표면 분류 준비(DMA/IE)

EPT 워크로 게스트에 매핑된 HPA를 수집하고, 추가로 게스트 페이지 테이블을 워크해 마지막 실행 게스트 명령이 있는 페이지(IE)와 PT 구조를 식별한다. 이로써 "공격자 제어 데이터가 담길 수 있는 DMA 영역"과 "IE(명령 디스어셈블용) 영역"을 구분할 준비를 한다.

HYPERPILL-Inspect

[2단계:동적]

활성 PIO/MMIO 식별

스냅샷을 재개하기 전 **VMCS와 게스트 메모리를 수정**하여 원하는 VM-Exit(=PIO/MMIO 접근)을 인위적으로 발생시킨다(PIO는 outb, MMIO는 대상 페이지로의 mov). 하이퍼바이저가 해당 VM-Exit을 처리하고 **L2를 vmresume**하는 순간까지 실행시킨 뒤, 즉시 스냅샷으로 **복원**한다.

논문에선 이걸 probing 이라 표현함

PIO: 16-비트 **전 포트 공간**을 순회하며 write 프로브를 주입한다.

MMIO: EPT 워크에서 모은 **후보 페이지들**에 write 프로브를 주입한다.

HYPERPILL-Inspect

[2단계:동적]

이런저런 인풋을 조지고 아래 3가지 아티펙트로 리엑션을 수집

Instruction Count:
찾고자하는 영역이 아닌 파트는 처리 명령 수가 작고 유사하지만, **장치 에뮬레이션**이 필요한 활성 PIO/MMIO는 명령 수가 크다. (뭔소린지 모르겠음 ㅋㅋ)

Unique PCs: 서로 다른 장치 범위 접근은 **새로운 프로그램 카운터** 집합을 만든다(장치별 에뮬 코드).

User-space 진입(sysret):
KVM/Hyper-V는 가상장치 코드를 **유저 프로세스**에서 실행하므로, **sysret 발생** 여부가 실제 장치 코드라는 근거다.

HYPERPILL-Inspect

[2단계:동적]

세 신호를 **병합하여** 최종 **활성 PIO/MMIO 리스트**를 산출
정적(EPT)으로 범위를 **좁히고**, 동적(exec·feedback)으로 **진짜 장치 주소만 남김**

API 훅킹 없이 **에뮬레이터 계측**만으로 가능

HYPERPILL-Fuzz

앞 단계에서의 결과를 필요로 하는 단계

(i) HYPERPILL-Snap이 만든 **하이퍼바이저 스냅샷**

(ii) HYPERPILL-Inspect가 산출한 **활성 PIO/MMIO 목록**과 **분류된 게스트 메모리 프레임(DMA/IE)**

이걸로 퍼징

MELTER의 fuzzing 모드에서 하이퍼바이저를 재실행하며, 입력 인터프리터가 퍼저 입력을 IO 연산 시퀀스로 해석해 VM-Exit 형태로 주입.

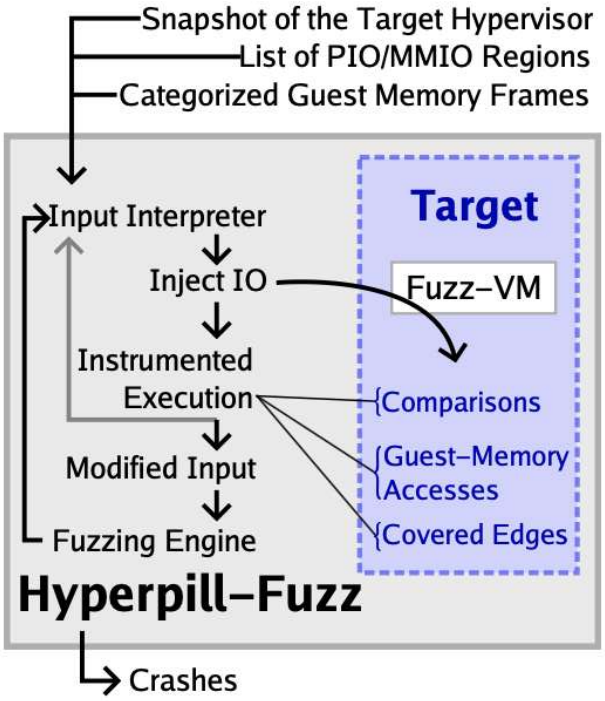
HYPERPILL-Fuzz

implementation 기여도 많고 아이디어도 많음

입력 포맷(연산 토큰) -> IO 연산 시퀀스로 분해. 5가지 핵심 연산을 지원
하이퍼콜(ABI 자동학습) -> 모든 레지스터에 난수 채워가면서 영향 받는 reg 찾는 듯
DMA(원자적 JIT 주입, API 비의존) ->

기존 공개 퍼저(Morphuzz/ViDeZZo 등)는 하이퍼바이저 소스코드에 DMA API 훅을 박아 해당 시점에 게스트 메모리를 JIT로 퍼저 데이터로 채움. 그러나 HYPERPILL은 소스코드 수정 없이 에뮬레이터에서 하이퍼바이저의 모든 메모리 접근을 추적하고, 접근 대상이 Inspect가 DMA로 라벨한 게스트 프레임이면 그 읽기 '직전' 해당 위치를 퍼저 데이터로 채워 넣음.

cov -> MELTER는 Bochs 기반이라 **명령 단위 계측**을 쉽게 붙일 수 있다고 함


Snapshot of the Target Hypervisor
List of PIO/MMIO Regions
Categorized Guest Memory Frames
Input Interpreter
Inject IO
Instrumented Execution
Modified Input
Fuzzing Engine
Target
Fuzz-VM
Comparisons
Guest-Memory Accesses
Covered Edges
Hyperpill-Fuzz
Crashes

Result

		Morphuzz		ViDeZZo		HYPERPILL		
		12 Cores 24 Hours						
Device	⊡	Branch Coverage (Executions/Second)						Bug
Block								
ahci	✓	42.43%	(25.68)	30.42%	(562.24)	**45.90%**	(26.18)	✓
nvme		29.12%	(23.82)			**36.44%**	(14.45)	✓
sdhci	✓	69.81%	(22.98)	**72.37%**	(107.22)	66.85%	(32.34)	
virtio-scsi	✓	27.96%	(23.83)	11.73%	(217.28)	**48.83%**	(51.68)	
Display								
cirrus		88.10%	(19.06)	83.42%	(138.78)	**88.67%**	(32.18)	✓
qxl	✓					**59.68%**	(26.96)	✓
virtio-gpu	✓	24.37%	(26.21)	2.77%	(222.42)	**45.52%**	(36.53)	✓
Networking								
e1000e	✓	50.27%	(24.83)	41.52%	(53.04)	**55.99%**	(42.22)	✓
igb	✓	29.73%	(25.63)			**35.93%**	(60.85)	✓
vmxnet	✓	50.75%	(27.01)	19.64%	(145.73)	**56.89%**	(48.14)	
USB								
ehci	✓	73.76%	(24.58)	**74.38%**	(177.08)	73.32%	(10.46)	
xhci	✓	55.54%	(28.83)	29.25%	(1061.36)	**76.64%**	(69.26)	✓
Geo. Mean		45.20%	(24.65)	28.00%	(203.07)	55.45%	(33.20)	

결과가 안좋았으면... 지금 읽고 있지도 않았겠죠

Result

		Morphuzz		ViDeZZo		HyperPill		
		12 Cores 24 Hours						
Device	⊡	Branch Coverage (Executions/Second)						Bug
Block								
ahci	✓	42.43%	(25.68)	30.42%	(562.24)	**45.90%**	(26.18)	✓
nvme		29.12%	(23.82)			**36.44%**	(14.45)	✓
sdhci	✓	69.81%	(22.98)	**72.37%**	(107.22)	66.85%	(32.34)	
virtio-scsi	✓	27.96%	(23.83)	11.73%	(217.28)	**48.83%**	(51.68)	
Display								
cirrus		88.10%	(19.06)	83.42%	(138.78)	**88.67%**	(32.18)	✓
qxl	✓					**59.68%**	(26.96)	✓
virtio-gpu	✓	24.37%	(26.21)	2.77%	(222.42)	**45.52%**	(36.53)	✓
Networking								
e1000e	✓	50.27%	(24.83)	41.52%	(53.04)	**55.99%**	(42.22)	✓
igb	✓	29.73%	(25.63)			**35.93%**	(60.85)	✓
vmxnet	✓	50.75%	(27.01)	19.64%	(145.73)	**56.89%**	(48.14)	
USB								
ehci	✓	73.76%	(24.58)	**74.38%**	(177.08)	73.32%	(10.46)	
xhci	✓	55.54%	(28.83)	29.25%	(1061.36)	**76.64%**	(69.26)	✓
Geo. Mean		45.20%	(24.65)	28.00%	(203.07)	55.45%	(33.20)	

virtio?
어택 벡터에 빠진거 아니냐?
라고 생각했는데

mmio, portio를 조지니까
virtio를 통한 surface 를 커버하는 듯

더 알아볼 필요가 있음

Result

우리가 봐야 할 파트 (= 우리가 논문쓰면 따라가야 함)

-> Evaluation

조사 대상 :
커버리지(브랜치/엣지 커버리지)-LLVM 기반 측정
실행 속도 (execs/sec)
발견된 취약점 수

타겟 :
QEMU: 12개 가상 디바이스 (virtio-scsi, virtio-gpu, virtio-net, virtio-block 등 포함)
Hyper-V, macOS VF: 네트워크/스토리지/그래픽 등 기본 장치

비교 대상 :
Morphuzz (USENIX Security 2022, QEMU 대상)
ViDeZZo (IEEE S&P 2023, 가상 디바이스 대상)

Result

CVE	Description	Morphuzz	ViDeZZo	HYPERPILL
CVE-2020-11869	ATI Integer-overflow	0:37	0:11	0:13
CVE-2020-25084	EHCI UAF	4:31	0:43	1:11
CVE-2020-25085	SDHCI Heap-Overflow	5:34	2:36	6:24
CVE-2020-25625	OHCI DoS	0:14	0:04	0:06
CVE-2021-20257	E1000 DoS	3:34	Timeout	1:29

Table 4: Mean times to find known bugs (h:mm)

비교는 1day

가치 입증을 0day

macOS: Out-of-Bounds in virtio-net. One of the 6 bugs HYPERPILL identified in macOS Virtualization-Framework is an out-of-bounds memory access in the virtio-net device. The crashing input performed three MMIO accesses to the device's MMIO regions. HYPERPILL detected accesses to four distinct DMA buffers (populating 47 bytes), to trigger the crash. To the best of our knowledge, there are no symbols publically available for macOS Virtualization Framework. As such, manually hooking macOS' DMA access APIs would require a considerable amount of work by a virtualization expert. However, HYPERPILL's use of feedback from the hardware-virtualization interface allows it to transparently fuzz complex attack-surfaces (e.g., DMA).