공학석사 학위논문

# 인텔 프로세서 트레이스를 이용한 실시간 다중 가상머신의 커널 제어 흐름 무결성 보장

2019년 2월

서울대학교 대학원

전기정보공학부

백 세 현

# 인텔 프로세서 트레이스를 이용한 실시간 다중 가상머신의 커널 제어 흐름 무결성 보장

지도 교수 백 윤 흥

이 논문을 공학석사 학위논문으로 제출함
2019년 2월

서울대학교 대학원
전기정보공학부
백 세 현

백세현의 공학석사 학위논문을 인준함
2019년 2월

위 원 장 김 장 우 (인)

부위원장 백 윤 흥 (인)

위 원 이 병 영 (인)

# 초 록

오늘날 클라우드 컴퓨팅 기술은 사물 인터넷 서비스, 인공지능 서비스 등 다양한 서비스를 위해 사용되고 잇다. 하지만 이렇게 클라우드에 많은 정보들이 처리되게 되면서 이러한 클라우드 컴퓨팅 환경에 대한 보안 문제에 대한 우려도 커지고 있는 상황이다. 이를 위해 클라우드 컴퓨팅 환경에서 가상머신의 무결성을 보장하기 위한 다양한 연구가 진행되었지만, 아직 가상머신 커널의 실행 흐름 무결성의 경우에는 커널 코드를 수정해야할 뿐 아니라 효율적으로 보호도 하지 못하였다. 또한 실제 클라우드 환경에서는 다수의 가상머신이 동시에 실행되며 각각의 가상머신이 서로 다른 커널로 동작할 수 있기 때문에 가상머신들 간의 구분을 필요로 한다. 이에 이번 논문에서는 클라우드 환경에서 가상머신 커널의 수정 없이 실시간으로 다수의 가상머신에 대해 효율적으로 가상머신 커널의 실행 흐름 무결성을 보호하는 RTC-VM을 제안한다. 이를 위해 RTC-VM은 동시에 실행되는 여러 개의 가상머신을 구분하여 각각에 대해 무결성을 검증할 수 있게 구현되었다. 또한, 가상머신 실행 흐름 정보에 대한 유실 없이 실시간으로 모니터링할 수 있다. 그리고 효율적으로 수행 흐름 정보를 얻기 위해 RTC-VM은 최근 인텔 아키텍처에서 지원하는 하드웨어 기능인 Processor Trace (PT)를 활용하였으며, 가상머신의 성능 실험에서도 7.5%의 적은 성능 오버헤드로 수행 흐름 무결성을 보장하였다.

**주요어 : 클라우드 컴퓨팅, 수행 흐름 무결성, 인텔 프로세서 트레이스, 코드 재사용 공격, 커널, 실시간**
**학 번 : 2017-20209**

# 목 차

# 표 목차



# 그림 목차

# 제 1 장 소개

## 제 1 절 연구의 배경

오늘날 음성인식 기술과 같은 인공지능 서비스부터 스마트 홈과 같은 사물 인터넷 서비스까지 많은 서비스들이 클라우드 컴퓨팅 환경을 통해 사람들에게 제공되고 있다. 하지만 이러한 서비스들이 늘어갈수록 다양한 중요 정보들이 클라우드 환경에서 처리되면서, 이로 인해 클라우드 컴퓨팅 환경의 보안 문제에 대한 우려도 커지고 있는 상황이다[1].

따라서 이러한 클라우드 컴퓨팅 환경에서의 보안성을 높이기 위한 한 가지 방법으로 클라우드 관리자는 개별 클라우드 사용자가 자신의 어플리케이션을 수행하는 가상머신 (Virtual Machine, VM)의 커널 무결성을 보장해 주어야 한다. 이를 위해 기존 연구들은 가상화를 통해 가상머신에 발생하는 하드웨어 이벤트들을 검사하거나 코드 혹은 바이너리 수정을 통해 가상머신의 커널이 특정 코드를 수행할 때의 정보를 추출하도록 하여 가상 머신 커널의 일부 메모리 무결성을 검증하였다[2, 3].

하지만 이러한 기존 연구들의 방식들은 가상머신 커널의 수행 흐름 무결성을 효율적으로 보장하지 못하였다. 여기서 수행 흐름 무결성이란 감시하는 프로그램이 적법한 수행 흐름으로 실행되는지를 검사하는 것이다. 기존 커널을 대상으로 하는 수행 흐름 무결성 연구[4, 5, 6]들은 이를 위해 커널 코드 및 바이너리를 수정하여 커널의 수행 흐름을 검증하는 방식으로 동작한다. 하지만 이러한 검증 방식은 실제 클라우드 환경에서 쓰이기 어려운 두 가지 이유가 존재한다. 먼저 클라우드 환경에서 사용 중인 모든 가상 머신 커널에 적용하기 어렵다는 문제가 존재한다. 일반적인 클라우드 컴퓨팅 환경에서 클라우드 관리자는 다양한 종류의 커널과 각각의 커널을 여러 버전으로 제공하게 되는데 이러한 커널 수정을 기반으로 하는 접근 방식은 새로운 커널이 등장하거나 커널이 패치될 때마다 클라우드 관리자로 하여금 매번 커널 코드를 수정해야 하므로 현실적으로 적용하기 힘들다. 다음으로 클라우드 사용자의 QoS (Quality of Service) 보장 측면에서도 문제가 될 수 있다. 클라우드 사용자는 클라우드 관리자와의 계약을 통해 가상머신에서 사용할 하드웨어 자원을 정하게 된다. 그리고 클라우드

사용자가 사용하는 동안 클라우드 관리자는 계약에 따라 일정한 QoS를 클라우드 사용자에게 제공해야 한다. 하지만 만약 수행 흐름 무결성을 위해 커널 코드가 수정될 경우 이로 인해 예측하지 못한 성능 부하가 발생하여 QoS를 보장하기 힘들어지는 문제가 발생할 수 있다.

이 논문에서는 기존 가상머신 커널을 위한 수행 흐름 무결성을 가상머신 커널의 수정 없이 효율적으로 간접 분기 명령어 (indirect control transfer instruction) 단위로 보호하는 RTC-VM (Real-Time Control-flow integrity of Virtual Machine)을 제안한다. 이를 위해 RTC-VM은 최근 인텔 아키텍처에서 제공하는 PT (Processor Trace)를 사용하였다. PT는 하드웨어적으로 현재 수행하는 프로세서의 수행 흐름 정보를 패킷의 스트림 형태로 제공하는데 이를 통해 RTC-VM은 가상머신 커널의 수정 없이 수행 흐름 정보를 얻을 수 있었다. 그리고 수행 흐름 무결성 검증 과정을 감시하는 가상머신의 수행 과정과 독립적으로 처리하게 함으로써 감시하는 가상머신의 성능 오버헤드를 최소화할 수 있었다. 또한, RTC-VM은 가상머신이 여러 개가 실행되는 실제 시스템에 적용할 수 있도록 각 가상머신을 구분하고 실시간으로 수행 흐름 무결성을 검증할 수 있다.

# 제 2 장 배경 및 공격 모델

## 제 1 절 인텔 프로세서 트레이스

| Packet name | Packet Description |
|---|---|
| Paging Information Packet (PIP) | 현재 사용 중인 Paging 정보를 담고 있는 CR3 레지스터의 변화가 있을 때마다 발생하는 패킷이다. |
| Flow Update Packet (FUP) | Interrupt와 exception이 발생할 경우의 소스 주소 (source address)를 기록하는 패킷이다. |
| Target IP Packet (TIP) | Indirect branch, exception, interrupt가 발생할 경우의 타겟 주소 (target address)를 기록하는 패킷이다. |
| TNT packet | Direct conditional branch의 taken/not-taken 여부를 기록하는 패킷이다. |

**표 1 RTC-VM에서 분석하는 주요 Intel PT packet들과 이에 대한 설명**

Intel PT는 CPU에서 생성된 실행 흐름을 효율적으로 추적하고 기록할 수 있도록 하는 최신 x86/64 아키텍처에서 지원하는 하드웨어 기능이다. 실행 트레이스는 데이터 패킷을 순차적으로 연결하여 형성되며, 각각 분기 타겟 및 분기점 표시와 같은 동적 제어 흐름 변화에 대한 정보로 인코딩된다. 표 1은 PT가 제공하는 실행 트레이스를 구성하는 여러 유형의 데이터 패킷을 설명한다. PIP와 FUP는 각각 CR3 레지스터가 업데이트되고 인터럽트와 같은 비동기 이벤트가 발생할 때 CPU에 의해 생성된다. 이러한 패킷은 트레이스에서 상대적으로 드물게 나타나지만, 문맥이 고유한 복수의 실행 단위인 가상머신이나 프로세스로 순차적 트레이스를 분리하는데 이용할 수 있다는 점에서 중요하다. TIP와 TNT는 각각 프로세스의 제어 흐름의 모든 변경을 나타내는 트레이스에서 대부분의 패킷을 차지한다. TIP는 간접 분기 (e.g. indirect jmp/call and ret)의 타겟 주소를 기록하고 TNT는 각 조건부 분기 (e.g. jz, je, loop 등)를 취하는지 여부를 표시한다. 한편 Intel PT는 사용 목적에 따라 패킷 생성 관련 설정이나 필터 관련 기능도 제공한다. 이와 관련한 자세한 내용은 관련 매뉴얼에

정리되어 있다[7].

## 제 2 절 공격 모델

공격자는 먼저 victim 사용자가 이용 중인 가상머신을 네트워크로 접속하여 공격 행위를 하는 원격 공격자나, 클라우드 컴퓨팅 환경을 공격하고자 가상 머신 커널을 공격하는 악성 사용자를 가정한다. 공격자는 가상 머신 커널에 존재한 취약점을 이용하여 수행 흐름 탈취 (Control Flow Hijacking) 공격을 행할 수 있다.

한편 부팅 시 커널 무결성이 보장되고 이를 확인한 직후 제안하는 RTC-VM이 효율적으로 돌아가기 시작한다고 가정한다. 또한 다양한 클라우드 대상 공격 가운데 내부 클라우드 관리자에 의한 공격은 고려하지 않는다. RTC-VM은 가상머신 커널의 수행 흐름 무결성에 대해서만 대응하였다.

# 제 3 장 디자인 및 구현

## 제 1 절 디자인 개요

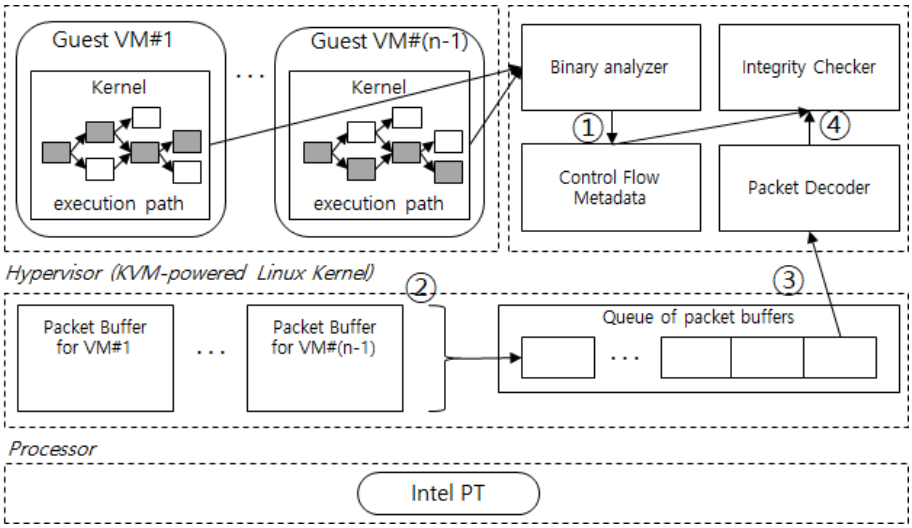


그림 1 RTC-VM의 디자인 개요

그림 1과 같이 RTC-VM의 동작은 크게 네 단계로 이루어져 있다. 먼저 binary analyzer는 오프라인에서 모니터링의 대상이 될 가상머신이 동작하기 전에 guest kernel binary를 분석하여 무결성 검증에 쓰일 control flow metadata를 생성한다. 이러한 control flow metadata에는 guest kernel code 내에 존재하는 각 basic block에 대한 정보를 추출하게 된다. 두 번째로 가상머신이 동작을 하면 각 가상머신을 식별하여 같은 가상머신에서 추출된 트레이스는 같은 버퍼에 기록되도록 관리한다. 기록이 완료된 버퍼는 RTC-VM이 관리하는 큐의 대기열에 전달된다. 세 번째로 전달받은 버퍼는 큐에서 관리되며 패킷 디코더에 차례대로 전달되고 디코드를 마친 버퍼는 사라지게 된다. 이를 통해 가상머신이 실행되는 동안 실시간으로 버퍼의 유실 없이 모니터링이 가능하게 된다. 마지막으로 runtime integrity enforcement 과정에서는 먼저 기록된 패킷들 가운데 전처리를 통해 guest kernel에 해당 하는 트레이스만을 추려내고 이를 integrity checker에게 전달한다. Integrity checker는 오프라인에 생성해 둔 control flow metadata를

이용해 전달 받은 트레이스의 수행 흐름 무결성을 검증하게 된다. 이러한 일련의 무결성 검증 과정은 가상머신의 수행과는 독립적인 host kernel의 프로세스로 동작함으로써 guest OS에 대한 수정 없이 효율적으로 수행될 수 있다.

## 제 2 절 오프라인 바이너리 분석

RTC-VM이 보장하는 Control Flow Integrity Policy에 따라 바이너리 분석을 통해 얻고자 하는 metadata가 달라질 것이다. RTC-VM의 Control Flow Integrity Policy에 대해 먼저 설명하고자 한다. Control flow hijacking attack의 경우, 공격자는 control flow를 변조하여 공격자 자신이 수행하고자 하는 악성코드를 실행하고자 한다. 이를 위해 공격자는 control transfer instruction을 사용할 수 밖에 없는데, control transfer instruction은 크게 두 가지로 나눌 수 있다. 먼저 direct control transfer instruction은 코드에 control transfer target address가 저장되어 있는 형태로 공격자는 이러한 코드를 변조하여 공격할 수 있다. 하지만 이러한 코드 변조 행위는 기존 W+X policy를 통해 쉽게 방어가 가능하다고 알려져 있다[8]. 한편 indirect control transfer instruction의 경우에는 control transfer target address가 레지스터에 의해 결정되므로 공격자는 코드의 변조없이 수행 흐름을 변조할 수 있다. 이를 막기 위한 다양한 수행 흐름 무결성 연구들이 존재하며, RTC-VM에서는 그 중 guest kernel이 수행하는 indirect control transfer instruction에 대하여 항상 basic block의 entry로 jump하는지를 enforce하였다. 한편 이러한 수행 흐름 무결성 검증 방법을 통해 RTC-VM은 비교적 적은 runtime overhead로 효율적인 실행 흐름 무결성을 보장할 수 있었다.

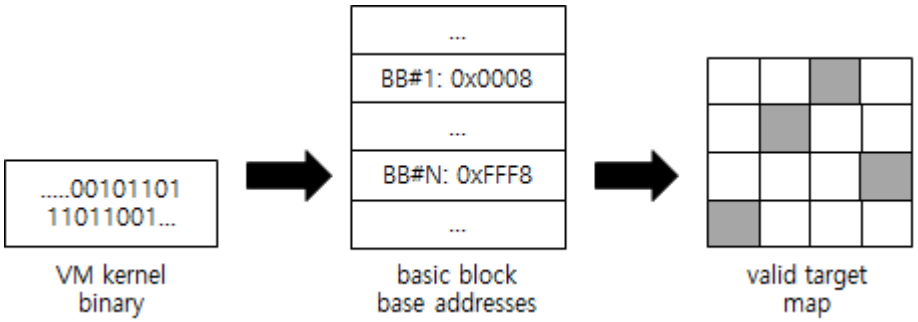


그림 2 control flow metadata 생성 과정

앞서 설명한 RTC-VM의 Control Flow Integrity Policy를 보장하기 위해서는 가상머신 커널 코드 내에 존재하는 basic block들의 시작 주소에 대한 정보를 알고 있어야 한다. 오프라인 바이너리 분석은 바로 이를 위한 과정으로, objdump 툴을 이용해 먼저 가상머신 커널 바이너리를 disassemble 한 뒤에 바이너리 내에 존재하는 모든 basic block의 시작 주소를 추출하였다. 이렇게 추출된 정보를 통해 그림 2와 같이 각 코드의 byte address마다 valid indirect branch target address인지를 하나의 bit로 표시하는 valid target map을 생성하였다. 이 valid target map이 integrity checker에게 전달될 control flow metadata가 된다.

## 제 3 절 가상머신 단위 버퍼 관리

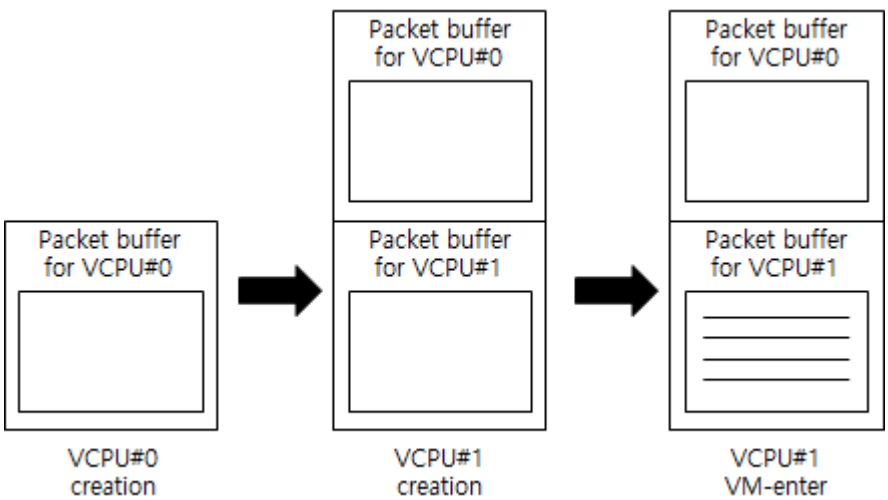


**그림 3 가상머신 단위 패킷 버퍼 분류**

RTC-VM은 여러 개의 가상머신이 동시에 실행되는 실제 가상 환경을 가정하고 있기 때문에 가상머신마다 기록되는 패킷들이 서로 섞일 가능성이 높다. 또한 모든 가상머신이 동일한 OS 커널을 사용하지 않는다는 점에도 유의해야 한다. 따라서 기록된 패킷은 디코드되어 integrity checker가 이용하기 전에 가상머신 별로 분류되어야 한다. 패킷이 일단 기록되면 각 패킷이 생성되는 가상머신을 나타내는 특정 식별자가 없기 때문에 패킷을 분류할 수 없다. 따라서 RTC-VM은 처음부터 가상머신에 의해 패킷을 정렬하기 때문에 하이퍼바이저의

CPU 가상화 단위인 VCPU의 형태로 각 가상머신에 CPU 리소스가 할당된다는 점을 이용한다. 특히 가상머신이 생성되어 하나 혹은 그 이상의 VCPU를 할당하면 RTC-VM은 그림 3과 같이 VCPU마다 트레이스 버퍼를 준비한다. 가상머신이 실행되는 동안 VM-enter이 host에서 해당 VCPU로 실행되는 guest로의 control이 전달되도록 하는 기능을 한다. 이 기능을 이용해 VCPU에서 VM-enter가 발생할 때마다 Intel PT를 동작시켜 이 VCPU에서 발생하는 패킷들을 해당 트레이스 버퍼에 기록하도록 한다. 그런데 VCPU와 실제 CPU 사이의 매핑이 동적으로 변경될 수 있다. 단순히 VCPU만을 고려하여 트레이스 버퍼에 기록하는 것이 아니라 VCPU가 매핑된 해당 CPU의 Intel PT를 동작시키도록 함으로써 동적으로 매핑이 변경되는 문제를 해결할 수 있다. 따라서 각각의 트레이스 버퍼들을 읽음으로써 RTC-VM은 시스템 내에서 동시에 돌아가는 모든 가상머신들에 대해 각각 패킷들을 처리할 수 있게 된다.

## 제 4 절 실시간성 보장

가상머신이나 프로세스의 수행이 끝나고 난 뒤에 무결성을 검증하는 방식은 이미 공격자로 인해 시스템이 장악된 이후이기 때문에 너무 늦다. 따라서 가상머신이 실행되는 동안 짧은 간격으로 주기적으로 모니터링할 필요가 있다. 이를 통해 실시간성을 보장하는 방법에는 긴 latency와 패킷 유실의 두 가지 문제점이 있다. RTC-VM은 이 문제들을 해결하여 무결성 검증을 실시간으로 가능하도록 보장하였다.

첫 번째 문제는 Intel PT를 동작시키기 위한 기존 툴을 이용했을 때 발생하는 긴 latency이다. PT를 조작하는 대표적인 툴은 perf와 simple-pt[9]가 있다. Perf는 리눅스의 성능 분석 툴이며 PT 동작에 대해서도 다양한 기능을 제공한다. Simple-pt는 PT에 관해 간단한 동작만을 지원하는 오픈소스 툴이다. 두 툴 모두 디코드를 위해 트레이스 버퍼를 파일로 덤프하는 과정이 불가피하며 파일에 읽고 쓰는 동작은 긴 latency를 초래한다. 또한 트레이스 버퍼는 커널 메모리 영역에 존재하기 때문에 커널과 유저를 오가는 동작들을 수행하며 긴 latency를 수반하게 된다. 따라서 기존 툴을 그대로 이용하여 실시간성을 보장하기에는 어려움이 있다. 이를 해결하기 위해 커널에서 트레이스 버퍼를 관리하는 부분과 유저에서 디코드를 하는 부분이 같이 접근할 수 있는 공유 메모리 영역을 설정했다. 공유 메모리에 트레이스

버퍼의 내용을 복사하고 이를 유저에서 디코드하는 방식을 통해 파일로 덤프하는 과정을 제거하고 파일 읽기/쓰기 동작을 메모리 읽기/쓰기 동작으로 치환함으로써 latency를 줄였다. 또한 공유 메모리의 사용으로 따로 유저와 커널 사이를 오가는 동작도 제거가 되었다. 따라서 트레이스 기록부터 디코드까지의 총 latency를 줄였다.

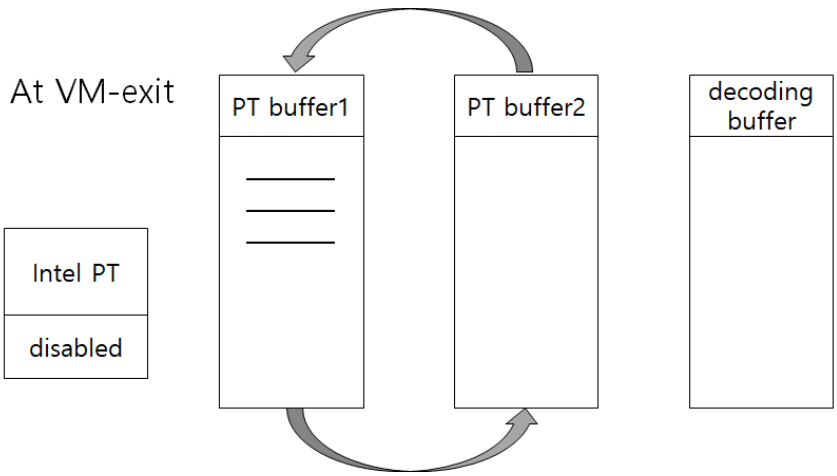


**그림 4 VM-exit에서의 더블 버퍼링**

두 번째 문제는 패킷의 유실 가능성이다. 패킷이 유실될 수 있는 상황은 패킷 단위의 유실과 트레이스 버퍼 단위의 유실 두 가지로 나눌 수 있다. 패킷 단위의 유실이 일어나는 상황은 가상머신이 동작하고 있는데 PT가 disable된 상태여서 이 동안의 가상머신의 동작을 나타내는 패킷들이 기록되지 않으면서 발생한다. 발생한 패킷의 양을 나타내는 offset 값을 얻기 위해서 PT가 disable되어야 하고 안전하게 디코드 버퍼에 복사할 때까지 disable 상태를 유지해야 한다. 이 과정에서 disable 동안 발생하는 패킷들이 유실된다. 이를 해결하기 위해 RTC-VM은 VM-exit에서의 더블버퍼링을 제안한다. 더블 버퍼링은 VCPU당 하나가 아니라 두 개의 버퍼를 둠으로써 offset 값을 얻기 위해 PT를 disable하고 디코드를 수행하는 동안 다른 버퍼에 패킷들을 기록하도록 하는 방법이다. 하지만 이 과정에서도 PT를 disable하고 버퍼를 바꾸는 작은 간격 사이에도 발생할 수 있는 유실 가능성을 무시할 수 없다. 이러한 가능성도 아예 제거하기 위해 VM-exit부터 다시 VM-enter로 들어가기 전까지 가상머신의 실행이 멈춘다는 사실에 착안하여, 버퍼를 바꾸는 순간을 VM-exit으로 설정했다.

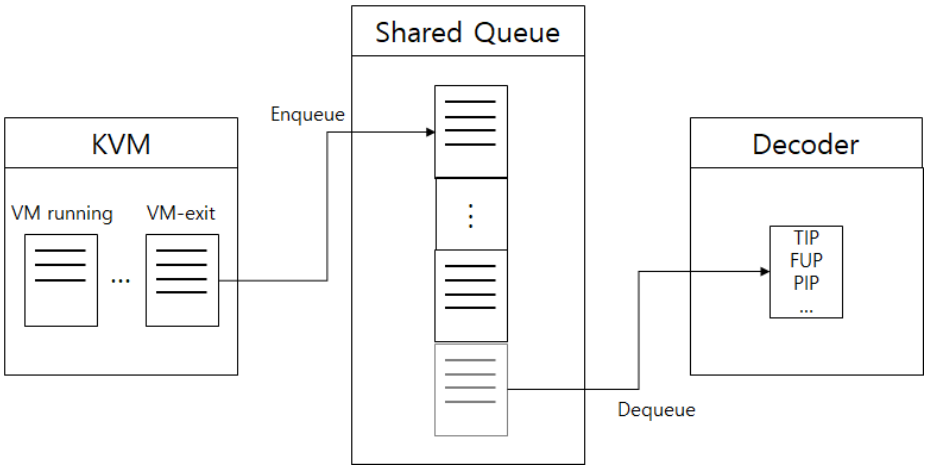


**그림 5 공유 큐 동작**

트레이스 버퍼 단위의 유실이 일어날 수 있는 두 번째 상황은 두 가지가 있다. 하나는 여러 개의 가상머신들에서 동시에 VM-exit이 일어나면 이 때 발생하는 트레이스 버퍼들 중 하나만 디코더에게 전달되면서 나머지가 유실된다. 나머지 상황은 디코드가 끝나지 않은 상황에서 VM-exit이 일어나면 해당 트레이스 버퍼는 유실된다. 이를 해결하기 위해 그림 5와 같이 RTC-VM은 트레이스 버퍼를 쌓아두는 KVM과 디코더 사이의 공유 큐를 이용한다. 각 VM-exit마다 해당 트레이스 버퍼가 공유 큐의 대기열에 입력된다. 디코더가 트레이스 버퍼를 디코드할 때마다 공유 큐의 대기열에서 해당 트레이스 버퍼가 제거된다. 따라서 앞서 언급한 여러 VM-exit이 동시에 발생하거나 디코드가 끝나지 않은 상황에서 VM-exit이 발생하더라도 공유 큐에 트레이스 버퍼를 넣어둠으로써 트레이스 버퍼 단위의 유실이 발생하는 가능성을 제거한다.

RTC-VM은 무결성 검증에 대한 실시간성을 보장하기 위해 검증에 관한 일련의 과정에 수반되는 latency를 줄이고 주기적인 모니터링에서 발생할 수 있는 패킷 유실의 가능성을 제거했다.

## 제 5 절 무결성 검증 알고리즘

RTC-VM에서는 무결성 검증을 위해 guest kernel을 수행하면서 발생하는 indirect branch target address를 추출하기 위해 Intel PT를 사용하였다. 이때 Intel PT에서 나오는 다양한 종류의 패킷 가운데

무결성 검증에 필요한 패킷만을 추출하고 분석하기 위해 packet decoder를 구현하였다. 그리고 integrity checker는 packet decoder에 의해 추출된 indirect branch target address를 가지고 guest kernel의 실행 흐름 무결성을 검증하는 역할을 담당한다.

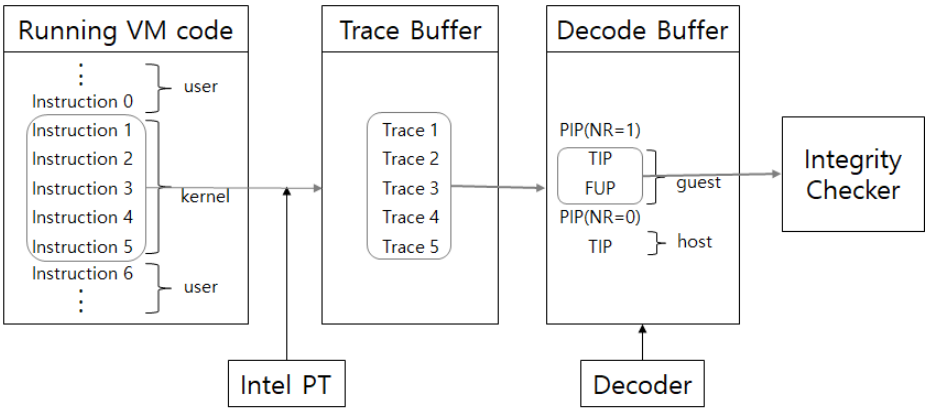


**그림 6 패킷 필터링**

RTC-VM은 가상머신의 커널 수행 흐름에만 관심이 있기 때문에 그림 6과 같이 패킷 필터링 과정을 통해 원하는 패킷만을 얻어낸다. 디코더에서는 이와 관련된 패킷을 전달하기 위해 먼저 VCPU가 커널 모드에서 실행될 때만 패킷 생성을 활성화함으로써 Intel PT의 Current Privilege Level (CPL) 기반의 selective tracing 기능을 이용한다. 그러나 가상화 시스템에서는 host와 guest 두 가지 유형의 커널이 있으며, VCPU가 host 또는 guest 커널 모드에서 실행될 때 두 경우 모두 CPL이 0으로 설정된다. 즉, CPL=0에 대해 PT가 selective tracing을 하도록 구성된 경우에도 디코더는 host와 guest 커널에 의해 생성된 트레이스 패킷을 수신하게 된다. 위에서 말한 바와 같이 RTC-VM은 guest 커널의 실행 흐름만 필요하기 때문에 이는 바람직하지 않다. 다행히 패킷 중에 PIP는 각 패킷이 host 또는 guest 커널에서 발생하는지 여부를 나타내는 non-root (NR) 비트를 전달한다. 따라서 단지 비트를 점검하는 것만으로도 host 커널에 해당하는 패킷을 제거하여 guest 커널의 패킷만을 저장할 수 있다. 다음으로 디코더에서는 수행 흐름 관련 패킷들 가운데 indirect branch의 target address 정보를 담고 있는 TIP 패킷과 asynchronous event의 source address 정보를 담고있는 FUP 패킷만을 추출하여 integrity checker에게 넘겨주도록 하였다.

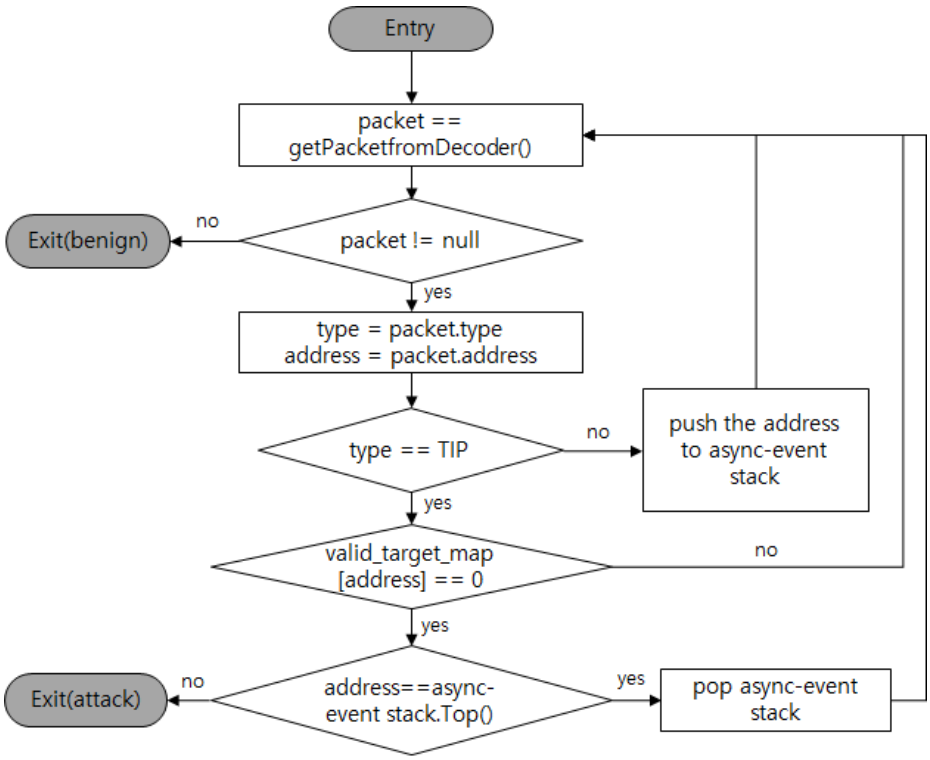


그림 7 Integrity Checker의 동작 알고리즘

Integrity Checker는 디코더가 전달해준 수행 흐름 정보를 통해 그림 5와 같은 방법으로 수행 흐름 무결성을 검증하였다. 먼저 디코더에서 패킷을 받아와서 패킷이 NULL인 경우는 프로그램이 안전하게 수행이 끝났다는 것으로 판단한다. 패킷이 NULL이 아니면 패킷의 타입을 판단하여 FUP 패킷인 경우 (TIP 패킷이 아닌 경우)에 해당 source address를 async-event stack에 push하여 저장하고 다음 패킷을 처리한다. 만약 TIP 패킷인 경우, target address에 대한 valid target map의 해당하는 비트를 확인하여 basic block의 base address 인지 아닌지를 판단한다. 만약 비트 값이 1인 경우 정상적인 수행 흐름의 분기이므로 다음 패킷을 처리한다. 만약 비트 값이 0인 경우에는 asynchronous event로 인한 수행 흐름 분기인지를 확인한다. 즉, async-event stack의 top element와 비교하여 일치하는지 확인한다. 이 때 target address가 top element의 다음 명령어를 가리키는 경우를 처리하기 위해서 차이가 0xf 이내일 경우 정상적인 경우로 처리하였으며, 이후 다음 패킷을 처리하도록 하였다. 만약 top element와도 대응되지 않을 경우 공격으로 간주하였다.

# 제 4 장 실험 결과

## 제 1 절 실험 환경 및 프로토타입 구현

RTC-VM의 프로토타입을 실제 물리 머신에서 구현하여 실험하였다. 사용한 머신은 4개의 Intel core i5-6600 CPU @ 3.30GHz와 4GB RAM의 스펙을 가지고 있고, 호스트 커널과 게스트 커널은 모두 4.8.0 버전을 사용하였다.

앞서 디자인에서 언급한 바와 같이 RTC-VM의 경우 게스트 커널을 수정하지 않고 하이퍼바이저 역할의 KVM 코드 수정 및 유저 어플리케이션으로 구현하였다. Binary analyzer는 objdump로 커널 바이너리를 덤프하여 나온 출력 파일을 파싱하여 구현하였으며, 107 SLoC으로 구현하였다. KVM 코드는 VCPU를 생성, 제거하는 함수와 VM-exit, VM-enter를 관리하는 함수 내에 VCPU 단위로 트레이스 버퍼를 관리하고 공유 큐를 관리하는 코드를 추가하여 구현하였으며, 221 SLoC으로 구현하였다. Decoder 및 integrity checker는 Intel Processor Trace Decoder Library[10]를 참고하여 구현하였으며, 314 SLoC으로 구현하였다.

## 제 2 절 성능 오버헤드

| benchmark | overhead |
|---|---|
| hanoi | 6.1% |
| arith | 10.2% |
| int | 6.8% |
| syscall | 5.8% |
| dhry | 5.7% |
| context1 | 10.5% |
| average | 7.5% |

**표 2 가상머신의 Unixbench 성능 오버헤드**

RTC-VM의 성능 부하를 측정하기 위해 RTC-VM이 동작하는 상태에서 실제 가상머신의 application benchmark의 성능을 측정하는

실험을 설계하였다. RTC-VM과 가상머신은 서로 다른 코어에서 수행하도록 하였다. 이는 기존의 Intel PT를 활용한 연구들[11, 12]에서 한 실험과 같은 가정이다. 표 1은 실제로 가상머신에서 Unixbench만 수행한 경우와 RTC-VM이 동작하는 동안 수행한 경우에 발생한 성능 오버헤드를 정리한 표이다. 그 결과 평균 7.5%의 성능 오버헤드가 측정되었다. 이러한 오버헤드는 RTC-VM이 별도의 코어에서 동작함에도 불구하고 메모리의 경우에는 RTC-VM과 가상머신이 함께 쓰는 컴퓨팅 자원이기 때문에 이를 가상화하는 과정에서 발생한 것과 Intel PT의 기본 동작에 의한 오버헤드가 반영된 것으로 보인다.

## 제 3 절 공격 탐지

RTC-VM은 가상머신 커널의 수행 흐름 무결성을 보장하기 위한 연구이다. 이번 장에서는 실제로 이러한 RTC-VM이 가상머신 커널에서 발생한 수행 흐름 변조 공격을 검출할 수 있는지를 검증하기 위해 간단한 rootkit 공격 코드를 작성하여 실험했다. 사용한 rootkit 공격 코드는 가상머신 커널이 관리하는 가상 파일 시스템(Virtual File System, VFS)과 관련된 자료구조를 변조하는 공격이다. 구체적으로 공격 코드는 이러한 자료구조가 나타내는 특정 파일의 파일 연산과 관련된 함수 포인터를 변조하게 된다. 그리고 추후에 이 파일을 접근하는 경우, 정상적인 파일 연산 함수가 아닌 공격자가 의도한 공격 코드가 수행되게 된다. 실험 결과 RTC-VM은 바로 이렇게 공격 코드가 실행되는 순간에 발생하는 indirect call 패킷을 분석하여 valid target map에 없는 것을 확인하여 공격으로 검출하였다.

# 제 5 장 관련 연구

이번 장에서는 RTC-VM이 Intel PT를 이용해 가상머신 커널을 대상으로 수행 흐름 무결성을 검증하는 것과 관련된 보안 분야의 논문들에 대해 살펴보고 각각에 대해 RTC-VM이 어떠한 차이가 있는지에 대해 살펴보고자 한다.

## 제 1 절 Virtual Machine Introspection

VMI 연구들은 클라우드 환경에서 각각의 가상머신의 상태 정보를 추출하여 보안 검증을 실시하였다[2, 3]. 이러한 연구들은 기본적으로 가상화 기술이나 바이너리 수정 기술을 사용하여 가상머신이 사용하는 하드웨어 자원들에 대해 모니터링 하거나, 가상머신이 동작하면서 발생하는 다양한 이벤트들에 대한 정보를 추출하여 사용하게 된다. 하지만 아직까지 가상머신 커널의 수행 흐름 정보를 효율적으로 추출하는 연구들은 없었으며, RTC-VM은 Intel PT를 이용해 이러한 정보를 효율적으로 추출하고 처리하여 가상머신 커널에 대한 수행 흐름 무결성을 보장하였다.

## 제 2 절 Control Flow Integrity using Intel PT

최근 Intel PT라는 새로운 하드웨어 기능을 통해 수행 흐름 정보를 효율적으로 추출할 수 있게 되면서 이를 이용해 수행 흐름 무결성을 지키려는 연구들이 등장하고 있다[11, 12, 13]. 하지만 이러한 연구들은 모두 사용자 어플리케이션을 위한 수행 흐름 무결성 연구들로 RTC-VM의 경우에는 클라우드 환경의 가상머신 커널을 대상한다는 점이 다르며, 구체적으로 이로 인해 asynchronous event로 인한 Intel PT packet을 처리하는 과정에서 차이가 존재한다.

## 제 3 절 Kernel Control Flow Integrity

KCoFI[14]는 Secure Virtual Architecture라고 하는 가상화된 환경에서 커널을 가상 명령어 (virtual instruction)로 컴파일한 뒤 커널 동작 중에 하이퍼바이저에서 이러한 가상 명령어를 실제 명령어로

변환하는 과정에서 수행 흐름 무결성을 검증하였다. Xingyang. El[15]은 커널의 분기 명령어마다 수행 흐름 무결성을 검증하는 코드를 삽입하여 구현하였으며 이 때 적법한 목적지 주소를 저장하는 테이블을 공격자로부터 지키기 위한 커널 코드 수정도 함께 하였다. 한편 krx[16]는 커널 코드를 읽기가 불가능하도록 하고, 코드의 위치를 임의로 섞는 방식으로 하여 커널에 대한 Just In Time Code Reuse Attack으로부터 방어하였다. 한편 메모리 페이지 단위로 가상머신 커널의 수행 흐름 무결성을 검증하려는 연구도 존재하였다[17]. 앞서 언급한 기술들은 모두 커널 코드를 재컴파일하거나 많은 부분을 수정해야 한다는 한계가 존재한다. 하지만 RTC-VM은 이러한 수정없이 가상머신 커널에 대한 실행 흐름 무결성을 보장하였다.

# 제 6 장 결론

RTC-VM은 여러 개의 가상머신이 실행되는 시스템에서 실시간으로 가상머신 커널에 대해 수행 흐름 무결성을 보장하는 연구이다. 실제 시스템과 같은 환경에 적용할 수 있도록 가상머신 단위로 구분하여 각각에 대해 무결성을 검증할 수 있다. 또한, 유실되는 데이터 없이 실시간으로 가상머신의 커널을 모니터링할 수 있다. 기존 게스트 커널 코드의 수정을 동반하는 커널 수행 흐름 무결성 연구들에 비해 RTC-VM은 Intel PT라고 하는 하드웨어 기능을 사용하여 이러한 커널 코드의 수정 없이 7.5%의 적은 오버헤드로 가상머신의 수행 흐름 무결성을 검증하였다.

# 참고 문헌

[1] Samarati, Pierangela, et al. “Cloud security: Issues and concerns.” Encyclopedia on Cloud Computing (2016): 1-14.
[2] Zeng, Junyuan, Yangchun Fu, and Zhiquiang Lin. “Pemu: a pin highly compatible out-of-VM dynamic binary instrumentation framework.” ACM SIGPLAN Notices. Vol. 50. No. 7. ACM, 2015.
[3] LibVMI, http://libvmi.com/
[4] Criswell, John, Nathan Dautenhahn, and Vikram Adve. “KCoFI: Complete control-flow integrity for commodity operating system kernels” Security and Privacy (SP), 2014 IEEE Symposium on. IEEE, 2014.
[5] Ge, Xinyang, et al. “Fine-grained control-flow integrity for kernel software.” Security and Privacy (EuroS&P), 2016 IEEE European Symposium on. IEEE, 2016.
[6] Pomonis, Marios, et al. “kR^X: Comprehensive kernel protection against just-in-time code reuse.” Proceedings of the Twelfth European Conference on Computer System. ACM, 2017.
[7] Guide, Part. “Intel® 64 and IA-32 Architectures Software Developer’s Manual.” Volume 3B: System programming Guide, Part2 (2011).
[8] Seshadri, Arvind, et al. “SecVisor: A tiny hypervisor to provide lifetime kernel code integrity for commodity OSes.” ACM SIGOPS Operating Systems Review. Vol. 41. No. 6. ACM, 2007.
[9] Simple-pt, http://github.com/andikleen/simple-pt/
[10] Intel® Processor Trace Decoder Library, http://github.com/01org/processor-trace/
[11] Ge, Xinyang, Weidong Cui, and Trent Jaeger. “Griffin: Guarding control flows using intel processor trace.” Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems. ACM, 2017.
[12] Gu, Yufei, et al. “PT-CFI: Transparent backward-edge control flow violation detection using intel processor trace.” Proceedings of the Seventh ACM on Conference on Data and

Application Security and Privacy. ACM, 2017.
[13] Liu, Yutao, et al. “Transparent and efficient cfi enforcement with intel processor trace.” High Performance Computer Architecture (HPCA), 2017 IEEE International Symposium on. IEEE, 2017.
[14] Criswell, John, Nathan Dautenhahn, and Vikram Adve. “KCoFI: Complete control-flow integrity for commodity operating system kernels.” Security and Privacy (SP), 2014 IEEE Symposium on. IEEE, 2014.
[15] Ge, Xinyang, et al. “Fine-grained control-flow integrity for kernel software.” Security and Privacy (EuroS&P), 2016 IEEE European Symposium on. IEEE, 2016.
[16] Pomonis, Marios, et al. “kR^X: Comprehensive kernel protection against just-in-time code reuse.” Proceedings of the Twelfth European Conference on Computer Systems. ACM, 2017.
[17] Zhan, Dongyang, et al. “Checking virtual machine kernel control-flow integrity using a page-level dynamic tracing approach.” Soft Computing (2017): 1-11.

## Abstract

# Kernel Control-Flow Integrity of Multiple Virtual Machines in Real-Time using Intel Processor Trace

Baek Sehyun
Electrical and Computer Engineering
The Graduate School
Seoul National University

Nowadays cloud computing technology is used for a variety of services, such as the internet of things and artificial intelligence. However, as more data is being processed in the cloud, there is growing concern about security issues in the cloud computing environment. To solve this concern, many studies have been conducted to ensure the integrity of virtual machines in a cloud computing environment. However, in the case of the control flow integrity for the virtual machine, existing studies are not only necessary to modify the kernel code, but also cannot protect it efficiently. Also, since multiple VMs which can have different kernel one another run simultaneously in real-world cloud computing environment, it is required to identify VMs. In this paper, we propose RTC-VM which efficiently protects the control flow integrity of VM kernel for multiple VMs without modification of VM kernel in real-time. For this purpose, RTC-VM is implemented to enforce control flow integrity of each VM with identifying multiple VMs which run concurrently. In addition, RTC-VM can monitor in real-time without loss of execution control-flow information. For efficient monitoring, RTC-VM utilizes Processor Trace (PT), a

hardware feature that is recently supported by Intel architecture. According to the experimental results, RTC-VM incurs on average 7.5% overhead.

Keywords : Cloud Computing, Control-Flow Integrity (CFI), Intel Processor Trace (Intel PT), Code Reuse Attack (CRA), Kernel, Real-Time
Student Number : 2017-20209