KIPS Trans. Comp. and Comm. Sys.
Vol.9, No.5 pp.101~106 pISSN: 2287-5891

https://doi.org/10.3745/KTCCS.2020.9.5.101

JMP+RAND: Mitigating Memory Sharing-Based Side-Channel Attack by Embedding Random Values in Binaries

Taehun Kim† · Youngjoo Shin††

ABSTRACT

Since computer became available, much effort has been made to achieve information security. Even though memory protection defense mechanisms were studied the most among of them, the problems of existing memory protection defense mechanisms were found due to improved performance of computer and new defense mechanisms were needed due to the advent of the side-channel attacks. In this paper, we propose JMP+RAND that embedding random values of 5 to 8 bytes per page to defend against memory sharing based side-channel attacks and bridging the gap of existing memory protection defense mechanism. Unlike the defense mechanism of the existing side-channel attacks, JMP+RAND uses static binary rewriting and continuous jmp instruction and random values to defend against the side-channel attacks in advance. We numerically calculated the time it takes for a memory sharing-based side-channel attack to binary adopted JMP+RAND technique and verified that the attacks are impossible in a realistic time. Modern architectures have very low overhead for JMP+RAND because of the very fast and accurate branching of jmp instruction using branch prediction. Since random value can be embedded only in specific programs using JMP+RAND, it is expected to be highly efficient when used with memory deduplication technique, especially in a cloud computing environment.

Keywords : Memory Sharing-based Side-channel Attack, Binary Rewriting, Memory Sharing, Cloud Computing, Countermeasure

JMP+RAND: 바이너리 난수 삽입을 통한 메모리 공유 기반 부채널 공격 방어 기법

김 태 훈† · 신 영 주††

요 약

컴퓨터가 보급된 이래로 정보보안을 달성하기 위해 많은 노력이 이루어졌다. 그중 메모리 보호 기법에 대한 연구가 가장 많이 이루어졌지만, 컴퓨터의 성능 향상으로 기존 메모리 보호 기법의 문제들이 발견되었고 부채널 공격의 등장으로 새로운 방어기법이 필요하게 되었다. 본 논문에서는 JMP+RAND 기법을 이용해 페이지(Page)마다 5-8byte의 난수를 삽입하여 메모리 공유 기반 부채널 공격을 방어하고 기존 메모리 보호 기법도 보완하는 방법을 제안한다. 기존 부채널 공격들의 방어기법과 달리 JMP+RAND 기법은 정적 바이너리 재작성 기법(Static binary rewriting)과 연속된 jmp 명령어, 난수 값을 이용해 사전에 부채널 공격을 방어한다. 우리는 메모리 공유 기반 부채널 공격이 JMP+RAND 기법이 적용된 바이너리를 공격하는 데 걸리는 시간을 정량적으로 계산하였고 현실적인 시간 내에 공격할 수 없다는 것을 보여주었다. 최근 아키텍처는 분기 예측(Branch prediction)을 이용해 jmp 명령어의 분기처리가 매우 빠르고 정확하므로 JMP+RAND 기법의 오버헤드가 매우 낮다. 특히 특정 프로그램에만 난수 삽입이 가능하므로 클라우드 컴퓨팅 환경에서 메모리 중복제거 기능과 함께 사용하면 높은 효율성을 보일 수 있을 것으로 기대한다.

키워드 : 메모리 공유 기반 부채널 공격, 바이너리 재작성, 메모리 공유, 클라우드 컴퓨팅, 방어 기법

1. 서 론

컴퓨터가 보급된 후 약 30년 동안 KASLR[1](Kernel Address Space Layout Randomization), ASLR, DEP (Data Execution Prevention)와 같은 메모리 보호 기법이 BOF(Buffer Overflow), ROP(Return Oriented Programming)의 변종을 방어하기 위해 만들어졌다. 하지만 메모리 보호 기법은 메모리 변조(Memory corruption) 공격을 방어하는 기법이기 때문에 최근 대두하는 부채널(Side-channel) 공격을 방어하기에 적합하지 않고 컴퓨터 성능 향상으로 KASLR, ASLR의 엔트로피(Entropy) 부족 문제가 발견되었다. 이로 인해 공격자들은 비교적 짧은 시간 내에 유효한 커널 주소와 공격대상이 되는 함수의 주소를 정확히 알아내어 메모리 변조 공격을 시도할 수 있었다. 특히 클라우드 컴퓨팅 환경에서 Flush+ Reload[2], Flush+Flush[3]와 같은 캐시 부채널 공격과 메모리 노출(Memory disclosure) 공격[4-8] 등 메모리 공유 기반 부채널 공격을 방어하기 위해 메모리 중복제거 기능을 사용하지 못하여 높은 오버헤드가 발생하고 있다. 또한 클라우드 컴퓨팅

※ 이 논문은 2019년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임(No.2019-0-00533, 컴퓨터 프로세서의 구조적 보안 취약점 검증 및 공격 탐지대응).

† 준 회 원 : 광운대학교 컴퓨터정보공학부 학사과정
†† 정 회 원 : 광운대학교 컴퓨터정보공학부 조교수

Manuscript Received : December 20, 2019
Accepted : February 18, 2020

* Corresponding Author : Youngjoo Shin(yjshin@kw.ac.kr)

환경이 아니더라도 Meltdown[9], Spectre[10], ZombieLoad [11] 등 CPU 부채널 공격은 데이터 획득 과정에서 캐시 부채널 공격을 이용해 신뢰성 있는 프로세서가 제공해야 할 원칙들을 무력화하여 데이터를 노출한다.

최근 이러한 부채널 공격을 방어하기 위해 캐시 부채널 공격들에 대한 방어기법 연구가 진행되고 있다. 하지만 대부분 방어기법은 캐시 부채널 공격 시 변화하는 카운터 값을 PCM[12] (Performance Counter Monitor) 또는 perf[13, 14]로 얻어 머신러닝에 적용해 캐시 부채널 공격을 탐지하는 연구들이 주를 이룬다. 이런 연구들은 공격을 탐지하기 위해 항상 백그라운드 환경에서 실행되어야 하며 실행환경에 따라 탐지율을 보장할 수 없다. 또한 클라우드 컴퓨팅 환경에서 발생할 수 있는 부채널 공격인 메모리 노출 공격에 대한 방어기법도 발견하지 못했다.

따라서 본 논문에서는 사전에 캐시 부채널 공격, 메모리 노출 공격을 일정하게 방어할 뿐만 아니라 엔트로피에 의존하는 기존의 메모리 보호 기법도 보완하는 JMP+ RAND 방어기법을 제안한다. JMP+RAND는 정적 바이너리 재작성 기법을 이용해 .text 섹션의 4KB 페이지마다 5-8byte의 난수를 삽입하여 간접적으로 엔트로피를 40bit 이상 증가시키는 방어기법이다. 페이지마다 삽입될 난수 바이트의 크기는 고정적이지 않기 때문에 공격자가 .text 섹션의 N번째 페이지의 내용을 알기 위해서 이전 N-1개의 페이지에 대해 각각 몇 바이트의 난수가 삽입되었는지를 반드시 알아야 한다. 또한 JMP+RAND는 프로그램을 입력값으로 받아 난수가 삽입된 프로그램을 출력하여 원하는 프로그램만 엔트로피를 증가시킬 수 있으므로 클라우드 컴퓨팅 환경에서 메모리 중복제거 기술을 사용하더라도 캐시 부채널, 메모리 노출 공격을 방어할 수 있는 방어기법이다.

본 논문의 구성은 다음과 같다. 2장에서는 정적 바이너리 재작성, 메모리 공유, 메모리 공유기반 부채널 공격에 대한 배경지식을 기술한다. 3장에서는 JMP+RAND 방어 기법과 문제점 해결 방안, 바이너리 난수 삽입 방법에 대해 기술하고 4장에서는 3장을 바탕으로 본 논문이 제시한 JMP+RAND 기법의 성능 평가와 기대효과를 기술한다. 마지막으로 5장에서는 결론을 기술한다.

2. 배경 지식

2.1 정적 바이너리 재작성

정적 바이너리 재작성은 소프트웨어 보안성을 증가시키기 위해 BOF, ROP 등의 공격 대상인 가젯(Gadget)을 변경하는 방법으로 정적 바이너리 계측(Static binary instrumentation), 바이너리 트램폴린(Binary trampoline)으로 불린다. 이러한 기법 대부분은 입력값으로 프로그램을 받아 보안성이 증가한 프로그램을 출력하는 프로그램으로 구현된다. 정적 바이너리 재작성은 프로그램의 바이너리를 수정하기 때문에 이미 컴파일러에 의해 정해진 재배치(Relocation), 심볼(Symbol) 정보들을 수정하지 않도록 주의해야 한다. 재배치, 심볼 정보들은 컴파일할 때 생성된 재배치 테이블(Relocation table)에 기록되어 있으므로 재배치, 심볼 정보의 오프셋(Offset), 데이터(Data) 등이 수정된다면 프로그램 실행 시 폴트(Fault)가 발생할 수 있다.

일반적으로 바이너리 재작성은 바이너리에 데이터와 코드를 삽입하고 기존의 코드를 패치하기 때문에 추가적인 .new_ text, .new_data 섹션이 필요하고 프로그램의 흐름을 유지하기 위해 제어 흐름(Control flow) 명령어(i.e., jmp)를 이용한다. 하지만 메모리 변조 공격의 대부분은 call 명령어처럼 재배치, 심볼 정보와 관련된 정보를 대상으로 공격하기 때문에 패치를 이용한 정적 바이너리 재작성은 적합하지 않다. 따라서 메모리 변조 공격을 방어하기 위해 재배치, 심볼 정보를 회복하며 정적 바이너리 재작성을 할 수 있는 UROBOROS[15, 16], Multiverse [17] 연구가 수행되었다. 또한 Spectre[10]와 같은 CPU 부채널을 방어하기 위한 방어기법으로 정적 바이너리 재작성을 사용한 oo7[18] 연구가 있다.

2.2 메모리 공유

메모리 공유는 프로세스 간에 메모리를 공유하는 기법이다. 사용자는 메모리 공유를 이용해 프로세스 간 통신도 가능하고 중복된 메모리에 대한 자원 낭비도 줄일 수 있다. 본 논문은 후자에 초점을 둔다. 메모리 공유 방법에는 content-aware sharing과 content-based sharing이 있다. Content-aware sharing은 프로그램이 디스크로부터 메모리에 적재될 때 공유 라이브러리, 코드 섹션처럼 같은 내용을 갖는 페이지를 공유하는 방법이며 운영체제에 의해 수행된다.

Content-based sharing은 메모리 중복제거(Memory deduplication)라고 불리며 하이퍼바이저에 의해 수행된다. 하이퍼바이저는 페이지 단위로 메모리를 스캔하면서 같은 페이지가 있으면 병합하여 하나의 공유 페이지로 만든다. 공유 페이지에 모든 사용자가 수정할 수 있으면 보안 문제가 생길 수 있으므로 병합된 페이지는 읽기 전용(Read-only) 권한을 가진다. 따라서 어떤 프로세스가 공유 페이지에 쓰기 접근(Write-access)을 시도할 때 페이지 폴트(Page fault)와 함께 CoW(Copy-on-Write) 메커니즘이 적용된다. 레드헷(Red Hat)은 KVM의 KSM(Kernel Samepage Merging), VMware는 ESXi의 TPS(Transparent Page Sharing), IBM의 PowerVM은 Active Memory Deduplication 이름으로 메모리 중복제거가 수행된다.

2.3 메모리 공유 기반 부채널 공격

캐시 부채널 공격 캐시 부채널 공격[2, 3]은 캐시 inclusive 정책과 경쟁 조건(Race condition)을 이용해 특정 캐시 메모리(i.e., Cache line, Cache set)를 관찰하는 공격이다. 캐시 inclusive 정책은 대부분의 CPU 제조사들이 선택한 속성이고 경쟁 조건은 메모리 공유를 이용해 발생시킬 수 있다. 공격자는 의도된 메모리 공유를 이용해 희생자와 공유한 캐시 메모리와 clflush 명령어로 인한 캐시 상태를 관찰하여 희생자가 특정 데이터에 접근했는지 접근하지 않았는지를 관찰할 수 있고 비밀키처럼 민감한 데이터를 유출할 수 있다[19].

[19]에서 비밀키를 얻는 데 사용한 Flush+Reload[2]는 캐시 부채널 공격 중 가장 정확한 공격이다. IaaS(Infrastructure as

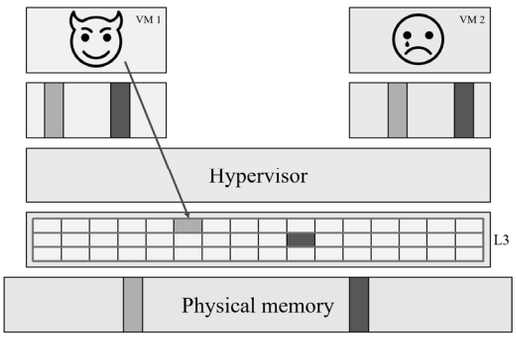

Fig. 1. Flush+Reload Attack Model

a Service)와 같은 클라우드 환경에서 서비스 제공자는 자원의 효율성을 위해 하이퍼바이저의 메모리 중복제거 기능을 사용한다. 메모리 중복제거 기능으로 공격자와 희생자가 갖는 같은 페이지들이 공유되거나 공격자는 희생자가 가지고 있을 것이라고 예상하는 페이지를 mmap() 함수를 이용해 메모리에 적재하여 메모리 공유를 유도할 수 있다. 또한 일반적인 서버 환경에서는 동일한 프로그램을 실행시킴으로써 메모리 공유를 할 수 있다. Flush+Reload는 이런 메모리 공유를 기반으로 캐시 라인을 공유해 희생자의 특정 활동을 관찰하거나 민감한 데이터를 유출할 수 있다. Intel 아키텍처의 대부분은 코어(Core)마다 L1, L2 캐시를 독립적으로 가지고 있고 L3 캐시를 공유하고 있다. 캐시 inclusive 정책과 특정 캐시 라인을 비우는 clflush 명령어를 이용해 메모리 공유된 페이지를 사용하는 희생자의 데이터 접근을 관찰할 수 있다.

가상화 환경에서 Flush+Reload 공격 모델은 Fig. 1과 같고 3가지 단계로 이루어진다. FLUSH: 공격자는 clflush 명령어를 통해 관찰할 L3 캐시 라인을 비운다(L3 캐시의 초록색 네모). 캐시 inclusive 정책에 의해 해당 캐시 라인과 관련된 L1, L2의 캐시 라인도 비워진다. WAIT: 정해진 시간 동안 희생자의 해당 캐시 라인 접근을 기다린다. RELOAD: 관찰한 캐시 라인을 로드하는 데 사용된 사이클 수를 측정한다. 만일 많은 사이클 수가 측정되었다면 희생자가 캐시 라인에 접근하지 않았음을 의미하고 적은 사이클 수가 측정되었다면 희생자가 캐시 라인에 접근했음을 의미한다. 이러한 메모리 로드 시간차를 부채널로 이용해 희생자의 데이터 접근을 관찰할 수 있다.

메모리 노출 공격 메모리 노출 공격[4-8]은 가상화 환경에서 하이퍼바이저가 제공하는 메모리 중복제거 기능을 이용해 공격자 VM이 희생자 VM의 메모리 레이아웃을 알아내는 공격이다. 공격자 VM은 희생자 VM과 특정 페이지를 공유하는 것을 목표로 하므로 희생자 VM이 가지고 있을 것이라고 예상하는 후보 페이지들을 mmap() 함수를 이용하여 메모리에 맵핑한다. 하이퍼바이저는 정기적으로 메모리를 스캔하면서 같은 페이지가 있으면 하나의 페이지로 병합하고 읽기 전용 권한을 부여한다. 따라서 중복 제거된 읽기 전용 페이지에 공격자 VM이 쓰기 접근을 시도할 때 페이지 폴트와 CoW 메커니즘으로 인해 일반 페이지에 비해 많은 사이클 수가 측정된다. 이러한 쓰기 접근을 시도할 때 발생하는 시간차를 부채널로 이용하여 희생자 VM의 메모리 레이아웃을 노출할 수 있다.

3. JMP+RAND 방어 기법과 문제점 해결

본 장에서는 기존의 메모리 보호 기법인 KASLR, ASLR을 보완하고 메모리 공유 기반 부채널 공격을 방어하는 JMP+RAND 기법과 바이너리 난수 삽입 시 발생하는 문제점 해결 방안에 대해 설명한다. JMP+RAND 기법은 두 가지 방법(i.e., 주입 기반, 패치 기반)으로 구현될 수 있다.

3.1 JMP+RAND 방어 기법

JMP+RAND는 페이지마다 5-8byte 난수를 삽입하여 메모리 공유 기반 부채널 공격을 방어하는 기법이다. 이 방어 기법은 페이지마다 크기가 고정되지 않은 난수를 삽입하기 때문에 .text 섹션의 N번째 페이지는 이전 N-1개의 페이지에 대해 의존성이 생겨 공격의 복잡성이 증가하는 특징을 가진다. 희생자 바이너리에 삽입된 난수의 크기는 공격자가 알아내야할 엔트로피가 되어 추가적인 공격 복잡도를 부여한다. 공격자는 바이너리에 삽입된 난수 값 때문에 페이지 내용이 달라져 희생자와 메모리 공유를 할 수 없고 메모리 공유 기반 부채널 공격을 시도하기 위해 무차별 대입 공격(Brute force attack)을 해야 한다.

3.2 바이너리 난수 삽입 문제점 해결

바이너리에 난수를 삽입할 때 두 가지 문제점이 존재한다. 첫 번째는 프로그램의 흐름에 영향을 주지 않고 난수를 삽입하는 방법이 매우 어렵다는 것이다. 예를 들어 .text 섹션에 난수가 삽입되면 CPU가 명령어를 페치(fetch)할 때 삽입된 난수를 명령어로 인식하고 페치하여 정상적으로 프로그램이 동작하지 않을 수 있다. 우리는 연속된 jmp 명령어와 난수로 이루어진 JMP+RAND를 제안해 첫 번째 문제를 해결했다. 두 번째 문제는 난수 삽입 시 변경되는 재배치, 심볼 정보로 프로그램의 흐름을 잃어버리는 문제이다. 본 논문은 3.3, 3.4절에서 제안하는 주입 기반의 난수 삽입, 패치 기반의 난수 삽입을 이용해 두 번째 문제를 해결한다.

3.3 주입 기반의 난수 삽입

주입 기반의 난수 삽입은 .text 섹션의 명령어와 명령어 사이에 JMP+RAND를 주입하는 방법이다. 이 방법은 JMP+RAND가 주입될 위치가 자유롭다는 장점이 있지만 재배치, 심볼 정보를 반드시 회복해야 하는 조건이 있다. Fig. 2A과 Fig. 2B를 비교하면 주입 기반의 난수 삽입을 더 분명하게 이해할 수 있다. JMP+RAND가 “mov %eax, %ebx” 명령어 이후에 주입된다면 Fig. 2B처럼 0x107번지에 jmp 명령어와 5byte의 난수가 삽입된다. 이때 jmp 명령어의 목적지 주소는 프로그램의 흐름을 유지하기 위해 난수를 건너뛴 바로 다음 명령어인 0x111번지를 가리킨다.

하지만 Fig. 2A와 Fig. 2B의 메모리 레이아웃을 비교하면 JMP+RAND가 삽입된 이후 명령어들은 모두 0xa만큼 번지수가 증가한 것을 확인할 수 있다. 이처럼 바이너리에 주입된 JMP+RAND로 인해 각종 심볼, 재배치 정보들의 오프셋 정보가 재배치 테이블과 일치하지 않아 폴트가 발생한다. 따라

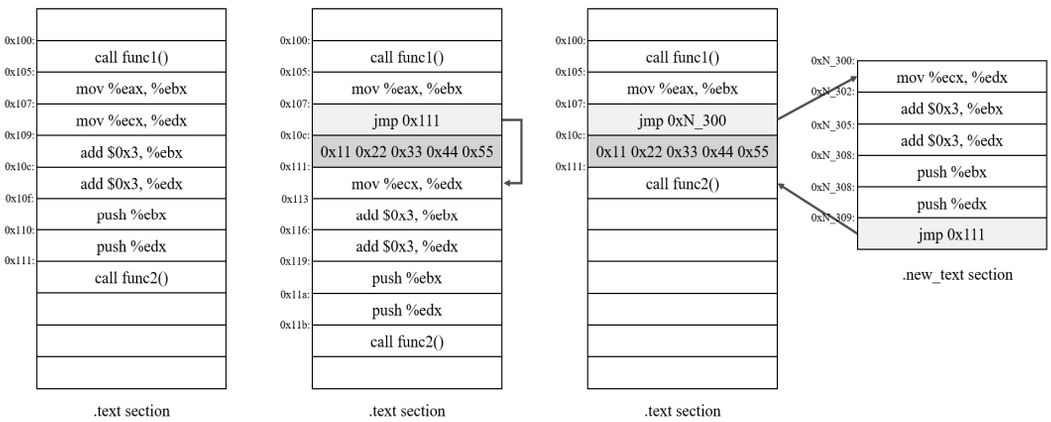

Fig. 2A. Origin Code Fig. 2B. Inject-based JMP+RAND Fig. 2C. Patch-based JMP+RAND

서 주입 기반의 난수 삽입 기법을 사용한다면 반드시 프로그램을 실행하기 전에 재배치와 심볼 정보 또는 재배치 테이블 정보를 증가한 오프셋에 맞게 수정해야 한다. jmp 명령어는 5byte의 길이를 가지고 5-8byte의 난수를 삽입하기 때문에 주입 기반의 난수 삽입 방법에서 4KB 페이지마다 10-13byte의 크기 증가와 한 번의 분기 명령어(i.e., jmp)가 추가된다.

3.4 패치 기반의 난수 삽입

주입 기반의 난수 삽입 방법은 반드시 재배치, 심볼 등의 정보를 회복해야 하는 단점이 있다. 이러한 정보 회복과 관련 없이 난수를 삽입하는 방법으로 패치 기반의 난수 삽입 방법을 제안한다. 패치 기반의 난수 삽입 방법은 .text 섹션의 페이지마다 10-13byte의 패치(Patch)될 후보 명령어들을 찾아 새로운 섹션으로 복사한 뒤 해당 명령어들을 JMP+RAND로 패치하는 기법이다. 패치 될 명령어는 재배치, 심볼 정보들과 관련 없이 정적으로 바이너리가 완성된 연속된 명령어들이다. 만일 페이지 내에 패치 될 명령어가 없다면 패치 기반의 난수 삽입 방법은 사용할 수 없다.

삽입될 난수의 크기가 5byte로 선정되었다면 재배치, 심볼 정보와 관련 없는 10byte의 연속된 명령어들을 찾아야 한다. Fig. 2A의 0x107-0x110번지까지의 명령어들이 패치될 명령어로 선정될 수 있고 Fig. 2C처럼 새로운 섹션(i.g., .new_text) 또는 .text 섹션의 맨 마지막 부분으로 복사될 수 있다. 복사될 때 반드시 jmp 명령어를 추가해 원래 프로그램의 흐름으로 돌아와야 한다. 이후 패치될 명령어는 JMP+RAND로 패치되어 난수를 삽입할 수 있다. Fig. 2A와 Fig. 2C를 비교하면 패치된 명령어를 제외한 나머지 명령어의 메모리 번지가 동일하기 때문에 패치 기반의 난수 삽입 방법에서는 패치될 명령어만 찾는다면 재배치 정보를 회복할 필요가 없다. 하지만 주입 기반의 난수 삽입 방법과 달리 2번의 jmp 명령어가 추가되어 페이지마다 15-18byte의 크기가 증가한다.

4. 성능 평가와 기대 효과

본 장에서는 JMP+RAND 기법이 메모리 공유 기반 부채널 공격의 방어기법으로써 적절한지에 대한 성능 평가와 기대 효과를 기술한다. 성능 평가는 페이지마다 삽입되는 난수 크기에 따른 프로그램 공격 시간을 수식과 그래프로 나타내었고 페이지마다 삽입되는 난수의 크기 변화로 페이지 의존성을 갖는 프로그램에 대한 공격 시간을 확률 분포를 이용해 평가한다. 본 장의 성능 평가에서는 명령어를 .text 섹션에 주입/패치하는 시간을 고려하기보다 삽입된 난수 크기에 의해 변화하는 공격 시간 복잡도에 대해 기술한다. 주입/패치 기반의 난수 삽입은 페이지마다 증가하는 크기가 다를 뿐 메커니즘은 같다. 따라서 주입/패치 기반의 방법의 차이를 고려하지 않고 충분한 성능 평가를 할 수 있다.

4.1 삽입되는 크기에 비례하는 시간 복잡도

JMP+RAND는 페이지마다 난수를 삽입하여 메모리 공유를 어렵게 만드는 방어 기법이다. 공격자는 바이너리에 삽입된 난수 때문에 희생자와 메모리 공유를 할 수 없고 메모리 공유 기반 부채널 공격을 시도하기 위해 무차별 대입 공격(Brute force attack)을 수행해야 한다. 우리는 삽입되는 난수 크기에 따라 변화하는 메모리 공유 기반 부채널 공격의 시간 복잡도를 계산하기 위해 다음의 실험을 진행했다. Gruss et al.[3]의 PoC 코드와 Kim et al.[8]을 바탕으로 Intel® CoreTM i5-7400 3.00GHz 프로세서와 16GB 메모리를 사용하는 환경에서 캐시 부채널 공격(i.e., Flush+Reload)을 수행하는데 약 0.03분, 클라우드 컴퓨팅 환경에서 메모리 노출 공격을 수행하는데 약 3분 정도 소비하는 것을 실험을 통해 알아냈다. 실험을 바탕으로 JMP+RAND가 적용된 바이너리를 공격하기 위해서 Fig. 3의 공격 시간 복잡도를 가지는 것을 확인했다.

Fig. 3은 N이 바이너리에 삽입되는 비트의 크기, Equation (1)로 얻는 T_{atk}가 메모리 공유기반 부채널 공격에 성공하는데 걸리는 시간이라고 할 때 얻는 그래프이다. N은 프로그램에 삽입될 비트의 크기이기 때문에 N이 증가할수록 프로그램의 크기도 증가한다. $S_{JMP+RAND}$를 JMP+RAND 기법이 적용된 프로그램의 텍스트 섹션 크기, S_{Origin}을 JMP+RAND 기법이 적용되지 않은 원본 프로그램의 텍스트 섹션 크기라고 할 때 Equation (2)를 이용해 $S_{JMP+RAND}$를 구할 수 있다. 따라서 너무 큰 난수 삽입은 메모리 자원의 오버헤드가 될 수 있다. 5-8byte의 난수 삽입은 공격자가 일반적인 서버, 클라

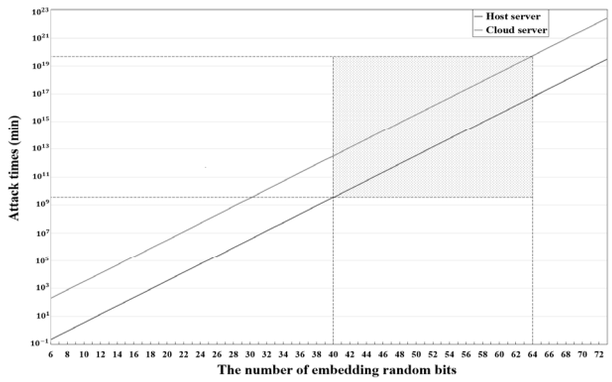

Fig. 3. Memory Sharing-based Side-channel Attack Time Complexity as to JMP+RAND

우드 컴퓨팅 환경을 공격하는데 최소 약 10^9분에서 최대 10^{19}분이 필요하고 바이너리의 텍스트 섹션 크기를 0.325% 증가시킨다(c.f., Fig. 3, Equation (1), (2)).

$$T_{atk} = \begin{cases} 2^{N\times 8}\times 0.03(m) \\ 2^{N\times 8}\times 3(m) \end{cases} \quad (1)$$

$$S_{JMP+RAND} = (1+\frac{5\times 8+N}{4000\times 8}\times 100)\times S_{Origin} \quad (2)$$

4.2 페이지 의존성에 비례하는 시간 복잡도

기존의 방어기법과 달리 JMP+RAND는 페이지마다 고정되지 않은 크기의 난수를 삽입함으로써 페이지들에 의존성을 부여한다. ΔK를 K번째 페이지에 삽입된 난수의 크기라고 정의할 때, 페이지들의 의존성을 Fig. 4처럼 나타낼 수 있다. JMP+RAND가 적용된 바이너리에 대해 메모리 공유 기반 부채널 공격을 시도하기 위해서 공격자는 K번째 페이지를 선택해야 한다. 공격자는 해당 바이너리의 변경 정보를 알 수 없으므로 K번째 페이지를 곧장 공격할 수 없고, 페이지마다 5-8byte의 난수가 삽입되었다는 정보를 바탕으로 1번째 페이지부터 Δ1을 알아내는 무차별 대입 공격을 수행해야 한다. 따라서 K번째 페이지를 공격하기 위해 Δ1-ΔK-1을 알아내는 공격이 K-1번 선행되어야 한다. 우리는 Equation (1)과 4.1절을 기반으로 페이지 의존성을 갖는 바이너리를 공격하는 데 필요로 하는 평균 공격 시간을 구할 수 있다. 페이지마다 5-8byte의 난수가 삽입되었을 때 캐시 부채널 공격과 메모리 노출 공격에 성공하는 데 필요한 시간을 확률변수를 각각 X_{SCA}, X_{MDA}라 하면 Table 1과 기댓값(m_{SCA}, m_{MDA})을 얻을 수 있다. 페이지마다 공격에 성공하기 위해 m_{SCA}, m_{MDA}

Table 1. Probability Distribution Table

X_{SCA}	$2^{5\times 8}\times 0.03(m)$	$2^{6\times 8}\times 0.03(m)$	$2^{7\times 8}\times 0.03(m)$	$2^{8\times 8}\times 0.03(m)$
X_{MDA}	$2^{5\times 8}\times 3(m)$	$2^{6\times 8}\times 3(m)$	$2^{7\times 8}\times 3(m)$	$2^{8\times 8}\times 3(m)$
P	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{1}{4}$

- $m_{SCA} = 2^{56}\times 0.03(m)$
- $m_{MDA} = 2^{56}\times 3(m)$

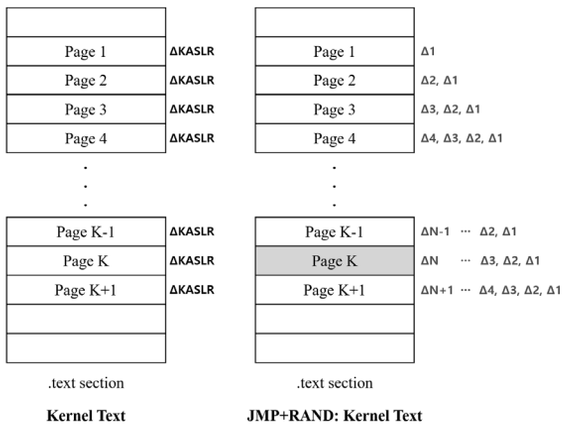

Fig. 4. The Dependency of Each Page

의 공격 시간이 필요하고 K번째 페이지에 대해 공격을 성공하기 위해 약 $K\times m_{SCA}$, $K\times m_{MDA}$의 시간이 필요하다. K가 증가할수록 공격 시간 복잡도가 증가하지만 첫 페이지는 K가 1이기 때문에 상대적으로 낮은 공격 시간 복잡도를 갖는 것은 문제가 될 수 있다. 이러한 문제점은 첫 페이지만 10byte의 난수를 삽입하여 극복할 수 있다. 10 byte의 난수 삽입으로 캐시 부채널 공격을 수행하는 데 최소 10^{24}분이 소비되고 이는 현실적으로 공격할 수 없는 시간이다.

4.3 기대 효과

커널 텍스트에 적용된 KASLR은 9bit, 스택에 적용된 ASLR은 19bit의 엔트로피를 갖는 기존의 메모리 보호 기법은 컴퓨터의 성능 향상으로 문제가 되고 있다. 낮은 엔트로피는 메모리 노출공격[4-8]에 취약할 수 있고, 캐시 부채널 공격[2, 3]에도 취약할 수 있다. JMP+RAND 기법은 낮은 오버헤드로 최소 40-64bit의 엔트로피를 간접적으로 증가시킬 수 있고, 메모리 공유를 기반으로 하는 부채널도 방어할 수 있다. 특히 가상화 환경에서 메모리 공유기반 부채널 공격 때문에 메모리 중복제거 기능을 사용하지 못한다. 하지만 JMP+RAND를 사용한다면 특정 프로그램에만 난수를 삽입할 수 있으므로 메모리 중복제거 기능을 사용하더라도 보안성을 보장받을 수 있을 것으로 예상한다.

5. 결 론

본 논문에서는 바이너리 난수 삽입을 이용한 메모리 공유 기반 부채널 공격 방어 기법인 JMP+RAND를 제안하였다. 기존의 메모리 보호기법을 보완하고 새로운 메모리 공유기반 부채널 공격에 대응하기 위해 정적 바이너리 재작성을 이용한 두 가지 방법을 설명했다. 재배치, 심볼 정보를 회복해야 하지만 명령어 삽입 공간에 제약이 없고 페이지마다 jmp 명령어를 한 번 사용하는 주입 기반의 난수 삽입을 제시했다. 비교적 구현이 쉽지만 패치될 명령어 선정에 제약이 있고, 페이지마다 jmp 명령어를 두 번 사용해야 하는 패치 기반의 난수 삽입 방법도 제안했다.

JMP+RAND 방어기법이 메모리 공유 기반 부채널 공격을 방어할 수 있는지 확인하기 위해 실험을 진행하였고 정량적으로 계산하여 현실적인 시간 내에 공격할 수 없음을 확인하였다. 더 구체적으로, JMP+RAND는 텍스트 섹션 크기를 약 0.325% 증가시키는 오버헤드를 갖고 성능 측면에서는 오버헤드가 거의 없지만 메모리 공유 기반 부채널 공격을 수행하기 위해 최소 10^{24}분의 시간이 소비됨을 확인했다. 특히 특정 프로그램에만 JMP+RAND를 적용할 수 있으므로 가상화 환경에서 메모리 중복제거 기능과 같이 사용되면 자원 효율성과 보안성 모두를 획득할 수 있다고 예상한다.

References

[1] Kernel Address Space Layout Randomization [Online], https:// lwn.net/Articles/569635/

[2] Yarom Yuval, and Katrina E. Falkner, Flush+ Reload: a High Resolution, Low Noise, L3 Cache Side-Channel Attack. USENIX Security, 2014.

[3] D. Gruss, C. Maurice, K. Wagner, and S. Mangard, "Flush+ Flush: A Fast and Stealthy Cache Attack," in DIMVA, 2016.

[4] D. Gruss, D. Bidner, and S. Mangard, "Practical Memory Deduplication Attacks in Sandboxed Javascript," In: Pernul G., Y A Ryan P., Weippl E. (eds) Computer Security ESORICS 2015.

[5] Kyniyasu Suzaki, Kengo lijima, Toshiki Yagi, and Cyrille Artho. Memory Deduplication as a Threat to the Guset OS, EUROSYS11, 2011.

[6] K. Suzaki, K. Iijima, Y. Toshiki, and C. Artho, "Implementation of a Memory Disclosure Attack on Memory Deduplication of Virtual Machines," *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences*, Vol.96, No.1, pp.215-224, 2013.

[7] Antonio Barresi, Kaveh Razavi, Mathias Payer, and Thomas R. Gross. "CAIN: Silently Breaking ASLR in the Cloud," 9th USENIX WOOT`15.

[8] Taehyun Kim, Taehun Kim, and Youngjoo Shin, "Breaking KASLR by using Memory Deduplication in Virtualized Environments," CISC-W`19.

[9] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown: Reading Kernel Memory from User Space," in USENIX Security Symposium (to appear), 2018.

[10] P. Kocher, J. Horn, A. Fogh, D. Genkin, G. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," In *S&P*, 2019.

[11] Michael Schwarz, Moritz Lipp, Daniel Moghimi, Jo Van Bulck, Julian Stecklina, Thomas Prescher, and Daniel Gruss. 2019. "ZombieLoad: Cross-PrivilegeBoundary Data Sampling," arXiv:1905.05726, 2019.

[12] B. Gulmezoglu, A. Moghimi, T. Eisenbarth, and B. Sunar, "FortuneTeller: Predicting Microarchitectural Attacks via Unsupervised Deep Learning, Cryptography and Security," 8 Jul. 2019.

[13] M. Chiappetta, E. Savas, and C. Yilmaz, "Real time detection of cache-based side-channel attacks using hardware performance counters," Cryptology ePrint Archive, 2015.

[14] M. Mushtaq, A. Akram, K. B. Muhammad. N. B. R. Rao, V. Lapotre, and G. Gogniat, "Run-time Detection of Prime+ Probe Side-Channel Attack on AES Encryption Algorithm," *2018 Global Information Infrastructure and Networking Symposium*, 23-25 Oct. 2018.

[15] Shuai Wang, Pei Wang, and Dinghao Wu. "Reassembleable disassembling. USENIX Security," 2015.

[16] Shuai Wang, Pei Wang, and Dinghao Wu. "UROBOROS: Instrumenting stripped binaries with static reassembling," IEEE 23rd SANER, 2016.

[17] Erick Bauman, Zhiqiang Lin, and Kevin W. Hamlen. Superset Disassembly: Statically Rewriting x86 Binaries Without Heuristics, NDSS, 2019.

[18] Guanhua Wang, Sudipta Chattopadhyay, Ivan Gotovchits, Tulika Mitra and Abhik Roychoudhury, "oo7: Low-overhead Defense against Spectre Attacks via Program Analysis," IEEE Transactions on Software Engineering, 2020.

[19] G. Irazoqui, M. S. Inci, T. Eisenbarth, and B. Sunar, "Wait a minute! a fast, cross-VM attack on AES," in RAID, Gothenburg, SE, pp.299-319. Sep. 2014.

김 태 훈

https://orcid.org/0000-0002-1887-7009
e-mail : taehunpb@gmail.com
2017년 ~ 현 재 광운대학교 컴퓨터정보공학부 학사과정
관심분야 : CPU, GPU Micro-Architecture Security, GPU Computing, OS Security

신 영 주

https://orcid.org/0000-0003-4831-7392
e-mail : yjshin@kw.ac.kr
2006년 고려대학교 컴퓨터학과(학사)
2008년 KAIST 전산학과(석사)
2014년 KAIST 전산학과(박사)
2008년 ~ 2017년 국가보안기술연구소 선임연구원
2017년 ~ 현 재 광운대학교 컴퓨터정보공학부 조교수
관심분야 : Applied Cryptography, CPU Micro-Architectural Security, Cloud Computing, SDN/NFV Security