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

김태훈, 신영주
광운대학교 컴퓨터정보공학부 정보 및 사이버보안 연구실
e-mail: milk8581@naver.com, yjshin@kw.ac.kr

JMP+RAND: Mitigating Memory Sharing-based Side-channel Attacks by Embedding Random Values in Binaries

Taehun Kim, Youngjoo Shin
Information and Cyber Security Lab, School of Computer and Information Engineering
Kwangwoon University

요 약

정보보안을 달성하기 위해서 컴퓨터가 보급된 이래로 많은 노력이 이루어졌다. 그중 메모리 보호 기법에 대한 연구가 가장 많이 이루어졌지만, 컴퓨터의 성능 향상으로 이전의 메모리 보호 기법들의 문제들이 발견되고, 부채널 공격의 등장으로 새로운 방어책이 필요 되었다. 본 논문에서는 프로그램에 정적 바이너리 재작성(Static Binary Rewriting) 기법을 통해 페이지(Page)마다 4~8byte 의 난수를 삽입하여 메모리 공유 기반 부채널 공격을 방어할 수 있는 2 가지 방법을 제시한다. 최근 아키텍처는 분기 예측(Branch Prediction)을 통해 jmp 명령어에 대한 분기처리가 매우 빠르고 정확하게 처리되기 때문에 난수를 삽입할 때 사용하는 jmp+rand 방식은 오버헤드가 매우 낮다. 또한 특정 프로그램에만 난수 삽입이 가능하므로 특히 클라우드 환경에서 중복제거 기능과 함께 사용하면 높은 효율성을 보일 수 있다고 예상한다.

1. 서론

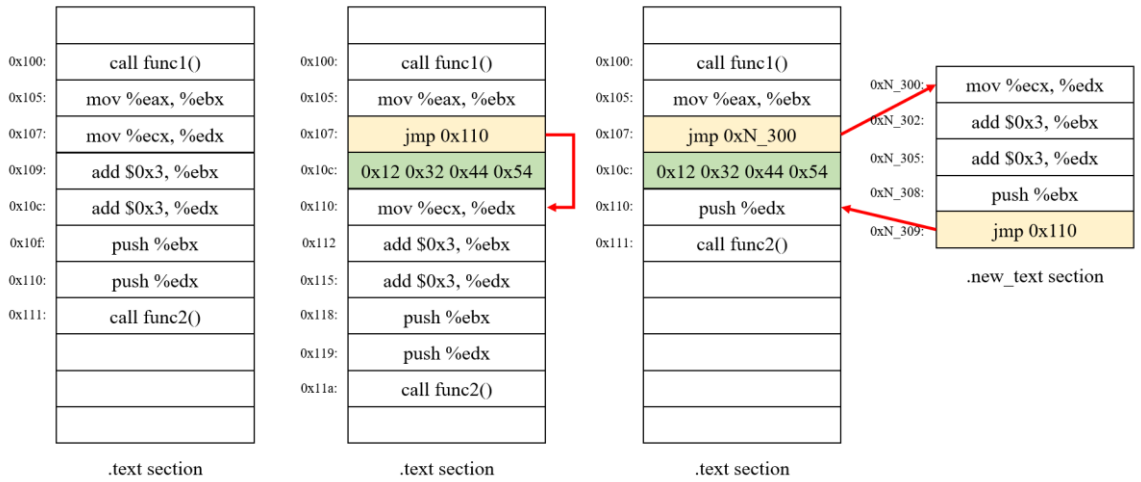
32-bit 컴퓨터가 보급된 30 년 동안 KASLR(Kernel Address Space Layout Randomization), ASLR, DEP(Data Execution Prevention)과 같은 메모리 보호 기법이 BOF(Buffer Overflow), ROP(Return Oriented Programming)의 변종을 방어하기 위해 생겨났다. 하지만 컴퓨터의 성능 향상으로 KASLR, ASLR 의 엔트로피(Entropy) 부족 문제가 생겼고 최근 부채널 공격이 대두되면서 새로운 방어책이 필요하다. 특히 클라우드 환경에서 FLUSH+RELOAD[1], CAIN[2] 등의 메모리 공유 기반 부채널 공격을 방어하기 위해 메모리 중복 제거 기능을 사용하지 못하여 높은 오버헤드가 발생하고 있다. 따라서 기존의 메모리 보호 기법을 보완할 뿐만 아니라 부채널 공격도 효과적으로 방어하는 방법을 제시하려 한다. JMP+RAND 는 정적 바이너리 재작성 기법을 이용해 .text 섹션의 4KB 페이지마다 4~8byte 의 난수를 삽입하여 간접적으로 엔트로피를 32bit 이상 높인다. 페이지마다 삽입될 난수 바이트의 크기는 고정적이지 않기 때문에 .text 섹션의 N 번째 페이지의 내용을 알기 위해서 1~N-1 번째 페이지 각각 몇 바이트의 난수가 삽입되었는지를 반드시 알아야 한다. 또한 프로그램을 입력 값으로 받아 난수가 삽입된 프로그램을 출력하기 때문에 원하는 프로그램만 엔트로피를 높여줄 수 있다. 즉, 클라우드 환경에서 메모리 중복 제거 기술을 사용하더라도 효과적으로 방어할 수 있다.

본 논문의 구성은 다음과 같다. 2 장에서는 정적 바이너리 재작성, 메모리 공유, 메모리 공유기반 부채널 공격에 대한 배경지식을 설명한다. 3 장에서는 바이너리 난수 삽입 방법에 대해서 주입 기반, 패치 기반 두 가지로 제시하고 기대효과에 대해 말한다. 마지막으로 4 장에서는 결론 및 향후 계획에 대해 기술한다.

2. 배경지식

2.1 정적 바이너리 재작성

정적 바이너리 재작성은 소프트웨어 보안성을 증가시키기 위해 ROP, BOF 등의 공격 대상인 가젯(Gadget)을 변형시키는 방법이다. 주로 입력 값으로 프로그램을 받아 보안성이 증가한 프로그램을 출력하는 형태로 구현된다. 정적 바이너리 재작성은 프로그램의 바이너리를 수정하기 때문에 이미 컴파일러에 의해 정해진 재배치(Relocation), 심볼(Symbol) 정보들을 주의해야 한다. 이런 정보들을 피하는 바이너리 재작성은 가젯을 변형시키는 데 한계가 있기 때문에 최근 재배치, 심볼 정보를 회복하며 바이너리 재배치를 할 수 있는 UROBOROS[3,4], Multiverse[5] 연구가 수행되고 있다.



(그림 1) origin code (그림 2) inject-based rewriting (그림 3) patch-based rewriting

2.2 메모리 공유

메모리 공유는 프로세스 간에 메모리를 공유하는 기법이다. 메모리 공유를 통해 프로세스 간 통신도 가능하고 중복된 메모리에 대한 자원 낭비도 줄 일 수 있다. 메모리 공유 방법에는 content-aware sharing 과 content-based sharing 이 있다. Content-aware sharing 은 프로세스들이 여러 기능을 수행할 때 공유 라이브러리처럼 같은 내용을 갖는 페이지를 공유하는 방법이며 운영체제에 의해 수행된다.

Content-based sharing 은 메모리 중복제거(memory deduplication)라고 불리며 하이퍼바이저에 의해 수행된다. 하이퍼바이저가 페이지 단위로 메모리를 스캔하면서 같은 페이지가 있으면 병합하여 하나의 공유 페이지로 만든다. 공유 페이지에 아무나 수정할 수 있으면 보안 문제가 생길 수 있기 때문에 병합된 페이지는 read-only 권한을 가진다. 따라서 어떤 프로세스가 공유 페이지에 write-access 시 page-fault 와 함께 COW(Copy-On-Write) 메커니즘이 적용된다.

2.3 메모리 공유기반 부채널 공격

FLUSH+RELOAD[1]는 메모리 공유기반 부채널 공격 중에서 가장 정확한 공격이다. IaaS(Infrastructure as a Service)와 같은 클라우드 환경에서 서비스 제공자는 자원의 효율성을 위해 하이퍼바이저의 중복 제거 기능을 킨다. 중복제거 기능을 통해 공격자와 희생자의 같은 페이지들이 공유되고, 공격자는 희생자가 가지고 있을 것이라고 예상하는 페이지를 mmap() 함수를 통해 메모리에 적재하여 의도적인 메모리 공유를 유도할 수 있다. FLUSH+RELOAD 는 이런 메모리 공유를 기반으로 하여 캐시 라인을 공유해 특정 활동을 확인할 수 있는 캐시 부채널 공격이다. Intel 아키텍처의 대부분은 코어(Core)마다 L1, L2 캐시를 독립적으로 가지고 있고 L3 캐시를 공유하고 있다. 캐시의 inclusive 정책과 캐시 라인을 비우는 clflush 명령어를 이용해 메모리 공유된 페이지를 사용하는 희생자의 캐시 라인 사용을 확인할 수 있다. FLUSH+RELOAD 공격은 3 가지 단계로 나뉜다. FLUSH: 공격자는 clflush 명령을 통해 관찰할 L3 캐시 라인을 비운다. 캐시의 inclusive 정책에 의해 해당 캐시 라인과 관련된 L1, L2 의 캐시 라인도 비워진다. WAIT: 희생자가 해당 캐시 라인에 접근할 때까지 기다린다. RELOAD: 해당 캐시 라인을 다시 접근해서 메모리 로드 속도가 빠르면 희생자가 접근한 것이고 메모리 로드 속도가 느리면 희생자가 접근하지 않았음을 알 수 있다.

3. 바이너리 난수 삽입 방법과 기대효과

기존의 메모리 보호 기법인 KASLR, ALSR 을 보완하고 부채널 공격을 막기 위해 프로그램의 .text 섹션에 4KB 페이지마다 4~8byte 의 난수를 삽입한다. 각 페이지마다 삽입될 바이트 수는 일정하지 않기 때문에 .text 섹션의 N 번째 페이지는 N-1 개의 페이지에 대해 의존성이 생긴다. 바이너리에 난수를 삽입하는 방법은 JMP+RAND 방식을 사용하였다.

3.1 주입 기반의 난수 삽입

주입 기반의 난수 삽입은 프로그램을 디스어셈블리(Disassembly) 도구를 사용해 .text 섹션의 명령어와 명령어 사이에 JMP+RAND 를 주입하는 방식이다. (그림 1)과 (그림 2)를 비교하면 차이를 더 분명하게 알 수 있다. 명령어들 사이에 명령어가 주입되기 때문에 주입된 명령어 이후로 재배치와 심볼 정보들의 오프셋이 바뀐다. 따라서 주입 기반의 난수 삽입 기법을 사용한다면 반드시 프로그램을 실행하기 전에 재배치와 심볼 정보를 바뀐 오프셋에 맞춰 수정해야 한다. jmp 명령어는 5byte 의 길이를 가지고 난수를 4byte 이상 삽입하기 때문에 4KB 페이지마다 9~13byte 의 크기증가와 한 번의 분기 명령이 추가된다.

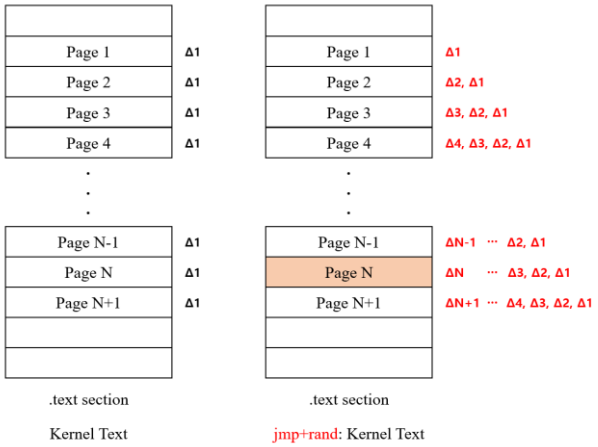
3.2 패치 기반의 난수 삽입

주입 기반의 난수 삽입 방법은 반드시 재배치, 심볼 등의 정보를 회복해야 한다. 이런 정보 회복과 관련 없이 난수를 삽입하는 방법이 패치 기반의 난수 삽입 방법이다. (그림 3)과 같이 .text 섹션에서 페이지마다 패치(Patch)될 명령어들을 찾는다. 패치 될 명령

어는 재배치와 심볼과 같은 정보들과 관련이 없어 정적으로 바이너리가 완성된 명령어들이 후보가 된다. 삽입될 난수의 크기에 따라 9~13byte 의 명령어를 찾아 JMP+RAND 로 패치 후 선정된 명령어들을 .text 섹션의 맨 마지막 부분이나 새로운 섹션에 복사한 후 마지막에 jmp 명령어를 통해 원래의 코드 흐름으로 돌아오는 형태이다. (그림 1)과 (그림 3)을 비교하면 패치된 명령어를 제외한 나머지 명령어의 주소 참고가 동일한 것을 확인할 수 있다. 페이지마다 2 번의 jmp 명령어가 추가되지만, 현대의 아키텍처에서 jmp 명령어를 통한 오버헤드는 거의 없을 것으로 예상한다.

3.3 기대효과

커널 텍스트 KASLR 는 9bit, 스택 ASLR 은 19bit 의 엔트로피를 갖는 기존의 메모리 보호 기법은 컴퓨터의 성능 향상으로 문제가 되고 있다. 낮은 엔트로피는 memory disclosure 공격[6]에 취약할 수 있고, 메모리 공유를 기반으로 하는 부채널 공격에도 취약할 수 있다. JMP+RAND 기법을 통해 낮은 오버헤드로 최소 32~64bit 의 엔트로피를 간접적으로 증가시킬 수 있고, 메모리 공유를 기반으로 하는 부채널도 방어할 수 있다. 또한 기존의 KASLR, ASLR 은 페이지마다 의존성이 존재하지 않는다. 특정 N 번째 페이지를 공격자가 중복제거를 통해 공격하고 싶다면 엔트로피에 맞춰 해당 페이지만 준비하면 되었다. 하지만 JMP+RAND 방식은 삽입되는 난수 값을 고정하지 않아 N 번째



(그림 4) 각 페이지들의 의존성

페이지를 공격하고 싶으면 (그림 4)와 같이 N-1 개의 의존성이 생긴다. 따라서 실제로 엔트로피는 32~64bit 보다 훨씬 클 것으로 예상한다. 특히 가상화 환경에서 메모리 중복 제거 기술과 함께 사용하면 높은 자원 효율성과 함께 커널과 보안을 필요로 하는 프로그램에 선택적으로 보안성을 증가할 수 있을 것으로 예상한다.

4. 결론 및 향후 계획

주입 기반의 난수 삽입, 패치 기반의 난수 삽입 방법을 정적 바이너리 재작성 기법을 토대로 구현해 실제로 낮은 오버헤드로 기존 프로그램의 의미를 갖춰 실행할 수 있는지 확인할 예정이다. Intel 과 같은 CISC 아키텍처는 명령어의 크기가 고정되지 않기 때문에 삽입될 난수 설정에 제약을 받지 않는데 ARM 과 같은 RISC 아키텍처는 명령어 길이가 정해져 있다. 따라서 본 논문에서 제시한 방법이 제대로 적용되지 않을 수 있으므로 아키텍처에 상관없이 적용할 수 있는 새로운 방안을 찾으려 한다.

본 논문에서는 바이너리 난수 삽입을 통한 메모리 공유 기반 부채널 공격 방어 기법에 대하여 제시하였다. 기존의 메모리 보호기법을 보완하고 새로운 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 제시했다.

하지만 아직 이론적인 부분이고 아키텍처에 의존적이기 때문에 아키텍처에 독립적으로 적용할 수 있는 방법을 찾아 실제로 구현하고, 여러 벤치마킹을 통해 프로그램의 성능을 측정하는 연구를 진행하려 한다.

Acknowledgement

이 논문은 2019 년도 정부(과학기술정보통신부)의 재원으로 정보통신기술진흥센터의 지원을 받아 수행된 연구임(2019-0-00533, 컴퓨터 프로세서의 구조적 보안 취약점 검증 및 공격 탐지 대응). 본 연구는 과학기술정보통신부 및 정보통신기술진흥센터의 SW 중심대학 지원사업의 연구결과로 수행되었음(2017-0-00096).

참고문헌

[1] Yarom Yuval, and Katrina E. Falkner. Flush+ Reload: a High Resolution, Low Noise, L3 Cache Side-Channel Attack. USENIX Security, 2014.
[2] Antonio Barresi, Kaveh Razavi, Mathias Payer, and Thomas R. Gross. CAIN: Silently Breaking ASLR in the Cloud. 9th USENIX WOOT`15.
[3] Shuai Wang, Pei Wang, and Dinghao Wu. Reassembleable disassembling. USENIX Security, 2015
[4] Shuai Wang, Pei Wang, and Dinghao Wu. UROBOROS: Instrumenting stripped binaries with static reassembling. IEEE 23rd SANER, 2016
[5] Erick Bauman, Zhiqiang Lin and Kevin W. Hamlen. Superset Disassembly: Statically Rewriting x86 Binaries Without Heuristics, NDSS. 2019
[6] Kyniyasu Suzaki, Kengo Iijima, Toshiki Yagi, and Cyrille Artho. Memory Deduplication as a Threat to the Guset OS, EUROSYS11. 2011