공유 메모리에의 최소 특권의 원칙 적용 기법에 대한 연구

유준승 [1], 방인영 [1], 백윤흥 [2]
[1] 서울대학교 전기정보공학과 박사과정, 반도체공동연구소
[2] 서울대학교 전기정보공학과 교수, 반도체공동연구소

jsyou@sor.snu.ac.kr, iybang@sor.snu.ac.kr, ypaek@sor.snu.ac.kr

A Study on Enforcing Principle of Least Privilege on Shared Memory

Jun-Seung You[1], Bang-In Young[1], Yun-Heung Paek[1]
[1]Dept. of Electrical and Computer Engineering and Inter-University Semiconductor Research Center (ISRC), Seoul National University

요 약

현대 시스템의 크기와 복잡도 증가로 인하여 시스템의 여러 구성 요소들이 공유하는 메모리에 대한 최소 특권의 원칙 적용의 필요성이 대두되었다. 제 3 자 라이브러리, 다중 쓰레드 등의 각 구성 요소들이 접근할 수 있는 메모리 권한을 다르게 적용함으로써 구성 요소들 중 하나에서의 취약점이 전체 시스템을 위협하는 것을 방지함과 동시에 각 요소들 간 효율적인 메모리 공유를 가능케 하기 때문이다. 본 논문에서는 공유 메모리에 대한 최소 특권의 원칙 적용 기법들의 분석과 더불어 각 기법들이 가지는 한계점을 제시한다.

1. 서론

다양한 하위 구성 요소들로 이루어진 현대 시스템의 효율적인 실행을 위해 공유 메모리는 필수불가결의 요소이며. 이에 따라 서로 다른 구성 요소들이 공통적으로 접근할 수 있는 공유 메모리가 적극적으로 사용되고 있다. 예를 들어, 커널과 드라이버는 효율적인 통신을 위하여 대부분의 버퍼와 디바이스 구조체를 공유하며, 많은 어플리케이션들은 제 3 자 라이브러리와의 데이터 공유를 통해 라이브러리들이 제공하는 함수들을 사용한다.

하지만 공유 메모리는 성능적으로 큰 장점을 가짐에도 불구하고, 필연적으로 보안적인 문제점을 동반한다. 여러 시스템 구성 요소들이 동시에 접근할 수 있는 메모리를 이용해 하나의 구성 요소가 가지는 취약점 또는 악의적인 구성요소가 전체 시스템을 공격 및 위협할 수 있기 때문이다. 이러한 문제점은 다양한 커널 드라이버를 이용한 커널 공격 [1], 제 3 자 라이브러리 취약점을 통한 개인정보 유출 [2] 등을 통해 지속적으로 확인되고 있다. 이에 따라 최소 특권의 원칙 적용을 위한 다양한 연구가 진행되어 왔다. 즉, 시스템 메모리에 대해 각 구성 요소들의 접근 권한을 다르게 부여함으로써 하나의 구성 요소에서의 취약점이 전체 시스템으로 전파되는 것을 방지하고자 하는 것이다.

본 논문에서는 현대 시스템 보호에 있어 중요한 요소 중 하나인 최소 특권의 원칙 기법들을 분석한다. 이와 함께 각 기법들의 공유 메모리에의 적용에 있어서의 한계점을 살펴본다.

2. 배경이론

최소 특권의 원칙은 현대 컴퓨터 시스템과 함께 발전되어 왔다. 허나, 초기 연구들은 공유 메모리에 대해 여러 구성 요소들이 다양한 접근 권한(read-only, read-write, not-accessible)을 가지는 것에 초점을 맞추기보다는 각 구성 요소가 접근할 수 있는 메모리를 격리하는 것에 집중되어 있다.

2.1. 프로세스 격리 기법

프로세스 격리는 시초의 운영체제가 가지는 원칙을 직접적으로 적용한 기법이다. 즉, 각 시스템 구성 요소들을 다른 프로세스에 위치시킴을 통해 구성 요소들의 서로 다른 주소 공간을 가지게 되며, 자연스럽게 요소 간 격리가 적용된다 [3]. 안타깝게도 해당 기법은 구성 요소간 데이터 전달 및 공유를 위하여 프로세스 간 통신 (Inter Process Communication, IPC)를 필

연적으로 동반하며, IPC 가 야기하는 큰 성능 오버헤드 때문에 데이터 공유가 자주 일어나는 현대 시스템에는 적합하지 못하다.

2.2. Software Fault Isolation (SFI)

SFI 기법은 소프트웨어적 변화를 통해 구성 요소들이 접근할 수 있는 메모리를 제어한다. 공통적으로 메모리 접근 명령어마다 추가 명령어 삽입을 통해 허용된 메모리 주소로만 접근하게끔 한다. 대표적인 SFI 기술 중 하나인 Native Client(NaCl)[4]는 마스킹을 통해 접근 가능한 주소 공간을 4GB 로 제한하며, 최근 각광받고 있는 WebAssembly(wasm)[5]은 접근 가능한 주소 하한과 상한을 설정하여 비교한다. 하지만 SFI 기법은 2.1 에서 언급되었듯이 접근 가능한 메모리를 제한하는 것에 초점이 맞춰져 있기 때문에 구성 요소 간의 데이터 전달 및 공유를 위해서는 메모리 복사를 필요로 하며, 이는 큰 오버헤드를 야기한다.

3. 공유 메모리에의 최소 특권의 원칙 적용 기법

2 장에서 설명된 메모리 격리에 초점이 맞춰져 있던 초기 기법들과는 달리, 최소 특권의 원칙 적용 기법들은 점차 동시에 다양한 접근 권한을 제공할 수 있게 끔 발전되었으며, 자연스럽게 공유 메모리 적용에 더 적합하게 되었다.

3.1. 페이지 테이블 기반 기법

페이지 테이블은 기본적으로 페이지를 접근할 수 있는 권한 (ro, rw, na) 비트들을 가지고 있다. 페이지 테이블 기반 기법은 해당 비트들을 변경하는 방식으로 구현된다. 즉, 기본적인 기법은 구성 요소 A 에서 B 로 넘어갈 때 A 에서 rw 였던 메모리 페이지를 ro 로 변경함으로써 B 가 권한 비트가 변경된 페이지에 대해 읽기 권한 밖에 가지지 못하게 하는 메커니즘을 지닌다. 하지만 이런 기본적인 기법은 페이지 접근 권한 비트를 변경하기 위한 시스템콜(mprotect)의 오버헤드가 너무 크다. 다시 말해, 매번 구성 요소 간 데이터 공유 및 전달이 있을 때마다 시스템콜을 호출해야 되기 때문에 현실적으로 사용되기 어렵다. 이를 해결하기 위해 크게 두 가지 방향의 연구가 진행되었다.

첫 번째 연구 방향[6]은 하나의 프로세스 내에 각 구성 요소 별로 다른 페이지 테이블을 가지는 기법을 제시하였다. 이와 함께, 구성 요소가 가지는 페이지 테이블은 유저스페이스(userspace)에서 변경 가능하게끔 하여 권한 변경이 값비싼 시스템콜이 아닌 단순한 라이브러리 API 호출로 되게끔 하였다. 하지만, 해당 기법은 구성 요소를 쓰레드로 한정했다는 한계가 있다. 즉, 요소 별 페이지 테이블 관리를 용이하게끔 하기 위하여 요소의 단위를 쓰레드로 한정했고, 이에 따라 보다 보편적인, 혹은 쓰레드가 최소 실행 단위가 아닌 요소들 (커널 드라이버, 제 3 자 라이브러리) 등에 대한 지원은 할 수 없다는 한계가 있다.

두 번째 연구 방향[7]은 페이지 테이블에 더블 매핑 기법 적용을 통해 최소 특권의 원칙을 적용한다. 페이지 테이블 접근 권한 비트를 사용하는 것은 공통적이나, 쓰레드 별 페이지 테이블을 가지는 첫 번째 방향과 달리 각 구성 요소가 동일한 메모리에 대해 서로 다른 가상 주소 페이지를 지니고, 각 페이지는 동일한 물리메모리에 매핑 된다. 즉, 하나의 페이지 테이블을 유지하면서 동일한 물리메모리에 대해 다른 접근 권한을 구현한다. 하지만, 해당 기법은 첫 번째 기법과 마찬가지고 구성 요소의 단위를 쓰레드로 한정하며, 각 구성 요소들이 동일한 메모리에 대해 서로 다른 주소를 가지게 된다는 단점이 있다. 이에 따라 메모리 공유 및 전달에 있어서 페이지 테이블을 공유하고 있음에도 불구하고 주소 변환 과정을 필요로 한다.

이와 더불어 두 연구 방향 공통적으로, 페이지 테이블 기반 기법은 접근 권한 제어의 최소 단위가 페이지라는 단점을 지닌다. 즉, 페이지 크기 이하의 메모리에 대한 접근 권한 설정은 불가하다. 이에 따라 공유 메모리가 작을 때에도 페이지 하나를 모두 할당해야 되어 메모리 오버헤드가 커질 수 있다는 단점이 있다.

3.2. 소프트웨어 기반 기법

소프트웨어 기반 기법은 2.2 에서 설명된 SFI 기법을 발전시켜 보다 공유메모리에 적합하게 제안된 기법이다. 즉, 소프트웨어적 변화 및 명령어 추가를 통해 메모리 접근의 권한을 제어 및 확인하는 것은 SFI 와 동일하다. 하지만 단순히 메모리 접근을 특정 메모리 영역에 제한하는 기존 SFI 기법들과 달리, 공유 메모리에 적합하게 고안된 기법들[8, 9, 10]은 공통적으로 그림 1 과 같은 access control list (ACL)이라는 자료구조를 사용하여 최소 특권의 원칙을 적용한다.

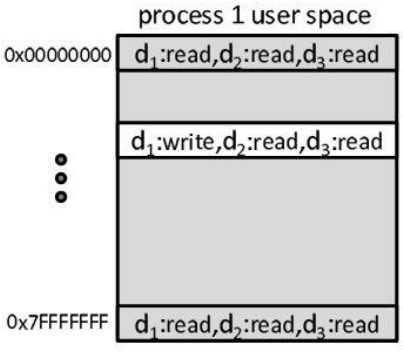

그림 1. Access Control List (ACL) 구조

다시 말해, 시스템 구성 요소들이 사용하는 전체 주소 공간에 대해 각 구성 요소 (그림 1 에서는 d_1, d_2, …로 표현됨) 들의 접근 권한을 지니는 자료구조 (테이블) 관리한다. 해당 테이블은 메모리 접근 명령어마다 추가 명령어 삽입을 통해 lookup 된다.

소프트웨어 기반 기법은 페이지 기반 기법과 달리 최소 바이트 단위로 접근 제한이 가능하다는 장점을 지니지만, 고질적인 문제는 성능이 느리다는 점이다. 명령어 단위로 매 메모리 접근마다 ACL 을 읽고 접근 권한을 비교해야 되기 때문이다. 이 때문에 해당 연구들은 보통 쓰기(write)에 대한 접근 권한만을 확인하는 방식으로 구현되어 있다.

3.3. 하드웨어 보조 기반 기법

3.1 및 3.2 에서 설명된 기법들 둘 다 본질적으로는 특정 소프트웨어적 자료구조 (3.1 – 페이지 테이블, 3.2 – ACL) 변경을 통해 최소 특권의 원칙을 실현하기 때문에 그 성능에 한계가 있다. 이를 극복하기 위해 최근 출시된 새로운 하드웨어 기능들을 사용한 기법들이 제시되었다.

해당 연구들이 기용한 하드웨어 기능은 근본적으로 페이지에 대한 접근 권한을 페이지 테이블보다 더 빠르게 바꿀 수 있는 기능이다. 즉, 페이지마다 하나의 키(key)를 할당하고, 해당 키를 가지고 있는 페이지에 대한 접근 권한을 간단하게 유저스페이스 레지스터 수정을 통해 수정할 수 있다. 즉, 구성 요소 A 와 B 가 공유하고자 하는 메모리 페이지에 키 1 번을 할당하고, A 에서 B 로 넘어갈 때 3.1 처럼 시스템콜이나 API 호출이 아닌 단일 명령어를 통한 레지스터에 키 1 에 대한 접근 권한을 수정함으로써 최소 특권의 원칙을 적용한다. 이러한 기능을 제공하는 하드웨어 기능은 대표적으로 인텔의 Memory Protection Keys (MPK)와 ARM 의 ARM Memory Domains (MD)가 있다.

하지만 해당 기법들은 기본적으로 3.1 의 페이지 테이블 기반 기법의 연장선인 만큼 페이지를 접근 제한의 기본 단위로 한다는 단점을 공유한다. 즉, 공유 및 전달하는 데이터의 크기가 작음에도 불구하고 페이지 하나를 할당해야 하기 때문에 메모리 오버헤드가 발생한다. 이와 더불어 사용할 수 있는 키의 수가 제한되어 있기 때문에, 많은 수의 구성 요소를 동시에 지원하지 못한다. 뿐만 아니라, 하드웨어 기능 특성 상, 해당 기능을 지원하지 않는 시스템에서는 이러한 기법들을 사용하지 못한다는 단점을 지닌다.

4. 결론

본 논문에서는 공유 메모리에의 최소 특권의 원칙 적용을 위한 기법들을 분석하고, 이들에 대한 한계점을 살펴보았다. 페이지 테이블 기반 기법들은 접근 제어 최소 단위가 페이지이고 구성 요소 최소 단위가 쓰레드라는 한계점이 있었으며, 소프트웨어 기반 기법은 자료구조 접근 및 비교에 의해 성능이 느리다는 한계점이 존재했다. 마지막으로 하드웨어 보조 기반 기법은 페이지 단위로 접근 제어가 된다는 점과 특정 하드웨어 기능이 없으면 사용할 수 없다는 한계점이 있었다. 공유메모리에 최소 특권의 원칙 적용은 현대 시스템 구성 상 지속적으로 대두될 문제점인만큼 기존 기법들의 한계점을 보완하는 새로운 연구가 계속 되어야 할 것이다.

4. Acknowledgement

이 논문은 2023 년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원과 (No.2020-0-01840, 스마트폰의 내부데이터 접근 및 보호 기술 분석, No.2021-0-00528, 하드웨어 중심 신뢰계산기반과 분산 데이터보호박스를 위한 표준 프로토콜 개발) BK21 FOUR 정보기술 미래인재 교육연구단에 의하여 지원된 연구임.

참고문헌

[1] CVE Details. Vulnerabilities in the Linux kernel by year. https://www.cvedetails.com/product/47/Linux-Linux-Kernel.html?vendor_id=33

[2] Github: Tunz. Case Study of JavaScript Engine Vulnerabilities. https://github.com/tunz/js-vuln-db.

[3] BITTAU, A., MARCHENKO, P., HANDLEY, M., AND KARP, B. Wedge: splitting applications into reduced privilege compartments. In Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation (Berkeley, CA, USA, 2008), USENIX Association, pp.309–322.

[4] Yee, Bennet, et al. "Native client: A sandbox for portable, untrusted x86 native code." *Communications of the ACM* 53.1 pp.91-99, 2010

[5] . Haas, Andreas, Andreas Rossberg, Derek L. Schuff, Ben L. Titzer, Michael Holman, Dan Gohman, Luke Wagner, Alon Zakai, and J. F. Bastien. "Bringing the web up to speed with WebAssembly." In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2017, pp. 185-200.

[6] Hsu, Terry Ching-Hsiang, et al. "Enforcing least privilege memory views for multithreaded applications." *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 2016, pp. 393-405

[7] Wang, Jun, Xi Xiong, and Peng Liu. "Between Mutual Trust and Mutual Distrust: Practical Fine-grained Privilege Separation in Multithreaded Applications." *USENIX Annual Technical Conference*. 2015, pp. 361-373

[8] Witchel, Emmett, Junghwan Rhee, and Krste Asanović. "Mondrix: Memory isolation for Linux using Mondriaan memory protection." *Proceedings of the twentieth ACM symposium on Operating systems principles*. 2005, pp. 31-44.

[9] Mao, Yandong, et al. "Software fault isolation with API integrity and multi-principal modules." *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*. 2011, pp. 115-128.

[10] Castro, Miguel, et al. "Fast byte-granularity software fault isolation." *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*. 2009, pp. 45-58.