Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.33, NO.2, Apr. 2023 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2023.33.2.319

ARM PA를 통한 경량화된 파일 디스크립터 권한 관리 시스템*

조 규 원,[1†] 이 호 준[2‡]
[1,2]성균관대학교 (대학원생, 교수)

Lightweight Capability-Based Access Control System on File Descriptor via ARM PA*

Kyuwon Cho,[1†] Hojoon Lee[2‡]
[1,2]SungKyunKwan University (Graduate student, Professor)

요 약

프로세스 내부 격리(Intra-process Isolation)에서 파일 디스크립터는 메모리 이외에 또 다른 공격 벡터로써 작용한다. 공격자는 파일 디스크립터를 변조함으로 주어진 권한 이상의 파일에 읽거나 쓰는 방식을 통해 격리된 환경을 벗어날 수 있다. 본 논문에서는 ARM 아키텍처의 하드웨어 보안 기술인 PA(Pointer Authentication) 기반의 경량화된 새로운 파일 디스크립터의 권한 관리 시스템을 제시한다. 우리의 무결성 보장 기법은 리눅스 커널 모듈의 형태로 제작되었으며 5% 정도의 오버헤드만으로 파일 디스크립터에 대한 권한 관리를 수행하였다.

ABSTRACT

In intra-process isolation, file descriptors work as another attack vector from the memory corruption attacks. The attacker can read or write by corrupting file descriptors so they can escape the isolation. In this paper, we propose new lightweight capability-based access control system on file descriptor using ARM's hardware extension, PA(Pointer Authentication). Our system was implemented on Linux kernel module, only shows 5% overhead to control the access on the file descriptor.

Keywords: ARM PA, Intra-process isolation, Capability

I. 서 론

IPI(Intra-process Isolation)는 프로세스 단위의 격리의 무거운 비용을 극복하기 위해 연구되기 시작한 분야이다. 한 프로세스 내에서 더 작은 단위로 컨텍스트를 쪼개 상호 간 통신을 검증된 방식으로만 수행하게 하여 하나의 객체가 오염된다고 하더라도 그 오염이 다른 객체에까지 전파되지 않게 차단하는 것을 주목표로 한다. 이를 달성하기 위해 소프트웨어 기반의 방법과[1] 하드웨어 기반의 방법[5-8] 등이 연구되고 있다.

이러한 기존 연구 대부분은 메모리 관점에서의 격리에만 집중하고 있지만 한 프로세스 내에서는 메모리 이외에도 다양한 자원이 공유되고 있다는 것을 이용하여 IPI에 대한 공격을 수행하는 연구가 최근 수행되었다[2]. 유닉스 계열의 운영체제에서 프로세스는 시스템과 환경과 파일을 통해 소통하는데, 이 파일 역시 프로세스 단위에서 공유되는 자원이기 때문에 오염된 컨텍스트가 파일을 통해 시스템과의 소통

Received(02. 10. 2023), Accepted(03. 06. 2023)
* 이 연구는 2023년도 정부의 재원으로 한국연구재단(교육부) 기초연구사업 (NRF-2022R1C1C1010494), 정보통신기획평가원(융합보안핵심인재양성 (No. 2019-0-01343), 국제공동연구사업 (No. 2020-0-00666))의 지원을 받아 수행된 연구임.
† 주저자, kyuwon.cho@skku.edu
‡ 교신저자, hojoon.lee@skku.edu(Corresponding author)

채널을 오염시킨다면 메모리 관점에서 격리된 다른 객체까지 오염이 전파될 수 있는 것이다.

ARM 아키텍처 v8.3부터 도입된 하드웨어 확장인 PA(Pointer Authentication)는 포인터의 무결성 검증을 위해 제안된 기술로 하드웨어 가속 암호화 회로를 통해 64bit 정수에 빠르게 검증 코드를 상위 비트에 삽입하고 이를 검증하는 ISA를 제공한다.

본 논문에서는 파일 내부 격리를 위해 PA를 사용한 파일 디스크립터에 대한 경량화된 권한 관리 시스템을 제안하고, 이에 대한 실제 구현을 통해 성능을 측정한다.

II. 연구 배경

2.1 프로세스 내부 격리와 우회법

한 프로그램이 여러 컨텍스트를 가지고 동작할 때 한 컨텍스트가 서로 다른 컨텍스트를 침범하지 못하도록 격리가 필요하다. 가장 간편한 격리 방법으로는 각 컨텍스트를 각각 하나의 프로세스로 독립시켜 커널 수준에서의 격리를 제공하는 방법이 있으나 이는 개발 비용 증가와 IPC로 말미암은 실행시간 증가로 이어진다. 이에 연구자들은 한 프로세스 내부에서 각 컨텍스트를 격리 실행시킬 수 있는 방법론들을 연구했다. 컨텍스트끼리의 오염 전파는 대부분 메모리 관점에서 이루어지기 때문에 대부분의 프로세스 내부 격리 연구 또한 각 컨텍스트가 어떻게 메모리에 접근할 수 있는지를 정의함으로써 수행되었다.

그러나 최근 연구[3-4]에서 컨텍스트간에 메모리 이외의 자원 또한 공유된다는 점에서 착안하여 파일과 관련된 시스템 콜에 대한 새로운 공격 및 방어 모델을 제시하였다. 그러나 이에 대한 방어 모델은 Intel MPK를 사용하는 모델로 ARM 아키텍처에는 동등한 하드웨어 기능이 존재하지 않기 때문에 ARM 아키텍처 위에서의 프로세스 내부 격리에 대해 파일과 관련한 시스템 콜을 오용한 공격에 대한 방어 기법이 필요한 상황이다.

2.2 ARM Pointer Authentication

ARM PA(Pointer Authentication)는 ARM v8.3부터 도입된 하드웨어 기반의 보안 기능이다. 이 기능의 본 목적은 포인터의 상위 바이트에 PAC(Pointer Authentication Code)이라 불리는 HMAC 검증 코드를 사용하고 포인터 참조할 때 코드를 검증함으로 포인터 변조 공격을 방지하는 데 있다. pac 명령어를 통해 포인터 상위 바이트를 암호화된 PAC으로 덮어쓰고, aut 명령어를 통해 PAC이 적용된 포인터를 검증한다. 이 때 검증 코드가 일치하지 않으면 검증 코드가 있던 자리를 유효하지 않은 값으로 덮어쓰게 된다. PA는 추가적인 방어를 위해 modifier를 피연산자로 사용하여 검증 코드에 맥락을 설정할 수 있다. 같은 값과 같은 키로 pac 명령어를 수행해도 다른 modifier를 사용하면 다른 검증 코드가 반환된다. 이러한 명령어들은 특별한 하드웨어 회로를 사용함으로 명령어 당 약 4사이클 정도만 소모할 정도로 매우 경량화되어있다[2].

III. 시스템 설계 및 구현

리눅스 등 유닉스 기반 시스템은 프로그램이 파일 계층을 통해 시스템과 소통하며 모든 열린 파일은 32비트 정수 디스크립터를 통해 접근한다. 본 장에서는 디스크립터에 추가적인 검증 코드를 추가해 컨텍스트 단위의 무결성 검증을 수행하는 시스템을 제안한다.

3.1 시스템 설계

본 시스템은 **1)유저 수준의 라이브러리와 2)커널 수준의 검증 모듈**로 이루어진다.

3.1.1 유저 수준 라이브러리

본 시스템의 유저 수준 라이브러리는 프로그래머에게 간편한 격리를 제공하기 위한 API 집합으로 이루어져 있다. 프로그래머는 init 함수를 통해 본 시스템을 초기화하고 초기화 이후부터 수행되는 모든 파일 관련 시스템 콜 (e.g., open, read 등)을 필터링함을 선언한다.

이후 enter 함수를 통해 본 시스템에 의하여 보호받는 컨텍스트로 진입할 수 있다. 시스템에 의해 보호받는 영역에서의 파일 디스크립터 생성 요청은 검증 코드가 부착된 capability 토큰 형태를 반환하며 파일 디스크립터 사용 시에는 항상 토큰에 부착된 검증 코드를 통해 올바른 권한을 갖는 요청인지 검증한다. 파일 디스크립터는 일반적으로 32bit 자료형으로 표현되는데, PA는 그 목적이 포인터 보호에 있

으므로 피연산자 자료형으로 64bit 정수를 사용한다. 본 시스템은 하위 호환성을 보장하기 위해 capability 토큰을 32bit 자료형으로 유지하되 pac 명령어를 사용할 때 64비트로 확장한 뒤 생성된 검증 코드를 다시 32bit의 상위 7비트에 삽입함으로 보호받는 파일 디스크립터를 생성하였다. 또한 공격자가 임의 코드 실행을 통해 enter 함수를 호출하여 보호받는 컨텍스트로 진입을 막기 위해서 init 함수에서 enter 함수를 호출하는 지점을 정의하고 enter 함수의 프롤로그에서 해당 지점 이외에서 enter 함수를 호출하면 진입에 실패하는 CFI를 적용한다.

프로그래머는 exit 함수를 통해 컨텍스트를 종료할 수 있다. 이후 해당 컨텍스트에서 발급받은 capability 토큰은 비활성화되며 더는 사용할 수 없다. 그러나 다시 enter를 통해 컨텍스트에 재진입하면 다시 토큰이 활성화되어 다시 사용할 수 있다. 다만 이 경우 재사용 공격에 취약할 수 있으므로 revocation 함수를 통해 사용이 끝난 컨텍스트와 그 안에서 생성된 토큰들을 안전하게 폐기할 수 있다.

```
init();
...
enter();
int fd = open(path, flags); // returns token
write(fd, buf, SIZE);
exit();
...
read(fd, buf, SIZE); // This syscall will fail.
```

Fig. 1. Example code using system library

3.1.2 커널 수준 모듈

본 시스템의 커널 수준 모듈은 실질적인 시스템 콜 필터링 및 컨텍스트 구현으로 이루어져 있다. 컨텍스트 구현은 ioctl 입력을 통해 라이브러리로부터의 요청을 해석하고 지정된 동작을 수행하는 식으로 이루어진다. 시스템 라이브러리가 init 함수를 통해 커널 모듈에 초기화 명령을 전송하면 커널 모듈은 시스템 콜 테이블을 덮어써 파일 입출력 시스템 콜들을 후킹한다. 또한 랜덤한 64bit 정수를 생성해 각 컨텍스트에 귀속된 modifier로 배정하며 이후 enter 요청이 들어오면 현재 modifier를 배정받은 값으로 변경한다. revocation 또한 modifier 변경으로 수행된다. revocation 요청이 들어오면 해당 컨텍스트에 배정된 modifier를 재생성한다. 추후 기존 토큰에 대한 재사용 공격이 수행되어도 서명에 사용된 modifier가 변경되었기 때문에 항상 검증 코드가 유효하지 않게 되고 따라서 재사용 공격에 내성을 가질 수 있다.

시스템 콜 후킹은 파일 디스크립터 생성 콜(e.g., open, openat 등)들은 모든 로직이 완료된 뒤에 반환할 디스크립터와 배정받은 modifier를 피연산자로 한 pac 명령어를 통해 얻은 검증 코드를 부착한 capability 토큰을 유저에게 반환한다. 또한 파일 디스크립터를 인자로 사용하는 콜(e.g., read, write, fstat등)들은 실제 로직이 수행되기 전에 modifier를 사용해 검증하고, 검증에 실패하면 실제 로직을 수행하지 않고 실패를 반환한다.

IV. 실험 및 결과 분석

본 장에서는 제안 시스템을 실제로 구현하고 성능을 실험한다. 주요 실험으로 시스템 구성요소에 대한 마이크로 벤치마크와 상용 웹 서버에 시스템을 적용하여 웹 서버 스루풋 측정을 수행하였다. 모든 실험은 pac 명령어를 실제 하드웨어로 구현한 Apple M1 Mac Mini에 Asahi Linux를 설치하여 수행되었다.

4.1 시스템 마이크로벤치

라이브러리 함수 호출의 비용은 Table 1과 같다. 위 세 함수는 본 제안 시스템에서 사용하는 API 구현이며 아래 두 항목은 비교를 위한 시스템 콜 호출 비용이다. enter와 exit 모두 getpid와 비슷한 수준의 사이클 비용을 소모하는 것을 볼 수 있다. 반대로 init 함수는 상당히 큰 비용이 드는데, 이는 프로그램 시작 1회만 드는 비용이기 때문에 웹 서버나

Table 1. Cycle count for our system library and common system call(*)

Lib function	Cycle count
init	29486
enter	2186
exit	2193
revocation	2195
*getpid	2086
*open	9554

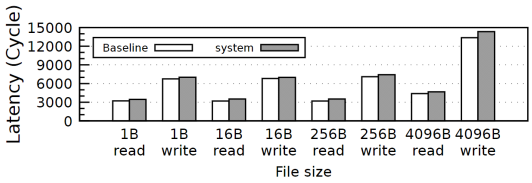

Fig. 2. System overhead in read/write system call for each file size

데몬과 같은 지속적인 실행을 요구하는 프로그램을 대상으로는 실행시간에 큰 영향을 미치지 않는다.

후킹된 시스템 콜 비용 증가를 알아보기 위해 read/write 함수를 각 파일 크기별로 1만 회 실행해 측정하였다. 검증 로직은 상수 시간이기 때문에 요청 크기가 작으면 작을수록 실행시간 증가 비율이 높은 경향을 보이지만 가장 극단적인 1B 요청도 7% 내외의 실행시간 증가를 보이는 등 우수한 성능을 확인할 수 있었다.

4.2 웹 서버 스루풋

본 장에서는 시스템을 상용 웹 서버인 lighttpd의 basic http authentication 로직을 보호하는데 사용하고 스루풋을 측정한다. 측정 방법으로는 네트워크 잡음을 무시하기 위해 로컬 네트워크에서 Apache Benchmark(ab) 툴을 사용해 1) 서로 다른 숫자의 클라이언트 개수 2) 서로 다른 파일 크기 별로 스루풋을 측정했다.

클라이언트 개수별 실험 결과는 최소 4.46%(12K)부터 최대 6.64%(500)의 오버헤드가 측정되었으며 파일 크기별 오버헤드는 최대 6.76%(100B), 최소 1.06%(50K)의 오버헤드를 기록했다. 이는 Intel MPK를 사용한 기존 연구[3]와 비슷한 수준이다.

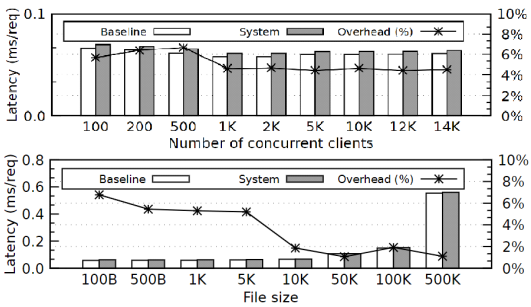

Fig. 3. Latency benchmark of baseline (unmodified lighttpd) vs our system enabled lighttpd using Apache Benchmark(ab)

V. 결 론

본 논문은 ARM의 하드웨어 암호화 기능인 PA를 사용해 파일 디스크립터에 대한 프로세스 내부 격리를 위한 무결성 검증 시스템을 5% 내외의 오버헤드만으로 구현했다. 향후 본 논문을 기반으로 모든 자원을 격리할 수 있는 완벽한 프로세스 내부 격리 시스템을 구현하고자 한다.

References

[1] Chen, Yaohui, et al. "Shreds: Fine-grained execution units with private memory." 2016 IEEE Symposium on Security and Privacy (SP). IEEE, 2016.

[2] Liljestrand, Hans, et al. "PAC it up: Towards Pointer Integrity using ARM Pointer Authentication." USENIX Security Symposium. 2019.

[3] Schrammel, David, et al. "Jenny: Securing Syscalls for {PKU-based} Memory Isolation Systems." 31st USENIX Security Symposium (USENIX Security 22). 2022.

[4] Connor, R. Joseph, et al. "PKU pitfalls: Attacks on pku-based memory isolation systems." Proceedings of the 29th USENIX Conference on Security Symposium. 2020.

[5] Hedayati, Mohammad, and Spyridoula Gravani. "Hodor: Intra-process isolation for high-throughput data plane libraries." Proceedings of the 2019 USENIX Annual Technical Conference. 2019.

[6] Wang, Zhe, et al. "Seimi: Efficient and secure smap-enabled intra-process memory isolation." 2020 IEEE Symposium on Security and Privacy (SP). IEEE, 2020.

〔7〕 Lee, Hojoon, Chihyun Song, and Brent Byunghoon Kang. "Harnessing the x86 Intermediate Rings for Intra-Process Isolation." IEEE Transactions on Dependable and Secure Computing (2022).

〔8〕 Park, Soyeon, et al. "libmpk: Software Abstraction for Intel Memory Protection Keys (Intel MPK)." USENIX Annual Technical Conference. 2019.

〈저자소개〉

조 규 원 (Kyuwon Cho) 학생회원
2021년 8월: 성균관대학교 졸업
2021년 8월~현재: 성균관대학교 석사과정
〈관심분야〉 정보보호

이 호 준 (Hojoon Lee) 정회원
2010년 12월: The University of Texas at Austin 졸업
2013년 8월: KAIST 석사
2018년 2월: KAIST 박사
2018년~2019년: CISPA 연구원
2019년~현재: 성균관대학교 조교수
〈관심분야〉 정보보호