ARM Pointer Authentication을 활용한 Data Pointer Integrity 보호 기법

진홍주*, 이지원*, 김선권**, 이동훈*
*고려대학교 정보보호대학원 정보보호학과
**고려대학교 일반대학원 정보보안학과
*realredwine@korea.ac.kr, *hisdory728@korea.ac.kr, **godmakessky@korea.ac.kr, *donghlee@korea.ac.kr

Data Pointer Integrity Protection Technique Using ARM Pointer Authentication

Hongjoo Jin*, Jiwon Lee*, Seon Kwon Kim**, Dong Hoon Lee*
*Graduate School of Information Security, Korea University.
**Graduate School of Cyber Security, Korea University.

요 약

시스템에 악영향을 끼치는 공격자는 메모리 취약점을 악의적으로 이용하여 프로그램의 제어 흐름을 조작하는 공격을 수행한다. 프로그램의 제어권을 탈취한 공격자는 ROP 공격과 같은 공격을 수행하여 시스템에 악영향을 주는 코드를 실행한다. ROP 공격으로부터 프로그램을 보호하기 위한 기법은 주로 Return Address 및 Function Pointer의 무결성을 보장하는 기법들이 연구되었으나, 메모리에 많이 상주하며 자주 사용되는 Data Pointer에 대한 보호 기법 연구는 미비하다. Data Pointer는 다른 포인터 객체보다 비교적 많이 선언되고 자주 사용되기 때문에 무결성 검증을 위한 비용이 많이 든다. 본 논문에서는 Data Pointer 보호에 대한 높은 보안성을 제공하면서 성능 영향을 대폭 줄이기 위해 하드웨어 지원 기능을 활용한다. ARM 아키텍처에서 제공하는 Pointer Authentication 기능은 하드웨어 기반 Keyed MAC 생성 및 검증 명령어를 제공하며, 이를 활용하면 포인터에 대한 무결성 검증을 효율적으로 수행할 수 있다. 컴파일 과정에서 Data Pointer의 선언 및 사용 부분을 분석하여 ARM PA 명령어를 삽입하고, 적절한 Context 값으로 MAC 값을 생성하는 기법을 제안한다. 제안 기법을 활용하면 효율적으로 Data Pointer의 무결성 검증을 수행할 수 있다.

Ⅰ. 서 론

메모리 취약점은 가상 메모리(Virtual Memory)에 저장되는 데이터의 무결성을 훼손하며, 공격자는 이를 악의적으로 이용하여 프로그램의 제어 흐름(Control-flow)을 조작하는 Control-flow Hijacking 공격을 수행한다. 리턴 지향 프로그래밍(Return-oriented Programming, ROP)[1][2]은 메모리 취약점을 악용하는 대표적인 공격 기법으로, 주로 스택 메모리에 저장된 포인터(e.g., Return Address, Function Pointer, Data Pointer)를 조작함으로써 프로그램의 제어 흐름을 변경한다. ROP 기법은 제어 흐름을 사전에 구성한 가젯 체인(Gadget Chain)으로 이동시켜 시스템에 악영향을 주는 코드를 실행한다.

ROP 공격으로부터 프로그램을 보호하기 위한 기법은 주로 Return Address의 무결성을 보장하는 Backward-edge Control-flow Integrity[3][4] 기법과 Function Pointer의 무결성을 보장하는 Forward-edge Control-flow Integrity[5] 기법이 연구되었다. 하지만, 공격자들은 Data Pointer를 조작하는 정교한 공격을 수행했으며[6], Data Pointer의 무결성을 보장하는 기법[7]은 높은 보안 수준을 달성하기 위해서는 극심한 성능 저하를 유발한다. Data Pointer는 다른 포인터 객체(Return Address, Function Pointer)보다 비교적 많이 선언되고, 자주 사용되기 때문에 무결성 검증을 위한 비용이 많이 든다. 때문에, 성능 오버헤드를 줄일 수 있는 효율적인 기법 설계가 필요하다.

최근, ARM 아키텍처 버전 8.3에 도입된 Pointer Authentication(PA)[8] 확장은 하드웨어 지원으로 포인터의 무결성을 검증하는 보안 확장이다. PA는 보호 대상 포인터에 대한 MAC(Message Authentication Code) 값을 생성 및 검증하는 명령어 체계를 지원하여 소프트웨어 기반 보호 기법 구현과 비교하여 월등한 성능 향상을 달성한다. 주로 저전력 환경을 위해 사용하는 ARM 아키텍처에서 PA 지원을 활용하여 보안 기법을 설계하면 발생하는 성능 오버헤드를 대폭 낮출 수 있다.

ARM PA를 활용하여 Return Address를 보호하는 기법[9]과 Function Pointer를 보호하는 기법[10]이 제안되었지만, Data Pointer 보호 연구는 미비한 상태이다. 본 논문에서는 ARM PA를 활용한 Data Pointer 보호 기법을 제안한다. PAC을 생성할 때 사용하는 Context 값은 MAC 값을 생성할 때와 MAC 값을 검증할 때 같은 값을 유지해야 한다는 제약조건이 있다. 따라서, 특별한 규칙성 없이 선언 및 사용되는 Data Pointer의 특성 때문에 알맞은 Context 값을 지정하는 것이 중요하다. 제안하는 Data Pointer Integrity 보호 기법은 컴파일 과정에 개입하여 Data Pointer의 MAC 값을 생성 및 검증하는 명령어를 추가하며, MAC 값 생성을 위한 적절한 Context 값을 생성한다. 제안 기법을 활용하면 효율적으로 Data Pointer 조작 공격을 막을 수 있다.

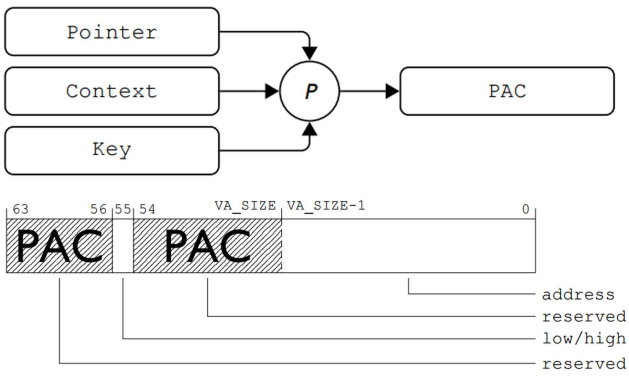

그림 1. ARM Pointer Authentication Code (PAC).

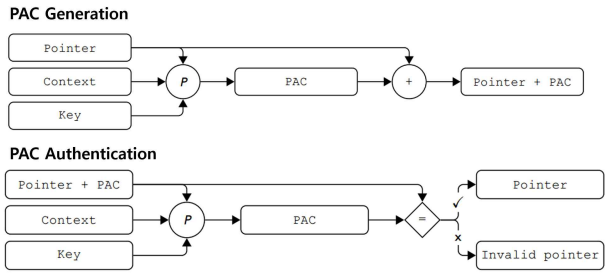

그림 2. PAC 생성 및 검증 프로세스.

Ⅱ. 본론

64-bit 아키텍처에서 2^{64} 주소 공간을 표현할 수 있지만, 실제 가상 메모리의 크기는 2^{48}을 사용해도 충분하기 때문에 주소 값을 저장하는 포인터 객체는 48-bit 부분만 사용한다. 포인터 객체의 나머지 16-bit은 다른 목적으로 예약하여 사용할 수 있는데, ARM PA는 이 공간을 포인터의 MAC 값을 저장하는 용도로 활용한다. ARM PA 지원은 포인터 값에 대한 MAC 값을 생성하기 위해 Seed 역할을 하는 Context에 128-bit 비밀키 값을 활용해 PAC을 생성한다(그림 1). PAC을 생성하기 위해 사용하는 128-bit 키는 하드웨어 기반 공간에 안전하게 생성 및 저장되며, PAC 생성 및 검증도 하드웨어 지원(명령어 지원)으로 구현되어 소프트웨어 기반 기법보다 효율적이다(그림 2).

ARM PA 지원은 MAC 값을 생성하는 *pacda* 명령어와 MAC 값을 검증하는 *autda* 명령어를 제공한다. 제공되는 명령어를 사용하기 위해서는 컴파일 과정의 어셈블리 파일(.s file) 생성 단계에서 추가하고자 하는 위치에 명령어들을 작성해야한다. 때문에, 우선적으로 컴파일 과정 중 IR 코드 생성 단계에서 각각의 함수에서 선언 및 사용되는 데이터 포인터에 대한 정적 분석 정보를 수집하고, 이를 활용하여 어셈블리 코드에서 데이터 포인터의 선언 시점에 PAC을 생성하고, 사용할 때마다 PAC 값을 검증하는 명령어를 작성한다. 이 과정은 실행 시간이 아닌 컴파일 시간에 발생하기 때문에 프로그램의 성능에는 영향을 주지 않으며, 오직 추가된 명령어가 부여하는 성능 영향만 적용된다.

데이터 포인터에 대한 PAC 값을 생성하기 위해 사용하는 Context 값은 데이터 포인터의 선언 시점 및 모든 사용 시점에 같은 값을 지니면서 고유성을 가진 값으로 설정해야 한다. 모든 PAC 값 생성 시 같은 Context 값을 사용하면 PAC 재사용 공격에 취약하다[9]. 따라서, 데이터 포인터를 위한 적합한 Context 값 설정은 중요하며, 최대한의 고유성을 가진 값으로 설정해야 한다. 제안하는 기법은 Context 값으로 해당 데이터 포인터가 선언된 함수의 이름과 해당 데이터 포인터의 타입 정보를 조합하여 활용한다. 이를 활용하면 모든 선언 및 사용 시점에 같은 값을 지니면서 값의 고유성을 최대한 이끌어 낼 수 있다(그림 3).

```
mov    Xmod,  #type_id
xor    Xmod,  #func_id
pacda  Xptr,  Xmod
str    Xptr,  <memory>
...

...

ldr    Xptr,  <memory>
mov    Xmod,  #type_id
xor    Xmod,  #func_id
autda  Xptr,  Xmod
```

그림 3. 제안하는 PAC 활용 기법의 명령어 수준 예시.

Ⅲ. 결론

본 논문에서는 ARM PA를 활용한 데이터 포인터의 무결성을 보호하는 기법을 제안한다. ARM PA를 활용하여 데이터 포인터의 PAC 값 생성 시 Context 선택의 어려움을 해결하였다. 제안 기법을 활용한다면 기존의 소프트웨어 기반 데이터 포인터 보호 기법보다 성능 오버헤드를 대폭 줄일 수 있다.

ACKNOWLEDGMENT

이 성과는 2021년도 과학기술정보통신부의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No.NRF-2021R1A2C2014428).

참 고 문 헌

[1] Prandini, M., & Ramilli, M. (2012). Return-oriented programming. IEEE Security & Privacy, 10(6), 84-87.

[2] Checkoway, S., Davi, L., Dmitrienko, A., Sadeghi, A. R., Shacham, H., & Winandy, M. (2010, October). Return-oriented programming without returns. In Proceedings of the 17th ACM conference on Computer and communications security (pp. 559-572).

[3] Burow, N., Zhang, X., & Payer, M. (2019, May). SoK: Shining light on shadow stacks. In 2019 IEEE Symposium on Security and Privacy (SP) (pp. 985-999). IEEE.

[4] Kuznetzov, V., Szekeres, L., Payer, M., Candea, G., Sekar, R., & Song, D. (2018). Code-pointer integrity. In The Continuing Arms Race: Code-Reuse Attacks and Defenses (pp. 81-116).Standard(AES),"FIPS PUB ZZZ, 2001, (http://www.nist.gov/aes).

[5] Park, M. C., & Lee, D. H. (2020). Random CFI (RCFI): Efficient fine-grained Control-Flow Integrity through random verification. IEEE Transactions on Computers, 70(5), 733-745.

[6] Hu, H., Shinde, S., Adrian, S., Chua, Z. L., Saxena, P., & Liang, Z. (2016, May). Data-oriented programming: On the expressiveness of non-control data attacks. In 2016 IEEE Symposium on Security and Privacy (SP) (pp. 969-986). IEEE.

[7] Song, C., Lee, B., Lu, K., Harris, W., Kim, T., & Lee, W. (2016, February). Enforcing Kernel Security Invariants with Data Flow Integrity. In NDSS.

[8] ARM, Pointer Authentication on ARMv8.3, https://www.qualcomm.com/media/documents/files/whitepaper-pointer-authentication-on-armv8-3.pdf

[9] Liljestrand, H., Nyman, T., Ekberg, J. E., & Asokan, N. (2019, June). Authenticated Call Stack. In Proceedings of the 56th Annual Design Automation Conference 2019 (pp. 1-2).

[10] Denis-Courmont, R., Liljestrand, H., Chinea, C., & Ekberg, J. E. (2020, July). Camouflage: Hardware-assisted CFI for the ARM Linux kernel. In 2020 57th ACM/IEEE Design Automation Conference (DAC) (pp. 1-6). IEEE.