RISC-V 아키텍처에서의 쉐도우 스택 성능평가

강하영 [1], 박성환 [2], 권동현 [3]
[1]부산대학교 정보융합공학과 석사과정
[2] 부산대학교 정보융합공학과 박사과정
[3]부산대학교 컴퓨터공학과 교수 (교신저자)

rkdgkdud12345@pusan.ac.kr, starjara@pusan.ac.kr, kwondh@pusan.ac.kr

Shadow stack performance evaluation in RISC-V architecture

Ha-Young Kang[1], Seong-Hwan Park[1], Dong-Hyun Kwon[2]
[1]Dept. of Information Convergence Engineering, Pusan National University
[2]School of Computer Science and Engineering, Pusan National University

요 약

본 연구에서는 RISC-V 아키텍처를 대상으로 쉐도우 스택을 적용한 벤치마크의 성능을 평가하였다. 이를 통해 RISC-V 아키텍처 상에서 쉐도우 스택이 가지는 성능 오버헤드를 측정하였다. 실험 결과, 평균 2.75%의 성능 오버헤드를 보여주었으며 이는 기준선 대비 무시할 만한 성능 오버헤드가 발생함을 보여주었다. 이러한 결과는 RISC-V 아키텍처에서 쉐도우 스택이 보안 강화에 유용하게 활용될 수 있음을 시사하며, 이를 통해 새로운 보안 메커니즘의 도입에 대한 가능성을 열어두고자 한다. 이 연구는 RISC-V 아키텍처를 기반으로 한 보안 강화 기법의 효과적인 적용에 대한 중요한 기여를 제공할 것으로 기대된다.

1. 서론

RISC-V 는 현재 컴퓨터 아키텍처에서 주목받는 새로운 플랫폼 중 하나이다. 이는 무료로 사용가능한 오픈소스 ISA(instruction set architecture)로, 다양한 하드웨어 구현에서 사용된다[1]. 더불어, RISC-V 는 모듈식 디자인을 사용하여 기본 ISA 코어에 기능을 선택적으로 추가할 수 있는 확장을 제공한다[2]. 이는 시스템 요구 사항에 따라 유연하게 확장이 가능하므로 소프트웨어 개발자들에게 안정적인 타겟을 제공하면서도 새로운 기능을 효율적으로 도입할 수 있는 환경을 제공한다[3].

IoT 장치 및 임베디드 시스템 분야에서는 주로 ARM 과 같은 상용 프로세서를 사용해왔다[4][5][6]. 그러나 RISC-V 의 등장으로, 이러한 분야에서는 더 이상 비용 문제나 라이선스 제한 등의 제약을 받지 않고 개방형 아키텍처인 RISC-V ISA 를 활용함으로써 비용 문제를 해결할 수 있게 되었다[7]. 이는 개방형 아키텍처인 RISC-V 가 ARM 같은 상용 프로세서에 대체제로서의 역할을 수행할 수 있음을 의미한다.

많은 산업 현장에서 RISC-V 를 사용되고 있다[8]. 소프트웨어 업계에서는 이미 RISC-V 아키텍처를 지원하기 시작했다. 예를 들어, Red Hat 은 Linux Fedora 배포판을 지원하고 있으며 Google 은 이에 대한 Android 지원을 발표했다. 하드웨어 산업도 새로운 RISC-V 기반 칩의 설계와 제조를 위해 움직이고 있는 흐름인데, 예를 들어, RISC-V 를 기반으로 하는 보드의 상용 구현을 제조하는 HiFive 제조업체가 있다.

쉐도우 스택[9]은 스택 무결성을 강화하여 반환 지향 프로그래밍(ROP, Return-Oriented Programming) 공격[10]으로부터 보호하고 반환 주소를 보호하는 메커니즘이다. 이 기법은 반환 주소를 공격자가 접근할 수 없는 별도의 격리된 메모리 영역에 저장한다. 프로그램이 반환될 때, 프로그램 스택의 반환 주소의 값을 받아오기 전에 쉐도우 스택의 보호된 반환 주소와 비교한다. 비교 결과가 일치하는 경우, 해당 반환 주소를 사용하여 제어흐름을 반환함으로써 오염된 주소로의 점프를 막는다. 이러한 쉐도우 스택 기법을 통해 반환 지향 프로그래밍 공격을 방지할 수 있다.

일반적으로 ARM 과 Intel 아키텍처에서 쉐도우 스택을 사용하는 경우 낮은 성능 오버헤드를 보이며, 실제 워크로드에서도 오버헤드는 2% 미만으로 나타나며 하드웨어 메커니즘을 사용하지 않고도 이를 달성할 수 있음을 보여준다[9][11][12][13][14].

ARM 이나 Intel 아키텍처에서는 다양한 기법을 활용한 쉐도우 스택의 구현과 성능을 비교한 결과가 제시되어 있다. 그에 비해 RISC-V 아키텍처에서 수행한 쉐도우 스택에 관한 연구나 실험에 대한 논문은 여전

히 부족하다. 따라서 본 논문에서는 RISC-V 아키텍처에서 쉐도우 스택을 적용한 벤치마크의 성능을 측정하여 결과를 정리하였다. 이를 위해 SPEC CPU 2017 벤치마크를 사용하여 실험을 수행한 결과, 평균적으로 2.75%의 작은 오버헤드가 발생하였다.

2. 쉐도우 스택 메커니즘[9][11]

1) 병렬(parallel) 쉐도우 스택 메커니즘

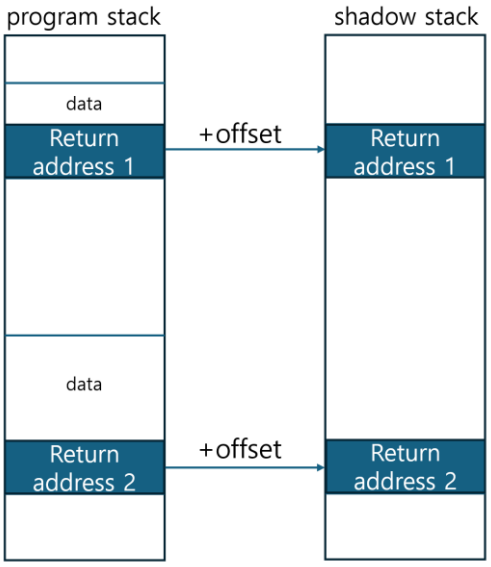

그림 1 병렬 쉐도우 스택 메커니즘 디자인

병렬 쉐도우 스택은 프로그램 스택의 반환 주소 위치를 사용하여 쉐도우 스택에서 해당 항목을 직접 찾는 방식이다. 이 메커니즘은 프로그램 스택과 쉐도우 스택이 같은 크기를 가지고, 간단한 오프셋 매핑으로 해당 오프셋만큼의 차이를 두고 프로그램 스택의 반환 주소와 쉐도우 스택의 반환 주소의 위치가 존재한다. 결과적으로 매우 간단하게 쉐도우 스택을 조회할 수 있어 성능 측면에서 우수하다.

하지만 이러한 병렬 쉐도우 스택 메커니즘은 프로그램 스택과 쉐도우 스택이 같은 크기를 가짐으로써 높은 메모리 오버헤드를 가지고, 오프셋이 유출될 경우 쉐도우 스택의 위치가 모두 노출되므로 낮은 보안성을 가진다는 단점이 존재한다.

2) 컴팩트(compact) 쉐도우 스택 메커니즘

병렬 쉐도우 스택의 높은 메모리 오버헤드를 줄일 수 있는 새로운 메커니즘이 제시되었다.

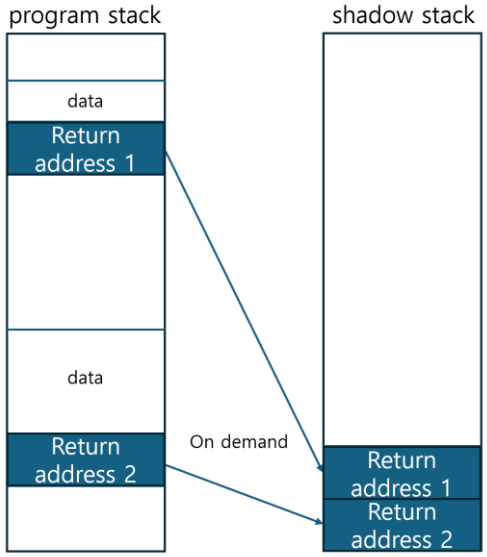

그림 2 컴팩트 쉐도우 스택 메커니즘 디자인

대안으로 제시된 컴팩트 쉐도우 스택 메커니즘은 전용 레지스터의 오프셋을 인코딩하여 쉐도우 스택에 대한 오프셋을 런타임에 결정할 수 있도록 한다. 또한, 이 전용 레지스터는 스레드 로컬로 사용되므로 오프셋은 각 스레드마다 다르게 설정할 수 있다. 이를 통해 함수 호출 당 오버헤드를 추가하지 않으면서도 보다 효율적인 쉐도우 스택 조회를 가능하게 한다.

컴팩트 쉐도우 스택은 쉐도우 스택에서 해당 항목을 찾을 때, 프로그램 스택의 반환 주소 위치를 사용하지 않고 쉐도우 스택의 마지막 항목을 가리키는 쉐도우 스택 포인터를 사용한다. 따라서 프로그램 스택을 복제하지 않고 반환 주소를 위한 공간만을 필요로 하므로 더 적은 메모리를 할당할 수 있다. 이를 통해 병렬 쉐도우 스택에서 발생한 높은 메모리 오버헤드를 완화할 수 있다. 또한 쉐도우 스택 포인터를 레지스터에 저장하여 포인터에 접근할 때 더욱 효율적으로 수행하여 성능을 향상시킬 수 있다.

3. 실험 환경

실험은 VCU 118 보드에서 진행하였다. 보드의 RAM 용량은 2GB 이며, CPU 로는 Rocket 코어를 사용하였다. CPU frequency 는 100MHz 이고, 실험에서는 clang (17.0.6 버전) 컴파일러를 사용하였다. 쉐도우 스택을 적용하지 않은 baseline 에는 rv64imafdc 아키텍처를 대상으로 하여 lp64d 를 어플리케이션 바이너리 인터페이스를 지정하였고 static 링킹을 통해 컴파일하였다. 쉐도우 스택을 적용하기 위해서는 baseline 옵션에 -fsanitize=shadow-call-stack -for-linker=--no-relax-gp 옵션을 추가하여 쉐도우 스택을 적용하여 컴파일 하였다. --no-relax-gp 는

GP(global pointer)를 쉐도우 스택의 최상단을 가리키기 위한 레지스터를 예약함으로써 쉐도우 스택의 값이 변경되는 것을 방지하는 역할을 하는 옵션이다.

4. 실험

실험에서는 SPEC CPU 2017 벤치마크를 사용하여 성능 측정을 진행하였다. 그 중에서 C 언어로 작성된 것과 동시에 속도측면에서 성능을 확인할 수 있는 벤치마크를 골랐다.

각각의 벤치마크르 총 5 회 실행하여 평균값을 계산하였다. 이후 기준선 대비 쉐도우 스택의 오버헤드를 계산하였다.

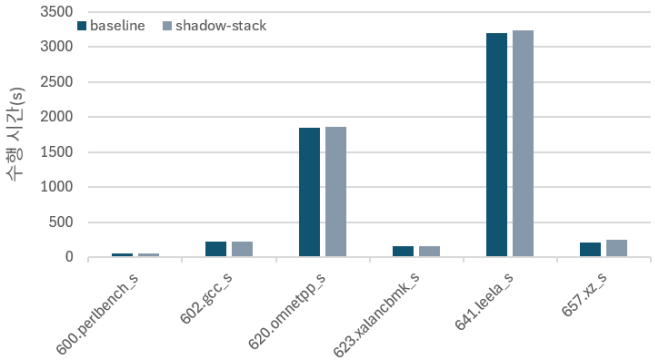

그림 3 수행시간 측정

각 벤치마크는 기준선 대비 -0.66%, 0.43%, 0.08%, 1.21%, 1.47%, 13.98%의 오버헤드가 발생했다. 총 6 가지 벤치마크를 종합하여 구한 평균 오버헤드는 약 2.75%로 적은 오버헤드를 가짐을 알 수 있다.

600.perlbench_s 벤치마크의 경우, 쉐도우 스택을 적용했을 때 오히려 더 낮은 오버헤드를 보인다. 쉐도우 스택은 재귀 호출과 관련된 함수 호출을 추적하는데 도움을 주기 때문에 재귀 호출이 많은 벤치마크에서는 기준선보다 쉐도우 스택을 구현한 벤치마크에서 더 좋은 성능을 보일 수 있다. 또한 perl 과 같은 스크립팅 언어는 다양한 작업을 수행하며 동적으로 함수를 호출하기 때문에, 쉐도우 스택을 사용하면 성능이 향상될 수 있다.

657.xz_s 벤치마크의 경우, 평균대비 높은 오버헤드를 보인다. 해당 벤치마크의 경우 데이터 압축 알고리즘을 사용하는데 이는 메모리 접근 패턴을 변경시킬 수 있고, 또한 압축 알고리즘은 많은 메모리 사용량을 요구하므로 쉐도우 스택을 적용할 경우 평균보다 더 높은 성능 저하가 발생할 수 있다.

그러나 그러한 벤치마크를 포함하더라도 2.75%의 평균 오버헤드를 보이므로 낮은 오버헤드를 발생시키는 것을 볼 수 있다.

이를 통해 쉐도우 스택을 적용함으로써 적은 성능 오버헤드로 보안성을 보장할 수 있음을 확인하였다. 또한 ARM 과 Intel 에서 제시한 작은 오버헤드를 가진다는 장점이 RISC-V 에서도 동일하게 적용됨을 알 수 있다.

5. 결과

현재 RISC-V 아키텍처의 장점으로 인해 쓰임이 계속해서 증가하고 있다. 반환 주소를 보호하기 위한 기법인 쉐도우 스택을 하드웨어를 활용하여 RISC-V 아키텍처에서 성능을 평가함으로써 확인하였다. 실험 결과는 RISC-V 보드에서 쉐도우 스택을 수행하는 것이 가능하며, ARM 과 Intel 이 제시한 쉐도우 스택의 장점을 RISC-V 보드 위에서도 동일하게 적용된다는 것을 확인하였다.

벤치마크를 통해 측정한 결과, 쉐도우 스택을 적용한 경우 기준선 대비 약 2.75%의 적은 오버헤드가 발생하였다. 이러한 결과는 RISC-V 아키텍처에서 쉐도우 스택의 사용이 효율적이며, 적은 오버헤드가 발생한다는 것을 시사한다. 따라서 RISC-V 아키텍처를 통한 보안 강화 메커니즘의 적용은 효과적일 수 있음을 확인하였다.

사사문구

이 논문은 2024 년도 정부(과학기술정보통신부)의 재원으로 정보통신기획평가원의 지원을 받아 수행된 연구임 (No.2021-0-00724, 임베디드 시스템 악성코드 탐지·복원을 위한 RISC-V 기반 보안 CPU 아키텍처 핵심기술 개발)

참고문헌

[1] Lee Y, Waterman A, Cook H, Zimmer B, Keller B, Puggelli A, Kwak J, Jevtic R, Bailey S, Blagojevic M, Chiu PF. An agile approach to building RISC-V microprocessors. ieee Micro. 2016 Mar 18;36(2):8-20.

[2] Mezger BW, Santos DA, Dilillo L, Zeferino CA, Melo DR. A survey of the RISC-V architecture software support. IEEE Access. 2022 May 10;10:51394-411.

[3] Rodrigues C, Marques I, Pinto S, Gomes T, Tavares A. Towards a heterogeneous fault-tolerance architecture based on arm and RISC-V processors. InIECON 2019-45th Annual Conference of the IEEE Industrial Electronics Society 2019 Oct 14 (Vol. 1, pp. 3112-3117). IEEE.

[4] Jung J, Kim B, Cho J, Lee B. A secure platform model based on ARM platform security architecture for IoT devices. IEEE Internet of Things Journal. 2021 Sep 1;9(7):5548-60.

[5] Huang M, Song C. ARMPatch: A binary patching framework for ARM-based IoT devices. Journal of Web Engineering. 2021 Sep;20(6):1829-52.

[6] Yu S, Chen W, Li L, Qin J. Development of ARM-based embedded system for robot applications. In2006 IEEE Conference on Robotics, Automation and Mechatronics 2006 Jun 1 (pp. 1-6). IEEE.

[7] Poorhosseini M, Nebel W, Grüttner K. A compiler

comparison in the risc-v ecosystem. In2020 International Conference on Omni-layer Intelligent Systems (COINS) 2020 Aug 31 (pp. 1-6). IEEE.

[8] Gómez-Sánchez G, Call A, Teruel X, Alonso L, Moran I, Pérez MÁ, Torrents D, Berral JL. Challenges and Opportunities for RISC-V Architectures Towards Genomics-Based Workloads. InInternational Conference on High Performance Computing 2023 May 21 (pp. 458-471). Cham: Springer Nature Switzerland.

[9] Zhou J, Du Y, Shen Z, Ma L, Criswell J, Walls RJ. Silhouette: Efficient protected shadow stacks for embedded systems. In29th USENIX Security Symposium (USENIX Security 20) 2020 (pp. 1219-1236).

[10] Prandini M, Ramilli M. Return-oriented programming. IEEE Security & Privacy. 2012 Dec 10;10(6):84-7.

[11] Burow N, Zhang X, Payer M. SoK: Shining light on shadow stacks. In2019 IEEE Symposium on Security and Privacy (SP) 2019 May 19 (pp. 985-999). IEEE.

[12] Zou C, Gao Y, Xue J. Practical software-based shadow stacks on x86-64. ACM Transactions on Architecture and Code Optimization (TACO). 2022 Oct 7;19(4):1-26.

[13] Li J, Chen L, Xu Q, Tian L, Shi G, Chen K, Meng D. Zipper stack: Shadow stacks without shadow. InEuropean Symposium on Research in Computer Security 2020 Sep 12 (pp. 338-358). Cham: Springer International Publishing.

[14] Choi W, Seo M, Lee S, Kang BB. SuM: Efficient shadow stack protection on ARM Cortex-M. Computers & Security. 2024 Jan 1;136:103568.