The Transactions of the Korea Information Processing Society
TKIPS Vol.13, No.9, pp.413~420
eISSN: 3022-7011
https://doi.org/10.3745/TKIPS.2024.13.9.413

Performance Evaluation and Consideration of Shadow Stack on RISC-V Architecture

Kang Ha Young† · Han Go Won†† · Park Sung Hwan††† · Kwon Dong Hyun††††

ABSTRACT

RISC-V is an open-source instruction set architecture, used in various hardware implementations, and can be flexibly expanded to meet system requirements through the RV64I base instruction set and 16 standard extensions. Currently, the RISC-V architecture employs the shadow stack technique to protect return addresses. This paper compares the performance of the compact shadow stack mechanism and the parallel shadow stack mechanism in the RISC-V architecture using the SPEC CPU 2017 and beebs benchmarks. Experimental results show that the parallel shadow stack mechanism exhibits higher overhead than the compact shadow stack mechanism. This suggests that the efficiency of the parallel mechanism is reduced due to the limitations of the RISC-V architecture, making the compact shadow stack more suitable for RISC-V. Additionally, this paper identifies the security limitations of the existing RISC-V shadow stack and proposes directions for enhancing the performance and security of shadow stack mechanisms to ensure a secure execution environment for RISC-V.

Keywords : RISC-V, Shadow Stack, Security

RISC-V 아키텍처 상에서의 쉐도우 스택 성능 평가 및 고찰

강 하 영† · 한 고 원†† · 박 성 환††† · 권 동 현††††

요 약

RISC-V는 오픈소스 명령어 집합 아키텍처로, 다양한 하드웨어 구현에서 사용되며, RV64I 기본 명령어 집합과 16개의 표준 확장을 통해 시스템 요구 사항에 맞게 유연하게 확장할 수 있다. 현재 RISC-V 아키텍처에서는 반환 주소를 보호하기 위해 쉐도우 스택 기법을 사용하고 있다. 본 논문에서는 RISC-V 아키텍처에서 컴팩트 쉐도우 스택 메커니즘과 병렬 쉐도우 스택 메커니즘의 성능을 SPEC CPU 2017 및 beebs 벤치마크를 사용하여 비교하였다. 실험 결과, 병렬 쉐도우 스택 메커니즘이 컴팩트 쉐도우 스택 메커니즘보다 더 높은 오버헤드를 보이는 것으로 나타났다. 이는 RISC-V 아키텍처의 한계로 인해 병렬 메커니즘의 효율성이 떨어짐을 시사하며, 따라서 컴팩트 쉐도우 스택이 RISC-V 아키텍처에 더 적합함을 보여준다. 또한 본 논문에서 기존 RISC-V 쉐도우 스택의 보안상 한계를 파악하고, RISC-V의 안전한 수행 환경을 보장하기 위해 쉐도우 스택 메커니즘의 성능과 보안성을 향상시키는 방향을 제시한다.

키워드 : RISC-V, 쉐도우 스택, 보안

1. 서 론

RISC-V는 현재 컴퓨터 아키텍처에서 주목받는 새로운 플랫폼 중 하나이다. 이는 무료로 사용 가능한 오픈소스 명령어 집합 아키텍처로, Rocket, BOOM, CVA6, 그리고 SHAKTI C-Class 등 다양한 하드웨어 구현에서 사용된다[1,2]. 더불어, RISC-V는 RV64I 기본 명령어 집합과 필요에 따라 기능을 추가할 수 있는, 총 16개의 표준확장을 제공한다. 표준확장은 하이퍼바이저 (hypervisor)를 위한 표준확장 H, 수퍼바이저 (supervisor) 수준 명령어를 위한 표준확장 S 등이 있으며, RV64IMAFD, RV64GCHS, RV64IMACH와 같이 활용된다[3]. 이러한 확장을 활용하여 저성능 컴퓨팅 환경에서는 필수적인 명령어 집합만을 포함시켜 자원 사용을 최소화하고, 고성능 환경에서는 개발 환경에 알맞게 확장을 적용할 수 있다. 따라서 시스템 요구 사항에 따라 유연하게 확장이 가능하므로 소프트웨어 개발자들에게 새로운 기능을 효율적으로 도입할 수

※ 이 과제는 부산대학교 기본연구지원사업(2년) 의하여 연구되었음.
※ 이 논문은 2024년 ACK 2024의 일반논문으로 "RISC-V 아키텍처에서의 쉐도우 스택 성능평가"의 제목으로 발표된 논문을 확장한 것임.
† 준 회 원 : 부산대학교 정보융합공학과 석사과정
†† 비 회 원 : 부산대학교 정보컴퓨터공학부 학사과정
††† 준 회 원 : 부산대학교 정보융합공학과 박사과정
†††† 종신회원 : 부산대학교 정보컴퓨터공학부 부교수
Manuscript Received : July 11, 2024
Accepted : July 23, 2024
* Corresponding Author : Kwon Dong Hyun(kwondh@pusan.ac.kr)

있는 환경을 제공한다[4].

RISC-V는 베이스에 오프셋 연산만을 사용하여 store 및 load를 수행하는 주소 지정 방식을 채택한다[5]. RISC-V에서는 다른 아키텍처와 비교하여 단순하고 규칙적인 인코딩을 제공하는 간단한 메모리 모델을 사용한다.

사물인터넷(IoT) 장치 및 임베디드 시스템 분야에서는 주로 ARM과 같은 상용 프로세서가 사용된다. ARM 프로세서는 전 세계 반도체 지식 재산권(IP) 시장에서 37% 이상의 점유율을 차지하며 지배적인 위치를 확보하고 있다[6-9]. 그러나 상용 프로세서는 지속적인 비용과 라이선스 문제가 발생한다. 이에 반해, 오픈 소스인 RISC-V는 이러한 비용 문제나 라이선스 제한 없이 활용할 수 있어 기존의 문제를 해결할 가능성을 제공한다[10]. 이러한 이점으로 인해 RISC-V는 ARM의 대체재로서 사물인터넷 장치 및 임베디드 시스템 분야에서 빠르게 성장하고 있다[6].

실제로 많은 산업 현장에서 RISC-V를 사용하고 있다[11]. 소프트웨어 업계에서는 이미 RISC-V 아키텍처를 지원하기 시작했다. Red Hat은 Linux Fedora 배포판을 지원하고 있으며 Google은 이에 대한 Android 지원을 발표했다. 하드웨어 산업도 새로운 RISC-V 기반 칩의 설계와 제조를 위해 움직이고 있는 흐름인데, 예시로 RISC-V를 기반으로 하는 보드의 상용 구현을 제조하는 HiFive 제조업체가 있다.

쉐도우 스택[12]은 스택 무결성을 강화하여 반환 지향 프로그래밍 (Return-Oriented Programming) 공격[13]으로부터 반환 주소 (return address)를 보호하는 기법이다. 이 기법은 프로그램 스택 외의 별도의 스택 공간을 생성해 별도의 격리된 메모리 영역에 반환 주소를 저장하여 보호한다. 함수 수행 이후, 프로그램 스택 내에 저장된 반환 주소 대신 쉐도우 스택 내에 저장된 반환 주소를 로드함으로써 공격자가 오염시킨 반환 주소로의 점프를 막는다.

RISC-V 아키텍처에서도 이러한 쉐도우 스택 기법을 사용하여 프로그램의 제어 흐름을 보호하고 있다. 본 논문에서는 RISC-V 아키텍처에서 현재 사용되는 쉐도우 스택 메커니즘인 컴팩트 쉐도우 스택 메커니즘을 분석한다. 이후 컴팩트 쉐도우 스택 메커니즘보다 효율적인 방식을 사용하여 빠른 수행시간을 가진다고 알려진 병렬 쉐도우 스택 메커니즘과 비교 분석하여 RISC-V 아키텍처 내에서 가장 효율적인 쉐도우 스택 메커니즘을 탐색한다. 이를 위해 SPEC CPU 2017, beebs 벤치마크를 사용하여 실험을 수행한 결과, 병렬 쉐도우 스택 메커니즘이 RISC-V 아키텍처의 한계로 인해 빠른 수행시간이 보장되지 않아 컴팩트 메커니즘의 수행시간 오버헤드(SPEC CPU 2017:1.54%, beebs:3.79%)보다 더 높은 오버헤드(SPEC CPU 2017:6.96%, beebs:6.71%)가 측정되었다.

ARM과 Intel에서는 더 효율적이고 안전한 쉐도우 스택 메커니즘에 대한 활발한 연구가 진행되고 있다. 현재로서는 RISC-V에서 컴팩트 메커니즘의 쉐도우 스택을 사용하고 있지만, 해당 메커니즘은 ARM이나 Intel에서 사용하고 있는 쉐도우 스택 메커니즘보다 보안성이 떨어진다. 따라서 본 논문은 향후 연구를 통해 RISC-V의 쉐도우 스택 보안을 강화하고, 이를 통해 RISC-V의 안전한 수행 환경을 보장하는 데 기여하고자 한다.

2. 쉐도우 스택 메커니즘

프로그램 내에서 함수의 수행이 끝났을 때 해당 함수가 불린 주소로 다시 되돌아가야 한다. 이때 그러한 되돌아갈 주소, 즉 반환 주소를 프로그램 스택 내에서 로드하여 반환하고, 해당 주소로 점프하게 된다.

Fig. 1은 보호 메커니즘을 활용하지 않았을 때, 반환 주소를 저장하고 로드하는 방식을 보여준다. sp (stack pointer) 레지스터를 사용하여 함수의 프롤로그에서 반환 주소를 저장하고 함수의 에필로그에서 저장된 반환 주소를 로드하여 반환한다. 그러나 반환 주소가 무결한지 검증하는 메커니즘이 없기 때문에 반환 지향 프로그래밍 공격에 취약하다. 반환 지향 프로그래밍 공격에서는 공격자가 버퍼 오버플로우 등의 취약점을 이용하여 스택에 일련의 실행 가능한 코드 조각 (gadget)을 삽입한다. 그런 다음 함수의 반환 주소를 이러한 코드 조각의 주소로 덮어쓰고, 함수가 반환될 때 스택에 삽입된 코드 조각들이 차례로 실행되도록 하여 악의적인 행위를 수행한다. 각 코드 조각은 단순한 동작을 수행하지만, 여러 코드 조각을 순차적으로 실행함으로써 복잡한 동작도 수행할 수 있다.

따라서 반환 주소는 프로그램의 흐름을 보호하기 위해 무결하게 보호되어야 할 데이터 중 하나이다.

2.1 컴팩트 쉐도우 스택

현재 RISC-V 아키텍처에서는 컴팩트 메커니즘을 활용한 쉐도우 스택을 사용하여 반환 주소를 보호하고 있다.

Fig. 2는 컴팩트 메커니즘을 적용하였을 때, 프로그램 스택 및 쉐도우 스택에서 발생하는 반환 주소의 흐름[14]을 보여준다. 컴팩트 메커니즘은 쉐도우 스택의 최상단을 가리키는 쉐도우 스택 포인터 (shadow stack pointer)를 사용하여 쉐도우 스택에 접근한다. 쉐도우 스택에 저장되는 반환 주소는 프로그램 스택에 저장되는 반환 주소의 순서에 따라 비워진 공간 없이 저장되게 된다. 즉, 이러한 메커니즘은 메모리 오버헤드가 적다는 장점이 있다.

prologue	`sd ra, 8(sp)`
epilogue	`ld ra, 8(sp)`

Fig. 1. RISC-V Base Instruction

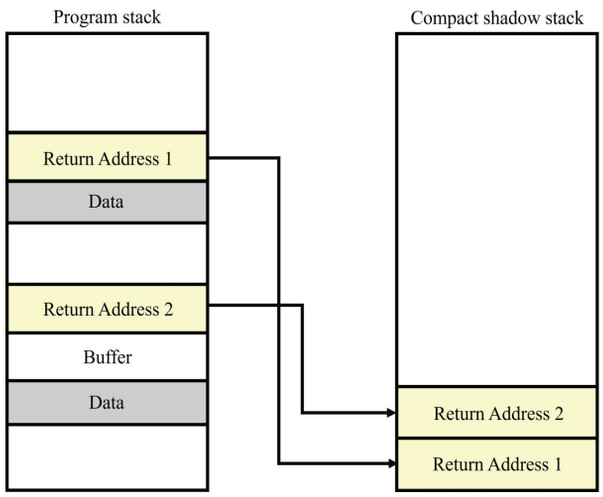

Fig. 2. Compact Mechanism Stack Overview

prologue	addi gp, gp, 8 sd ra, -8(gp)
epilogue	ld ra, -8(gp) addi gp, gp, -8

Fig. 3. RISC-V Compact Shadow Stack Instruction

Fig. 3은 컴팩트 메커니즘의 쉐도우 스택을 활용하여 반환 주소를 저장하고 로드하는 방식을 보여준다. 쉐도우 스택 포인터는 gp (global pointer) 레지스터를 예약하여 저장한다. 따라서 gp 레지스터는 쉐도우 스택의 최상단 주소를 가리키게 되고, 함수의 프롤로그에서 반환 주소를 저장할 때마다 gp 레지스터의 값을 증가시켜 저장한다. 함수의 에필로그에서 반환 주소를 로드할 때 쉐도우 스택에 저장된 반환 주소를 로드하고 gp 레지스터의 값을 감소시켜 메모리를 효율적으로 사용한다.

이러한 컴팩트 메커니즘을 활용하여 쉐도우 스택을 구현하면 큰 메모리 오버헤드 없이 반환 주소를 보호할 수 있으므로 RISC-V에서는 이러한 메커니즘을 채택하여 기본적으로 사용하고 있다.

2.2 병렬 쉐도우 스택

쉐도우 스택 메커니즘 중 컴팩트 메커니즘보다 수행 시간 측면에서 효율적인 방식을 사용하여 더 빠른 수행이 가능하다고 알려진 병렬 메커니즘[15]이 있다.

Fig. 4는 병렬 메커니즘을 적용하였을 때, 프로그램 스택 및 쉐도우 스택에서 발생하는 반환 주소의 흐름을 보여준다. 병렬 메커니즘은 sp 레지스터에 오프셋 연산을 수행하여 쉐도우 스택에 접근한다. 프로그램 스택에 반환 주소를 저장하는

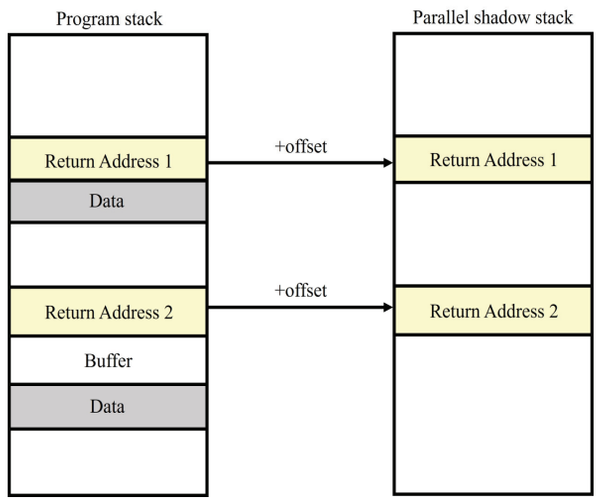

Fig. 4. Parallel Mechanism Stack Overview

메모리 주소에서 오프셋만큼 차이가 나는 주소에 반환 주소를 저장한다. 예를 들어 0xfff8 (프로그램 스택)에 반환 주소가 저장되고 오프셋 값을 0xf000이라고 한다면 0xfff8으로부터 오프셋만큼 차이가 나는 0x0ff8 (쉐도우 스택)의 주소에 반환 주소가 저장된다.

1) x86

Fig. 5A는 x86 아키텍처에서 병렬 메커니즘의 쉐도우 스택을 활용하여 반환 주소를 저장하고 로드하는 방식[16]을 보여준다. r15 레지스터에 오프셋 값을 저장하여 rsp가 가리키는 값으로부터 오프셋 값만큼 차이 나는 주소에 반환 주소를 저장하고 로드한다. x86 명령어 인코딩 구조 Fig. 6A에 따라 rsp 레지스터 값에 r15 오프셋 값을 더한 후 해당 주소에 반환 주소를 저장하는 명령어를 생성할 수 있다. 따라서 x86 아키텍처에서는 단일 명령어로 간단한 오프셋 연산을 수행함으로써 수행 시간을 줄이는 병렬 메커니즘의 장점을 잘 활용할 수 있다.

prologue	mov [rsp + r15], rax
epilogue	mov rax, [rsp + r15]

(A) x86

prologue	sub sp, sp, r15
	sd ra, 0(sp)
	add sp, sp, r15

epilogue	sub sp, sp, r15
	ld ra, 0(sp)
	add sp, sp, r15

(B) RISC-V

Fig. 5. Parallel Shadow Stack Instruction

x86 Instruction encoding

instruction prefixes	opcode	mod	reg/opcode	r/m	scale	index	base	displacement	imm

(A) x86

RISC-V Instruction encoding - store

imm[11:0]	rs1(base)	funct3	rd(dest)	opcode

RISC-V Instruction encoding - load

imm[11:5]	rs2(src)	rs1(base)	funct3	imm[4:0]	opcode

(B) RISC-V

Fig. 6. Instruction Encoding Structure

[16]에서는 x86 아키텍처를 기반으로 spec2006 벤치마크를 활용하여 컴팩트 메커니즘과 병렬 메커니즘의 쉐도우 스택 성능을 비교 실험하였다. 실험 결과, 병렬 메커니즘은 쉐도우 스택 조회 시 단순한 기법을 사용함으로써 성능 면에서 전반적으로 컴팩트 메커니즘보다 우수한 결과를 나타냈다.

2) RISC-V

그러나 RISC-V에서는 아키텍처 상의 한계로 인해 단일 명령어로는 오프셋 연산을 구현하지 못한다. Fig. 5B는 RISC-V 아키텍처에서 병렬 메커니즘의 쉐도우 스택을 활용하여 반환 주소를 저장하고 로드하는 방식을 보여준다. RISC-V 명령어 인코딩 구조 Fig. 6B에 따르면 store와 load 명령어에는 레지스터를 사용하여 연산을 할 수 없고 오프셋 (immediate)값만 사용하여 연산이 가능하다. 또한 그러한 오프셋 값은 12 bit로 설정되어 있으므로 -2048 ~ 2047의 값만 적용할 수 있어서 제한적이다. 따라서 RISC-V 아키텍처에서 충분한 크기의 오프셋 값을 주기 위해서는 레지스터에 오프셋 값을 저장하여 연산을 수행해야 한다. sp 레지스터를 r15 (오프셋)만큼 감소(sub)시킨 후 해당 주소에 저장 및 로드 (sd, ld)를 하고 다시 sp 레지스터를 r15만큼 증가 (add)시켜 다시 복구시키는 방법을 사용한다. 즉, RISC-V 아키텍처에서 병렬 메커니즘을 구현하기 위해서는 store 및 load 명령어 외에 r15를 사용하여 오프셋 연산을 하는 명령어가 추가로 필요하므로 병렬 메커니즘이 가지는 장점을 활용하지 못한다.

추가적인 명령어 사용으로 발생하는 오버헤드를 완화하기 위한 방안을 4.1장에서 다룬다.

2.3 기존 RISC-V 쉐도우 스택 메커니즘의 보안 한계

병렬 메커니즘에서는 오프셋 값이 유출될 경우 공격자가 쉐도우 스택의 위치를 파악할 수 있으며, 이를 통해 쉐도우 스택에 저장된 리턴 주소를 덮어쓸 수 있게 된다. 반면, 컴팩트 쉐도우 스택은 오프셋 값을 사용하지 않고 임의의 주소에 쉐도우 스택 공간을 할당한다. 이 방식에서는 gp 레지스터를 통해서만 해당 공간에 접근할 수 있어 병렬 메커니즘보다 보안상 이점이 있다. 그러나 컴팩트 메커니즘의 쉐도우 스택도 공격자가 접근하고 덮어쓸 수 있는 공간에 할당되므로, 보안은 확률적인 요소에 의존하게 된다. 또한 확률적인 요소에 의존하여 보호하고 있는 쉐도우 스택 내의 반환 주소를 검증 로직 없이 그대로 신뢰하여 해당 반환 주소로 점프하므로 여전히 취약한 점이 존재한다.

ARM과 Intel 아키텍처에서는 하드웨어 기반 보안 기능을 활용하여 이러한 취약점을 방어할 수 있는 메커니즘이 개발되고 있다. 이러한 메커니즘의 구체적인 적용 기법은 4.2장에서 자세히 다룬다.

3. 실 험

RISC-V 아키텍처에서 컴팩트 쉐도우 스택 메커니즘과 병렬 쉐도우 스택 메커니즘의 성능을 비교하기 위한 실험을 진행하였다. 실험에는 SPEC CPU 2017 벤치마크와 beebs 벤치마크가 사용되었다. SPEC CPU 2017에서는 C/C++ 언어로 작성된 벤치마크를 선택하여 속도 측면에서 성능을 평가하였다. Test input을 사용하였고 다만 보드의 한계로 인해 수행되지 않는 벤치마크는 제외하였다. 각 벤치마크는 총 3회 실행되었으며, 평균값을 계산하여 컴팩트 메커니즘과 병렬 메커니즘을 적용한 쉐도우 스택의 오버헤드를 기준선과 비교하였다. beebs 벤치마크에서는 코드 크기가 큰 벤치마크들을 사용하였고, 총 100회 수행하여 각 쉐도우 스택 메커니즘의 오버헤드를 기준선과 비교하였다.

3.1 실험 환경

실험은 VCU 118 보드에서 진행하였다. 보드의 RAM 용량

은 2GB이며, CPU로는 Rocket 코어를 사용하였다. CPU frequency는 100MHz이고, 실험에서는 clang (17.0.6 버전) 컴파일러를 사용하였다. 쉐도우 스택을 적용하지 않은 baseline에는 rv64imafdc 아키텍처를 대상으로 하여 lp64d를 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 컴파일하였다. --no-relax-gp는 gp 레지스터를 쉐도우 스택의 최상단을 가리키기 위한 레지스터를 예약함으로써 쉐도우 스택의 값이 변경되는 것을 방지하는 역할을 하는 옵션이다.

3.2 실험 결과

1) SPEC CPU 2017

Fig. 7은 SPEC CPU 2017 벤치마크의 실험 결과를 보여준다. 컴팩트 메커니즘은 기준선 대비 평균적으로 1.54%의 오버헤드를 보인 반면, 병렬 메커니즘은 기준선 대비 평균적으로 6.96%의 오버헤드를 보였다.

2) beebs

Fig. 8은 beebs 벤치마크의 실험 결과를 보여준다. 컴팩트

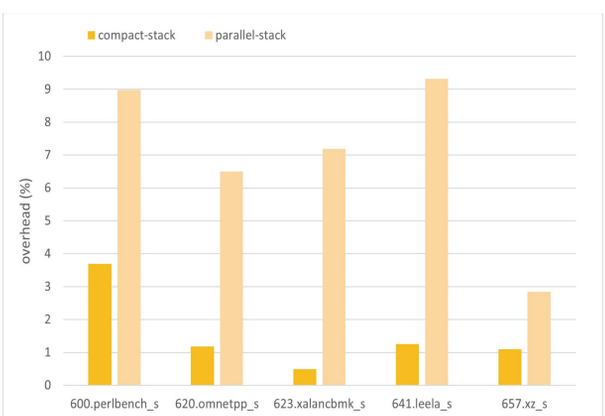

Fig. 7. Overhead of Shadow Stack Mechanism (spec2017)

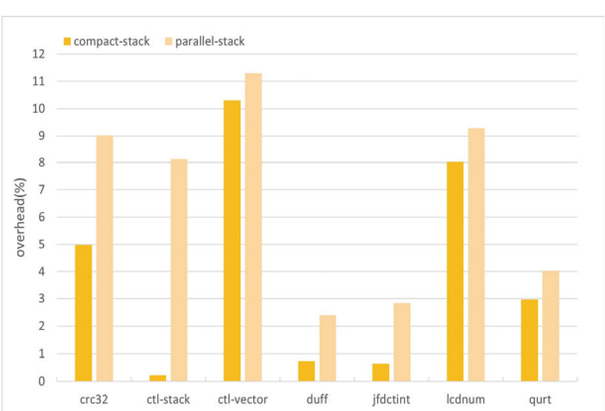

Fig. 8. Overhead of Shadow Stack Mechanism (beebs)

메커니즘은 기준선 대비 평균적으로 3.79%의 오버헤드를 보인 반면, 병렬 메커니즘은 기준선 대비 평균적으로 6.71%의 오버헤드를 보였다.

3) 결과 분석

병렬 메커니즘은 짧은 수행 시간을 장점으로 가짐에도 불구하고 모든 벤치마크에서 병렬 메커니즘이 컴팩트 메커니즘보다 수행 시간 측면에서 더 높은 오버헤드를 보였다. 이는 병렬 메커니즘의 짧은 수행 시간의 장점이 RISC-V 아키텍처에서는 나타나지 않음을 의미한다.

또한, 프로그램 스택 전체의 메모리만큼의 공간을 필요로 하는 병렬 메커니즘은 반환 주소만 저장하는 컴팩트 메커니즘보다 더 높은 메모리 오버헤드를 유발하므로, RISC-V 아키텍처에서는 병렬 메커니즘보다 컴팩트 메커니즘을 적용하는 것이 적합하다. 컴팩트 메커니즘을 적용하였을 때 발생하는 오버헤드는 무시할만한 수준이므로, RISC-V에서 이러한 메커니즘의 쉐도우 스택을 적용함으로써 보다 안전한 환경에서 프로그램을 수행할 수 있다.

4. 고 찰

4.1 병렬 쉐도우 스택 메커니즘의 오버헤드 완화

RISC-V는 32비트 및 64비트 아키텍처에서 32비트 명령어를 사용하며, 이때 immediate 값은 12비트로 제한된다. 병렬 메커니즘을 구현하기 위해서는 큰 오프셋 값을 필요로 하므로 immediate 값을 오프셋으로 적용하기에 어려움이 있다. 따라서 CSIC 명령어 대비 추가적인 명령어를 필요로 하고 이에 따라 더 높은 오버헤드를 발생시킨다.

1) immediate 값으로 더 넓은 비트 폭 활용

현재 RISC-V 표준에서는 16비트 및 32비트 길이의 명령어만을 지원하지만, 추가적인 구현을 통해 64비트, 128비트, 256비트 등의 더 긴 길이의 명령어를 구현할 수 있다[17]. 이를 통해 immediate 값이 12비트로 제한되지 않고 더 큰 값을 사용할 수 있다. 불필요한 add 및 sub 명령어의 사용을 줄이고 단일 load 및 store 명령어를 사용하여 병렬 메커니즘을 구현함에 따라 오버헤드를 완화할 수 있다.

2) 레지스터 오프셋을 사용하는 명령어 사용

사용자 정의 명령어(custom instruction)를 설계하여 store 및 load 명령어에서 레지스터 오프셋을 사용하도록 정의할 수 있다. 이러한 접근 방식은 제한된 immediate 값 비트 폭의 한계를 해결하는 데 기여할 수 있다.

4.2 다른 아키텍처에서의 쉐도우 스택 메커니즘

1) x86 제어흐름 시행 기술

Intel의 제어흐름 시행 기술(CET, Control-flow Enforcement

Technology) [18-20]은 하드웨어 기반의 보안 기술로, 쉐도우 스택 메커니즘을 사용하여 프로그램의 제어 흐름을 보호한다. 이 기술은 쉐도우 스택 페이지(shstk 페이지)라는 새로운 메모리 페이지 유형을 도입하여, 이 페이지에 쉐도우 스택 공간을 할당한다. shstk 페이지는 페이지 테이블 항목의 읽기/쓰기(R/W) 비트를 0으로 설정하여 이 페이지에 대해 읽기 전용(Read-Only) 권한만 부여한다. 이로 인해 일반적인 메모리 접근 명령어가 shstk 페이지의 내용을 수정하거나 삭제할 수 없게 되어 공격자가 반환 주소를 덮어쓸 수 없으므로, 쉐도우 스택의 무결성이 유지된다.

또한, 함수가 반환될 때 프로그램 스택의 반환 주소와 쉐도우 스택의 반환 주소를 비교하는 검증 과정이 수행한다. 두 주소가 일치하는 경우에만 해당 주소로 점프하여, 오염된 주소로의 점프를 방지한다.

Intel의 제어흐름 시행 기술에서 제공하는 쉐도우 스택 메커니즘은 공격자가 쉐도우 스택의 공간도 덮어쓸 수 있는 기존 RISC-V 쉐도우 스택 메커니즘과 달리, 읽기 전용으로 설정된 shstk 페이지를 사용하여 공격자가 해당 페이지에 반환 주소를 덮어쓸 수 없게 한다. 또한 쉐도우 스택 내의 반환 주소와 프로그램 스택 내의 반환 주소를 비교하는 과정을 수행하여 반환 지향 프로그래밍 공격이나 스택 버퍼 오버플로우 공격 등 제어 흐름 공격을 방어한다.

2) aarch64 포인터 인증 기법

포인터 인증 (Pointer Authentication) 기법 [21-22]은 ARMv8.3-A에서 도입된 하드웨어 기반 보안 기능으로, 포인터의 무결성을 최소한의 성능 오버헤드로 검증할 수 있게 한다. 이 기법은 함수 호출과 반환 과정에서 중요한 포인터, 특히 반환 주소를 보호하는 데 사용된다.

함수의 프롤로그에서, ARM 프로세서는 paciasp 명령어를 사용하여 현재 sp 값을 비밀 키처럼 활용하여 암호화 서명(PAC, Pointer Authentication Code) 을 생성하고 이를 반환 주소에 추가한다. 이후 암호화 서명이 추가된 반환 주소가 스택에 저장된다.

함수의 에필로그에서 반환될 때, autiasp 명령어를 사용하여 반환 주소에 추가된 암호화 서명을 검증한다. 이 명령어는 이전에 생성된 암호화 서명이 올바른지 현재의 sp 값을 사용하여 확인한다. 서명 검증이 성공하면 암호화 서명을 제거하여 반환 주소를 원래 상태로 복원한다. 만약 서명이 검증에 실패하면, autiasp 명령어는 예외를 발생시키고, 이후의 잘못된 메모리 접근을 차단한다.

aarch64 아키텍처에서 제공하는 포인터 인증 기법은 쉐도우 스택의 공간을 할당해줘야 했던 기존 RISC-V 쉐도우 스택 메커니즘과는 달리 쉐도우 스택의 공간을 따로 할당하지 않고 반환 주소를 보호할 수 있으므로 메모리를 효율적으로 사용 가능하다. 또한 함수를 반환할 때의 sp 값이 함수 호출 시의 sp 값과 일치할 때만 반환 주소가 유효하도록 보장함으로써 반환 주소의 재사용을 방지한다. 이러한 방식으로, 포인터 인증 기법은 반환 주소의 무결성을 유지하고, 악의적인 공격자가 스택을 오염시켜 잘못된 반환 주소로 점프하는 것을 효과적으로 방지한다.

5. 결 론

본 논문에서는 RISC-V 아키텍처에서 사용되는 쉐도우 스택 메커니즘을 분석하고, 이를 다른 메커니즘과 비교하여 가장 효율적인 쉐도우 스택 메커니즘을 탐색하였다. RISC-V 아키텍처는 무료로 사용 가능한 오픈 소스 명령어 집합 아키텍처로, 다양한 하드웨어 구현에서 채택되고 있으며, 유연성과 확장성 덕분에 저성능부터 고성능까지 다양한 컴퓨팅 환경을 지원한다.

쉐도우 스택은 반환 주소의 무결성을 보호하는 중요한 보안 기법으로, RISC-V에서도 이를 활용하여 프로그램의 제어 흐름을 보호하고 있다. 본 논문에서는 RISC-V에서 사용되는 컴팩트 메커니즘과 그보다 빠른 수행시간의 장점을 가진다고 알려진 병렬 메커니즘을 비교 분석하였다. 병렬 메커니즘은 오프셋 연산을 통해 반환 주소를 저장하며, 이론적으로 수행시간이 단축될 수 있으나 RISC-V 아키텍처에서는 구현의 한계로 인해 실제로는 효율성이 떨어지는 것으로 나타났다.

실험 결과, 컴팩트 메커니즘을 적용함으로써 발생하는 오버헤드는 무시할 만한 수준이므로, RISC-V에서 이를 적용하여 보다 안전한 프로그램 수행 환경을 제공할 수 있다. 그러나 여전히 보안상 취약점이 존재하므로, 향후 연구에서는 공격자가 쉐도우 스택에 접근하여 반환 주소를 덮어쓰지 못하도록 하거나, 반환 주소를 반환하기 전에 검증하는 프로세스를 추가하는 등의 방안들을 모색할 필요가 있다.

References

[1] Y. Lee et al., “An agile approach to building RISC-V microprocessors,” *IEEE Micro*, Vol.36, No.2, pp.8-20, 2016.

[2] A. Dörflinger et al., “A comparative survey of open-source application-class RISC-V processor implementations,” In *Proceedings of the 18th ACM International Conference on Computing Frontiers*, pp.12-20, 2021.

[3] T. Chen and D. A. Patterson, “Risc-v geneology,” *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-6*, 2016.

[4] C. Rodrigues, I. Marques, S. Pinto, T. Gomes, and A. Tavares, “Towards a heterogeneous fault-tolerance architecture based on arm and RISC-V processors,” *InIECON*

2019-45th Annual Conference of the IEEE Industrial Electronics Society, IEEE, Vol.1, pp.3112-3117, *2019.*

[5] D. Kanter, "RISC-V offers simple, modular ISA," *Microprocessor Report*, Vol.1, pp.1-5, 2016.

[6] William Li, Dale Gai, "RISC-V to Shake up $8.6-Billion Semiconductor IP Market," September 13, 2021 [online] https://www.counterpointresearch.com/insights/riscv-semiconductor-ip-market-2025/

[7] J. Jung, B. Kim, J. Cho, and B. Lee, "A secure platform model based on ARM platform security architecture for IoT devices," *IEEE Internet of Things Journal*, Vol.9, No.7, pp.5548-60, 2021

[8] M, Huang and C. Song, "ARMPatch: A binary patching framework for ARM-based IoT devices," *Journal of Web Engineering*, Vol.20, No.6, pp.1829-52, 2021.

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

[10] M, Poorhosseini, W, Nebel, and K. Gruttner, "A compiler comparison in the risc-v ecosystem, In *2020 International Conference on Omni-layer Intelligent Systems (COINS)*, IEEE, pp.1-6, 2020.

[11] G, Gomez-Sanchez et al., "Challenges and opportunities for RISC-V architectures towards genomics-based workloads," *InInternational Conference on High Performance Computing*, Cham: Springer Nature Switzerland, pp.458-471, 2023.

[12] J. Zhou, Y. Du, Z, Shen, L, Ma, J, Criswell, and R. J. Walls, "Silhouette: Efficient protected shadow stacks for embedded systems," In *29th USENIX Security Symposium (USENIX Security 20)*, pp.1219-1236, 2020.

[13] M, Prandini and M. Ramilli, "Return-oriented programming," *IEEE Security & Privacy*, Vol.10, No.6, pp.84-7, 2012.

[14] J. Li, L. Chen, Q. Xu, L. Tian, G. Shi, K. Chen, and D. Meng, "Zipper stack: Shadow stacks without shadow," In *European Symposium on Research in Computer Security*, Cham: Springer International Publishing, pp.338-358, 2020.

[15] C. Zou, Y. Gao, and J. Xue, "Practical software-based shadow stacks on x86-64," *ACM Transactions on Architecture and Code Optimization*, Vol.19, No.4, pp.1-26, 2022.

[16] N. Burow, X. Zhang, and M. Payer. "SoK: Shining light on shadow stacks," In *Proceedings of IEEE Symposium on Security and Privacy (Oakland)*, 2019.

[17] N. M. Qui, C. H. Lin, and P. Chen, "Design and implementation of a 256-bit RISC-V-Based dynamically scheduled very long instruction word on FPGA," *IEEE Access*, Vol.8, pp.172996-173007, 2020.

[18] B. V. Patel, "A Technical Look at Intel's Control-Flow Enforcement Technology," 2020. [online] https://www.intel.com/content/www/us/en/developer/articles/technical/technical-look-control-flow-enforcement-technology.html?wapkw=control-flow%20enforcement%20technology

[19] M. Xie et al., "CETIS: Retrofitting Intel CET for generic and efficient intra-process memory isolation," in *Proc. 29th ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, pp.2989-3002, 2022, doi: 10.1145/3548606. 3559344

[20] H. Kim, J. Lee, S. Kim, S. Jung, and S. K. Cha, "How'd security benefit reverse engineers? the implication of intel cet on function identification," In *Proceedings of the International Conference on Dependable Systems Networks*, pp.559-566, 2022.

[21] H. Liljestrand, T. Nyman, K. Wang, C. C. Perez, J. E. Ekberg, and N. Asokan, "PAC it up: Towards pointer integrity using ARM pointer authentication," In USENIX Security '19. USENIX Association.

[22] S. Yoo, J. Park, S. Kim, Y. Kim, and T. Kim, "In-kernel control-flow integrity on commodity oses using ARM pointer authentication," In *31st USENIX Security Symposium*, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022, K. R. B. Butler and K. Thomas, Eds., USENIX Association, pp. 89-106, 2022.

강 하 영

https://orcid.org/0009-0002-8983-9090
e-mail : rkdgkdud12345@pusan.ac.kr
2024년 부산대학교 수학과(학사)
2024년 ~ 현 재 부산대학교
정보융합공학과 석사과정
관심분야 : 시스템 보안, 정보보안

한 고 원

https://orcid.org/0009-0002-4074-6539
e-mail : gowonisgood@pusan.ac.kr
2023년 ~ 현 재 부산대학교
정보컴퓨터공학부 학사과정
관심분야 : 시스템 보안

박 성 환

https://orcid.org/0009-0006-5607-8423
e-mail : starjara@pusan.ac.kr
2021년 동서대학교 컴퓨터공학부(학사)
2021년 ~ 현 재 부산대학교 정보융합공학과 박사과정
관심분야 : 시스템 보안, H/W 아키텍처

권 동 현

https://orcid.org/0000-0002-7507-3111
e-mail : kwondh@pusan.ac.kr
2012년 서울대학교 정보컴퓨터공학부(학사)
2019년 서울대학교 정보컴퓨터공학부(박사)
2023년 부산대학교 정보컴퓨터공학부 조교수
2024년 ~ 현 재 부산대학교 정보컴퓨터공학부 부교수
관심분야 : 시스템 보안, 소프트웨어 보안