Poison value 기반 sanitizer 기술 연구 동향 분석

강승민 [1], 박재열 [2], 권동현 [3]
[1] 부산대학교 정보컴퓨터공학부 학부생
[2] 부산대학교 정보융합공학과 석박사통합과정
[3] 부산대학교 정보컴퓨터공학부 교수*

tmdals010126@pusan.ac.kr, woduf0628@pusan.ac.kr, kwondh@pusan.ac.kr

A Survey on Sanitizer Techniques Based on Poison Values

Seung-Min Kang[1], Jae-yeol Park [2], Dong-hyun Kwon[3]
[1]Dept. of Computer Science and Engineering, Pusan National University
[2]Dept. of Information Convergence engineering, Pusan National University
[3]Dept. of Computer Science and Engineering, Pusan National University

요 약

Sanitizer 는 메모리 오류, 타입 불일치, 초기화되지 않은 값 사용 등 취약점을 탐지하는 동적 분석 도구로, 소프트웨어 테스트에 널리 활용된다. Address Sanitizer 와 같은 메타데이터 기반 도구는 높은 탐지율을 보이지만, 메모리 사용과 실행 시간 측면에서 한계가 있다. 이를 보완하기 위해 최근에는 poison value 를 활용한 경량 sanitizer 가 제안되고 있다. 논문은 이러한 기법들을 정리하고, 탐지 정확도, 오탐률, 오버헤드 관점에서의 특성과 실용 가능성을 고찰한다.

1. 서론

C/C++ 프로그램에서 발생하는 런타임 메모리 오류는 중요한 보안 취약점 중 하나로 꼽힌다. 이러한 문제를 탐지하기 위한 동적 분석 도구인 “Sanitizer”는 테스트 단계에서 오류를 조기 발견하는 데 널리 활용된다. 대표적인 예로 Address Sanitizer (ASAN) [1]은 shadow memory 기반의 접근 방식으로 높은 탐지 정확도를 제공하지만 평균 3.37 배의 메모리 오버헤드와 1.73 배의 실행 시간이라는 문제점이 존재한다.

이러한 문제를 해결하기 위해 별도의 메타데이터 없이 redzone 에 특수 값을 삽입하여 변조를 감지하는 poison value 기반 sanitizer 가 제안되었다. 초기에는 메타데이터를 최소화하면서 높은 성능과 호환성을 확보한 LBC[2]가 등장하였고 이후 ReZZan[3]은 메타데이터를 완전히 제거하였다. 최근에는 FloatZone[4]이 FPU 특성을 활용해 검사 연산 자체의 부하를 줄이는 방법을 제시하였다.

본 논문은 poison value 를 활용한 sanitizer 기술들의 설계 방식과 성능 특성을 비교 분석하고, 이들의 한계와 향후 연구 방향을 모색한다.

2. Poison value 기반 sanitizer 연구

2.1 LBC (Lightweight bounds checking)

LBC 는 메모리 객체 주변에 guard zone 을 삽입하고, 포인터 역참조 시 이 영역에 접근했는지를 검사하는 sanitizer 이다. Guard zone 은 접근이 발생하면 안 되는 보호용 메모리 공간으로, 내부에 삽입된 poison value 를 통해 손상 여부를 감지한다. 이는 redzone 과 기능적으로 동일하며, LBC 에서는 이를 guard zone 이라 지칭하여 오류를 탐지한다.

Fast-path 에서는 메모리 접근 시 poison value 만을 비교해 빠르게 오류를 탐지하며 slow-path 에서는 guard map 을 조회해 guard zone 침범을 확인한다.

Guard map 은 각 메모리 주소가 정상 객체인지 guard zone 인지를 나타내는 최소한의 메타데이터다.

SPECint 2000 기준 LBC 는 평균 23% 실행 오버헤드와 8.5% 메모리 오버헤드를 기록했으며 기존 bounds checker 대비 뛰어난 성능과 낮은 자원 소모를 달성했다. 또한 포인터 자체를 추적하는 별도 instrumentation 없이 메모리 접근 시점에서만 검사해 기존 코드와 높은 호환성을 유지할 수 있다[2].

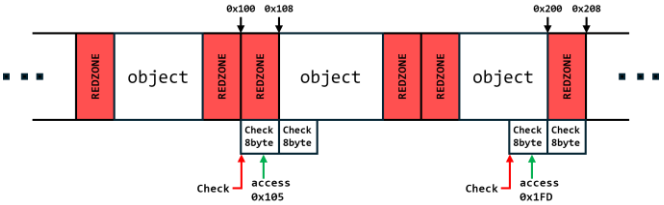
2.2 ReZZen

ReZZan 은 redzone 에 Poison value 를 삽입하고, 해당 값의 변조 여부를 통해 메모리 경계 침범을 탐지

* Corresponding Author

하는 경량 sanitizer 이다. <그림 1>의 메모리 구조에서 볼 수 있듯이 Stack, heap, global 영역의 각 메모리 객체의 전후에 redzone 을 배치하며 프로그램 실행 시 고정된 값을 poison value 로 설정한다. 이후 메모리 접근 시 <그림 1>에서 화살표로 표기된 것과 같이 해당 주소에 저장된 값이 poison value 와 일치하는지를 검사함으로써 overflow 나 underflow 여부를 판단한다.

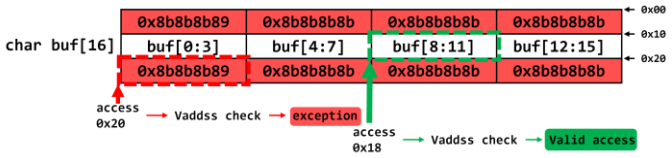
ReZZan 은 별도의 shadow memory 나 메타데이터를 사용하지 않기 때문에 기존 방식 대비 메모리 접근 오버헤드가 낮고 구현이 단순하다. AFL++ 기반 Fuzzing 실험에서 native 실행 대비 ASAN 은 약 2.36 배, ReZZan 은 1.27 배의 실행 오버헤드를 보였다[3].



<그림 1> Rezzan 의 redzone 과 redzone check
access 는 초록색, check 는 붉은색 화살표

2.3 FloatZone

FloatZone 은 부동소수점 연산 장치인 Floating Point Unit (FPU)의 예외 처리 특성을 이용해 메모리 오류를 탐지하는 경량 sanitizer 이다. 이 기법은 redzone 에 poison value 로 0x8b8b8b8b 을 <그림 2>와 같이 삽입하고 메모리 접근 시 주소에 저장된 값과 0x0b8b8b8a 를 더하는 연산(vaddss)을 수행하여 underflow exception 이 발생하는지를 확인함으로써 redzone 접근을 판별한다.



<그림 2> FloatZone 에서의 redzone 과 redzone check
exception 발생은 붉은색, valid access 는 초록색 화살표

이러한 방식은 별도의 메타데이터 없이 탐지가 가능해 shadow memory 기반 도구보다 낮은 오버헤드를 가진다. SPEC CPU 2006 기준으로 ASAN 은 native 실행 대비 평균 77.8%의 실행 시간 오버헤드와 237%의 메모리 오버헤드를 기록한 반면, FloatZone 은 36.4%의 실행 시간 오버헤드와 182%의 메모리 오버헤드를 보이며 효율적인 성능을 달성하였다[4].

3. 결론

본 논문에서는 LBC, ReZZan, FloatZone 기반 경량 sanitizer 를 비교 분석하였다. 이들 기법은 shadow memory 없이 메모리 오류를 탐지하여 성능과 적용성에서 기존 기법 대비 강점을 보인다.

LBC 는 guard zone 과 최소한의 메타데이터를 활용하여 정확도를 확보하며 ReZZan 은 고정된 poison value 만으로 검사를 수행해 메타데이터를 제거했다. FloatZone 은 FPU 의 부동소수점 연산 특성을 이용해 redzone 접근을 감지하여 연산 부하를 줄였다. 이러한 경량 sanitizer 들은 임베디드 시스템, IoT 기기, 실시간 운영체제(RTOS) 환경에서도 활용 가능하다.

그러나 사용자가 사용하는 production 환경에서는 한계가 존재한다. ReZZan 과 FloatZone 은 고정된 poison value 사용으로 인해 redzone 을 poison value 로 덮어쓰는 방식의 우회 공격에 취약하며 FloatZone 은 FPU 연산 오차로 인한 오탐 및 미탐 가능성이 존재한다. LBC 는 intra-object overflow, 즉 하나의 객체 내 배열과 필드 사이의 침범을 탐지하지 못하는 문제가 있다.

최근 SANRAZOR[5], ASAP[6]등 production 환경 적용을 위한 sanitizer 연구가 활발히 진행되고 있기에 향후에는 이러한 한계를 극복하고 실시간 보호가 가능한 경량 sanitizer 설계가 필요하다.

사사문구

본 연구는 과학기술정보통신부 및 정보통신기획평가원의 융합보안핵심인재양성사업의 연구 결과로 수행되었음 (RS-2022-II221201)

참고문헌

[1] Serebryany, Konstantin, Derek Bruening, Alexander Potapenko, and Dmitry Vyukov, "AddressSanitizer: A fast address sanity checker," 2012 USENIX Annual Technical Conference (USENIX ATC 12), Boston, USA, 2012, pp. 309–318.

[2] Hasabnis, N., Misra, A., and Sekar, R., "Light-weight bounds checking," Proceedings of the Tenth International Symposium on Code Generation and Optimization (CGO 2012), San Jose, CA, USA, 2012, pp. 135–144.

[3] Ba, J., Duck, G. J., and Roychoudhury, A., "Efficient greybox fuzzing to detect memory errors," Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering (ASE), Rochester, MI, USA, 2022, pp. 1–12.

[4] Gorter, F., Barberis, E., Isemann, R., Van Der Kouwe, E., Giuffrida, C., and Bos, H., "FloatZone: Accelerating memory error detection using the floating point unit," 32nd USENIX Security Symposium (USENIX Security 23), Anaheim, CA, USA, 2023, pp. 805–822.

[5] Zhang, J., Wang, S., Rigger, M., He, P., and Su, Z., "SANRAZOR: Reducing redundant sanitizer checks in C/C++ programs," Proceedings of the 15th USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2021, pp. 479–494.

[6] Wagner, J., Kuznetsov, V., Candea, G., and Kinder, J., "High system-code security with low overhead," Proceedings of the 2015 IEEE Symposium on Security and Privacy (S&P), 2015, pp. 866–879.