안드로이드 부트로더 퍼징을 위한 에뮬레이션 연구

김상윤 [1], 이병영 [2]
[1] 서울대학교 전기정보공학부 박사과정
[2] 서울대학교 전기정보공학부 부교수

sangyun.kim@snu.ac.kr, byoungyoung@snu.ac.kr

A Study on Android Emulation for Bootloader Fuzzing

Sang-Yun Kim[1], Byoung-Young Lee[2]
[1,2]Dept. of Electrical and Computer Engineering, Seoul National University

요 약

본 연구에서는 안드로이드 부트로더의 취약점을 효과적으로 발견하기 위한 에뮬레이션 기반 퍼징 프레임워크를 제안한다. 부트로더는 높은 권한을 가지고 실행되기 때문에 취약점이 존재할 경우 심각한 보안 위협이 될 수 있다. 그러나 부트로더는 다양한 하드웨어와 상호작용하기 때문에 에뮬레이션 하기에 어려움이 많다. 이에 본 연구에서는 QEMU 에뮬레이터를 기반으로 부트로더의 주변 장치를 모델링하여 효율적인 퍼징을 수행하였다. 실험 결과, 에뮬레이션을 통해 실제 기기 대비 높은 퍼징 속도를 달성하였으며, 지속적으로 새로운 코드 영역을 발견할 수 있음을 확인하였다. 본 연구의 프레임워크는 향후 부트로더 취약점 분석 및 보안 검증에 활용될 수 있을 것으로 기대된다.

1. 서론

컴퓨터나 모바일 기기에서 부트로더는 하드웨어를 초기화하고 운영체제를 로드하는 핵심 소프트웨어로, 시스템이 정상적으로 부팅되기 위해서는 부트로더의 역할이 필수적이다. 이러한 중요성으로 인해 부트로더는 매우 높은 권한을 가지고 실행된다.

그러나 이러한 높은 권한은 보안상 위험요소가 될 수 있다. 만약 공격자가 부트로더의 취약점을 악용하여 권한을 획득한다면, 공격자는 시스템 전체를 장악할 수 있게 된다. 따라서 부트로더의 보안은 매우 중요하며, 이를 위해 다양한 보안 기법들이 제안되어 왔다.

대표적인 부트로더 보안 기법으로는 보안 부팅 (Secure Boot) [1]이 있다. 해당 기법은 부트로더의 무결성을 검증하고 악성코드 실행을 차단하는 데 초점을 맞추고 있다. 그러나 매년 새로운 부트로더 취약점들이 지속적으로 발견되고 있어, 부트로더 보안의 중요성은 계속 대두되고 있다. 예를 들어 2020 년에 발견된 BootHole 취약점 [2]은 보안 부팅을 우회하여 서명 되지 않은 코드를 커널 공간에 로드 할 수 있었다.

이에 본 연구에서는 랜덤 테스팅 기법인 퍼징 [3]을 활용하여 부트로더의 취약점을 사전에 발견하고자 한다. 퍼징은 임의의 입력 데이터를 생성하여 프로그램을 테스팅하는 기법으로, 알려지지 않은 취약점을 발견하는 데 효과적이다. 특히 부트로더와 같이 소스 코드가 공개되지 않은 시스템의 취약점을 찾아내기 위해서는 퍼징이 유용할 것으로 기대된다.

그러나 부트로더는 다양한 하드웨어를 초기화해야 하므로, 실제 기기에서 퍼징을 수행하는 것이 일반적이다. 부트로더가 다양한 하드웨어와 상호작용하는 복잡한 과정을 에뮬레이션하기는 매우 어렵기 때문이다. 따라서 실제 기기에서 퍼징을 수행하는 경우, 퍼징 속도가 느리고 효율성이 낮다는 한계가 있다.

이에 본 연구에서는 부트로더에 활용되는 다양한 하드웨어를 에뮬레이션하여, 효율적인 퍼징 환경을 구축하고자 한다. 이를 통해 부트로더의 취약점을 보다 신속하게 발견하고, 이를 바탕으로 부트로더 보안 강화 방안을 모색할 수 있을 것으로 기대된다.

2. 기존 부트로더 퍼징 환경 조사

부트로더 퍼징 도구를 설계하기에 앞서, 기존에 활용되었던 부트로더 퍼징 환경을 조사하였다. 부트로더 퍼징을 위한 실험 환경은 크게 1) 실제 기기 환경과 2) 에뮬레이션 환경으로 구분할 수 있다.

실제 기기 환경 실제 기기 환경에서 퍼징을 수행하면 소프트웨어와 하드웨어 간의 높은 일관성을 가질 수 있다. 이를 통해 부트로더에 실제로 전달될 수 있는 하드웨어 입력을 활용하여 퍼징을 진행할 수 있으므로, 거짓 양성(false positive)을 효과적으로 제거할 수 있다.

그러나 실제 기기 환경에서는 몇 가지 한계점이 존

재한다. 먼저, 부트로더의 경우 일반적으로 소스코드가 공개되어 있지 않아, 다양한 퍼징 기법 적용이 어려워 퍼징 성능을 높이기 어렵다. 또한 다수의 실제 기기를 확보해야 하므로 확장성이 낮다는 문제가 있다.

에뮬레이션 환경에서는 부트로더의 다양한 하드웨어 상호작용을 관찰하고 입력으로 활용할 수 있다는 장점이 있다. 이를 통해 보다 광범위한 퍼징이 가능하다.

하지만 에뮬레이션의 정확성 한계로 인해 거짓 양성(false positive) 발생 가능성이 있다. 또한 에뮬레이션을 위한 주변장치 에뮬레이션 작업이 요구되므로 구현 복잡도가 높고, 주변장치에 대한 정보가 한정적이기 때문에 간접적인 방법으로 해결하는 경우가 많다.

이러한 장단점을 고려하여, 본 연구에서는 에뮬레이션 환경에서 부트로더 퍼징 도구를 수행하였다. 이를 통해 제한된 하드웨어 환경에서도 퍼징을 수행할 수 있었다.

3. 부트로더 에뮬레이션 설계

부트로더는 다양한 하드웨어를 초기화를 수행하는 코드를 포함하고 있다. 이러한 코드들은 주변 장치들과 데이터를 교환하는데, 이때 부트로더가 접근하는 주변 장치의 레지스터 주소는 타겟 기기마다 다른 값을 가지고 있다. 따라서 부트로더 에뮬레이션의 첫 단계는 타겟 기기의 메모리 매핑 정보를 알아내는 것이다.

메모리 매핑 정보를 파악하는 방법에는 세 가지가 있다. 첫째, 해당 칩셋의 Device Tree 정보를 분석하는 것이다. ARM 아키텍처의 경우 주변 기기 정보가 DTB(Device Tree Blob) 형태로 제공되는데, 이를 DTC(Device Tree Compiler)를 사용해 DTS(Device Tree Source) 형태로 변환하여 메모리 매핑 정보를 추출할 수 있다.

둘째, 같은 제조사의 이전 칩셋 DTS 정보를 활용하여 추론하는 것이다. 칩셋이 달라져도 내부 주변 장치들은 유사한 경우가 많으므로, 이전 칩셋의 DTS 를 참고하여 현재 칩셋의 메모리 매핑을 추정할 수 있다.

마지막으로, 부트로더 코드 리버싱을 통해 문자열 정보를 분석하여 메모리 매핑을 추론하는 방법도 있다.

본 연구에서는 Exynos 9820 칩셋을 대상으로 설계를 진행하였으며, 공개된 Exynos 9820 DTS 정보를 활용하여 메모리 매핑 정보를 획득하였다.

한편, 부트로더 에뮬레이션을 위해서는 부트로더가 접근하는 주변 장치에 대한 에뮬레이션이 필요하다. 이를 위해 앞서 분석한 메모리 매핑 정보를 활용하여 주변 장치의 MMIO(Memory-Mapped I/O) 및 DMA(Direct Memory Access) 영역을 모델링하였다.

주요 주변 장치로는 USB 컨트롤러를 선정하였다. 이에 대한 장치 설명서 및 드라이버 코드 분석을 통해 각 레지스터의 기능과 동작을 에뮬레이션하였다. 레지스터는 제어, 데이터, 상태 레지스터로 구분되며, 이에 따라 이벤트 핸들러를 설계하였다. 또한 DMA 영역의 경우 메모리 덤프 기능을 추가하였다.

이와 같이 타겟 기기의 메모리 매핑 정보와 주요 주변 장치의 에뮬레이션 설계를 통해, 부트로더 에뮬레이션 환경을 구축하였다.

4. 부트로더 퍼징 프레임워크 설계

퍼저의 전체적인 구조는 그림 1 에 나타내었으며, 각 모듈 별 설명은 다음과 같다. 첫째, 문법 기반의 입력 생성 및 변이 모듈이 있다. 이 모듈은 UART 와 USB 등 부트로더가 사용하는 다양한 입력 채널을 지원하도록 설계되었다. 둘째, 부트로더 에뮬레이션 모듈이 있다. 이 모듈은 앞서 제 3 장에서 설계한 메모리 매핑 정보와 주변 장치 에뮬레이션 기능을 포함한다. 셋째, 메모리 버그 탐지기 모듈이 있다. 이 모듈은 QEMU [4]의 TCG 기술을 활용하여 부트로더의 메모리 안정성을 검사한다.

이 세 가지 핵심 모듈은 상호 유기적으로 동작하도록 구현되었다. 먼저, 입력 생성 및 변이 모듈에서 생성된 임의의 입력은 에뮬레이션 모듈로 전달된다. 에뮬레이션 모듈은 입력을 처리하고, 메모리 버그 탐지기 모듈은 이 과정에서 발생하는 메모리 접근을 감시한다. 만약 메모리 오류가 발견되면 관련 정보를 수집하여 보고한다.

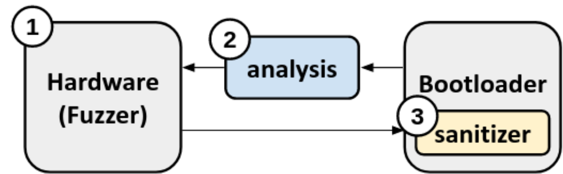

그림 1. 부트로더 퍼징 프레임워크 구조도

5. 실험 환경 구성

본 연구의 실험은 EPYC 7552 CPU 와 256GB RAM 이 장착된 Ubuntu 20.04.6 LTS 환경에서 진행되었다. S-Boot 부트로더를 x86 환경에서 실행하기 위해 에뮬레이터가 필요하며, 이를 위해 외부 펌웨어 아카이브 사이트에서 바이너리를 받아 이미지를 생성하였다. 본 연구에서 타겟으로 설정한 부트로더 버전은 갤럭시 S10 의 Exynos 9820 칩셋에 맞춘 부트로더이다.

실험에 사용된 펌웨어는 갤럭시 S10 펌웨어이며, 외부 웹 사이트를 통해 다운로드 받았다. 이후 각 LUN 에 어떤 이미지가 포함되어 있는지를 나타내는 정보를 담고 있는 pit 파일을 파싱하여 각 LUN 이미지를 생성할 수 있는 유틸리티 프로그램과 스크립트를 제작하였다.

또한, 에뮬레이터 실행 시 BL1 과 BL2 의 동작을 생략하고 바로 BL3 로 실행하기 위해, 펌웨어 파일에서 BL3 에 해당하는 영역을 추출하는 작업을 수행하였다. 갤럭시 S10 부트로더의 경우, BL3 이미지는 0xA4000 이후에 위치하고 있다.

이렇게 생성된 이미지와 BL3 바이너리를 활용하여

에뮬레이터를 실행하면 S-Boot 의 부팅 메시지를 확인할 수 있다. 이를 통해 에뮬레이션 환경에서 S-Boot 부트로더를 구동할 수 있음을 확인하였다.

부트로더 퍼징을 위해 AFL 을 선택하였다. AFL 은 커버리지 기반 퍼징과 forkserver 를 통한 실행 속도 최적화 기능을 제공한다. 이러한 기술들을 S-Boot 부트로더에 적용하기 위해 AFL 과 QEMU 를 수정하였다. 추가적으로, 부트로더 퍼징 성능을 저하하는 가장 큰 원인인 재부팅 과정을 해결하기 위해 QEMU 의 스냅샷 기능을 활용하였다. 스냅샷을 통해 부팅 이후의 특정 시점의 메모리 상태를 저장하고 복원함으로써 약 4-5 배의 성능 향상을 달성하였다.

퍼징 입력 생성을 위해, 부트로더에 전달할 수 있는 두 가지 하드웨어 인터페이스인 UART 와 USB 컨트롤러를 선정하였다. UART 의 경우, 부팅 과정 중 S-Boot Shell 에 진입하였을 때 다양한 명령어를 입력으로 전달할 수 있다. USB 컨트롤러는 부트로더의 여러 부팅 모드에서 허용되는 USB 패킷을 생성하여 전달한다. 각 인터페이스에서 사용 가능한 입력들은 부트로더 바이너리에 대한 리버스 엔지니어링을 통해 식별하였으며, 이를 바탕으로 퍼징 입력 생성기를 구현하였다.

6. 실험 결과 및 평가.

본 과제에서는 에뮬레이터를 통한 부트로더 퍼징 프레임워크를 설계하였다. 해당 프레임워크의 성능을 퍼징 속도와 커버리지 증가량 두 가지 측면에서 평가하였다. 실험은 1 개의 퍼저를 사용하여 1 core 로 수행되었다.

AFL 퍼징 속도를 측정한 결과, 평균 0.46 exec/s 의 속도로 퍼징이 수행되었다. 이는 입력 처리 시간보다 부팅 시간이 상당 부분을 차지하고 있어, 일반적인 유저 레벨 애플리케이션에 비해 퍼징 속도가 현저히 낮게 나타난다.

그러나 실제 기기와 비교했을 때, 에뮬레이션을 통한 퍼징은 상당한 성능 향상을 보여준다. 실제 기기에서는 부트로더에 새로운 입력을 넣기 위해서는 재부팅을 필요로 하며, 재부팅 과정은 수 초의 시간이 소요된다.

또한, 에뮬레이션 기반 퍼징은 확장성(scalability) 측면에서도 장점을 가진다. 예를 들어, 10 개의 코어를 사용하여 퍼징을 수행한다면 총 실행 시간 관점에서 4.6 exec/s 의 성능을 얻을 수 있다. AFL 은 여러 개의 AFL 인스턴스를 하나의 마스터 AFL 이 관리하며 시드(seed)를 공유하는 실행 모드를 지원한다. 이를 활용하면 많은 수의 코어를 가진 서버 환경에서 안정적인 퍼징 성능을 확보할 수 있다.

본 연구에서 구현한 퍼징 프레임워크를 통하여 24 시간 동안 약 74 개의 새로운 코드 영역에 도달하였다. 이는 유저 레벨 애플리케이션에서 AFL 이 보여주는 성능에 비해 적은 수치이지만, 커버리지 증가량이 수렴하지 않고 지속적으로 증가하는 것을 확인하였다. 이는 퍼저가 계속해서 새로운 코드 영역을 발견하고 있음을 의미한다. 본 장에서 언급한 확장성 높은 퍼저 환경을 구축한다면 커버리지 증가량을 더욱 높일 수 있을 것으로 기대된다.

커버리지 증가량은 퍼징 속도뿐만 아니라 장치에 적용되는 입력 문법에도 크게 영향을 받는다. 퍼징 초기(약 1 시간 이내)에 많은 커버리지를 확보할 수 있었던 것은 부트로더의 장치 드라이버에 입력할 포맷을 사전에 설정해주었기 때문으로 볼 수 있다. 따라서 현재 제공된 문법을 더욱 정교하게 개선한다면 커버리지 증가량을 높이는 데 큰 도움이 될 것이다.

7. 결론

본 연구에서는 안드로이드 부트로더의 취약점을 효과적으로 발견하기 위한 에뮬레이션 기반 퍼징 프레임워크를 제안하였다. QEMU 에뮬레이터를 기반으로 부트로더의 주변 장치를 모델링하여 실제 기기 대비 높은 퍼징 속도를 달성하였으며, 지속적으로 새로운 코드 영역을 발견할 수 있음을 확인하였다. 비록 유저 레벨 애플리케이션 대비 낮은 퍼징 속도와 커버리지 증가량을 보였지만, 에뮬레이션 환경의 확장성을 활용하여 다수의 퍼저를 병렬로 수행하고 입력 문법을 개선한다면 퍼징 성능을 더욱 높일 수 있을 것으로 기대된다. 본 연구의 프레임워크는 향후 부트로더 취약점 분석 및 보안 검증에 활용될 수 있을 것이다.

8. 사사문구

이 논문은 2024 년도 삼성전자의 재원으로 "퍼징을 활용한 Chipset 및 firmware 보안 검증 기술" 과제의 지원을 받아 수행된 연구임.

참고문헌

[1] CVE-2020-10713, Boothole Vulnerability. https://nvd.nist.gov/vuln/detail/CVE-2020-10713

[2] Verified Boot. https://source.android.com/docs/security/features/verifiedboo

[3] Manes, V. J., Han, H., Han, C., Cha, S. K., Egele, M., Schwartz, E. J., and Woo. The art, science, and engineering of fuzzing: A survey. arXiv preprint arXiv:1812.00140. 2018.

[4] QEMU Emulator. https://www.qemu.org/