creative commons
COMMONS DEED

저작자표시-비영리-변경금지 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:

저작자표시. 귀하는 원저작자를 표시하여야 합니다.

비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.

변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 이용허락규약(Legal Code)을 이해하기 쉽게 요약한 것입니다.

Disclaimer

dCollection

공학석사학위논문

하드웨어 브레이크포인트를 이용한 커널 수준 결함 주입 기법

2016년 5월

서울대학교 대학원

공과대학 컴퓨터공학부

이 남 구

하드웨어 브레이크포인트를 이용한 커널 수준 결함 주입 기법

지도교수 하 순 회

이 논문을 공학석사학위논문으로 제출함

2016년 5월

서울대학교 대학원

공과대학 컴퓨터공학부

이 남 구

이남구의 석사학위논문을 인준함

2016년 5월

위 원 장 김 지 홍 (인)

부 위 원 장 하 순 회 (인)

위 원 이 창 건 (인)

요약

결함 주입 기법은 하드웨어의 오류를 모방(emulate)하거나 소프트웨어의 신뢰성을 검증하여 시스템의 견고성을 높이는 데에 사용된다. 결함 주입 기법은 임베디드 시스템 등에서 사용되는 리눅스 커널에도 적용할 수 있다. 리눅스 커널에 대한 기존의 연구에는 결함 주입 대상이 되는 부분을 수정해야 한다거나 한정된 종류의 결함만을 주입할 수 있다는 제약점이 있었다. 본 연구에서는 하드웨어 브레이크포인트를 이용하여 커널 수준에서 다양한 종류의 결함 주입 실험을 수행할 수 있는 결함 주입 기법을 소개한다. 이 기법은 결함 주입 대상이 되는 커널 코드의 수정 없이 결함 주입 기능을 가진 새로운 커널 모듈을 추가하는 방식으로 구현되었다. 제안한 기법을 이용해 안드로이드 운영체제가 동작하는 ARMv7-A 아키텍처 기반 ODROID-XU4 보드의 내장형 멀티미디어 카드(eMMC) 제어장치를 대상으로 실험을 진행하였다.

주요어: 결함 주입 기법, 리눅스 커널, 하드웨어 브레이크포인트, ARMv7-A, ODROID-XU4, eMMC 제어장치, 디바이스 드라이버

학 번: 2014-22687

목차

그림 목차

표 목차

제 1 장 서론

결함 주입 기법은 예상치 못한-시스템 개발 과정에서 디버깅 등의 방법으로 발견하지 못한-오류에 대해 시스템이 얼마나 잘 감내하는지 검증하기 위해, 즉 시스템의 견고성과 신뢰성을 검증하기 위해 이용되는 역사가 오래된 시스템 개발 방법이다. 결함 주입 기법은 우주 환경과 같은 극한의 환경을 가정하여 하드웨어에 직접 물리적인 결함을 주입하는 방식에서부터 시작하였으나 최근에는 하드웨어의 복잡도가 증가하면서 소프트웨어를 이용해서 하드웨어 수준의 결함을 모방(emulate)하거나 소프트웨어 자체의 결함에 대해서 연구하는 방식이 주로 이용된다[1].

본 논문에서는 하드웨어 브레이크포인트(이하 HWBP)를 이용해서 리눅스 커널에 결함을 주입하는 기법을 소개한다. 그리고 하드웨어 수준에서 제공되는 자원인 HWBP를 활용하여 기존의 연구에서 다뤄지지 않았던 새로운 결함 모델에 대한 실험을 소개한다. 본 논문의 결함 주입 대상인 리눅스는 PC나 서버를 비롯하여 스마트폰, 임베디드 시스템 등의 다양한 장치에서 널리 사용되는 운영체제이다. 리눅스가 동작하는 임베디드 환경의 경우, 전력 제약 등에 따라 설계 마진이 제한되어 있고, 특히 커널 영역의 경우 소프트웨어 디버깅 환경이 열악한 편이다. 결함 주입 기법은 하드웨어의 오류를 모방하거나 디바이스 드라이버의 오류를 재현하여 이러한 임베디드 장치의 안정성을 검증하기 위해 사용할 수 있다.

리눅스 커널을 대상으로 한 기존의 연구들은 소프트웨어 수준에서의

결함 주입을 목표로 하고 있다. 커널 자체의 안정성을 파악하기 위한, 혹은 유저 수준 프로그램의 안정성을 파악하기 위한 목적으로 연구가 진행되었다. 반면 하드웨어와 커널 사이의 인터페이스 수준에서의 결함을 다루는 연구는 부족하다. 본 연구에서는 HWBP를 이용하여 하드웨어와 커널 사이의 인터페이스에서 발생할 수 있는 결함을 재현하고 결함 발생 시에 시스템의 반응을 살펴볼 수 있는 기법을 소개한다.

리눅스 커널에서의 결함 주입 기법에 대한 기존의 연구들은 결함 주입 대상 모듈을 수정하고 다시 컴파일해야 하거나 한정된 종류의 소프트웨어 결함만을 주입할 수 있다는 제한이 있다. 본 논문에서 소개하는 기법은 결함 주입 대상 모듈에 대한 수정이 필요 없으며 맞춤형으로 정의된 결함 주입 함수를 이용하여 다양한 종류의 결함을 주입할 수 있다. 또한 하드웨어 수준에서 제공되는 자원을 이용하기 때문에 결함 주입 실험이 시스템 전체에 미치는 영향을 최소화 할 수 있다.

본 논문에서 소개하는 기법을 검증하기 위해 리눅스 기반의 안드로이드 운영체제가 동작하는 ODROID-XU4 보드의 내장형 멀티미디어 카드(이하 eMMC) 제어 장치를 대상으로 실험을 진행하였다. 실험에 사용된 도구는 ODROID-XU4 보드뿐만이 아니라 ARMv7-A 아키텍처로 구현된 다른 프로세서에도 이식될 수 있다.

본 논문의 구성은 다음과 같다. 2장에서 본 논문이 제시하는 기법이 풀고자 하는 문제를 정의 하고 3장에서 관련 연구를 소개한다. 4장에서 본 논문에서 제시하는 기법의 원리와 실제 구현에 대해서 설명하고 5장에서는 구현된 도구를 이용해 진행한 실험을 소개한다. 마지막으로 6장에서 연구의 결론을 내린다.

제 2 장 문제 정의

2.1 커널 수준에서의 결함 주입 기법

본 논문에서는 소프트웨어 수준에서의 결함 주입을 통해서 하드웨어 수준에서의 결함을 모방(emulate)하는 기법에 대해서 연구한다. 그리고 소프트웨어 중에서도 응용 수준이 아닌 커널 수준에서 결함을 주입하는 것을 목표로 한다. 실험 대상으로 사용되는 보드 ODROID-XU4는 4개의 Cortex-A15 코어와 4개의 Cortex-A7 코어로 이루어져 있고 8개의 코어 모두 ARMv7-A 아키텍처를 기반으로 한다[2]. 해당 보드에서 동작하는 리눅스 커널의 버전은 3.10.9이다.

2.2 결함 주입 대상 모듈을 수정할 필요가 없는 기법

본 논문에서는 ARMv7-A 아키텍처로 구현된 프로세서에서 제공하는 HWBP를 이용하여 결함을 주입하는 기법을 소개한다. ODROID-XU4 보드의 각 코어는 6개씩의 HWBP를 지원한다. 본 논문에서 소개하는 기법에서는 HWBP를 조작하는 커널 수준의 결함 주입 모듈을 이용하여 커널 내부의 임의의 위치에 존재하는 명령어(instruction)에 HWBP를 설정할 수 있다. 그리고 커널이 HWBP가 설정된 명령어를 실행하려고 하는 상황을 감지하여 결함 주입 대상이 되는 모듈에 결함을 주입한다. 이러한 방식을 이용하면 결함 주입 대상 모듈을 수정하지 않고도 결함 주입 실험을 수행할 수 있다.

2.3 결함 주입 위치를 지정하는 방식

HWBP를 이용하는 본 논문의 기법에서는 결함을 주입하고자 하는 함수에 HWBP를 설정함으로써 결함을 주입하는 위치와 시점을 설정한다. 이렇게 함으로써 임의의 위치에 결함을 주입하여 통계적인 분석을 하는 방식이 아닌 특정한 위치에 결함을 주입하는 방식의 실험이 가능하다. 이러한 방식은 결함 주입의 결과 시스템이 이상 반응을 보일 경우 그 원인을 쉽게 추적할 수 있다는 장점이 있다.

2.4 주입하는 결함의 종류

① 비트플립(Bit-flip): 비트플립은 메모리의 특정 비트의 값이 바뀌는 결함이다. 본 논문에서는 한 개 이상의 비트가 바뀌는 경우도 포함한다고 가정하여 커널이 소유한 변수들(커널의 전역 변수 및 커널 함수의 인자 등)의 값이 바뀌는 경우도 비트플립 결함으로 정의한다.

② 지연(Delay): 지연은 커널이 실행되는 도중 일정 시간 동안 실행이 멈추는 것을 의미한다. 본 논문에서는 커널 함수가 실행되는 도중 결함 주입 모듈이 인위적으로 함수의 수행을 지연시키는 기법을 소개한다.

③ 맞춤형(Custom): 맞춤형 결함은 결함을 주입하고자 하는 상황에 맞게 맞춤형으로 정의된 결함 주입 함수를 이용하는 기법이다. 자세한 사례는 5장에서 소개한다.

제 3 장 관련 연구

표 1에 리눅스 커널을 대상으로 하는 기존의 커널 수준 결함 주입 기법들을 정리하였다.

[3], [4]에서 소개하는 결함 주입 기법은 커널의 소스 코드를 직접 수정하여 결함을 주입하는 기법으로 코드 변형(code mutation)을 통해서 결함이 주입된다. 결함이 있는 채로 컴파일된 커널 모듈을 만들어서 시스템이 어떻게 반응하는지 알아보는 것이다. 각기 다른 부분에서 결함이 있는 커널 모듈을 대량으로 만들어서 각각의 경우에 대한 시스템의 반응을 알아보아야하기 때문에 하나의 결함을 넣을 때마다 매번 컴파일을 해야 하는 단점이 있다. 본 논문에서 제시하는 기법은 결함 주입 대상이 되는 모듈에 수정을 가할 필요가 없고, 맞춤형으로 결함 주입 함수를 새로 정의할 때만 컴파일이 필요하다.

[5], [6]에서는 함수 호출을 가로채는 기법(function call interception)을 사용하여 결함 주입 대상 모듈이 어떤 함수를 호출하려고 할 때 원래 함수 대신 따로 정의된 실험용 함수를 호출하게 한다. 따라서 특정 드라이버에서 사용하는 커널 함수가 잘못 동작했을 때 커널이 이를 잘 감지하고 처리할 수 있는지 확인하는 용도로 사용할 수 있다. 함수 호출을 가로채는 방법으로는 심볼 테이블을 수정하는 방법과 이진 코드를 해석(decode)하여 call, jmp와 같은 명령어의 호출 주소를 변경하는 방법을 제시하였다. 이 기법은 결함 주입 대상 모듈을 수정하지만 매 결함 주입 실험마다 대상 모듈을 다시 컴파일하지 않아도 되는 장점이 있다. 하지만 논문에서도 언급되었듯이 그 대상 모듈이 실행되는

도중에는 안전하게 변경을 가하기가 쉽지 않다는 단점이 있다. 본 논문에서 제시하는 기법은 하드웨어와 밀착된 수준에서 작동하기 때문에 대상 모듈이 실행되는 도중에도 함수 호출을 가로채는 효과를 쉽게 넣고 제거할 수 있다.

[4], [6]에서 볼 수 있듯이 결함 주입 기법과 관련된 최신의 연구들은 미리 잘 정의된 결함 주입 기법을 이용해서 효율적으로, 그리고 광범위하게 소프트웨어 결함을 주입하는 방법을 연구하고 있다. [6]에서는 리눅스 디바이스 드라이버의 견고성을 검증하기 위해 자동화된 방식으로 효율적인 결함 주입 시나리오 집합을 생성하는 방법에 대해서 연구하였다. [4]에서는 [1]의 기법을 이용해 동시에 여러 개의 소프트웨어 결함 주입 실험을 진행했을 때 실험들 사이에 간섭 효과가 있는지 검증하였다. 그래서 결함 주입 실험의 병렬화를 통해 전체 검증 시간을 줄일 수 있는지에 대한 연구를 진행하였다. 이 연구도 역시 리눅스 디바이스 드라이버를 실험 대상으로 하여 진행되었다.

본 논문에서 제시하는 기법은 [7], [8]처럼 결함 주입 대상에 수정을 가하지 않고 별도의 커널 모듈을 이용해 결함을 주입하는 기법이다. 기존의 연구들은 기본적인 비트플립 결함 주입 기법만을 제공하거나 커널의 특정 영역에만 결함을 주입할 수 있지만, 본 논문은 기존 연구에서 보여주지 못했던 종류의 결함을 커널의 임의의 위치에 주입할 수 있는 기법을 제시한다.

리눅스 커널을 대상으로 하는 기존의 커널 수준 결함 주입 기법들은 소프트웨어 수준에서 커널 자체나 유저 수준 프로그램의 안정성을 파악하기 위한 목적으로 연구가 진행되었고 하드웨어의 영향을 고려한 연구는 부족하였다. 본 연구는 하드웨어와 커널 사이의 인터페이스에서

표 1. 리눅스 커널 대상 결함 주입 기법

논문	결함 주입 대상 수정 필요성	커널 컴파일 필요성	결함 주입 방식	주입하는 결함의 종류
[3], [4]	O	O	code mutation	맞춤형
[5], [6]	O	X	function call interception	메모리 할당 오류, DMA/PCI 인터페이스의 오류
[7], [8]	X	X	function call interception	커널 메모리 영역에 비트플립
제안 기법	X	△	function call interception	비트플립, 지연, 맞춤형

발생할 수 있는 결함을 재현하고 결함 발생 시에 시스템의 반응을 살펴볼 수 있는 기법을 소개한다.

본 연구는 하드웨어적으로 제공되는 자원을 활용했다는 점에서 [9], [10]과 유사하다. [9]에서는 PowerPC 601 프로세서를 대상으로 하는 결함 주입 도구인 Xception을 소개하였다. Xception은 PowerPC 601의 하드웨어 브레이크포인트를 이용하여 프로세서의 각 기능 모듈(정수 계산 모듈, 부동 소수점 계산 모듈, 분기 모듈 등)에서 발생하는 결함을 모방하는 실험을 하는 데 사용되었다. [10]에서는 우주 환경에서 사용되어 신뢰도가 높아야 하는 Thor 프로세서를 대상으로 하는 결함 주입 도구인 FIMBUL(Fault Injection and Monitoring using BUilt in L

ogic)을 소개하였다. Thor 프로세서에는 프로세서 내부의 메모리 요소들(캐시 및 레지스터)과 프로세서 칩의 핀 값을 읽고 수정할 수 있게 하는 내부 로직(built in logic)이 존재한다. FIMBUL은 이처럼 프로세서 내부에 미리 구현되어 있는 테스트 로직을 이용해서 우주 환경의 이온화 입자들이 하드웨어에 결함을 가하는 상황을 모방하는 실험을 하는데 사용되었다. 앞의 두 연구는 모두 하드웨어 브레이크포인트를 이용하여 결함 주입 시점과 위치를 지정한다는 점에서 본 연구와 유사하다. 본 연구에서 제시하는 기법은 근래에 많이 사용되는 ARM 아키텍처 기반의 프로세서를 대상으로 하며 맞춤형으로 정의된 결함 주입 함수를 이용하여 다양한 종류의 결함을 주입할 수 있다는 점에서 앞의 두 연구와 차이가 있다.

제 4 장 기법 소개 및 구현

본 논문에서 제안하는 기법은 HWBP를 이용한다. ARMv7-A 기반 프로세서에서 HWBP가 설정되어 있는 명령어를 실행하려고 하면 하드웨어적으로 prefetch abort 예외상황(exception)이 발생한다[11]. 이 때 예외상황이 발생하는 것을 감지하고, 예외상황이 발생한 시점에서 미리 정의된 '결함 주입 함수'를 실행하는 방식으로 커널에 결함을 주입하게 된다. 이러한 기능을 가진 '결함 주입 모듈'은 커널 수준에서 작동해야 하는데 ARMv7-A에서 HWBP를 다루는데 사용되는 명령어는 커널 수준에서만 사용될 수 있기 때문이다. 그리고 이 '결함 주입 모듈'과 연동된 '유저 수준 프로그램'을 도입하여 HWBP를 다루는 작업을 유저 수준에서 처리할 수 있게 하였다.

그림 1은 본 논문에서 제안하는 결함 주입 기법의 간략한 흐름도이다. 그림에서 굵은 상자로 되어 있는 부분(결함 주입 모듈 및 유저 수준 프로그램)이 결함 주입을 위해 새로 구현된 부분이다. 결함 주입 모듈은 코어별로 제공되는 HWBP를 관리하며 유저 영역 프로그램의 명령을 받아 HWBP를 직접 설정하고 해제하는 역할을 한다. 결함 주입 모듈은 리눅스의 디바이스 드라이버 형식의 모듈로 구현되어 있고 유저 영역과의 인터페이스는 ioctl() 함수를 통해서 이루어진다. 유저 수준 프로그램은 이 ioctl() 명령을 통해서 유저 수준에서 HWBP를 설정하고 해제할 수 있게 한다.

그림 1의 각 단계에 대한 자세한 설명은 다음과 같다.

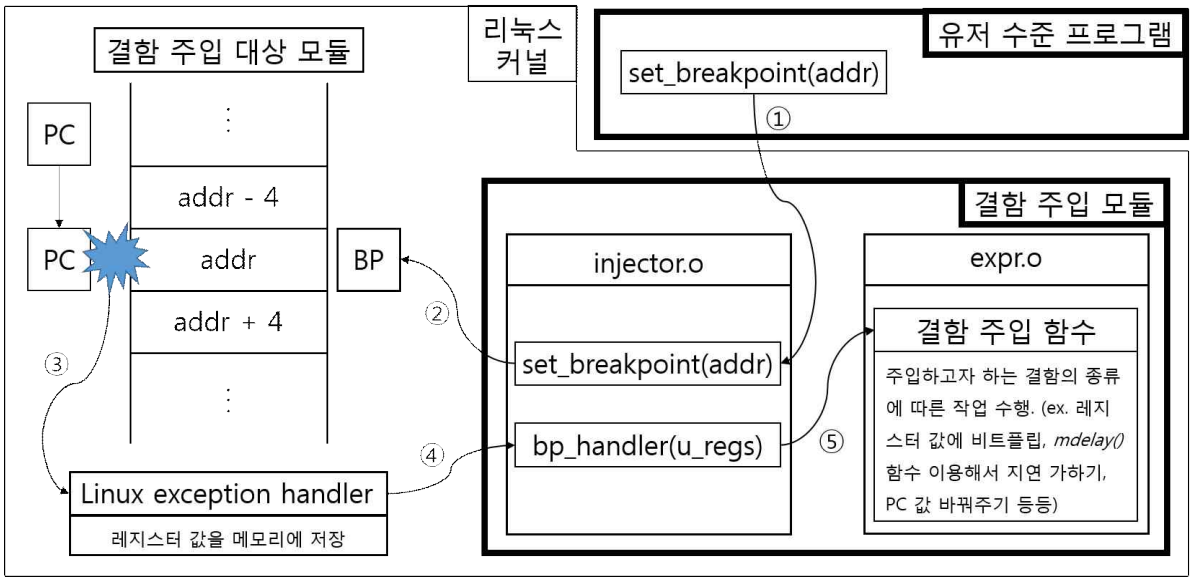

그림 1. 결함 주입 흐름도

① 사용자가 유저 수준 프로그램을 이용하여 커널의 특정 위치에 HWBP를 설정하라는 명령을 내린다. 유저 수준에서 커널 수준으로의 명령 전달은 ioctl() 함수를 통해 이루어진다. 본 기법에서 HWBP를 설정할 수 있는 위치에는 제약이 없고 커널 내부의 임의의 위치에 HWBP를 설정할 수 있다.

② 결함 주입 모듈이 이 명령을 받아서 사용자가 지정한 위치에 HWBP를 설정한다.

③ HWBP가 설정된 명령어가 실행되면 예외상황이 발생하여 리눅스 커널에서 기본적으로 제공되는 예외상황 처리루틴(exception handler)이 실행된다. 여기서 예외상황이 발생할 때의 레지스터 값이 메모리에 저장되어 추후 결함 주입 함수에서 레지스터 값을 변경

할 수 있게 된다. 레지스터의 값을 메모리에 저장하는 기능은 리눅스 커널에서 이미 구현되어 있는 기능이다. 이 기능은 하드웨어와 매우 가까운 수준에서 작동하기 때문에 리눅스 커널에서도 어셈블리어로 구현되어 있다. 이 단계에서 메모리에 저장되는 레지스터 목록은 다음과 같다.

- 범용 레지스터로 쓰이는 R0~R12
- SP (스택 포인터, R13)
- LR (링크 레지스터, R14)
- PC (프로그램 카운터, R15)

만약 본 연구에서 도입한 결함 주입 모듈이 없다면 리눅스 커널은 더 이상 아무런 일도 하지 않고 다시 HWBP가 심어졌던 명령어부터 실행을 재개한다. 본 기법에서는 리눅스 커널이 결함 주입 모듈의 bp_handler()를 부르도록 설정해 놓아서 ④로 넘어가게 한다. 이후 ⑤까지 다 진행한 후에 다시 HWBP가 심어졌던 명령어부터 실행을 재개하게 된다.

④ 리눅스 커널이 결함 주입 모듈의 HWBP 처리루틴(bp_handler())을 부른다. 예외상황 처리루틴에서 메모리에 저장해 놓은 레지스터 값이 인자로 전달이 된다.

ARM 호출 규약에서는 함수에 전달되는 인자를 레지스터에 저장하라고 규정하고 있다. 만약 HWBP가 함수의 시작 부분에 설정되어 있다면 예외상황 처리루틴에서 전달한 레지스터 값을 살펴봄으로써 함수에 전달된 인자의 값을 알 수 있다. 또한 이 값

을 수정하면 나중에 HWBP가 심어졌던 명령어부터 실행을 재개하게 될 때 레지스터 값이 수정된 채로 명령이 재개 된다.
위에서 언급하였듯이 메모리에 저장되어 값을 수정할 수 있는 레지스터에는 프로그램 카운터도 포함되는데 이 값을 수정하면 처음에 HWBP를 설정한 명령어가 아닌 다른 명령어부터 실행을 재개하도록 설정할 수 있다. 이러한 기능을 이용한 실험에 대해서는 5장에서 자세히 설명한다.

⑤ HWBP 처리루틴은 미리 정의되어 있는 결함 주입 함수를 호출한다. 이 결함 주입 함수에서 주입하고자 하는 결함의 종류에 따른 작업을 수행한다. 비트플립 결함과 지연 결함의 경우 미리 정의되어 있는 결함 주입 함수에 옵션 값(비트플립 위치, 지연 시간 등)만 다르게 설정함으로써 원하는 결함을 주입할 수 있다. 한편 맞춤형 결함의 경우 결함을 주입하고자 하는 상황에 맞게 이 결함 주입 함수를 구현해야 되어서 새로운 결함 주입 함수를 만들 때마다 컴파일을 할 필요가 있다.

HWBP를 해제하는 것도 HWBP를 설정하는 과정과 유사한 방식으로 진행된다. 단, 유저 수준 프로그램을 통해서만 해제되는 것이 아니고 사용자가 정해놓은 횟수만큼 결함이 주입되면, 즉 사용자가 정해놓은 횟수만큼 HWBP가 설정된 명령어가 실행되면 결함 주입 모듈이 자동으로 HWBP를 해제하게 된다. 결함 주입 횟수는 사용자가 임의로 정할 수 있고 횟수에 제한 없이 결함을 주입 하도록 설정할 수도 있다.

이 기법을 ODROID-XU4 보드에서 구현함에 있어서 고려해야 할 이슈가 두 가지 있다. 하나는 ODROID-XU4 같은 멀티 코어 시스템

의 경우 커널 코드가 어떤 코어에서 실행될 것인지 사전에 알 수 없다는 점이다. 따라서 HWBP를 설정할 때와 해제할 때는 각 코어의 HWBP가 동시에 설정되고 해제될 수 있도록 해야 본 기법이 올바르게 작동할 수 있다. 또한 전체 결함 주입 횟수가 지정되어 있을 경우 각 코어에서 결함이 주입된 횟수를 합산하여 시스템 전체적으로 결함이 주입된 횟수를 확인하는 과정을 거쳐야 HWBP 해제 작업이 올바로 진행될 수 있다.

다른 하나의 이슈는 Cortex-A15 코어와 Cortex-A7 코어의 HWBP 작동 양식에 미세한 차이가 있다는 점이다. Cortex-A15 코어의 경우 전력 관리 시스템에 의해서 HWBP가 주기적으로 초기화되는 현상이 발견되지만 Cortex-A7 코어에서는 이러한 현상이 발견되지 않는다. 그래서 Cortex-A15 코어의 HWBP가 초기화되더라도 결함 주입 실험이 진행 중이면 HWBP를 재설정 하는 기능이 결함 주입 모듈에 포함되어야 한다.

HWBP는 ARMv7-A 아키텍처를 바탕으로 프로세서를 만들 때 프로세서에 구현하도록 명세한 기능이기 때문에 본 논문에서 실험을 하는데 사용한 ODROID-XU4 보드 이외에도 ARMv7-A 아키텍처를 사용하는 프로세서에서 이 기법을 이용하여 결함 주입 실험을 할 수 있다.

제 5 장 실험

이 장에서는 본 논문에서 소개하는 기법을 이용하여 수행한 결함 주입 실험 3가지를 소개한다. 실험은 리눅스 기반의 안드로이드 운영체제가 동작하는 ODROID-XU4 보드에서 수행되었다. 표 2에 ODROID-XU4 보드의 사양을 정리하였고 그림 2에서 실험 환경을 도식적으로 나타내었다. 결함 주입 실험의 대상이 되는 하드웨어는 ODROID-XU4 보드의 eMMC 제어장치이고, 이 장치는 리눅스 커널과 eMMC 사이에서 인터페이스 역할을 한다. 또한 리눅스 커널은 eMMC 제어장치를 다루는 디바이스 드라이버를 갖고 있다. 이 디바이스 드라이버에 HWBP를 이용해서 결함을 주입하여 eMMC 제어장치에서 발생하는 결함을 모방하는 것이 목표이다. 그림 2에서 번개 모양 표식은 실험에서 결함을 주입하는 위치를 나타낸다.

표 2. ODROID-XU4 사양

SoC	Samsung Exynos 5422 Octa
CPU	4x Cortex-A15 + 4x Cortex-A7
Architecture	ARMv7-A
eMMC Controller Version	5.0
OS	Linux 3.10.9 / Android 4.4.4

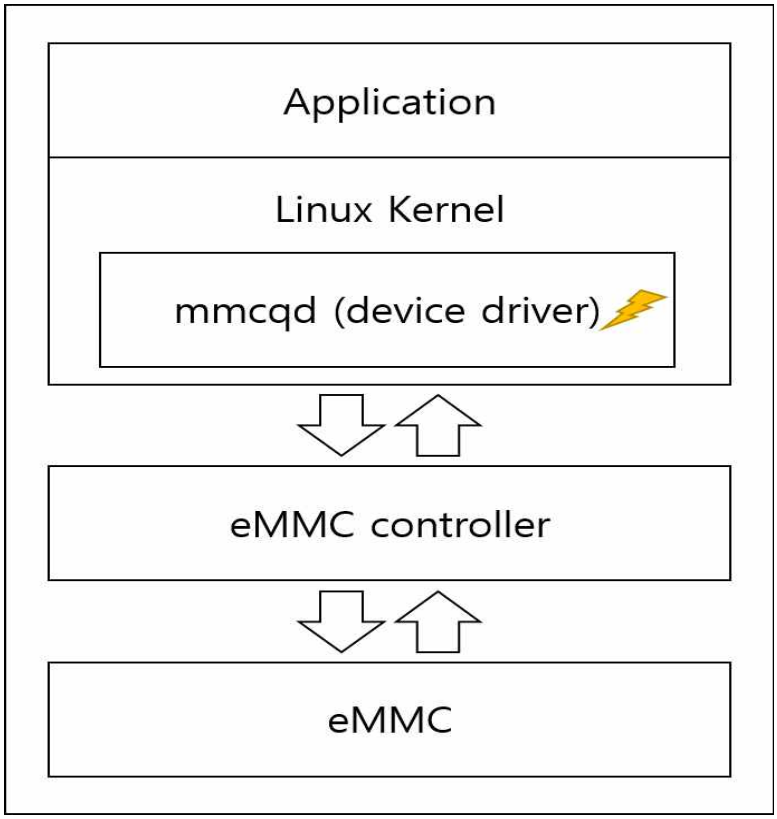

그림 2. 실험 환경
(번개 모양 표식은 결함이 주입되는 위치를 나타냄)

5.1절의 실험은 지연 결함을 주입하는 실험이고, 5.2절은 맞춤형으로 비트플립 결함을 주입하는 실험이다. 5.3절은 맞춤형 주입 기법으로 비트플립과 지연 이외에도 다양한 종류의 결함을 주입할 수 있음을 보여주는 실험이다.

5.1 리눅스 커널과 eMMC 제어장치 사이에서 발생하는 통신 지연 결함을 모방하는 실험

eMMC 제어장치와의 인터페이스를 담당하는 디바이스 드라이버는 mmcqd라는 이름의 커널 쓰레드를 관리한다. eMMC 제어장치에 읽거나 쓰는 명령을 내리는 것은 mmcqd를 통해서 이루어진다. mmcqd는 리눅스 커널이 eMMC 제어장치에 보내는 요청을 큐 형태로 관리한다. 이 요청 큐(reqeust queue)에 요청이 들어오면 mmcqd는 eMMC 제어장치에 명령을 내리고 휴면 상태(sleep)가 된다. eMMC 제어장치는 mmcqd로부터 받은 명령을 수행하고 나면 인터럽트 신호를 발생시킨다. 리눅스 커널은 이 인터럽트 신호를 받으면 휴면 상태에 있는 mmcqd를 깨워서 eMMC 제어장치로부터 온 응답을 처리하게 한다. 그 후 mmcqd는 요청 큐에서 다음 요청을 받아 다시 eMMC 제어장치에 명령을 내리는 과정을 반복한다.

리눅스 커널과 eMMC 제어장치 사이에서 발생하는 통신 지연을 모방하기 위해 mmcqd가 eMMC 제어장치에 명령을 내릴 때 호출하는 함수 중에 하나를 선택해서 HWBP를 설정하였다. 그리고 결함 주입 함수에서는 mdelay()를 이용해서 HWBP가 설정된 함수의 수행을 지연시켰다. 결함 주입 횟수는 무제한으로 설정하여 실험자가 HWBP를 해제하기 전까지 해당 함수가 불릴 때마다 계속 지연이 발생하도록 하였다. 즉 지연 시간을 1초로 설정하면 HWBP가 설정된 함수가 불릴 때마다 매번 1초의 지연이 발생하는 것이다. 지연 시간은 100ms부터 시작해서 5초 까지 늘려가면서 시스템의 반응을 살펴보았다.

위와 같이 HWBP를 설정하고 안드로이드의 Gallery 응용을 이용해 지연 결함 주입의 효과를 살펴보았고 응용 수준에서 지연이 발생함을 확인할 수 있었다. 즉 스크린 샷을 찍고, Gallery 응용을 실행하여 찍은 스크린 샷을 열람하고, 스크린 샷을 지우는 동작을 수행할 때 커널

에서의 지연으로 인해서 이러한 동작들도 느리게 수행되는 것을 관찰할 수 있었다. Gallery 응용만 지연될 뿐 시스템 상에서의 다른 이상 현상은 발견되지 않았다. 또한 HWBP를 해제하여 지연 결함을 가하는 것을 중단하면 Gallery 응용은 지연 없이 정상적으로 실행됨을 확인할 수 있었다.

이 실험을 통해서 본 논문에서 제안하는 기법을 이용하여 커널 수준에서 지연 결함을 주입할 수 있음을 확인할 수 있다. 또한 이 기법은 결함 주입 여부를 시스템이 실행되는 도중에 켜고 끄면서 시스템의 반응을 볼 수 있다는 장점을 가짐을 알 수 있다.

5.2 eMMC 제어장치로부터 오는 응답에 에러가 있는 상황을 모방하는 실험

mmcqd는 eMMC 제어장치로부터 명령에 대한 응답을 받은 후에 mmc_blk_err_check()에서 그 응답에 에러가 있는지 확인하게 된다. 이 때 응답의 내용은 함수의 인자로 넘어가기 때문에 이 함수의 시작 지점에 HWBP를 설정하면 인자의 내용에 접근할 수 있다.

그림 3은 결함 주입 대상이 되는 함수의 실제 코드이고 그림 4는 이 실험에서 사용하는 결함 주입 함수이다. 그림 3의 함수의 시작 부분에 HWBP를 설정하면 이 함수가 시작되기 전에 그림 4의 함수가 실행되어 eMMC 제어장치로부터 넘어온 응답에 수정을 가하게 된다. 그림 3에서 사각형으로 둘러싸인 부분이 eMMC 제어장치로부터의 응답에 에러가 있는지 확인하는 부분인데 그림 4에서 사각형으로 둘러싸인 부분을 보면 이 값을 에러가 있는 것으로 바꾸어 준다. 이러한 방식으로

```
static int mmc_blk_err_check(struct mmc_card *card,
                             struct mmc_async_req *areq)
{
        struct mmc_queue_req *mq_mrq = container_of(areq, struct mmc_queue_req,
                                                    mmc_active);
        struct mmc_blk_request *brq = &mq_mrq->brq;
        struct request *req = mq_mrq->req;
        int ecc_err = 0;
        if (!card->ext_csd.cmdq_mode_en) {
                if (brq->sbc.error || brq->cmd.error || brq->stop.error ||
                    brq->data.error) {
                        switch (mmc_blk_cmd_recovery(card, req,
                                                     brq, &ecc_err)) {
                        case ERR_RETRY:
                                return MMC_BLK_RETRY;
                        case ERR_ABORT:
                                return MMC_BLK_ABORT;
                        case ERR_NOMEDIUM:
                                return MMC_BLK_NOMEDIUM;
                        case ERR_CONTINUE:
                                break;
                        }
                }
        }
```

그림 3. 응답에 에러가 있는 상황을 모방하는 실험의 결함 주입 대상 함수

eMMC 제어장치로부터의 응답에 에러가 있는 상황을 모방할 수 있다.

mmcqd가 이러한 에러를 발견했을 때 하는 행동은 일단 동일한 명령을 재시도하는 것이다. 그러다가 반복해서 에러가 발생한다면 그 명령을 취소한다. 이 실험을 통해서 인위적으로 이러한 상황을 발생시킬 수 있었다. 또한 취소되는 명령의 수가 누적되면 리눅스 커널이 패닉(panic) 메시지를 출력하고 시스템이 죽는 것을 관찰할 수 있었다. 이 실

```
static int artificial_err_mmc_0_func(struct pt_regs *regs, void *UNUSED)
{
        struct mmc_async_req *areq;
        struct mmc_blk_request *brq;
        struct mmc_queue_req *mq_mrq;

        areq = (struct mmc_async_req *)regs->ARM_r1;
        mq_mrq = container_of(areq, struct mmc_queue_req, mmc_active);
        brq = &mq_mrq->brq;
        brq->cmd.error = -EILSEQ;

        return 0;
}
```

그림 4. 응답에 에러가 있는 상황을 모방하는 실험의 결함 주입 함수

험은 결함 주입 횟수를 어떻게 설정하느냐에 따라 시스템의 반응이 달라진다. mmcqd가 실패한 명령을 다시 시도해보는 횟수보다 적게 결함 주입 횟수를 설정하면 리눅스 커널은 죽지 않고 정상적으로 작동되지만 그보다 많게 결함 주입 횟수를 설정하면 위에서 설명하듯이 종국에는 시스템이 죽게 된다.

이 실험을 통해서 본 논문에서 제안하는 기법을 사용하면 상황에 따라 다양하게 정의될 수 있는 결함 주입 함수를 이용해 실험을 진행하는 것이 가능함을 알 수 있다.

5.3 eMMC 제어장치로부터 응답이 없는 상황을 모방하는 실험

이 실험에서는 eMMC 제어장치로부터 인터럽트 신호를 받은 리눅스 커널이 mmcqd를 깨우는 것을 막아서 eMMC 제어장치로부터 응답이

```
...
regs->ARM_pc = kallsyms_lookup_name("mmc_request_done") + 0xa8;
...
```

그림 5. 실행 재개될 명령어를 조작하는 결함 주입 함수 코드

없는 상황을 모방한다. mmcqd를 깨우는 작업은 mmc_request_done()에서 수행되는데 이 함수 시작 부분에 HWBP를 설정해서 mmcqd를 깨우는 명령어를 건너뛰도록 한다.

이를 위해서는 그림 5에서 설명하는 바와 같은 결함 주입 함수가 필요하다. Pre-fetch abort 예외상황 처리가 끝나면 기본적으로 HWBP가 설정된 명령어부터 실행을 재개하게 되지만 실행이 재개될 명령어를 조작함으로써 특정 명령어를 건너뛰게 할 수 있다.

이 실험을 수행하면 리눅스 커널이 더 이상 mmcqd를 깨울 수 있는 방법이 없게 되어 eMMC 카드에 대한 접근이 필요한 프로세스(예컨대 콘솔에서의 cp 명령이나 스크린 샷을 찍는 명령)는 성공적으로 실행될 수 없게 된다. 하지만 eMMC 카드에 대한 접근이 필요 없는 프로세스(ls나 cd 명령, 혹은 /proc 디렉터리에 대한 접근)는 성공적으로 실행됨을 관찰할 수 있었다.

제 6 장 결론

본 논문에서는 하드웨어 수준에서 제공되는 자원인 하드웨어 브레이크포인트를 이용하여 커널 수준에서 결함을 주입하는 기법을 소개하였다. 이 기법은 결함 주입의 대상이 되는 모듈의 수정 없이 결함 주입 실험을 수행할 수 있고, 커널의 특정 위치를 지정하여 결함을 주입할 수 있다는 장점을 가진다. 이 기법을 이용하여 다양한 종류의 결함을 주입할 수 있음을 실제 보드에서 진행된 실험을 통해 보였고, 특히 결함을 주입하고자 하는 모듈을 분석하여 맞춤형(custom) 결함 주입 실험을 수행할 수 있어서 다양한 상황에 맞게 결함을 주입할 수 있음을 보였다. 이 기법을 이용하면 ARMv7-A 아키텍처에서 구현된 임베디드 리눅스 환경에서 하드웨어 수준의 오류를 재현함으로써 시스템의 견고성과 신뢰성을 높이는데 기여할 수 있다.

참 고 문 헌

[1] R. Natella, D. Cotroneo, J. Duraes, and H. Madeira, “On Fault Representativeness of Software Fault Injection”, in IEEE Transactions on Software Engineering, vol. 39, no. 1, 2013

[2] ODROID, ODROID-XU4 wiki. http://odroid.com/dokuwiki/doku.php?id=en:odroid-xu4.

[3] T. Jarboui, J. Arlat, Y. Crouzet, and K. Kanoun, “Experimental Analysis of the Errors Induced into Linux by Three Fault Injection Techniques”, in Proceedings of the International Conference on Dependable Systems and Networks (DSN), 2002.

[4] S. Winter, O. Schwahn, R. Natella, N. Suri, and D. Cotroneo, “No PAIN, no gain? The Utility of PArallel Fault INjections”, in IEEE International Conference on Software Engineering (ICSE), 2015.

[5] V. Rubanov and E. Shatokin, “Runtime Verification of Linux Kernel Modules Based on Call Interception”, in IEEE International Conference on Software Testing, Verification and Validation, 2011.

[6] K. Cong, L. Lei, Z. Yang, and F. Xie, “Automatic Fault Injection for Driver Robustness Testing”, in Proceedings of the International Symposium on Software Testing and Analysis (ISSTA), 2015.

[7] 박순교, 김영필, 유혁, “최신 리눅스 커널 적용 가능한 장치 드라이버 결함 주입기와 분석 도구의 설계 및 구현”, 정보과학회논문지 시스템 및 이론, vol. 41, no. 1, 2014.

[8] A. Albinet, J. Arlat, and J. Fabre, “Characterization of the Impact of Faulty Drivers on the Robustness of the Linux Kernel”, in Proceedings of the International Conference on Dependable Systems and Networks (DSN), 2004.

[9] J. Carreira, H. Madeira, and J. Silva, “Xception: A Technique for the Experimental Evaluation of Dependability in Modern Computers”, in IEEE Transactions on Software Engineering, vol. 24, no. 2, 1998

[10] P. Folkesson, S. Svensson, and J. Karlsson, “A Comparison of Simulation Based and Scan Chain Implemented Fault Injection”, in Proceedings of the International Symposium on Fault-Tolerant Computing, 1998

[11] ARM, ARM Architecture Reference Manual ARMv7-A and ARMv7-R edition, 2014.

Abstract

Kernel-level Fault Injection Technique Using Hardware Breakpoint

Namgoo Lee
Department of Computer Science and Engineering
The Graduate School
Seoul National University

The fault injection technique is used to enhance system's robustness by emulating hardware fault or by verifying software reliability. The technique can be applied to the Linux kernel which is widely used in the embedded systems. Previous works on the Linux kernel have some constraints: the fault injection target must be mutated at the code level or the diversity of the injected fault is limited. This paper introduces a fault injection technique using hardware breakpoint that can be employed to various kernel level fault injection experiments. This technique is implemented by adding a new kernel module with fault injection capability without modifying the target kernel code. The suggested technique is used

to carry out some experiments on the eMMC controller inside the ODROID-XU4 board based on the ARMv7-A architecture running Android.

Keywords: fault injection, Linux kernel, hardware breakpoint, ARMv7-A, ODROID-XU4, eMMC controller, device driver

Student Number: 2014-22687