JKSCI

한국컴퓨터정보학회논문지
Journal of The Korea Society of Computer and Information
Vol. 27 No. 6, pp. 1-9, June 2022
https://doi.org/10.9708/jksci.2022.27.06.001

Fault Injection System for Linux Kernel Modules

Sunghoon Son*

*Professor, Dept. of Computer Science, Sangmyung University, Seoul, Korea

[Abstract]

In this paper, we propose a general-purpose fault injection system for Linux loadable kernel modules. The fault injection system enables software developers and testers to inject various kinds of faults easily into user-specified kernel modules in user-controlled manner. The proposed system also provides workload generation in order to make injected faults be exposed effectively, By experiments, we show that the fault injection system correctly injects faults into Linux kernel modules. The proposed system can be utilized as a useful tool for testing during kernel module development It is also useful for studies on kernel behaviour analysis and fault isolation and recovery.

▸ **Key words**: Linux, Kernel module, Fault injection, Test tool

[요 약]

본 논문에서는 리눅스 커널 모듈을 대상으로 다목적으로 사용할 수 있는 폴트 주입 시스템을 제안한다. 제안된 폴트 주입 시스템은 사용자가 지정한 커널 모듈을 대상으로 다양한 유형의 폴트를 사용자가 지정한 방식으로 발생시킬 수 있다. 또한 일단 커널 모듈에 폴트가 주입된 후에는 시스템의 동작 과정에서 주입된 폴트가 잘 드러날 수 있도록 하는 워크로드를 생성하는 기능도 함께 제공한다. 일련의 시험을 통해 제안된 폴트 주입 시스템이 효과적으로 동작함을 확인했다. 제안된 폴트 주입기는 커널 모듈 개발 및 테스트, 커널 동작에 대한 분석 연구, 디바이스 드라이버 등에 대한 폴트 격리 및 복구 시스템 연구 등에서 유용한 도구로 활용될 수 있을 것이다.

▸ **주제어**: 리눅스, 커널 모듈, 폴트 주입, 테스트 도구

• First Author: Sunghoon Son, Corresponding Author: Sunghoon Son
*Sunghoon Son (shson@smu.ac.kr), Dept. of Computer Science, Sangmyung University
• Received: 2022. 05. 11, Revised: 2022. 06. 08, Accepted: 2022. 06. 08.

http://www.ksci.re.kr pISSN:1598-849X | eISSN:2383-9945

I. Introduction

폴트 주입(fault injection)은 컴퓨터 시스템에 비정상적인 방법으로 스트레스가 가해지는 상황에서 시스템이 어떤 방식으로 동작하는가를 이해하기 위한 시험(test) 기법이다. 폴트 주입에는 보통 하드웨어적인 폴트 주입, 소프트웨어 기반 폴트 주입, 두 방법을 결합한 하이브리드 폴트 주입 등 다양한 기법이 존재한다. 과거에는 주로 하드웨어 기반의 폴트 주입 기법들이 사용되었으나 최근에는 소프트웨어 개발 과정에서 다양한 소프트웨어 기반 폴트 주입 기법들이 사용되고 있다. 소프트웨어 테스트 시 인위적으로 폴트를 주입하여 다양한 코드 실행 경로가 실행되도록 함으로써 테스트의 커버리지를 확장할 수 있다. 특히 프로그램 내의 에러 처리 함수와 같이 평상시에는 자주 실행되지 않는 부분까지 시험하는데 유용하다. 폴트 주입은 스트레스 테스트 과정에서도 사용되며 견고한 소프트웨어를 개발하는데 중요한 도구로 여겨진다.

폴트(fault)가 시스템의 고장(failure)으로 전파되는 과정은 흔히 전형적인 일련의 순환 과정을 따르는 것으로 알려져 있다[1]. 시스템이 동작하는 과정에서 폴트는 에러(error)의 원인이 되기도 한다. 폴트에 의해 발생한 에러가 다른 에러들을 일으킬 수도 있고, 시스템 경계까지 전파되어 사용자나 관리자에게 관측될 수 있게 된다. 이렇게 관찰된 에러를 고장이라고 한다. 이를 폴트-에러-고장의 순환 과정이라고 하며, 이는 컴퓨터 시스템의 신뢰도에 중요한 영향을 미치는 메커니즘이다.

본 논문에서는 리눅스의 커널 모듈(kernel module)을 대상으로 테스트 등을 위해 인위적으로 폴트를 주입할 수 있는 시스템을 제안한다. 제안된 폴트 주입 시스템은 우선 별도의 명령(command)을 제공하여 사용자가 폴트를 주입하고자 하는 대상 커널 모듈, 폴트의 유형, 폴트 발생 주기 등을 지정할 수 있도록 한다. 이러한 정보를 바탕으로 폴트 주입 시스템은 폴트 발생을 에뮬레이션할 수 있는 커널 모듈을 생성하여 이를 커널에 동적으로 로드하게 된다. 이 커널 모듈은 로드 과정에서 리눅스 커널의 Kprobes 기능을 활용하여 타겟 커널 모듈에 폴트를 주입하게 된다. 제안하는 폴트 주입 시스템은 또한 주입된 폴트가 시스템 동작 중에 잘 발현될 수 있도록 폴트 주입과 관련된 커널 서브시스템이 동작시키는 시스템 콜을 호출하는 프로그램을 실행시킴으로써 주입된 폴트가 즉시 드러날 수 있도록 했다.

실험을 통해 제안된 폴트 주입 시스템이 효과적으로 동작함을 확인했다. 우선 실험용 타겟 디바이스 드라이버 모듈을 제작하여 리눅스 커널에 로드하고, 이 디바이스 드라이버 모듈에 세 가지 유형의 폴트가 일정한 주기로 발생하도록 각각 주입한 후, 이 디바이스 드라이버 모듈이 구동될 수 있도록 응용 프로그램을 수행하여 정상적으로 폴트가 발생함을 확인했다. 제안된 폴트 주입기는 커널 모듈의 테스트나 커널 동작에 대한 분석 연구, 폴트 격리 및 복구 시스템 연구 등에서 유용한 도구로 활용될 수 있을 것으로 기대한다.

본 논문은 다음과 같이 구성된다. 먼저 2장에서는 폴트 주입 시스템의 필요성을 제시하고 이와 관련된 기존의 연구들을 살펴본다. 3장에서는 본 논문에서 제안하는 폴트 주입 시스템을 자세히 소개하고, 4장에서는 폴트 주입 시스템에 대한 실험 결과를 설명하며, 마지막으로 5장에서 결론을 맺는다.

II. Preliminaries

이 장에서는 리눅스 시스템의 커널 모듈을 위한 폴트 주입 시스템의 필요성에 대해 설명하고, 폴트 주입 기법에 대한 기존의 관련 연구들을 소개한다.

폴트 주입기는 소프트웨어를 테스트하는데 중요한 도구이다. 특히 일반적인 응용 프로그램이 아닌 커널 모듈과 같은 운영체제 커널의 개발과 시험 과정에서 다음과 같은 다양한 분야에서 도구로 사용될 수 있다. 첫째, 커널 모듈이 운영체제의 동작에 미치는 기능적, 성능적 영향을 연구하는데 사용될 수 있다. 커널 모듈은 일단 로드된 후에는 커널의 일부로 동작한다. 따라서 커널 모듈의 동작이 커널 전체에 영향을 미칠 수 있고, 오동작 하는 경우 시스템 크래쉬 등 치명적인 결과를 초래할 수도 있다. 폴트 주입기는 이러한 커널 모듈의 오류가 커널 전체에 미치는 영향에 대한 연구를 수행하는데 있어 매우 유용한 도구가 될 수 있다[2]. 폴트 주입기가 없는 경우 사용자 수준에서 부적절한 인자를 포함하는 시스템 콜을 호출하는 방식을 사용하거나, 소스 코드, 메모리 덤프, 에러 로그 등을 분석하는 간접적이고 피상적인 방법만으로 운영체제의 동작을 평가할 수밖에 없다.

둘째, 커널 모듈의 개발자나 테스트 담당자가 해당 커널 모듈을 시험하는 과정에서 폴트 주입기를 사용할 수 있다. 만일 개발 과정 중이라면 개발자가 직접 프로토타입 수준의 커널 모듈의 소스 코드 상에서 다양한 실행 경로가 빠짐없이 시험될 수 있도록 적절한 시험 데이터를 주입하는 방식으로 화이트 박스 테스트를 수행할 수 있다. 한편 개발의 마지막 단계에서는 개발된 커널 모듈의 아래 계층에

대한 에뮬레이션 모듈을 제작하여 해당 커널 모듈을 시험하는 일종의 블랙박스 테스트를 진행할 수 있다. 예를 들면 디바이스 드라이버 커널 모듈에 대해서 해당 드라이버가 동작하는 하드웨어의 에뮬레이션 모듈을 작성한 후, 이 에뮬레이션 모듈에 폴트를 주입하여 해당 디바이스 드라이버를 테스트하는 방식이 여기에 해당한다.

셋째, 폴트 주입기는 디바이스 드라이버의 폴트 격리 및 복구 시스템에서 활용될 수 있다. 보통 디바이스 드라이버는 커널 모듈의 형태로 제작된다. 연구에 따르면 운영체제 크래쉬의 대부분은 디바이스 드라이버의 폴트에 의한 것이라고 한다. 특히 리눅스에서는 디바이스 드라이버 소스 코드의 버그가 다른 커널 코드의 버그에 비해 몇 배 이상이 되는 것으로 조사되기도 했다. 이로 인해 디바이스 드라이버에 폴트가 발생하더라도 전체 시스템은 정상적으로 동작하는 가운데 해당 드라이버 모듈만 복구하는 기법들도 연구된 바 있다[3]. 이러한 디바이스 드라이버에 대한 폴트 격리 및 회복 기법들에 대한 연구에서도 폴트 주입기가 유용한 도구로 사용될 수 있을 것이다.

그간 폴트 주입에 대한 연구가 다양하게 진행되었다. [4]는 다양한 폴트 주입 기법과 주요 도구들에 대한 전반적인 리뷰를 제공한다. 또한 [5]에서는 SW 기반 폴트 주입 기법들을 중심으로 각 기법들에 대해 자세한 개요를 제공하고 신뢰도(dependability) 측정을 하는 목적에 따라 어떤 폴트 주입 기법을 선택해야 하는지 기준을 제시했다.

Xception[6]은 중단점 (breakpoint) 레지스터와 같이 하드웨어가 제공하는 기능을 사용하는 디버깅 도구와 성능 모니터링 도구를 사용해 효과적인 폴트 주입기를 구현했다. [7]에서는 분산 환경을 포함하여 신뢰성 연구를 지원하는 범용 도구를 개발했다. FAUmachine 프로젝트는 디스크 읽기/쓰기 폴트, 네트워크 패킷 손실, 레지스터나 메모리의 비트 폴트 등의 폴트를 주입할 수 있는 하드웨어 에뮬레이터를 제공한다. [8]은 리눅스의 디바이스 드라이버에 폴트를 주입하고 이 폴트가 커널 내에서 전파되는 과정 및 결과에 대해 분석했다. 리눅스 커널 v2.6.20 이후에서는 주요 커널 서브시스템의 특정 지점에 폴트를 주입할 수 있는 기능이 추가되었다[9]. 폴트 주입이 가능한 지점은 리눅스 커널 중 슬랩 할당 함수, 페이지 할당 함수 등이 해당한다. [10]은 GDB 디버거를 활용하여 응용 프로그램의 바이너리에 대해 레지스터 값이나 분기 명령의 타겟의 비트를 플립할 수 있는 폴트 주입기를 소개하고 있다.

특정 응용 분야를 위한 폴트 주입기도 널리 연구되고 있다. [11]은 파이썬 프로그램을 위한 폴트 주입 도구를 소개하고 있다. [12]는 머신 러닝 플랫폼 TensorFlow 상의 응용을 위한 폴트 주입기를 제안하였는데, 자율 주행과 같이 안전이 중요한 도메인의 응용을 시험하는데 유용한 도구이다. [13]에서는 표준 라이브러리 함수에 폴트를 주입하는 방식으로 자동화된 크래쉬 복구 솔루션을 제안하였다. 커널 수준에 적용할 수 있는 폴트 주입기로는 주로 디바이스 드라이버에 폴트를 주입하는 방식으로 버그를 찾아내는 연구들도 진행되었는데, 우선 [14]에서는 소프트웨어를 테스트하는 기법 중 자주 실행되지 않는 코드를 강제로 수행시킬 수 있는 입력을 생성해내는 퍼징(fuzzing) 기법을 적용하여 리눅스 디바이스 드라이버의 오류 처리 함수를 테스트하는 기법을 제안하고 있다. 다만 이 연구는 전체적인 테스트 기법 등에 중점을 두고 있고, 폴트 주입 자체는 별도의 도구를 사용하기보다 단순한 코드 인스트루멘테이션(instrumentation) 방식을 통해 수행하고 있다. [15]는 폴트 주입 기법을 사용하여 리눅스의 상용 USB 디바이스 드라이버들에서의 결함을 찾아내는 POTUS 시스템을 제안하고 있다.

과거에는 하드웨어를 기반으로 하거나 하드웨어와 소프트웨어가 결합된 형태의 폴트 주입기가 대세를 이루었으나 최근 소프트웨어 기반 폴트 주입기가 많이 연구되는 추세이다. 최근의 폴트 주입 도구에 대한 연구는 대부분 일반적인 응용 프로그램을 대상으로 하거나 특정 응용 분야에서만 사용할 수 있는 폴트 주입기가 주류를 이루고 있고, 커널을 대상으로 하는 폴트 주입 도구는 상대적으로 적다. 일부 커널을 대상으로 하는 연구들은 대부분 디바이스 드라이버의 결함을 찾는 목적으로 폴트 주입이 이루어지는 경우이다. 특히 이러한 연구들은 테스트 대상이 되는 디바이스 드라이버에 대해 효율적으로 결함을 찾아낼 수 있는 테스트 케이스를 설계하는데 중점을 두고 있으며 폴트 주입 자체는 비중 있게 다루지 않고 있다. 이에 반해 본 논문에서는 리눅스 운영체제 상에서 본격적으로 커널 코드를 대상으로 하는 폴트 주입 도구를 제안하고자 한다. 제안하는 폴트 주입 도구는 특정 하드웨어 기능의 사용을 배제한 순수한 소프트웨어 기반의 폴트 주입 도구로서, 디바이스 드라이버 뿐만 아니라 일반적인 리눅스 커널 모듈을 대상으로 별도의 소스 코드의 수정 없이 사용자의 요구에 따라 폴트 주입 범위나 유형, 주기 등을 바꿔가며 다양한 방식으로 폴트를 주입할 수 있는 특징을 가진다.

III. Fault Injection System

1. Overall structure of fault injection system

우선 폴트 주입 시스템에 대해 본격적으로 소개하기에 앞서 제안하는 시스템의 설계 목표부터 밝히고자 한다. 본 폴트 주입 시스템은 다음과 같은 목표들을 가지고 설계하였다.

커널 모듈 단위의 폴트 주입: 폴트 주입의 대상을 리눅스 커널 전체로 하기보다 동적으로 로드된 커널 모듈을 대상으로 했다. 따라서 시스템 전 범위를 폴트 주입 대상으로 하던 기존의 사례들에서 사용하던 폴트 타입과 폴트 발생 주기라는 두 가지 핵심 인자 외에 '타겟 모듈'이라는 인자를 추가하여 타겟 모듈, 폴트 타입, 폴트 발생 주기의 세 가지 인자를 바탕으로 모듈 단위 폴트 주입 방법을 제시하고자 했다.

타겟 모듈의 소스 코드 수정 금지: 타겟 모듈의 소스 코드를 변경하지 않는 폴트 주입을 목표로 했다. 폴트를 주입하기 위해 타겟 커널 모듈의 소스 코드를 수정하고 그에 따라 컴파일을 다시 할 필요 없이 타겟 모듈이 커널에 적재되어 동작하는 상태에서 동적으로 폴트를 주입할 수 있도록 했다.

사용자 수준에서 제어 가능한 폴트 주입 인터페이스 제공: 폴트 주입 시스템의 사용자는 커널 모듈 개발자나 시험 담당자이다. 해당 폴트 주입기 사용자는 개발과 시험이라는 본래의 목적에 집중할 수 있도록 폴트 주입의 범위, 유형, 그리고 주기 등을 사용자 수준에서 제어할 수 있도록 정형화된 폴트 주입 인터페이스를 제공하고자 했다.

타겟 모듈의 특성을 고려한 부하 생성 기능 제공: 폴트가 주입된 타겟 모듈이 지정된 조건에 도달해 폴트가 발생하는 것을 촉진하기 위해 폴트 주입 시스템은 타겟 모듈에 부하를 가할 수 있다. 이 과정에서 타겟 모듈이 가지는 특성을 분석해 폴트와 관련 없는 커널 서브시스템에 대한 부하는 최소화하고 타겟 모듈이 속한 서브시스템에 부하를 집중함으로써 타겟 모듈에 대해 보다 정교한 폴트 발생이 가능하도록 했다.

하드웨어가 제공하는 특수 기능의 사용 배제: 앞서 사례 연구에서도 볼 수 있듯이 기존의 폴트 주입 시스템은 CPU가 제공하는 디버그 레지스터와 같은 기능을 활용하는 경우가 많다. 제안하는 폴트 주입 시스템에서는 특정한 하드웨어 기능을 가정하지 않는 기법을 목표로 했다.

기존 리눅스 시스템의 기능을 최대한 활용: 제안하는 폴트 주입 시스템은 이미 기존 리눅스 시스템에서 제공하는 기능을 최대한 그대로 사용할 수 있도록 했다.

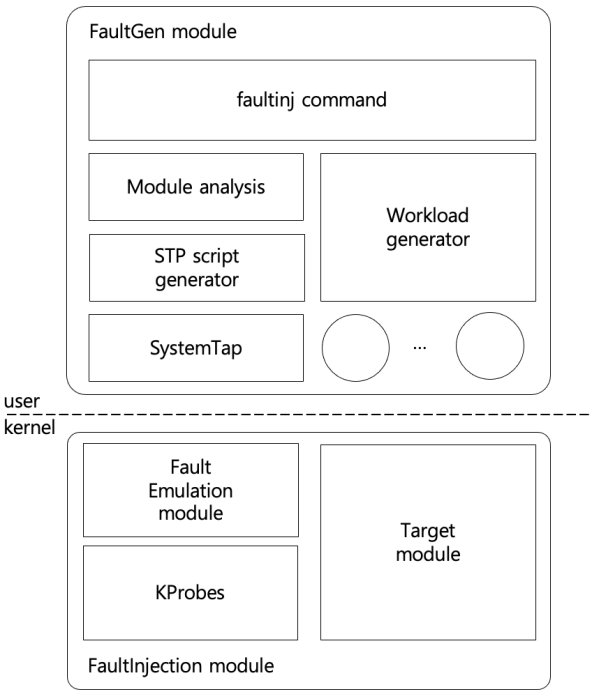

Fig. 1. Fault injection system

본 논문에서 제안하는 폴트 주입 시스템의 전체 구조는 Fig. 1과 같다. 그림에서 보는 바와 같이 폴트 주입 시스템은 사용자 수준(user-level)에서 동작하는 FaultGen 모듈과 커널 수준(kernel-level)에서 동작하는 FaultInjection 모듈로 구성된다. 우선 FaultGen 모듈은 사용자가 폴트를 주입할 수 있도록 명령(command)을 제공한다. FaultGen 모듈은이 명령을 통해 전달된 폴트 요구사항에 따라 폴트 에뮬레이션 커널 모듈 (fault emulation module)의 소스 코드를 생성하고, 이를 빌드하여, 커널에 로드한다. 이 폴트 에뮬레이션 모듈이 커널에 로드되는 과정에서 타겟 커널 모듈에 실제 폴트가 주입된다. FaultGen 모듈의 또 다른 기능은 폴트가 주입되고 폴트 에뮬레이션 모듈이 동작하고 있는 상태에서 타겟 커널 모듈에 주입된 폴트가 실제로 발현될 수 있도록 타겟 커널 모듈과 연관된 커널의 서브시스템에 적절한 부하를 가하는 것이다.

한편 커널 수준에서 동작하는 FaultInjection 모듈은 FaultGen 모듈에 의해 동적으로 로드된 폴트 에뮬레이션 모듈과 리눅스 커널이 제공하는 기능인 KProbes의 결합으로 구성된다. 폴트 에뮬레이션 모듈은 타겟 모듈 내에 사용자가 지정한 지점에 일종의 중단점을 설정하고, 타겟 커널 모듈의 실행이 이 중단점에 도달하게 되면 제어를 가로채서 사용자가 원하는 폴트 에뮬레이션 동작을 수행하게 된다. 폴트 에뮬레이션 모듈은 이 과정에서 KProbes가 제공하는 프로그래밍 API를 사용하여 커널 내 특정 함수의 결과나 변수 값 등을 임의로 모니터링하는 기능을 수행한다.

2. FaultGen module

앞서 언급한대로 FaultGen 모듈은 폴트 에뮬레이션 모듈의 생성 및 로드 기능과 타겟 모듈에 대한 부하 생성 기능을 수행한다. 이를 위해 폴트 주입의 대상이 되는 타겟 커널 모듈의 이름, 발생을 원하는 폴트의 유형, 폴트 발생 주기 등의 정보가 필요하다. FaultGen 모듈은 사용자가 이러한 정보들을 명시할 수 있도록 명령(command)을 제공하여 사용자로부터 해당 정보를 얻는다. FaultGen 모듈의 세부 구조는 Fig. 2와 같다.

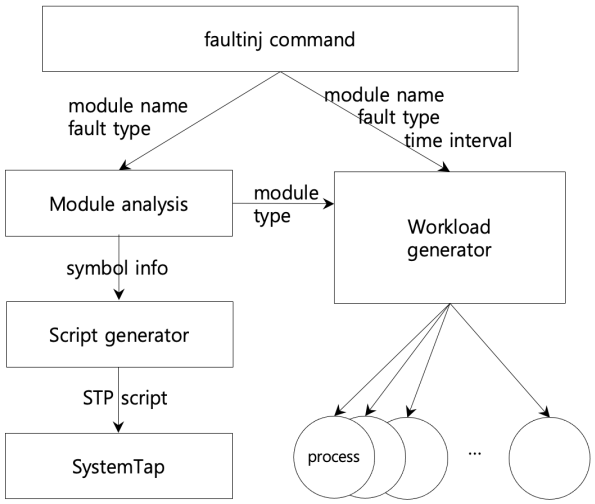

Fig. 2. FaultGen module

FaultGen 모듈은 커널 모듈을 생성하기 위해 리눅스의 SystemTap 기능을 활용한다. SystemTap은 폴트 에뮬레이션 모듈과 같은 동적 커널 모듈을 생성하고 실행하기 위해 SystemTap 스크립트(STP script)를 필요로 한다. FaultGen 모듈이 사용자로부터 얻은 정보로부터 STP 스크립트를 생성하는 과정은 다음과 같다. 우선 사용자로부터 얻은 정보를 타겟 커널 모듈의 심볼들과 비교하여 일치되는 심볼들에 대한 목록을 작성한다. 이어서 사용자가 제시한 폴트 유형을 기반으로 모델 라이브러리 저장소로부터 STP 스크립트를 구성하는 기본 함수들의 템플릿을 추출한다. FaultGen 모듈의 코드 생성기는 심볼 목록을 이용하여 함수 템플릿들을 각각 유효한 함수로 전환하게 된다. 마지막으로 생성된 모든 함수들을 통합함으로써 SystemTap의 입력이 되는 단일 STP 스크립트가 만들어진다. 이어서 SystemTap은 STP 스크립트를 커널 모듈에 대한 C 소스 코드로 변환하게 되는데, 이것이 바로 폴트 에뮬레이션 커널 모듈의 소스 코드이다. SystemTap은 C 컴파일러 수행하여 폴트 에뮬레이션 모듈을 빌드한 후, 이를 커널에 동적으로 로드하는 작업까지 수행한다. 폴트 에뮬레이션 커널 모듈은 커널에 로드되는 과정에서 타겟 모듈 내의 원하는 지점에 프로브를 심고 해당 프로브를 활성화시킨다. 폴트 에뮬레이션 모듈은 추후 해당 프로브가 활성화되면 실행되는 사용자 정의 프로브 핸들러 함수를 포함하고 있다.

FaultGen 블록의 또 다른 기능은 주입된 폴트가 실제 발생하기 위한 환경을 제공하는 것이다. 이러한 역할은 타겟 커널 모듈이 속한 커널 서브시스템에 부하를 가함으로써 가능하다. 워크로드 생성기는 사용자가 제공한 워크로드 프로그램 수행 정보를 기반으로 실제 부하 발생을 위한 프로세스를 생성함으로써 적절한 부하가 발생되도록 한다. 여기서 워크로드 프로그램으로는 사용자가 직접 제작한 테스트 프로그램, 상용 프로그램, 공개 소스 기반의 시험 도구 등 다양한 프로그램이 사용될 수 있다.

3. FaultInjection module

FaultInjection 모듈은 FaultGen 모듈이 커널에 적재한 폴트 에뮬레이션 모듈과 리눅스 커널의 KProbes 시스템으로 구성된다(Fig. 3).

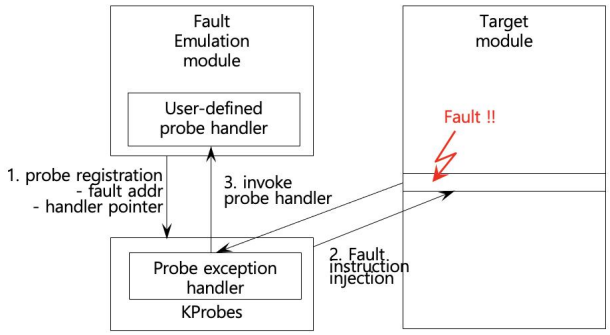

Fig. 3. FaultInjection module

FaultInjection 모듈의 동작은 크게 세 단계로 이루어진다. 첫번째 단계는 폴트 에뮬레이션 모듈이 커널 영역으로 동적 로드되는 과정에서 오류 발생 주소인 프로브 포인트(probe point)를 등록하는 것이다(그림의 1). 이 때 제공되는 정보는 폴트 발생 주소와 폴트 주입 에뮬레이션 기능을 제공하는 사용자 지정 프로브 핸들러 함수(probe handler function)에 대한 함수 포인터 등이다. 두번째 단계는 KProbes 내의 KProbe 관리자가 폴트 에뮬레이션 모듈의 초기화 과정에서 제공된 정보를 이용하여 관련 커널 자료 구조를 생성한 후, 대상 커널 모듈 내의 프로브 포인트에 위치한 명령(instruction)을 폴트 주입 명령(fault injection instruction)으로 대치함으로써 폴트를 주입하는 것이다(그림의 2).

세번째 단계는 타겟 커널 모듈이 동작하는 중에 제어가 사전에 지정된 프로브 포인트에 도달하는 경우 폴트 주입 명령이 실행되면서 일어난다. 이 명령이 실행되는 순간 KProbes의 예외 처리 메커니즘에 의해 폴트 발생 주소와 관련된 해당 probe 자료 구조에 등록된 사용자 정의된 프로브 핸들로 함수로 제어가 넘어가게 된다(그림의 3). 그러면 사용자 정의 프로브 핸들러 함수는 타겟 모듈에 정의된 폴트의 주입을 에뮬레이션하게 된다.

IV. Experiments

본 장에서는 제안한 폴트 주입 시스템의 기능의 정상적으로 동작하는지 여부를 확인하기 위해 실험을 설계하고 진행한 결과를 소개한다.

1. Fault injection command

폴트 주입 기능을 테스트하기 위하여 리눅스 커널 모듈에서 흔히 발생하는 몇 가지 전형적인 폴트에 대해 실험했다. 여기에는 커널 함수 kmalloc에서의 메모리 할당 관련 폴트, kmem_cache_alloc 함수에서의 메모리 할당 관련 폴트, alloc_pages 함수에서의 페이지 할당 폴트, 함수 kfree에서의 할당된 메모리 해제 폴트 등이 해당한다. 이들은 모두 리눅스 커널 내의 메모리 관련 함수들이다.

각 유형의 폴트는 실제 커널 동작 중 다양한 형태로 나타날 수 있다. 이 유형의 폴트들에 대해 실제 리눅스 커널에서 발생할 수 있는 커널 에러 메시지들은 다음과 같다.

- Use potential NULL pointers returned from routines
- Use freed memory
- Dereference user pointers
- Leak memory by updating pointers with potentially NULL "realloc" return values
- Do inconsistent assumptions about whether a pointer is NULL
- Allocate large stack variables (> 1K) on the fixed-size kernel stack
- Allocate insufficient memory to hold the type for which you are allocating
- Exceed bounds of array indices and loop bounds derived from user data

사용자가 원하는 폴트를 의도하는 방식으로 발생시킬 수 있도록 명령줄(command-line) 기반의 명령 faultinj가 제공된다. Table 1은 이 명령이 제공하는 옵션들이다.

Table 1. faultinj command

option	function
module	module name
module type	module type
fault type	fault type; 0 for kmalloc, 1 for kmem_cache_alloc, 2 for alloc_pages
fail-times	max number of times that the process may exit
interval	number of successful hits between potential failures
probability	probability of potential failure. Valid value range is from 0 to 1000
fail-space	number of successful hits before the first failure
total-time	duration of fault injection session in ms(mili-second) unit
device-path	full pathname of device for the faulty target module
workload	workload program name
workload-args	command line options for workload program

2. Demonstration

폴트 주입 시스템의 동작을 시험하기 위하여 폴트의 유형에 따라 세 가지 시험을 실행했다. 먼저 kmalloc 함수에 폴트를 주입하는 실험을 진행했다. 구체적인 실험 내용은 Table 2와 같다.

Table 2. kmalloc fault injection test

Test	kmalloc fault test
Criteria	After injecting a fault into target module, verify that a fault occurs in the system.
Procedure	

(1) Steps
① Compile the test device driver pobox using CONFIG_POBOX_MAIN=kmalloc
$ vi Makefile # set CONFIG_POBOX_MAIN=kmalloc
$ make modules # compile the module
② Install the target module
$ make modules_install
③ After unloading (if any) existing module, load the newly-installed module
$ rmmod pobox; modprobe pobox
④ Inject the fault into the target module
$ faultinj -m pobox -t 0 -i 200 -s 200 -c 30000 / opt/faultinj/bin/pobox_workload
 	
(2) Commands
• pobox: Target device driver module for sending and receiving messages using circular queue
• pobox_workload: User program to generate workload for target module pobox
 	
(3) Verification
• By identifying the values of 'Probe hits' and 'Failed times', confirm that the fault is correctly injected according to fail-times, interval, and total-time.	

이 실험의 수행 결과는 fig. 4와 같다. 명령 실행 시, 대상 모듈은 pobox, 폴트 타입은 kmalloc, 폴트 주입 시작 시점 200, 폴트 간결 200, 수행 시간 30초로 설정하고 폴트 주입을 시험했다. 폴트 주입 모듈이 정상적으로 만들어지고 제대로 동작하는지 확인한 후, 로그 파일의 Probe hits값과 Failed times 값을 통해 주어진 옵션을 만족하는 폴트 주입이 발생했는지 확인했다. 그 결과 Probe hits는 552, Failed times은 2였다. 폴트 주입 시작 시점의 값이 200이고, 폴트 간격이 200이기 때문에 552의 hits인 경우에 주입된 폴트는 2번이 되어야 하는데, Failed times 값이 2이기 때문에 폴트 주입이 정상적으로 이루어졌다고 판단할 수 있다.

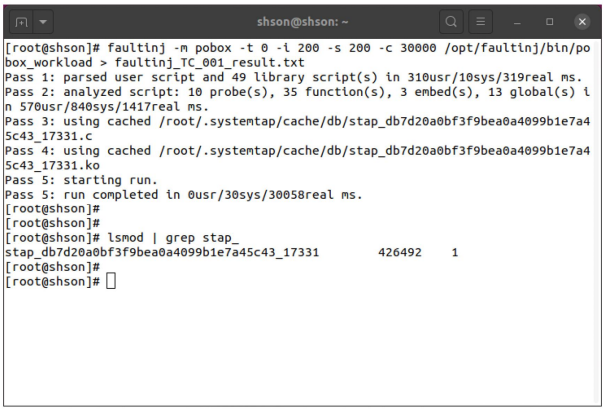

Fig. 4. Demonstration of kmalloc fault

다음은 kmalloc_cache_alloc 함수에 폴트를 주입하는 실험을 수행했다. 구체적인 실험 내용은 Table 3과 같다.

Table 3. kmem_cache_alloc fault injection test

Test	kmem_cache_alloc fault test
Criteria	After injecting a fault into target module, verify that a fault occurs in the system.
Procedure	

(1) Steps
① Compile the test device driver pobox using CONFIG_POBOX_MAIN=kmem_cache_alloc
$ vi Makefile # set CONFIG_POBOX_MAIN=kmem #_cache_alloc
$ make modules # compile the module
② Install the target module
$ make modules_install
③ After unloading (if any) existing module, load the newly-installed module
$ rmmod pobox; modprobe pobox
④ Inject the fault into the target module
$ faultinj -m pobox -t 1 -i 200 -s 200 -c 30000 /opt/faultinj/bin/pobox_workload
 	
(2) Commands
• pobox: Target device driver module for sending and receiving messages using circular queue
• pobox_workload: User program to generate workload for target module pobox
 	
(3) Verification
• By identifying the values of 'Probe hits' and 'Failed times', confirm that the fault is correctly injected according to fail-times, interval, and total-time.	

이 실험의 수행 결과는 fig. 5와 같다. 명령 실행 시, 대상 모듈은 pobox, 폴트 타입은 kmem_cache_alloc, 폴트 주입 시작 시점 200, 폴트 간격 50, 수행 시간 30초로 설정하고 시험했다. 폴트 주입 모듈이 정상적으로 만들어지고 동작하는지 확인한 후, 로그 파일의 Probe hits 값과 Failed times 값을 통해 주어진 옵션을 만족하는 폴트 주입이 발생했는지 확인한 결과, Probe hits값은 542, Failed times 값은 7이었다. 폴트 주입 시작 시점의 값이 200이고, 폴트 간격이 50이기 때문에 542의 hits가 발생한 경우 주입된 폴트는 7번이 되어야 하는데, Failed times 값이 7이기 때문에 폴트 주입이 정상적으로 이루어졌다고 판단할 수 있다.

Fig. 5. Demonstration of kmem_cache_alloc fault

다음은 alloc_pages 함수에 폴트를 주입하는 실험을 진행했다. 구체적인 실험 내용은 Table 4와 같다.

Table 4. alloc_pages fault injection test

Test	alloc_pages fault test
Criteria	After injecting a fault into target module, verify that a fault occurs in the system.
Procedure	

(1) Steps
① Compile the test device driver pobox using CONFIG_POBOX_MAIN=alloc_pages
$ vi Makefile # set CONFIG_POBOX_MAIN=alloc # _pages
$ make modules # compile the module
② Install the target module
$ make modules_install	

③ After unloading (if any) existing module, load the newly-installed module
$ rmmod pobox; modprobe pobox
④ Inject the fault into the target module
$ faultinj -m pobox -t 2 -i 200 -s 200 -c 30000 /opt/faultinj/bin/pobox_workload

(2) Commands
- pobox: Target device driver module for sending and receiving messages using circular queue
- pobox_workload: User program to generate workload for target module pobox

(3) Verification
- By identifying the values of 'Probe hits' and 'Failed times', confirm that the fault is correctly injected according to fail-times, interval, and total-time.

이 실험의 수행 결과는 fig. 6과 같다. 명령 실행 시, 대상 모듈은 pobox, 폴트 타입은 alloc_pages, 폴트 주입 시작 시점 100, 폴트 간격 100, 시험 수행 시간 30초로 설정하고 폴트 주입을 시험했다. 폴트 주입 모듈이 정상적으로 만들어지고 제대로 동작하는지 확인한 후, 로그 파일의 Probe hits값과 Failed times 값을 통해 주어진 옵션을 만족하도록 폴트가 주입되었는지 확인했다. 그 결과 폴트 로그 파일의 Probe hits값은 822, Failed times 값은 8이었다. 폴트 주입 시작 시점의 값이 100이고, 폴트 간격이 100이기 때문에 822의 hits인 경우에 주입된 폴트는 8번이 되어야 하는데, Failed times 값이 8이기 때문에 폴트 주입이 정상적으로 이루어졌다고 판단할 수 있다.

Fig. 6. alloc_pages type fault

V. Conclusions

폴트 주입기는 소프트웨어 개발 시 유용한 시험 도구가 될 수 있다. 본 논문에서는 리눅스 커널 모듈을 대상으로 사용할 수 있는 폴트 주입 시스템을 제안했다. 제안된 폴트 주입 시스템은 사용자가 지정한 방식에 따라 원하는 커널 모듈에 폴트를 주입할 수 있다. 또한 시스템이 동작하는 과정에서 주입된 폴트가 잘 드러날 수 있도록 해당 커널 모듈과 연관된 커널 서브시스템을 동작시키는 워크로드를 생성하는 기능도 제공한다. 일련의 실험을 통해 제안된 폴트 주입 시스템이 유용함을 확인했다. 제안된 폴트 주입기는 커널 모듈 개발 과정에서의 시험, 커널 동작의 분석 연구, 폴트 격리 및 복구 시스템 개발 등에서 폭넓게 활용될 수 있다.

REFERENCES

[1] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr, "Basic Concepts and Taxonomy of Dependable and Secure Computing," IEEE Transactions on Dependable and Secure Computing, vol. 1, no. 1, Jan.-Mar. 2004.

[2] T. Yoshimura, A Study on faults and error propagation in the linux operating system, Ph.D Thesis, Keio University, Mar. 2016.

[3] M. M. Swift, M. Annamalai, B. N. Bershad, and H, M. Levy, "Recovering device drivers," ACM Trans. on Computer Systems, Vol. 24, No. 4, pp. 333-360, Nov. 2006.

[4] Mei-Chen Hsueh, Timothy K. Tsai, and Ravishankar K. Iyer, "Fault injection techniques and tools," IEEE Computer, 30(4):75-82, Apr 1997.

[5] R. Natella, D. Cotroneo, and H. S. Madeira, Assessing Dependability with Software Fault Injection: A Survey, ACM Computing Surveys, Vol. 48, No. 3, Article 44, Feb. 2016.

[6] Jo~ao Carreira and Henrique Madeira and Jo~ao Gabriel Silva, "Xception: A Technique for the Experimental Evaluation of Dependability in Modern Computers," IEEE Transactions on Software Engineering, 24(2), February 1998.

[7] S. Potyra, V. Sieh, and M. Dal Cin, "Evaluating fault-tolerant system designs using FAUmachine," in Proceedings of the 2007 Workshop on Engineering Fault Tolerant Systems (EFTS'07), pp.1-9,New York, NY, USA, 2007.

[8] W. Cao, X. Zhao, and Y. Fayou, Research on Faults Propagation in Linux OS, in Proceedings of the 2nd International Conference on Cryptography, Security and Privacy, pp. 165-169, Mar. 2018.

[9] Linux fault injection capabilities infrastructure, http://www.kernel.org/doc/Documentation/faultinjection/fault-injdection.txt.

[10] H. A. Ahmad, Y. Sedaghat and M. Moradiyan, "LDSFI: a Lightweight Dynamic Software-based Fault Injection," in Proceedings of 9th International Conference on Computer and Knowledge Engineering (ICCKE), pp. 207-213, 2019.

[11] D. Cotroneo, L. De Simone, P. Liguori and R. Natella, "ProFIPy:

Programmable Software Fault Injection as-a-Service," in Proceedings of 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), 2020, pp. 364-372.

[12] Z. Chen, N. Narayanan, B. Fang, G. Li, K. Pattabiraman and N. DeBardeleben, "TensorFI: A Flexible Fault Injection Framework for TensorFlow Applications," in Proceedings of IEEE 31st International Symposium on Software Reliability Engineering (ISSRE), pp. 426-435, 2020.

[13] K. Bhat, E. Kouwe, H Bos, and C. Giuffrida, “FIRestarter: Practical Software Crash Recovery with Targeted Library-level Fault Injection,” in Proceedings of 51st Annual IEEE/IFIP International Conference on Dependable Systems and Networks, Jun. 2021.

[14] Zu-Ming Jiang and Jia-Ju Bai, Kangjie Lu, Shi-Min Hu, “Fuzzing Error Handling Code using Context-Sensitive Software Fault Injection,” in Proceedings of the 29th USENIX Security Symposium, Aug. 2020.

Authors

Sunghoon Son received his B.S., M.S. and Ph.D. degrees in Computer Science from Seoul National University, Seoul, Korea, in 1991, 1993 and 1999, respectively. Dr. Son joined the faculty of the Department of Computer Science at Sangmyung University, Seoul, Korea, in 2004. He is currently a Professor in the Department of Computer Science, Sangmyung University. He is interested in system software, embedded system, and virtualization.