Fuzz Testing을 통한 위성 SW 분석

종합설계1 Week11

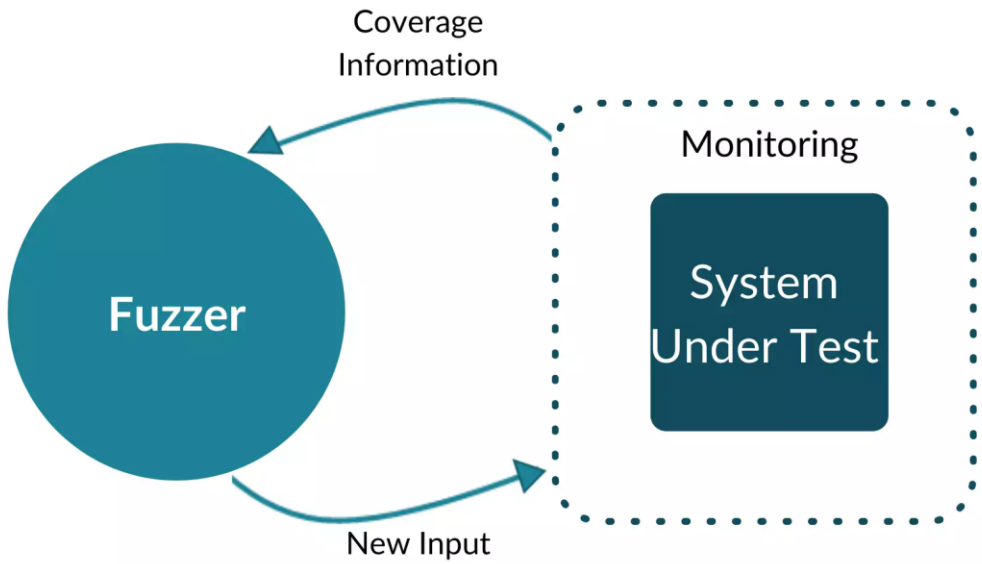
202002473 김승혁
201902733 이정윤
202002699 조민기

연구 배경

왜 필요한가 ?

- 위성 소프트웨어는 복잡한 구조와 FPP 기반 통신 방식을 사용해, 기존 수작업 테스트 방식으로는 잠재적 결함을 찾는 데 한계가 있음.

- 예측 불가능한 입력을 자동으로 생성하는 퍼징을 적용해 문제를 보완하고자 함.

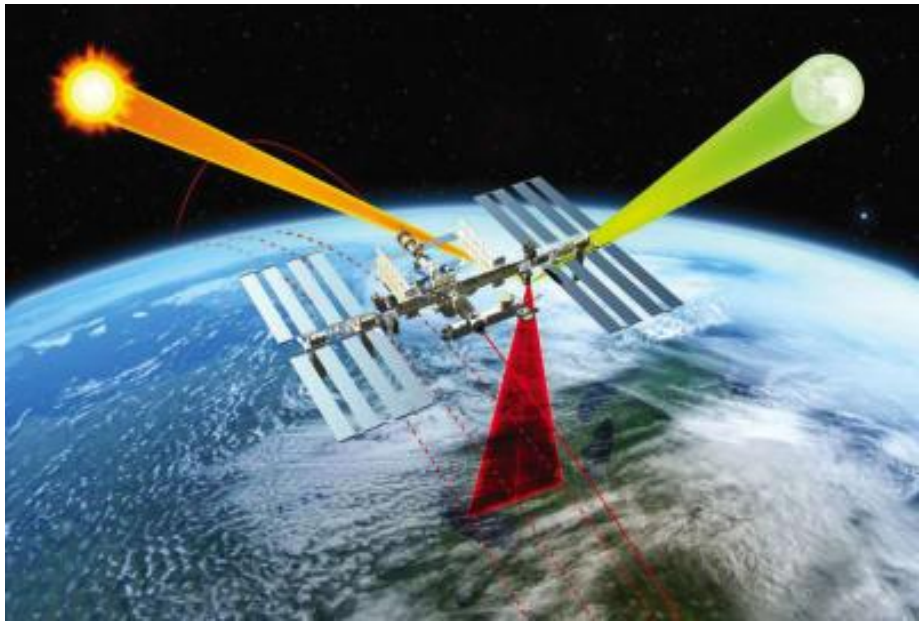


연구 목적

- Fprime 환경에서 다양한 fuzzing 도구를 비교 분석

- 컴포넌트 기반 구조에 최적화된 fuzzing 전략을 제시하는 것

-〉 발사 전 지상 테스트 단계에서 결함을 미리 잡아내고, 실제 위성 운영 중 발생할 수 있는 문제를 사전에 차단하는 것이 목표



연구 질문 및 가설

연구 질문

RQ1

- Fuzzing 기법이 기존 방식보다 결함을 더 잘 탐지할 수 있는가?

RQ2

- Fuzzing 도구의 유형이 Fprime이라는 컴포넌트 기반 아키텍처, fpp 통신 방식에서 결함 탐지 성능에 어떤 차이를 보일까?

연구 가설

H1

- 퍼징을 활용한 결함 탐지 활동은 기존 방식보다 결함 탐지율을 유의미하게 향상시킬 것이다.

H2

- Generation-Based 퍼저는 컴포넌트 간 복잡한 통신을 다루는데 더 효과적이어서, 더 많은 시스템 상태를 탐색하고 더 높은 결함 탐지 성능을 보일 것이다.

독립/종속 변수 정의

독립 변수

- 테스트 방식

1. 기존 수동 테스트 방식,
2. Generation-based 방식의 BooFuzz
3. Mutation-based 방식의 AFL++,
4. Coverage-guided 방식의 LibFuzzer

유형 별 동작 방식

- Generation-based 유형: 프로그램이 처리할 수 있는 입력 데이터에 대한 모델을 기반으로 테스트 케이스 생성

- Mutation-based 유형: 기존의 유효한 입력 데이터를 수집한 후, 이를 다양한 방식으로 변형하여 테스트 케이스 생성

- Coverage-Guided 유형: 코드 실행 중 커버리지 정보를 수집하여 달성하지 못한 경로에 도달할 수 있도록 테스트 케이스 생성

독립/종속 변수 정의

종속 변수

1. 결함 탐지 수 (충돌, 오류, 로그 기반 비정상 동작 수 등)
2. 테스트 종료 후 코드 커버리지
3. 테스트 수행 시간
4. 자원 소모량 (CPU, 메모리 등)

실험 대상

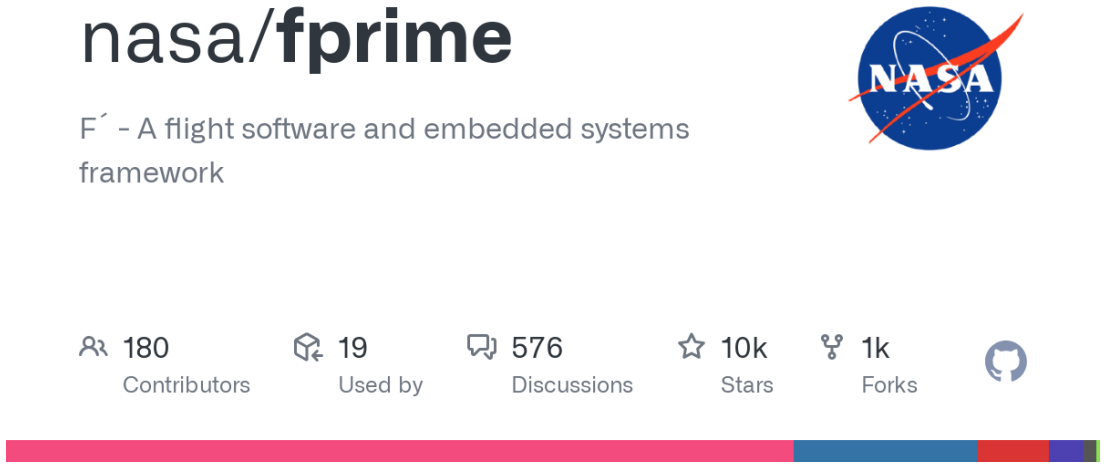
Fprime의 핵심 구성 모듈인 cmdDispatcher, Health, ActiveLogger를 우선 테스트한 다음 범위를 넓힐 것. Fuzzer는 BooFuzz, AFL++, LibFuzzer 사용.

선정 이유

cmdDispatcher: 위성 시스템으로 전송된 명령을 수신하고 분배하는 모듈. 입력 기반 공격에 매우 취약함.

Health: 여러 모듈들의 상태를 모니터링하고 오류나 비정상 상태를 감지하는 모듈. 타 모듈과의 상호작용이 많아 테스트에 적합.

ActiveLogger: fprime 시스템에서 발생하는 다양한 로그를 저장하는 모듈. 각 모듈에서 발생한 메시지를 수집하고 저장, 전송하는 역할. 로그 구조가 복잡하고 입력이 다양해 예상치 못한 입력이 발생하기 쉬움.



실험 대상/환경

실험 환경

- Ubuntu 20.04 기반 도커 컨테이너에서 수행. 자원은 4코어 CPU, 8GB 메모리로 동일하게 구성

입력 데이터

- Fprime 명령어 또는 시퀀스 포맷 기반 입력을 바탕으로 각 퍼저에 맞게 변형해 사용

테스트 관리 – 실험 절차

실험 절차

1. Fprime 대상 모듈을 별도 타겟 바이너리 또는 모듈 단위로 분리
2. 각 퍼저별 테스트 환경 구성(별도 harness 구성 및 입력 corpus 설정)
3. 동일 자원 조건에서 각 fuzzer를 6시간 이상 수행(Docker compose를 통한 병렬 컨테이너 실행)
4. 결함 탐지 수, 커버리지, 자원 사용량 등을 기록

테스트 관리 – 측정 지표 및 도구

정량 지표

- Crash 수
- 코드 커버리지
- 경로 다양성
- 자원 사용량

정성 지표

- 로그 분석
- 적용 난이도

테스트 케이스 명세

- cmdDispatcher를 기존 테스팅 방식을 통해 진행.
- 이후 cmdDispatcher를 선정된 3가지 퍼저를 이용해 각 테스트 데이터를 생성하여 테스트 진행
- 다른 모듈에도 같은 방식을 통해 테스트 진행

Id	대상(모듈)	실험 조건	테스트 데이터	평가지표	예상 결과
TC-H 1-1	CmdDispatcher	기존 테스트 코드 실행	정상 명령어 시퀀스	결함 수, 커버리지	낮은 탐지율
TC-H 2-1	CmdDispatcher	BooFuzz	유효한 명령어 시퀀스	충돌 수, 커버리지, 경로 다양성	경로 다양성 높음
TC-H 2-2	CmdDispatcher	AFL++	기존 시드 시퀀스 파일의 변형본	충돌 수, 커버리지, 경로 다양성	경로 다양성 낮음
TC-H 2-3	CmdDispatcher	libFuzzer	무작위 바이트 스트림	충돌 수, 커버리지, 경로 다양성	커버리지 높으나 의미 해석 부족
TC-H 2-4	CmdDispatcher	BooFuzz, AFL++, libFuzzer	유효한 CmdDispatcher 제어 패킷	충돌 수, 커버리지, 경로 다양성	도구별 탐지 차이 관찰
TC-H 2-5	Health	BooFuzz, AFL++, libFuzzer	유효한 Health 제어 패킷	충돌 수, 커버리지, 경로 다양성	도구별 탐지 차이 관찰
TC-H 2-6	ActiveLogger	BooFuzz, AFL++, libFuzzer	유효한 ActiveLogger 제어 패킷	충돌 수, 커버리지, 경로 다양성	도구별 탐지 차이 관찰

검증 기준

지표	정의	측정 방법	단위
충돌 수	퍼징 중 발견된 고유한 충돌 수	퍼저 내 크래시 로그	건
코드 커버리지	실행된 소스 코드의 라인/분기 비율	gcov, llvm-cov 도구 활용	%
경로 다양성	퍼저가 탐색한 고유 실행 경로 수	퍼저 내부 경로 해시, 커버리지 map	건
자원 사용량	평균 CPU 및 메모리 사용량	docker stats	%, MB

Thank You