ROS2(Robot Operating System2) 애플리케이션 테스트를 위한 기호 실행 기법 적용에 관한 연구

최장섭 [1], 이호준
[1] 성균관대학교 소프트웨어학과 학부생
[2] 성균관대학교 소프트웨어학과 교수

kgh010529@skku.edu, hojoon.lee@skku.edu

A Study on Application of Symbolic Execution for ROS2 Application Test

Jangseop Choi[1], xxx[2],
[1]Dept. of Computer Science and Engineering, Sungkyunkwan University (SKKU)
[2] Dept. of Computer Science and Engineering, Sungkyunkwan University (SKKU)

요 약

ROS2(Robot Operating System)은 로봇 애플리케이션 개발을 위한 소프트웨어 라이브러리 모음으로, 다양한 사이버-물리 시스템(Cyber-Physical System, CPS) 개발에 폭넓게 활용되고 있다. 본 연구에서는 ROS2 의 구조적 특성에 착안하여, 기존에 실용적인 한계가 존재했던 기호 실행(Symbolic Execution) 기법을 ROS2 애플리케이션에 적용하기 위한 새로운 방법론을 제안한다. 또한, 간단한 ROS2 애플리케이션에 기호 실행을 적용한 사례 분석을 통해 새로운 방법론의 실용 가능성을 탐색한다.

1. 서론

최근 드론, 로봇, 자율주행차 등 사이버-물리 시스템(Cyber-Physical System, CPS)의 본격적인 상용화가 진행되면서, 소프트웨어 결함이 물리적 사고로 이어질 수 있는 위험성이 대두되고 있다. 이에 따라 CPS 보안은 핵심 연구 과제로 부각되고 있다.

ROS2(Robot Operating System 2) [1]는 로봇 애플리케이션 개발을 위한 대표적인 소프트웨어 라이브러리 모음이다. ROS2 는 발행자-구독자(Publisher-Subscriber) 패턴을 채택하고 있어, 발행자는 메시지를 송신하고, 이를 수신한 구독자는 사전에 등록된 콜백(callback) 함수를 실행함으로써 동작을 수행한다. 본 연구는 이러한 ROS2 의 구조적 특성을 활용하여, 기존에 실용적인 한계에 직면해 있던 기호 실행(Symbolic Execution) 기법을 ROS2 애플리케이션에 적용하기 위한 새로운 방법론을 제안한다.

2. ROS2 기호 실행 기법 배경

콜백함수 기반 테스트. ROS2 에서 발행자와 구독자는 단순히 메세지의 송수신만을 담당하며, 핵심 기능은 콜백 함수가 수행한다. 즉, 발행자와 구독자가 담당하는 메세지 송수신 계층을 제거하고 보면, ROS2 애플리케이션은 콜백 함수들 간 메세지를 주고 받는 방식으로 구현되어 있음을 알 수 있다. 본 연구는 이러한 통찰을 바탕으로 단일 콜백 함수를 대상으로 기호 실행을 적용하는 기법을 제안한다. 더 나아가, 애플리케이션 내의 모든 콜백 함수를 단일 기호 실행하는 것만으로도 애플리케이션 전체의 커버리지를 달성할 수 있는 가능성을 제시한다.

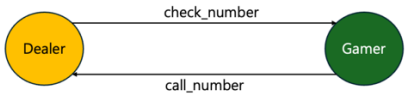
함수 기반 기호 실행 기법. 단일 함수에 대한 기호 실행(Symbolic Execution) 방법론은 UCKLEE [2]에서 처음 제안되었으나, 기존 소프트웨어에 이를 적용하는 데에는 두 가지 주요한 한계가 있다. 첫째는 콜러-콜리 구조에 따른 의존성 문제였고, 둘째는 함수 호출의 사전 조건을 고려하지 않아 오탐지(False Positive)가 발생하는 문제였다. 하지만 ROS2 는 고유한 실행 방식 덕분에 이러한 문제를 우회할 가능성을 제공한다. ROS2 는 메시지 발행 방식으로 콜백 함수가 호출되며, 이는 함수의 실행이 종료된 후에 결정된다. 이 구조는 콜백 함수의 원자적 실행을 보장한다. 또한 ROS2 는 분산 시스템이므로, 임의의 메시지 주입을 고려한 위협 모델에서 콜백 함수의 안정성을 증명할 필요가 있다. 이로 인해 사전 조건을 별도로 고려하

지 않고도 기호 실행이 가능하다.

3. ROS2 기호 실행 기술 개발

본 연구는 ROS2 라이브러리의 모델링과 실행기 개발을 통하여, ROS2 에 대한 기호 실행을 가능케하고 이를 실험을 통해 평가하고자 한다.

대상 어플리케이션. 사례 연구를 소개하기에 앞서, 기호 실행이 적용된 평가 대상 애플리케이션을 설명한다. 해당 애플리케이션은 '업다운 게임(Up-Down Game)'의 동작을 모사한 간단한 ROS2 애플리케이션으로, 두 쌍의 발행자-구독자(Publisher-Subscriber) 구조를 기반으로 구현되어 있다. 두 개의 노드가 각각 게이머(Gamer)와 딜러(Dealer)의 역할을 수행하며, 게이머는 임의의 숫자를 예측하여 메시지로 발행하고, 딜러는 해당 숫자를 정답과 비교한 후 결과를 'up', 'down', 또는 '정답'으로 판별하여 게이머에게 다시 전달한다.



(그림 1) 평가 대상 애플리케이션 구조

ROS2 라이브러리 모델링. ROS2 는 주로 동적 라이브러리(dynamic library)로 구현되어 있어, 중간 단계(middle-end)에서 계측(instrumentation)을 시도하는 기호 실행에 적용하기 어려움이 있다. 이를 해결하기 위해, 본 연구에서는 ROS2 라이브러리를 모사하는 Fake-ROS 라이브러리를 개발하였다.

Fake-ROS 는 ROS2 를 애플리케이션 계층에서 테스트하기 위한 라이브러리이므로 네트워크 설정 등을 담담하는 하위 계층의 함수들을 모두 공백 함수(dummy function)으로 변경해주었다.

```
#define private public

int main(int argc, char** argv) {

    int num;
    std::cin.read(reinterpret_cast<char*>(&num), sizeof(num));

    auto gamer =std::make_shared<Gamer>(num);

    auto msg =std::make_shared<std_msgs::msg::String>();
    int data;
    std::cin.read(reinterpret_cast<char*>(&data), sizeof(data));
    msg->data = data;

    int min;
    std::cin.read(reinterpret_cast<char*>(&min), sizeof(min));
    gamer->min = min;

    int max;
    std::cin.read(reinterpret_cast<char*>(&max), sizeof(max));
    gamer->max = max;

    gamer->predict_number(msg);

    return 0;
}
```

```
#define private public

int main(int argc, char** argv)
{
    int key_num;
    std::cin.read(reinterpret_cast<char*>(&key_num), sizeof(key_num));

    auto dealer =std::make_shared<Dealer>(key_num);

    auto msg =std::make_shared<std_msgs::msg::Int8>();
    int data;
    std::cin.read(reinterpret_cast<char*>(&data), sizeof(data));
    msg->data = data;

    dealer->check_number(msg);
    return 0;
}
```

(그림 2) 콜백 함수에 대한 기호 실행기 구현 코드

기호 실행을 위한 실행기(Executor) 작성. (그림 2)는 평가 대상 애플리케이션의 두 개 콜백 함수 각각에 대한 기호 실행기(symbolic executor)의 구성 과정을 나타낸다. 기호 실행 도구로는 symCC [3]를 사용하였으며, symCC 로 컴파일된 실행 파일은 표준 입력으로 전달되는 변수들을 자동으로 기호화(symbolize)하고, 실행 시 자동으로 기호 실행이 수행된다.

콜백 함수의 기호 실행을 위해서는 먼저 소속 노드 객체의 초기화가 필요하며, 이때 생성자의 파라미터는 실행 환경의 구성 파라미터로서 기호화된다. 또한, 콜백 함수가 수신하는 입력 메시지와 내부적으로 참조하는 노드의 상태 값 역시 함수 동작에 영향을 미치는 주요 요인이므로 기호화 대상에 포함된다.

요약하자면, 기호 실행기의 구성은 다음과 같은 절차를 따른다: 1) 노드 생성자의 파라미터 기호화, 2)노드 객체 생성, 3) 메시지 기호화, 4) 콜백 함수에서 참조하는 노드 상태 값 기호화, 5) 콜백 함수 호출

4. 실험 결과

각 콜백 함수에 대해 기호 실행기를 수행한 뒤, 생성된 테스트 입력을 기반으로 콜백 함수를 재실행하고, gcov 도구를 이용해 코드 커버리지를 측정하였다. <표 1>은 해당 실험을 통해 수집된 코드 커버리지 결과를 나타낸다. 두 콜백 함수가 달성한 커버리지를 종합적으로 분석한 결과, 전체 애플리케이션 수준에서의 코드 커버리지를 확보할 수 있었음을 확인하였다. 이를 통해, 개별 콜백 함수에 대한 기호 실행만으로도 프로그램 전반에 대한 효과적인 테스트가 가능함을 실험적으로 입증할 수 있었다.

<표 1> 기호 실행 기반 코드 커버리지 측정 결과

	'predict_number' executor	'check_number' executor
dealer.cpp	-	100.00%
gamer.cpp	100.00%	-

5. 결론

본 연구는 ROS 2 기반 애플리케이션의 보안 및 신뢰성 강화를 위한 정적 분석 기법으로서 기호 실행 적용 가능성을 제시하였다. 특히 ROS 2 의 발행자-구독자 구조에서 핵심 동작 단위인 콜백 함수에 주목하여, 단일 콜백 함수 수준에서 기호 실행을 수행하는 새로운 테스트 방법론을 제안하였다. 더불어, 간단한 사례 연구를 통해 개별 콜백 함수에 대한 기호 실행만으로도 전체 프로그램 수준의 테스트가 가능함을 실험적으로 입증하였다.

참고문헌

[1] Open Robotics, *"ROS2 Foxy Documentation"*, https://docs.ros.org/en/foxy/index.html, 2025.

[2] D. A. Ramos and D. Engler, "Under-Constrained Symbolic Execution: Correctness Checking for Real Code," *Proceedings of the 24th USENIX Security Symposium*, Washington, D.C., 2015.

[3] S. Poeplau and A. Francillon, "Symbolic Execution with SymCC: Don't Interpret, Compile!," *Proceedings of the 29th USENIX Security Symposium*, Boston, MA, USA, 2020.