Journal of The Korea Institute of Information Security & Cryptology ISSN 1598-3986(Print)
VOL.32, NO.6, Dec. 2022 ISSN 2288-2715(Online)
https://doi.org/10.13089/JKIISC.2022.32.6.1091

ROS 2의 이벤트 기반 런타임 모니터링을 활용한 실시간 공격 탐지 시스템*

강 정 환,[1†] 서 민 성,[2] 박 재 열,[2] 권 동 현[3‡]
[1,2,3]부산대학교 (대학원생, 학생, 교수)

Real-Time Attack Detection System Using Event-Based Runtime Monitoring in ROS 2*

Jeonghwan Kang,[1†] Minseong Seo,[2] Jaeyeol Park,[2] Donghyun Kwon[3‡]
[1,2,3]Pusan National University (Graduate student, Student, Professor)

요 약

로봇 시스템은 지난 10년 간 매우 빠른 속도로 발전했다. Robot Operating System은 로봇 시스템 및 애플리케이션의 효율적인 개발을 위한 오픈소스 기반의 소프트웨어 프레임워크이며, 다양한 연구 및 산업 현장에서 널리 사용되고 있다. ROS 애플리케이션은 다양한 취약점을 내재하고 있을 수 있다. 이러한 ROS 애플리케이션의 실행을 런타임 모니터링 하기 위해 다양한 연구가 진행되어 왔다. 본 연구에서는 ROS 2에서의 이벤트 기반 런타임 모니터링을 활용한 실시간 공격 탐지 시스템을 제안한다. 우리의 공격 탐지 시스템은 ros2_tracing의 tracetools를 확장하여 ROS 2 미들웨어 계층의 주요 라이브러리에 이벤트 계측을 삽입하고 런타임 중에 이벤트를 모니터링함으로써 API의 비순차적 실행을 통한 애플리케이션 계층에서의 공격을 탐지한다.

ABSTRACT

Robotic systems have developed very rapidly over the past decade. Robot Operating System is an open source-based software framework for the efficient development of robot operating systems and applications, and is widely used in various research and industrial fields. ROS applications may contain various vulnerabilities. Various studies have been conducted to monitor the excution of these ROS applications at runtime. In this study, we propose a real-time attack detection system using event-based runtime monitoring in ROS 2. Our attack detection system extends tracetools of ros2_tracing to instrument events into core libraries of ROS 2 middleware layer and monitors the events during runtime to detect attacks on the application layer through out-of-order execution of the APIs.

Keywords: Robot Operating System, Event-based Runtime Monitoring, Real-time Attack Detection

I. 서 론

로봇 시스템은 지난 10년 간 매우 빠른 속도로 발전했다. 특히, 로봇 시스템은 여러 복잡한 환경에서 무인비행체(UAV), 무인지상차량(UGV), 휴머노이드 등의 다양한 모습과 크기, 그리고 기능을 가진 형태로 빠르게 발전하고 있다[1]. ROS(Robot Operating System)는 로봇 시스템 및 애플리케이션의 효율적인 개발을 위해 2007년에 처음 공개된 오픈소스 기반의 소프트웨어 프레임워크이다. ROS

Received(09. 16. 2022), Modified(10. 20. 2022), Accepted(10. 20. 2022)
* 이 과제는 부산대학교 기본연구지원사업(2년)에 의하여 연구되었음
† 주저자, jeonghwan@pusan.ac.kr
‡ 교신저자, kwondh@pusan.ac.kr(Corresponding author)

는 다양한 연구 기관과 산업 현장에서 널리 사용되며 사실상 대표적인 오픈소스 로봇 개발 프레임워크로 자리 잡아가고 있다.

ROS 1은 보안을 충분히 고려하지 않고 설계되었다. 이로 인해 ROS 커뮤니티에서는 꾸준히 보안에 대한 관심이 증가하였다. 대표적인 예로 단일 공격 지점(single point of failure)로서 ROS 1의 마스터 노드에 대한 보안 이슈가 제기되었다[2][3]. 또 다른 예로 보안 기능을 담은 패치인 SROS(Secure Robot Operating System) 패키지가 새롭게 개발되어 추가되었다[4]. 하지만 SROS는 추가로 개발되어 도입된 기능이었기에 이를 지원하는 ROS 1의 클라이언트 라이브러리가 제한되는 한계가 있었다.

반면에 ROS 2는 네트워크 보안 측면에서도 보다 많은 요구사항을 반영하여 설계되었다. ROS 2에서는 데이터 통신을 수행하기 위한 미들웨어로서 OMG(Object Management Group)이 산업용으로 표준화한 DDS(Data Distribution Service)를 채택하였기 때문에 DDS Security 표준을 사용할 수 있게 되었다. DDS Security는 메시지 인증을 비롯하여 암호화, 액세스 제어와 같은 네트워크 기반의 보안 기능을 제공한다[5]. 따라서 ROS 2는 DDS Security를 적용하여 네트워크 계층의 보안을 향상했다.

그럼에도 불구하고, 실시간 시스템이나 소형 임베디드 시스템과 같은 로봇 개발환경을 고려하여 설계된 ROS 2 기반의 로봇 애플리케이션은 일반적으로 C++로 작성되기 때문에 버퍼 오버플로를 비롯한 다양한 소프트웨어 취약점을 내포할 수 있다[6][7][8]. 또한 ROS 애플리케이션은 일반적으로 소규모 그룹 혹은 개인이 개발하고 배포하는 경우가 많다. 이러한 애플리케이션은 보안보다는 기능 구현과 성능 향상에 중점을 두기 때문에 소프트웨어 취약점을 내재하고 있을 가능성이 있다[9].

이러한 취약점을 지닌 ROS 애플리케이션의 실행을 런타임에 모니터링하기 위해 다양한 기존 연구가 진행되어 왔다[10][11][12]. 특히 ROS 1에 대해 런타임 모니터링을 수행하는 ROSRV[10]와 ROSMonitoring[11]은 모두 모니터링 도구가 애플리케이션에 노드로서 직접 포함되어 있기 때문에 애플리케이션의 취약점을 악용한 공격으로부터 모니터가 안전하지 않다. 또한 이러한 런타임 모니터링 시스템은 로봇 시스템의 성능에 끼치는 영향을 최소화해야 할 필요가 있다. 하지만 해당 연구들[10][11]에서 구현된 모니터는 모두 로봇 애플리케이션에 상에 노드로 포함되어 다른 노드와 직접적으로 상호작용하는 방식으로 런타임 모니터링을 수행하기 때문에 ROS 애플리케이션의 노드 간의 통신 메커니즘에 직접적인 영향을 끼친다.

반면에 ROS 2에 대한 런타임 모니터링을 수행하는 기존 연구인 ROS-FM[12]은 eBPF(Berkeley Packet Filter)와 XDP(eXpress Data Path)를 활용해 ROS 2의 미들웨어 계층에서 발생하는 네트워크 패킷을 가로챈 후에 이를 필터링한다. 이를 통해 ROS 2의 노드 API의 직접적인 실행을 악용한 공격을 방어할 수 있다. 하지만, 해당 연구는 ROS 2의 통신 미들웨어인 DDS에서 제공하는 DDS-Security와 같은 네트워크 보안 기법에 직접적인 영향을 끼치게 되며, 특히 네트워크 패킷에 대한 암호화 기법을 적용할 경우에 해당 연구에서 제안한 모니터링 기법은 사용할 수 없다.

이에 본 연구에서는 ROS 2에서의 이벤트 기반 런타임 모니터링을 활용한 공격 탐지 기법을 제안한다. 우리가 제안한 기법은 ROS 애플리케이션에서 발생하는 이벤트를 계측하여 런타임 모니터링하고 이를 화이트리스트와 비교함으로써 비순차적 API 실행을 통한 공격을 탐지한다. ROS 애플리케이션에서 호출되는 이벤트에 대한 계측을 수행하기 위해 ros2_tracing[13]에서 제공하는 tracetools를 확장하여 이벤트에 대한 계측 지점을 수정 및 추가 삽입하고 LTTng[14]를 통해서 추적 데이터를 생성한다. 이렇게 생성된 추적 데이터를 실시간으로 모니터링하여 화이트리스트에 정의되지 않은 위반 행위를 공격으로서 탐지한다.

본 연구의 기여는 다음과 같다.

- ○ 우리가 제안한 시스템은 ROS 미들웨어 계층에서 발생하는 이벤트를 기반으로 공격을 탐지하므로 특정 ROS 애플리케이션에 의존하지 않는다.
- ○ 우리가 제안한 시스템은 노드의 직접적인 API 호출과 실행에 대한 이벤트의 발생을 감시하기 때문에 공격 행위를 보다 정확하게 특정할 수 있다.
- ○ 우리가 제안한 시스템은 ROS 2의 DDS Security 표준의 패킷 암호화와 같은 네트워크 보안 기술과 함께 사용할 수 있다.

II. 배경 지식

2.1 Robot Operating System 2

ROS 2는 Fig. 1.과 같이 여러 추상화 계층으로 구성되어 있다. 클라이언트 라이브러리는 서로 다른 언어로 작성된 프로그램 간의 일관된 동작을 보장하는 API를 지원하며, 주요 클라이언트 라이브러리로서 rclcpp와 rclpy가 있다. 이러한 라이브러리는 공식적으로 각각 C++ 및 Python을 지원하는데 중점을 두고 있다. 또한, 노드 간의 메시지 전송을 비롯한 데이터 통신을 위해 DDS를 미들웨어로 채택하였다. 또한, 미들웨어 라이브러리인 rmw는 DDS 미들웨어와 클라이언트 라이브러리 간의 통신을 위한 미들웨어 추상화 인터페이스의 역할을 수행한다. ROS 2에는 Intra-process API가 별도로 탑재되어 있으므로 DDS에 대한 의존 없이 intra-process communication을 수행한다.

ROS 2의 애플리케이션은 노드라고 하는 독립적인 프로세스들로 구성된다. 노드 간의 통신은 토픽, 서비스, 액션이라는 인터페이스를 통해 수행된다. 이때, Fig. 2.와 같이 노드 간의 통신은 기본적으로 토픽의 발행-구독 모델(publish-subscribe model)을 응용하며, 이는 ROS 1과 ROS 2 모두 마찬가지이다. 이때, 발행이나 구독과 같은 메시지 전송은 각 노드 내에 구현된 콜백 함수를 통해 수행된다. 따라서, 각 노드는 하나 이상의 콜백 함수를 포함하는 방식으로 구현된다.

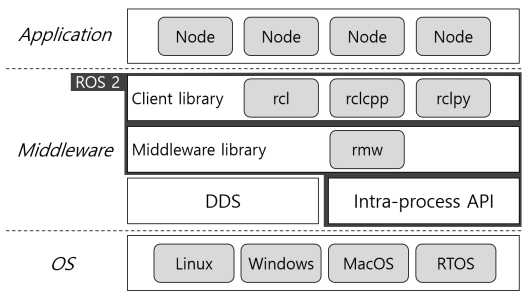

Fig. 1. The architecture of ROS 2

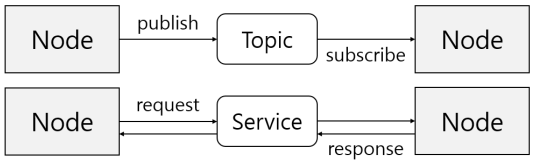

Fig. 2. Communication between nodes

2.2 ros2_tracing

ros2_tracing[15]은 ROS 2에 대한 실행 정보를 실시간으로 수집하기 위한 계측 및 추적 도구를 포함한 다목적 프레임워크이다. ros2_tracing은 ROS 2의 다양한 라이브러리 패키지에 대해 계측 지점을 삽입하기 위해서 핵심 계측 도구인 tracetools 패키지를 지원한다. Tracetools 패키지는 ROS 2에 대한 다양한 계측 지점을 제공하며, 이러한 계측 지점을 트리거하는 패키지이다. 미들웨어 계층의 클라이언트 라이브러리와 미들웨어 라이브러리를 포함한 ROS 2의 핵심 패키지들은 tracetools에 대한 함수 호출을 통해 계측된다. 사용자는 tracetools 패키지를 통해 이러한 이벤트를 수집하는 계측 지점을 확장하여 새로운 추적점을 정의할 수 있다.

기본적으로 tracetools에서 제공하는 계측 지점은 크게 초기화 이벤트와 런타임 이벤트의 두 가지 유형이 있다. 초기화 이벤트는 발행자(publisher), 구독자(subscriber)의 생성이나 서비스의 생성과 같은 일회성 정보를 수집하며, 런타임 이벤트는 메시지 발행이나 콜백 실행과 같은 정보를 수집한다.

특히 런타임 이벤트는 다양한 고유 식별자를 포함하고 있다. 예를 들어, 한 노드에 포함된 발행자가 메시지를 발행할 경우에 발생한 이벤트는 대상 토픽 이름, 발행자의 노드 이름과 같은 고유 식별자를 포함하며 이는 모두 초기화 단계에서 발생한 초기화 이벤트에서 수집한 정보와 일치한다.

ros2_tracing는 상대적으로 낮은 오버헤드와 높은 실시간 호환성을 가지고 있는 LTTng를 추적기(tracer)로 사용한다[13][14]. tracetools를 통해 계측된 ROS 2 패키지는 실행 중에 각 계측 지점에서 발생하는 이벤트를 실시간으로 LTTng 백엔드로 전달하며, LTTng는 이러한 이벤트를 CTF(Common Trace Format) 형식의 추적 데이터로 변환하여 파일에 저장한다[15]. 이때, 생성된 CTF 파일을 human-readable한 파일로 변환하기 위해서 babeltrace2[16]를 사용할 수 있다. babeltrace2를 통해 LTTng를 비롯한 다양한 추적 도구가 생성하는 CTF의 구문을 분석할 수 있다.

III. 위협 모델

본 연구에서는 C++로 작성된 ROS 2 애플리케이션에 포함된 특정 노드에 버퍼 오버플로와 같은 소프트웨어 취약점이 존재하여 이를 악용한 공격을 수행할 수 있으며 공격자는 이러한 취약점을 알고 있다고 가정한다. 따라서, 우리가 가정한 공격자는 취약점이 존재하는 피해자 노드에 악성 페이로드를 공급하여 공격을 수행한다. 또한, 공격자는 구체적으로 ROS 2 애플리케이션에 보다 다양한 공격을 수행하기 위해 비순차적으로 ROS 2 미들웨어 계층의 라이브러리들이 제공하는 API를 직접적으로 실행하여 다음과 같이 ROS 애플리케이션 구성을 변조하는 공격을 수행한다. (1)피해자 노드에서 임의의 토픽 혹은 서비스에 대한 통신을 시도한다. (2)정상적으로 실행 중인 피해자 노드를 강제로 종료한다. (3)악성 페이로드를 탑재한 공격자 노드를 생성한다.

우리가 제안한 시스템은 ROS 2 핵심 라이브러리에서 계측된 이벤트와 화이트리스트를 비교하여 기존의 애플리케이션 구성을 위반하는 이벤트 발생을 공격으로서 탐지한다. 또한, 본 연구에서는 rclcpp, rcl을 포함한 ROS 2의 클라이언트 라이브러리와 rmw와 같은 ROS 2 미들웨어 추상화 라이브러리, 그리고 리눅스 운영체제는 신뢰한다. 또한, ROS 2 애플리케이션을 구성하는 모든 노드는 사전에 정적으로 정의되어 실행된다고 가정한다. 그리고 사전에 정의된 노드 중에 컨테이너 프로세스나 컴포넌트와 같이 런타임 중에 동적으로 실행되는 노드는 없다고 가정한다.

IV. 공격 탐지 시스템 설계

본 디자인 장에서는 우리가 제안한 공격 탐지 시스템의 전체적인 개요와 동작 방식을 설명하고, 시스템을 구성하는 각 모듈에 대해 상세히 설명한다.

4.1 공격 탐지 시스템의 개요

우리가 제안한 공격 탐지 시스템의 전체적인 디자인은 Fig. 3.와 같다. 시스템의 구성은 크게 이벤트 수집부(event collection module)와 공격 탐지부(attack detection module)로 나뉜다. 이벤트 수집부에서는 ROS 2에서 발생하는 이벤트를 수집하고 LTTng의 라이브 모드(live mode)를 통해 실시간으로 추적 데이터를 공격 탐지부로 전달한다. 공격 탐지부는 config.yaml 파일로 입력 받은 애플리케이션 구성을 화이트리스트로 변환하고 이벤트 수집부로부터 전송받는 실시간 추적 데이터를 비교하여 공격을 탐지한다.

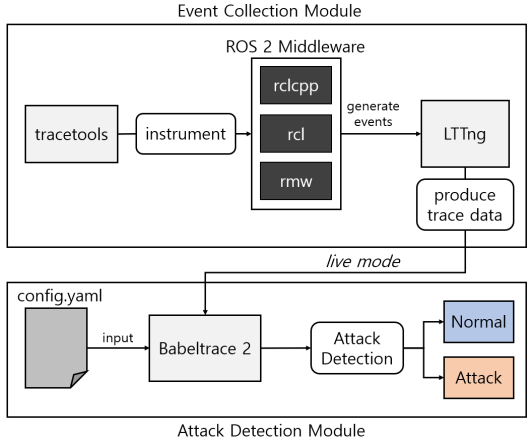

Fig. 3. The overview of our system

4.2 이벤트 수집부

이벤트 수집부는 크게 컴파일 단계, 수집 단계의 2가지 단계로 동작한다. 컴파일 단계에서는 ros2_tracing[13]의 tracetools 패키지를 활용하여 ROS 2 주요 클라이언트 라이브러리에 계측 지점들을 컴파일 타임에 삽입한다. 런타임 단계에서는 실행시간 동안에 애플리케이션이 호출하는 API에 대한 이벤트를 생성하고 LTTng의 live mode를 통해 공격 탐지부로 실시간 추적 데이터를 전달한다.

Table 1.은 우리 시스템에 적합한 이벤트 기반 런타임 모니터링을 수행하기 위해 tracetools 패키지에서 기존에 제공하는 계측 지점을 일부 포함하여 우리가 확장한 계측 지점을 정의한다. Layer는 각 계측 지점이 삽입되는 ROS 2 라이브러리를 나타내며, Instrumentation Points는 각 계측 지점의 이름이다. 또한 Type에는 I와 R의 두 가지 유형이 있으며, 이는 각각 초기화 이벤트와 런타임 이벤트를 구분한다. 예를 들어, rclcpp_publish는 런타임 이벤트로서 발행자가 메시지 발행을 수행할 때 발생하는 이벤트이며, rcl_publisher_init은 초기화 이벤트로서 발행자가 생성될 때 발생하는 이벤트이다.

Table 1.에서 정의된 계측 지점들은 컴파일 타임에 ROS 2 미들웨어의 클라이언트 라이브러리(rcl,

Table 1. Instrumentation points in our system

Layer	Instrumentation Points	Type
rclcpp	rclcpp_publish rclcpp_take rclcpp_send_request rclcpp_handle_request rclcpp_send_response rclcpp_handle_response	R
rcl	rcl_node_init rcl_node_fini rcl_publisher_init rcl_subscrbier_init rcl_service_init rcl_client_init	I

rclcpp)에 삽입되고, 로봇 시스템에서 애플리케이션이 실행되면 초기화 이벤트와 런타임 이벤트를 생성하고 LTTng는 발생한 이벤트를 추적 데이터로 변환하여 라이브 모드를 통해 실시간으로 공격 탐지부로 전송한다.

4.3 공격 탐지부

공격 탐지부는 사전에 유저에게 입력받은 애플리케이션 구성 파일을 통해 실행과 동시에 화이트리스트를 생성하고 이벤트 수집부로부터 실시간으로 전송받는 추적 데이터를 화이트리스트와 비교하여 공격을 탐지한다. 공격 탐지부의 모든 소스코드는 babeltrace 2 소스코드[17]의 내부에 C로 구현하였다. 이벤트 수집부의 LTTng는 기가바이트 단위의 데이터 생성을 위한 최적화된 파일 형식인 CTF(Common Trace Format) 파일 형식의 추적 데이터를 실시간으로 전송하기 때문이다[15]. 우리는 babeltrace 2를 이러한 CTF 형식의 추적 데이터를 사람이 읽을 수 있는 텍스트 형식의 추적 데이터로 변환할 수 있는 기능을 활용하면서 보다 빠른 공격 탐지를 위해 babeltrace 2 내에 우리의 공격 탐지부를 구현하였다. 따라서 이벤트 수집부로부터 전달받은 CTF 추적 데이터를 babeltrace에서 텍스트 형식으로 전환하자마자 곧바로 공격 탐지를 수행한다.

공격 탐지부는 크게 입력 단계, 초기화 단계, 런타임 단계의 3가지 단계로 동작한다.

입력 단계에서는 유저가 작성한 애플리케이션 구성 파일을 입력 받는다. 우리의 공격 탐지 시스템은 rcl 및 rclcpp와 같은 ROS 2 클라이언트 라이브러리에서 계측되는 이벤트를 기반으로 공격 탐지를 수행하기 때문에 C++로 작성된 대부분의 ROS 애플리케이션에 대해 탐지를 수행할 수 있다. 특히, 유저의 입장에서 사용성을 향상하기 위하여 ROS 애플리케이션의 구성을 나타내는 config.yaml 파일을 입력 받고 공격 탐지부의 실행과 동시에 화이트리스트를 생성한다. Fig. 4.는 rqt 플러그인을 통해 그래프로 표현한 간단한 토픽 예제이며, 이를 표현한 애플리케이션 구성 파일인 config.yaml 파일은 Fig. 5.와 같이 표현된다.

초기화 단계는 로봇 시스템에 애플리케이션이 처음 실행되는 단계에서 수행한다. ROS 애플리케이션의 초기 실행 과정에서 노드와 토픽, 서비스 등의 초기화가 이루어지고, 이로 인해 rcl 계층의 초기화 이벤트가 발생하기 때문에, 공격 탐지부는 이러한 초기화 이벤트와 화이트리스트를 비교하여 화이트리스트에 정의되지 않은 노드 생성과 같은 유저가 의도하지 않은 애플리케이션 구성요소의 실행을 탐지하고 노드나 토픽의 정상적인 생성을 검증한다. 또한, 초기화 이벤트의 고유 식별자에 대한 변수 데이터를 화이트

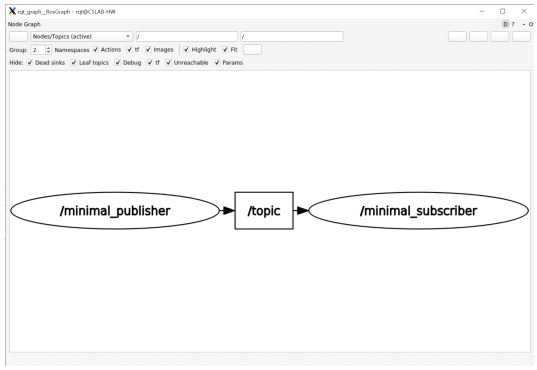

Fig. 4. The rqt graph of the topic example

```
1  Application:
2      Nodes:
3          - name: minimal_publisher
4          - name: minimal_subscriber
5      Topics:
6          - name: topic
7            publish_side:
8              - minimal_publisher
9            subscribe_side:
10             - minimal_subscriber
```

Fig. 5. The configuration file(config.yaml) to configure the topic example

Table 2. List of unique identifiers for each event

Event Type	Instrumentation Points	Unique Identifier			
Runtime Events	rclcpp_publish	node_name	publisher_handle	topic_name	
	rclcpp_take	node_name	publisher_handle	topic_name	
	rclcpp_send_request	node_name	client_handle	service_name	
	rclcpp_handle_request	node_name	service_handle	service_name	
	rclcpp_send_response	node_name	service_handle	service_name	
	rclcpp_handle_response	node_name	client_handle	service_name	
Initialization Events	rcl_node_init	node_name	node_handle		
	rcl_node_fini	node_name	node_handle		
	rcl_publisher_init	node_name	publisher_handle	node_handle	topic_name
	rcl_subscrbier_init	node_name	subscriber_handle	node_handle	topic_name
	rcl_service_init	node_name	service_handle	node_handle	service_name
	rcl_client_init	node_name	client_handle	node_handle	service_name

리스트에 저장함으로써 config.yaml 파일에 작성된 애플리케이션 구성과 실제 실행 중인 애플리케이션의 구성요소에 대한 런타임 정보를 결합한다. Table 2.는 모든 이벤트에 대한 고유 식별자의 리스트이다.

마지막으로, 런타임 단계에서는 애플리케이션 구성요소의 초기화가 완료된 시점부터 발생하는 모든 런타임 이벤트를 검증한다. 이때 런타임 이벤트를 화이트리스트의 애플리케이션 구성과 비교하고 런타임 이벤트의 고유 식별자를 초기화 이벤트의 고유 식별자와 비교함으로써 런타임 이벤트를 검증한다. 이와 같이, 애플리케이션의 구성을 위반한 초기화 이벤트의 발생과 각 이벤트의 고유 식별자의 불일치화를 공격으로 간주하여 탐지한다.

V. 실험 결과

5.1 실험 환경

본 연구에서 진행한 모든 구현 및 실험은 ROS 2 Foxy 릴리즈에서 수행되었다. 우리는 로봇 시스템에서의 보안 평가를 수행하기 위해 turtlebot3 burger 모델에 SBC(Single Board Computer)로서 raspberry pi 4B를 탑재하여 보안 평가를 위한 구현 및 실험을 수행하였다. 구현에 사용된 raspberry pi 4B 모델은 Cortex-A72 SoC, LPDDR4 8GB 메모리를 갖고 있으며 Ubuntu 20.04 LTS 버전을 운영체제로 사용했다.

또한, 우리 시스템의 성능 평가 및 부하 평가를 수행하기 위해 워크스테이션에 우리의 공격 탐지 시스템을 구현하였다. 해당 워크스테이션은 Intel i9-10900k CPU와 64GB의 메모리로 구성되어 있으며 Ubuntu 20.04 LTS를 운영체제로 사용했다.

5.2 보안 평가

우리의 공격 탐지 시스템의 보안 평가를 수행하기 위해 turtlebot 3 burger 로봇에 SBC를 탑재하고 우리의 이벤트 수집부와 공격 탐지부를 SBC 내에 구현하였다. 실제 ROS 2 기반의 로봇 시스템과 유사한 애플리케이션에서의 공격 탐지를 위해 turtlebot3_bringup 패키지와 turtlebot3_slam 패키지를 실행하여 Fig. 6.와 같은 ROS 애플리케이션을 구성하였다. 우리는 공격 탐지부에 해당 애플리케이션의 노드, 토픽, 서비스를 포함한 구성요소에 대한 화이트리스트를 자료구조로 생성하였다. 아래의 Table 3.에서는 노드 API의 비순차적 실행을 통한 공격 수행과 이에 대한 탐지 결과를 요약한다.

이와 같이, 악성 페이로드를 탑재한 새로운 노드를 생성하거나 실행 중인 정상 상태의 노드를 종료함

Table 3. Results of security evaluation

Add node	detected
Remove node	detected
Add publisher	detected
Add subscriber	detected
Add client	detected
Add service	detected

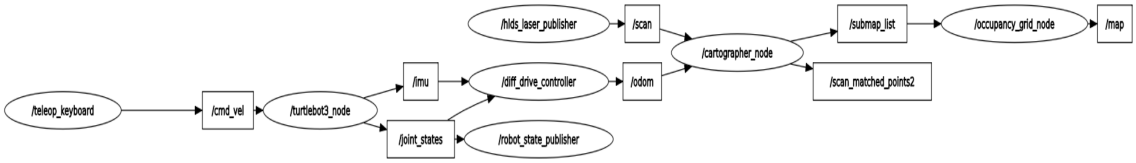

Fig. 6. The ROS 2 application example for security evaluation

으로써 애플리케이션의 구성을 변조하는 공격을 탐지했고, 기존의 토픽과 서비스에 대한 새로운 발행자, 구독자, 서비스 클라이언트 및 서버를 초기화하여 연결을 시도하는 공격을 성공적으로 탐지해 냈다.

5.3 성능 평가

우리의 공격 탐지 시스템의 성능을 평가하기 위해 로봇 시스템의 ROS 2 애플리케이션에서 공격이 발생한 시점으로부터 이를 탐지하는 시점까지 소요된 공격 탐지 시간을 측정하였다. 구체적으로 애플리케이션 상에 임의의 공격자 노드를 생성하며 타임스탬프를 기록하고, 해당 공격자 노드의 생성을 탐지하는 시점에 타임스탬프를 기록함으로써 두 타임스탬프 간의 시간 차이를 기록하였다. 이를 반복 수행함으로써 공격 탐지 시간의 평균값을 도출해 내었다.

또한, ROS 2 애플리케이션의 크기에 따른 공격 탐지 시간의 변화를 파악하기 위해 애플리케이션에서 실행 중인 노드의 개수에 변화를 주면서 탐지 소요 시간을 측정하였다. 노드 개수에 따른 실험 케이스별로 300초간, 총 30회의 공격을 수행하였다. 이때, 애플리케이션에서 실행 중인 노드들은 각각 1:1로 토픽 메시지 통신을 수행하고 각 발행자의 메시지 발행 주기는 500ms 이다. 예를 들어, 노드 10개가 실행 중인 경우는 실험 케이스는 5개의 토픽에 각각

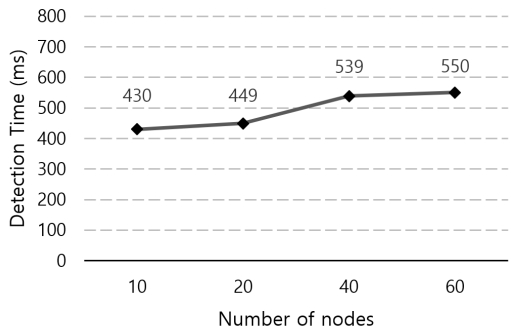

Fig. 7. The average attack detection time

발행자 노드와 구독자 노드가 하나씩 연결되어 있는 형태이다.

Fig. 7.은 애플리케이션의 크기에 따른 공격 탐지 시간을 측정한 결과이다. 평균 공격 탐지 시간은 각 실험 케이스에 따라 430ms, 449ms, 539ms, 그리고 550ms로 측정되었다. 실행 중인 노드가 많아질수록 평균 공격 탐지 시간이 조금씩 증가하는 경향을 확인할 수 있으나 그 폭이 매우 적다. 또한 애플리케이션에 포함된 노드의 개수가 증가함에 따라 이벤트의 발생 빈도가 큰 폭으로 증가하였음에도 불구하고 공격 탐지의 누락은 발생하지 않았다.

5.4 부하 평가

우리의 공격 탐지 시스템으로 인해 발생하는 부하를 평가하기 위해 로봇 시스템에 탑재된 우리 시스템에서 런타임 중에 발생하는 CPU 및 Memory 부하를 측정했다. 이때, 이벤트 수집부에서 컴파일 타임에 실행되는 tracetools를 통한 계측 지점 삽입 과정에 대한 부하는 측정하지 않았다. 우리 시스템으로 인한 부하를 평가하기 위해 런타임 중에 수행되는 이벤트 계측, 추적 데이터 생성, 실시간 공격 탐지 등의 모든 프로세스를 고려하였다.

우리는 baseline으로서 기존의 ROS 2 미들웨어와 애플리케이션을 구성하는 노드들의 CPU 및 메모리 사용량을 측정하였다.

또한, 이벤트 수집부로 인한 부하를 측정하기 위해, 계측 지점이 삽입된 ROS 2 미들웨어, ROS 2 애플리케이션, 그리고 이벤트 수집부의 LTTng 프로세스들의 CPU 및 메모리 사용량을 측정하였다.

마지막으로, 공격 탐지부를 포함한 전체 시스템으로 인한 부하를 측정하기 위해 계측 지점이 삽입된 ROS 2 미들웨어, ROS 2 애플리케이션, LTTng 프로세스들, 그리고 공격 탐지부의 Babeltrace 2 프로세스의 CPU 및 메모리 사용량을 측정하였다.

앞선 성능 평가와 마찬가지로 각 실험 케이스 별로 노드 개수를 변화시켰다. 동일하게 발행자 노드와

구독자 노드는 1:1로 구성되며, 각 발행자의 메시지 발행 주기는 500ms 이다. 또한, CPU 및 메모리의 사용량은 각 실험 케이스마다 300초간 측정하였다.

Fig. 8.은 baseline의 CPU 사용량과 이벤트 수집부만을 포함한 CPU 사용량, 그리고 공격 탐지부를 비롯한 전체 시스템을 포함한 CPU 사용량을 비교한다. baseline에 대해 이벤트 수집부로 인한 CPU 부하는 각 실험 케이스 별로 3.9%, 4.6%, 12.1%, 30.9%이다. 또한, baseline에 대해 이벤트 수집부와 공격 탐지부를 모두 포함한 CPU 부하는 각 실험 케이스 별로 29.6%, 33.0%, 38.4%, 49.2%이다. 우리는 노드의 개수가 증가함에 따라 발생하는 CPU 부하가 증가하는 것을 확인하였다.

우리의 전체 시스템을 탑재할 경우, 최소 29.6%에서 최대 49.2%의 CPU 부하가 발생하며, 이는 노드의 개수가 증가하면서 공격 탐지 시스템의 런타임 모니터링 연산이 빈번해지기 때문이다. 하지만, 오직 이벤트 수집부로 인해 발생하는 CPU 부하는 최소 3.9%에서 최대 30.9%이며, 이는 공격 탐지부를 로봇 시스템의 외부에 구성하는 경우에 로봇 시스템에 가해지는 CPU 부하가 상당히 줄어들 것이다.

Fig. 9.는 baseline의 메모리 사용량과 이벤트 수집부만을 포함한 메모리 사용량, 그리고 공격 탐지부를 비롯한 전체 시스템을 포함한 메모리 사용량을 비교한다. baseline에 대해 이벤트 수집부로 인한 메모리 부하는 각 실험 케이스 별로 3.1%, 4.3%, 5.1%, 6.9%이다. 또한, baseline에 대해 이벤트 수집부와 공격 탐지부를 모두 포함한 메모리 부하는 각 실험 케이스 별로 3.4%, 4.7%, 5.5%, 7.2%이다. 우리는 노드의 개수가 증가함에 따라 메모리 부하가 약간 증가하는 것을 확인하였고, 또한 이벤트 수집부만을 포함한 경우와 전체 시스템을 모두 포함한 경우에서 메모리 부하는 크게 차이가 발생하지 않는 것을 확인하였다.

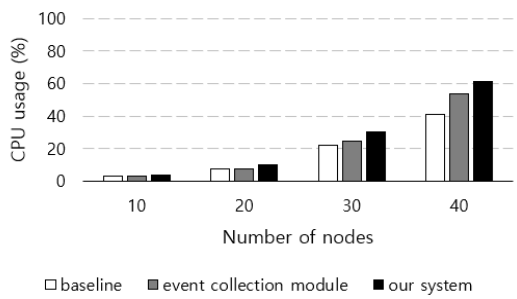

Fig. 8. The CPU overhead of our system

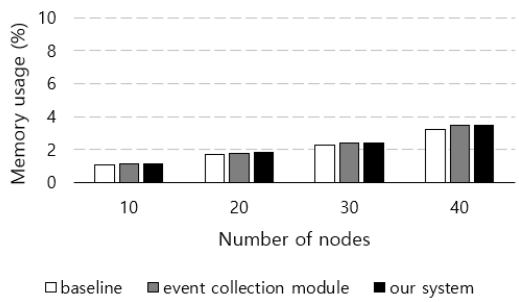

Fig. 9. The memory overhead of our system

VI. 관련 연구

6.1 학습 모델 기반 이상 탐지

학습 기반 이상 탐지는 시스템에서 수집한 동작 데이터가 사전 훈련된 기계 학습 모델을 만족하는지 확인하여 비정상적인 로봇 동작과 공격 행위를 탐지하는 기법이다[18][19].

Narayanan 등[20]은 ROS 애플리케이션에 대한 학습 기반 이상 탐지기를 제안했다. 구체적으로 산업용 로봇 팔이라는 타겟에 대하여 정상 및 비정상 작업에 대한 동작 데이터를 수집하고, 데이터를 정상 또는 비정상 행위로 분류하는 support vector machine과 같은 기계 학습 모델을 훈련하였다. 이러한 학습 모델을 기반으로 산업용 로봇 팔의 동작을 런타임 모니터링하여 이상 행위를 탐지한다. 이들은 ROS ADM(Anomaly Detection Module)이라는 탐지기를 구현하고 이를 ROS 미들웨어에 통합하였다.

이러한 기계 학습 기반의 탐지 시스템은 일반적으로 추가적인 하드웨어를 요구하지 않는다는 장점이 있으며, 또한, 유저가 정상 및 비정상 행위에 대한 사전 정의를 할 필요가 없다는 장점이 있다. 이러한 장점들로 인해 학습 기반 이상 탐지는 여러 로봇 시스템에 쉽게 확장될 수 있다는 장점이 있다.

6.2 시퀀스 유사도 기반 이상 탐지

시퀀스 유사도 기반 이상 탐지는 데이터의 시퀀스 구조를 고려하여 순차적으로 발생하는 이벤트 시퀀스의 유사도를 기반으로 이상을 탐지하는 기법이다[21]. 구체적으로 테스트 시퀀스 데이터베이스에서

비정상적인 시퀀스를 감지하거나, 길게 이어진 시퀀스 내에서 변칙적인 연속 하위 시퀀스를 감지하거나, 테스트 시퀀스에서 비정상적인 발생 빈도에 대한 패턴을 감시한다[22].

서강욱 등[23]은 시퀀스 유사도 기반 무인 비행체 이상 탐지 시스템을 제안하였다. 이들은 무인 비행체에 UAVCAN 프로토콜 기반의 네트워크를 구성하고 내부 네트워크에서 세 가지의 악의적인 메시지 주입 공격을 수행하였으며, nLCS, Levenshtein Distance, Jaccard Similarity, Jaro Similarity 와 같은 시퀀스 유사도 측정 알고리즘을 적용하여 시퀀스 유사도 기반 이상 탐지를 수행하였다. 또한, 이들은 실험을 통하여 위의 네 가지 알고리즘을 통한 시퀀스의 유사도 점수를 계산하여 정상 시퀀스와 비정상 시퀀스를 효과적으로 분류하였다. 결과적으로, 이들은 무인 비행체와 지상 통제 시스템 간의 통신 프로토콜과 무인 비행체의 일반적인 시스템 구성을 비롯한 다양한 특징을 고려하여 무인 비행체에서 발생하는 메시지 주입 공격을 성공적으로 탐지해냈다.

하지만, 해당 연구는 필수적인 모듈만이 구성된 소형 드론에 대한 탐지를 수행하였다. 농업용, 감시용, 군사용 등 다양한 목적과 요구사항을 가진 무인 비행체는 다양한 모듈과 복잡한 네트워크 구성을 가지며, 목적에 따라 다양한 기능을 수행한다[24][25]. 따라서, 해당 연구에서 한계점으로 서술한 바와 같이 복잡한 구성과 다양한 기능을 수행하는 중대형 무인 비행체에 대한 효용성 검증이 필요하다.

VII. 고 찰

성능 평가에 의하면 우리의 공격 탐지 시스템에서 공격 발생부터 탐지까지 소요되는 평균 시간은 약 60개의 노드가 동작하는 비교적 큰 애플리케이션에 대하여 550ms이었으며, 이는 10개의 노드로 구성한 실험의 평균 공격 탐지 시간인 430ms로부터 크게 지연되지 않았다. 다만, 우리의 공격 탐지 시스템으로 인해 로봇 시스템에서 발생하는 CPU 부하는 최소 29.6%에서 최대 49.2%로 애플리케이션의 규모에 따라 부하의 발생이 다소 커지는 것을 확인하였다. 29.6%의 CPU 부하는 ROS 2에 대한 모니터링을 수행하는 기존 연구에 비해 크게 벗어나는 수치는 아니지만, 실시간 기능을 지원하거나 임베디드 시스템으로 구성되는 일부 로봇 시스템에서는 여전히 큰 부담이 될 수 있다.

이러한 부하를 최소화하기 위해서 런타임 모니터링 및 공격 탐지 연산을 수행하는 공격 탐지부를 원격 서버와 같은 외부 시스템으로 분리하여 탑재함으로써 로봇 시스템에서 발생하는 부하를 최소화할 수 있다. 우리의 실험에 따르면 공격 탐지부를 제외한 CPU 부하는 최소 3.9%에서 최대 30.9%로서 로봇 시스템에 대한 CPU 부하를 크게 줄일 수 있다.

하지만, 이를 위해서는 로봇 시스템과 공격 탐지부가 탑재된 외부 시스템을 네트워크로 연결해야만 한다. 따라서 공격 탐지 시스템의 공격 탐지 시간은 네트워크의 사양과 구성 환경에 직접적인 영향을 받는다. 이로 인해 로봇 시스템에 우리의 모든 공격 탐지 시스템이 구현된 경우에 비해서 공격 탐지 시간은 필연적으로 증가할 수밖에 없다. 이는 성능과 부하 간에 trade-off 관계가 성립한다고 볼 수 있다. 따라서 공격 탐지의 성능 측면과 로봇 시스템의 부하 측면을 모두 고려하여 시스템을 구현할 필요가 있다.

VIII. 결 론

본 연구에서는 ROS 2에서의 이벤트 기반 런타임 모니터링을 활용한 실시간 공격 탐지 시스템을 제안하였다. 우리의 공격 탐지 시스템은 기존 연구와 달리 애플리케이션과 노드 통신 메커니즘에 영향을 끼치지 않으며, ROS 2의 네트워크 보안 표준과 무관하게 동작한다. 본 연구에서는 노드의 취약점을 악용하여 노드 API의 직접적인 비순차적 실행을 통한 공격을 탐지해 냈으며, 10개의 노드가 동작하는 시스템에서 평균 430ms의 공격 탐지 시간이 소요되었으며 누락된 공격 탐지는 발생하지 않았다. 또한, 실험 결과에 따르면 우리의 공격 탐지 시스템으로 인한 CPU 부하는 최소 29.6%에서 최대 49.2%로 발생하였으며, 이벤트 수집부만이 구현된 경우 최소 3.9%에서 최대 30.9%가 발생하였다.

향후 연구 계획으로 로봇 시스템에 가해지는 부하를 최소화하기 위해 이벤트 수집부의 최적화 및 소형화에 대한 연구를 진행할 것이다. 더불어, 모델 학습 기반 이상 탐지, 이벤스 시퀀스 유사도 기반 이상 탐지 등 이벤트 로그 기반의 다양한 런타임 보안 연구로 확장할 것이다.

References

[1] Rizk, Yara, Mariette Awad, and Edward W. Tunstel. "Cooperative heterogeneous multi-robot systems: A survey," ACM Computing Surveys (CSUR), vol. 52, no. 2, pp. 1-31, Apr. 2019.

[2] Twinkle Jain and Gene Cooperman. "DMTCP: Fixing the single point of failure of the ros master", ROSCon 2017, Sep. 2017.

[3] Pushyami Kaveti and Hanumant Singh. "ROS rescue: fault tolerance system for robot operating system." Robot Operating System (ROS), Studies in Computational Intelligence, vol 895, Springer, Cham, pp. 381-397, 2021.

[4] Ruffin White, Henrik Christensen, and Morgan Quigley, "SROS: Securing ROS over the wire, in the graph, and through the kernel." arXiv preprint arXiv:1611.07060, Nov. 2016.

[5] Jongkil Kim, J. M. Smereka, Calvin Cheung, Surya Nepal, and Marthie Grobler, "Security and performance considerations in ros 2: A balancing act." arXiv preprint arXiv:1809.09566, Sep. 2018.

[6] Patrice Lacroix and Jules Desharnais. "Buffer Overflow Vulnerabilities in C and C++." Rapport de Recherche DIUL-RR-0803, Université Laval, Québec, Canada, Aug. 2008.

[7] José D'Abruzzo Pereira, Naghmeh Ivaki, and Marco Vieira. "Characterizing Buffer Overflow Vulnerabilities in Large C/C++ Projects." IEEE Access, vol 9, pp. 142879-142892, Oct. 2021.

[8] Saleh M. Alnaeli et al. "Vulnerable C/C++ code usage in IoT software systems." 2016 IEEE 3rd World Forum on Internet of Things (WF-IoT), IEEE, pp. 348-352, Dec. 2016.

[9] Marc Pichler, Bernhard Dieber, and Martin Pinzger. "Can i depend on you? mapping the dependency and quality landscape of ros packages." 2019 third IEEE international conference on robotic computing (IRC), IEEE, pp. 78-85, Feb. 2019.

[10] Jeff Huang, et al. "ROSRV: Runtime verification for robots." International Conference on Runtime Verification. Springer, Cham, pp. 247-254, 2014.

[11] Angelo Ferrando, et al. "ROSMonitoring: a runtime verification framework for ROS." Annual Conference Towards Autonomous Robotic Systems. Springer, Cham, pp. 387-399, Dec. 2020.

[12] Sean Rivera, et al. "ROS-FM: Fast Monitoring for the Robotic Operating System (ROS)." 2020 25th International Conference on Engineering of Complex Computer Systems (ICECCS). IEEE, pp. 187-196, Oct. 2020.

[13] Christophe Bédard, Ingo Lütkebohle, and Michel Dagenais. "ros2_tracing: Multipurpose Low-Overhead Framework for Real-Time Tracing of ROS 2." IEEE Robotics and Automation Letters, vol. 7, no. 3, pp. 6511-6518, Jul. 2022.

[14] Mathieu Desnoyers and Michel R. Dagenais. "The lttng tracer: A low impact performance and behavior monitor for gnu/linux," Proceedings of the Linux Symposium, OLS (Ottawa Linux Symposium), pp. 209-224, Jul. 2006.

[15] The LTTng Project, "The LTTng Documentation", https://lttng.org/docs/v2.1

1/, 2021
[16] Babeltrace, “Babeltrace2 documentation”, https://babeltrace.org/docs/v2.0/man7/babeltrace2-intro.7/,
[17] efficios/babeltrace, “babeltrace”, https://github.com/efficios/babeltrace, 2021
[18] Salima Omar, Asri Ngadi, and Hamid H. Jebur. "Machine learning techniques for anomaly detection: an overview." International Journal of Computer Applications, vol. 79, no. 2, pp. 33-41, Oct. 2013.
[19] Qiang Liu, Tao Han, and Nirwan Ansari. "Learning-assisted secure end-to-end network slicing for cyber-physical systems." IEEE Network, vol. 34, no. 3, pp. 37-43, Jun. 2020.
[20] Vedanth Narayanan and Rakesh B. Bobba. "Learning based anomaly detection for industrial arm applications." Proceedings of the 2018 Workshop on Cyber-Physical Systems Security and PrivaCy, pp. 13-23 Oct. 2018.
[21] Varun Chandola, Arindam Banerjee, and Vipin Kumar. "Anomaly detection: A survey." ACM computing surveys (CSUR), vol. 41, no. 3, pp. 1-58, Jul. 2009.
[22] Varun Chandola, Arindam Banerjee, and Vipin Kumar. "Anomaly detection for discrete sequences: A survey." IEEE transactions on knowledge and data engineering, vol. 24, no. 5, pp. 823-839, May. 2012.
[23] Kang-uk Seo, and Huy-kang Kim, “Sequence Based Anomaly Detection System for Unmanned Aerial Vehicle”, Journal of The Korea Institute of Information Security and Cryptology, 32(1), pp. 39-48, Feb. 2022.
[24] Anam Tahir, et al. "Swarms of unmanned aerial vehicles—a survey." Journal of Industrial Information Integration, vol. 16, Dec. 2019
[25] Hazim Shakhatreh, et al. "Unmanned aerial vehicles (UAVs): A survey on civil applications and key research challenges." IEEE Access, vol. 7, pp. 48572-48634, Apr. 2019.

〈저자소개〉

강 정 환 (Jeonghwan Kang) 학생회원
2021년 2월: 한국항공대학교 항공전자정보공학부 학사
2021년 3월~현재: 부산대학교 정보융합공학과 석박사통합과정
〈관심분야〉 시스템 보안, 하드웨어 보안

서 민 성 (Minseong Seo) 학생회원
2020년 3월~현재: 부산대학교 정보컴퓨터공학부 학사과정
〈관심분야〉 시스템 보안, 정보보호

박 재 열 (Jaeyeol Park) 학생회원
2021년 3월~현재: 부산대학교 정보컴퓨터공학부 학사과정
〈관심분야〉 시스템 보안, 네트워크 보안

권 동 현 (Donghyun Kwon) 정회원
2012년 2월: 서울대학교 전기컴퓨터공학부 학사
2019년 2월: 서울대학교 전기컴퓨터공학부 박사
2019년 3월~2020년 2월: 한국전자통신연구원 연구원
2020년 3월~현재: 부산대학교 정보컴퓨터공학부 조교수
〈관심분야〉 시스템 보안, 소프트웨어 보안