B. Wang, R. Wang and H. Song, "Toward the Trustworthiness of Industrial Robotics Using Differential Fuzz Testing," in IEEE Transactions on Industrial Informatics, 2022, doi: 10.1109/TII.2022.3211888.

https://doi.org/10.1109/TII.2022.3211888

# Toward the Trustworthiness of Industrial Robotics Using Differential Fuzz Testing

Bingqing Wang, Rui Wang, and Houbin Song.

***Abstract*—Intelligent robots are a current application in Industrial Internet of Things (IIoT), with their trustworthiness being a topic of considerable research interest. Vulnerabilities in robot software may affect the trustworthiness of robotics. To detect these vulnerabilities in robot software, this study proposes a differential fuzz testing method. The main idea is to continuously execute test cases for different versions of software packages to detect inconsistencies among outputs and eventually discover vulnerabilities. First, test cases are generated, combining seed generation and mutation, after which the measured model of the packages in RVIZ is built and the generated seeds are executed. The differences among inconsistent outputs are calculated and the causes of the differences analyzed. Two evaluation metrics for the inconsistencies and seeds are presented. This method is applied to the crucial package in ROS-MoveIt!. The results show that the *arm.go()* of *moveit_commander* has joint angle overflow.**

***Index Terms*—differential fuzz testing, IIoT, robotics, trustworthiness**

## I. Introduction

WITH the development of information technology and industry, the applications of Industrial Internet of Things (IIoT), such as intelligent robots and sensors in the aircraft and medical device manufacturing industries to name a few, have increased. The deep integration of interconnected devices and physical controls exposes systems that use this technology to new forms of attack and therefore, the application of the IIoT presents challenges such as security issues. Vulnerabilities are defects in the specific implementation of hardware, software, protocols, and system, which can enable attackers to gain unauthorized access or destroy the system. If the software of robots has such vulnerabilities, the intelligent robots will cause high-risk situations and even be life-threatening. As guaranteeing the trustworthiness of robot software is extremely important, it is a key research topic. For example, ROS (robot operating system) is a universal robot development platform [1], [2], [3], which runs many software packages; hence, vulnerabilities hidden in any package version of ROS would affect multiple users. When the static analysis tool Coverity was used to test various ROS communication function packages, such as *ros_comm*, *actionlib*, and *roslib*, multiple dangerous vulnerabilities, such as buffer overflow, integer overflow and unsafe YAML loading, were discovered. The alarming fact is that static analysis is a traditional software testing method that does not run the target program [11] but rather checks the correctness of the program by analyzing and viewing the syntax, structure, process, and interface of the source program [12]. Hence, it can only detect a limited set of common errors, such as array binding operations and potential deadlock problems [13]. Its main disadvantages include a high false positive rate and the poor readability of test results.

Over the past few years, fuzzing [30], a gray box testing method [14], has been widely applied in the software testing field, such as operating systems [18], databases [19], web applications [20] and blockchain [21]. Fuzzing finds software vulnerabilities by providing unexpected inputs and monitoring abnormal results [16] and is thus, one of the most common software testing techniques for detecting bugs and vulnerabilities. The effectiveness of fuzzing mainly depends on the quantity and quality of the generated test cases. At present, the main generation methods of test cases can be classified into generation-based and mutation-based methods. Under generation-based methods, there are some popular fuzzers such as ContractFuzzer [24] and SPIKE [25], which generate inputs based on specific formats. In mutation-based methods, fuzzers mutate existing test cases to generate new test cases without any input grammar. Fuzzing does not consider the internal implementation of the target program, but rather uses the malformed inputs to cause the target program to produce anomalies and find vulnerabilities [17]. Unlike traditional testing methods, fuzzing [22] is not limited to the internal implementation details and complexity of the system. Hence, it not only saves time and effort, but it also has a low false alarm rate [31]. Therefore, fuzzing is an efficient and convenient test method to ensure the trustworthiness of the ROS packages. However, due to the particularity of ROS, two challenges remain.

- The basic execution unit of ROS is a node and it has both asynchronous and synchronous data stream communication. As different communication mechanisms have different data formats, specific test cases for each mechanism must be generated.

Manuscript received on April 29, 2022; revised on June 10, 2022 and July 29, 2022; accepted on September 20, 2022. This work was supported by the National Natural Science Foundation of China under Grants 61877040, National Key R&D Plan of China (2019YFB1309900).

B. Wang and R. Wang are with the Beijing Key Laboratory of Electronic System Reliability Technology, College of Information Engineering, Capital Normal University, Beijing 100048, China (e-mail: 3301369886@qq.com;rwang04@cnu.edu.cn).

H. Song is with the Department of Information Systems, University of Maryland, Baltimore County, Baltimore, MD 21250 USA (email: h.song@ieee.org).

- Some software packages often incorporate RVIZ (3D visualization tool for ROS) to monitor and control the behaviors of robotics. Therefore, this study cannot use current fuzzing tools to test the software packages.

Although some packages are implemented in different languages or different versions, they can achieve the same function. For these packages, this study proposes a differential fuzz testing method to verify the accuracy of various functions in practical applications. The concept of differential fuzz testing is to continuously provide invalid, unexpected or random data as inputs of several programs with the same functions and monitor these programs to catch "different behavior" in terms of certain outputs. First, a fuzz testing framework for ROS packages is established. According to different requirements, the generation and mutation strategies to generate test cases are designed, after which the concept of differential testing [21] is integrated into the framework. This study uses a unified test case file as the fuzzing input and opens the visualization tools to execute the test. The main idea is to continually generate test cases for different communication mechanisms so that as many inconsistencies as possible can be found among execution results. Finally, the method is applied to experiments on robotic arm control and the function packages that implement motion planning in robot joint space and workspace are tested: *moveit_commander* implemented by python and *move_group_interface* implemented by C++. This study generates 15246 seeds using fuzzer. It is found that 9266 seeds triggered inconsistencies between *moveit_commander* and *move_group_interface*. After analysis, *move_group_interface* was found to be more accurate than *moveit_commander* on some function implementation. The *arm.go()* of *moveit_commander* has joint angle overflow.

***Contributions*** This study makes the following contributions.

- It implements a differential fuzz testing framework RROSFuzz (differential **Fuzz** testing combining **R**VIZ with **ROS** packages) to efficiently find the differences and vulnerabilities among different versions of packages.
- Two evaluation metrics for differential fuzzing are introduced and four generation strategies and three mutation strategies for test case generation are defined.
- RROSFuzz is applied to test the most widely used software packages in ROS: *move_group_interface* and *moveit_commander*. Many inconsistencies are found and *move_group_interface* is verified as more accurate than *moveit_commander*.

***Paper Organization*** The rest of paper is organized as follows. Section II provides a background on ROS and robot software package, Section III introduces the design of RROSFuzz, Section IV shows the evaluation results, and Section V proposes future directions for improvement. Section VI surveys related work and finally, Section VII concludes the paper.

## II. Background

### A. Robot operating system

ROS is a flexible framework for writing robot software [4], which integrates a large number of tools, libraries and protocols, and thus, can help in considerably simplifying the creation of complex tasks and stable behavior control under diverse robot platforms [6], [7]. The ROS open-source community has many software packages that can achieve the same function; however, these packages are implemented by different languages and institutions. The ROS runtime "graph", a peer-to-peer distributed communication mechanism, creates a network that connects all processes [29] and it is through the network that nodes can interact with each other and obtain information published by other nodes. The computation graph implements other communication mechanisms, such as topic, service, and parameter server communication mechanisns. The message is the data format used by the topic communication mechanism and each message corresponds to a data type. ROS messages not only support standard data types (integer, float and boolean), but also include array and custom data types.

The rapid development of ROS has made it a standard of the robotics field. Therefore, the vulnerability hidden in any package version of ROS might result in serious consequences. For example, MiR robots use ROS to expose the runtime "graph" without any authentication, which allows attackers to arbitrarily command the robot. The ROS communication package *ros_comm* has a buffer overflow vulnerability, which allows attackers to cause denial of service.

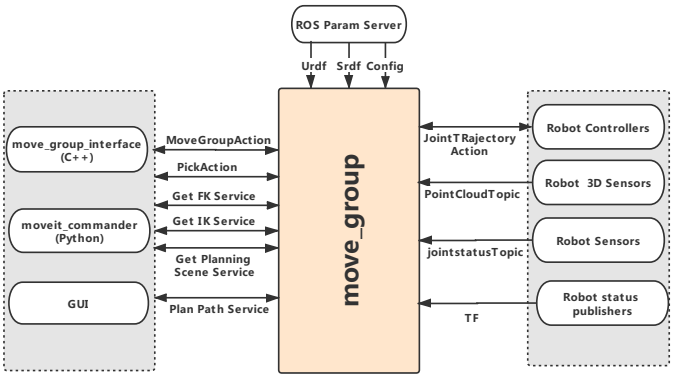



Fig. 1. Architecture diagram of MoveIt!.

### B. Robot software package

- MoveIt! is the most advanced software for movement operations [5] and is widely used in industry, business, development, and other fields. As shown in Fig. 1, MoveIt! provides three interfaces, including C++, python, and GUI. The C++ and python interfaces can use the API provided by *move_group_interface* and *moveit_commander* to implement motion planning in the joint space and workspace of the robot. These three interfaces can be used to interact with *move_group* through the communication of action and service. *Move_group* is the core node of MoveIt! and it can integrate other independent functional components to provide users with action instructions and services [8]. By the communication topic and service, *move_group* receives point cloud messages, joint status messages, and robot TF coordinate transformation from the robot. In addition, *move_group* requires the ROS parameter server to provide the kinematic parameters of

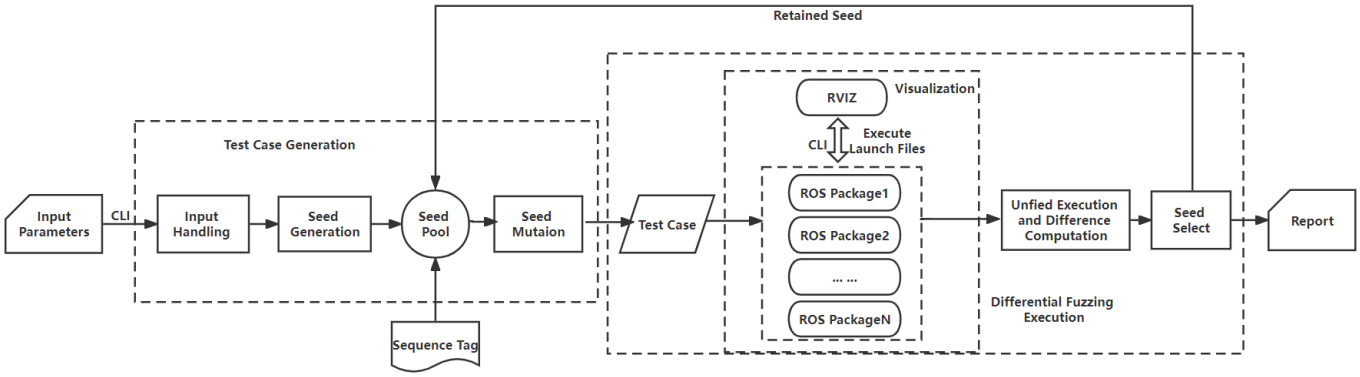



Fig. 2. Overview of RROSFuzz: It consists of the test case generation module and differential fuzzing execution module.

the robot. These parameters will be generated based on the unified robot description format (URDF) file during the the setup_assistant. The URDF is a standard XML file. It defines a series of labels (such as links and joints) to describe the robot model. The URDF file needs to be written and implemented by the users.

- RVIZ is a 3D visualization tool, which is compatible with various robot platforms. In RVIZ, XML can be used to describe the size, quality, position, material, joints, and other attributes of any physical objects such as robots and surrounding objects. At the same time, RVIZ can also graphically display the information of the robot's sensors, movement status and the changes in the surrounding environment in real time.

## III. RROSFuzz Design

An overview of RROSFuzz is given in Fig. 2. It consists of two major parts: test case generation and differential fuzz execution. The input parameters refer to the message type and topic name. They are passed to the test case generation module by the command line interface (CLI). The test case generation module oversees handling input, seed generation and mutation. The sequence tag enables each set of seeds to obtain a unique number, which facilitates the review and analysis of subsequent steps. Seed generation and mutation are mainly based on seed generation and mutation strategies. After the test cases are generated, RROSFuzz enters the execution module for testing. The execution module uses the topic mechanism to realize node communication (II.A), and combines with the ROS visualization tool RVIZ to realize human–computer interaction. According to the results of the execution, the module will calculate the difference information. Seed selection will keep and sort candidate seeds for the next iteration mutation based on the difference. When the execution output is inconsistent, the potential exception is recorded for cause analysis and a report is obtained. The implementation of the important components are depicted in Fig. 2. This study also introduces the evaluation indicators to improve the quality of seeds in differential fuzzing execution.

### A. Input Handling

Input handling creates a dictionary and dynamically imports ROS message modules. By executing shell instruction, this study obtains arguments from CLI, which are then parsed to achieve msg_type. RROSFuzz receives msg_type as input and determines the type. When the type is not known, it will create a dictionary via dynamically loading the ROS message file. The dictionary contains ROS message type and its parent module. This type of file ends with .msg and describes the fields of the ROS message. The ROS message fields can be used to generate source codes based on different programming languages. Then, the RROSFuzz will generate the ROS message class.

### B. Seed Generation

Seed generation mainly involves generating initial seeds based on the generation hypothesis strategies for a given ROS message type. Algorithm 1 first creates a strategy dictionary (Algorithm 1 line 3). Next, the algorithm obtains a list via combining the itertable fields in the message class into a new iterator through the zip function (Algorithm 1 line 4). Then, it traverses the list, creates a type dictionary for the message type, and maps the type dictionary to a strategy dictionary (Algorithm 1 line 6-8). Finally, the message class is filled according to the strategy dictionary to generate initial seeds (Algorithm 1 line 10). RROSFuzz decorates the seeds with *given()*, calls the seed generation module to randomly generate test cases, and sets max_examples in the *setting()* decorator to control the number of test cases. Details of the generation strategies are shown in Table 1.

**Algorithm 1** Seed Generation

1: **Input:** a class describes properties of msg_type $C$
2: **Output:** generated initial seeds $S$
3: *strategy_dict* ← *empty-dict*
4: *iterator* ← *combine(C.name, C.type)*
5: */* Mapping type dictionary to strategy dictionary */*
6: **for** *name,type* in *iterator* **do**
7: *type_dict* ← *ros_type_to_dict(s_type)*
8: *strategy_dict* ← *mapping_strategies(type_dict[name])*
9: **end for**
10: $S$ ← *strategy_generator(msg_class,strategy_dict)*

**Generation Strategies.** As shown in Table I, four strategies are defined, including array, time, string, and combination strategies based on the *st* module of the hypothesis package

TABLE I
GENERATION STRATEGIES

| Message Type | The corresponding function of The strategy dictionary | Call strategy |
|---|---|---|
| string | string() | st.text |
| time | time() | _Time |
| array | array() | st.lists |
| complex | ros_type_to_dict() | combination strategy |

in Python to guide seed generation. For example, a *st.text* strategy is defined to guide seed generation for string. For the complex message type such as PointStamped in ROS, a combination strategy is defined. This strategy will split the message type, each part corresponding to its own strategy, and finally combine them together and return a combination strategy corresponding to the complex message type.

### C. Seed Mutation

Seed mutation can also be used to generate seeds. As Algorithm 2 shows, the algorithm first sets up *mutation_seeds* and *mapping_dict* (Algorithm 2 line 3-4). The initial state of the *mutation_seeds* and *mapping_dict* is empty. The *mapping_dict* represents the mutation strategy used for each seed. Then, the algorithm traverses strategy_list and selects the corresponding mutation strategy to mutate the seeds (Algorithm 2 line 6-9). The mutated seeds will be saved in the *mutation_seeds* list (Algorithm 2 line 8). To facilitate the evaluation of the effectiveness of the mutation strategy, the algorithm saves the seed and mutation strategy as key and value in *mapping_dict* respectively (Algorithm 2 line 9). Finally, the mutated seeds will be written into the file (Algorithm 2 line 11).

**Algorithm 2** Seed Mutation

1: **Input:** initial seed *S*, mutation strategies *strategy_list*
2: **Output:** mutated seeds file *f*
3: *mutation_seeds* ← *empty-list*
4: *mapping_dict* ← *empty-dict*
5: */*Mutate and retain seeds and mutation strategy used*/*
6: **for** *i=0* → *len(strategy_list)* **do**
7: *S'* ← *mutate(S,strategy_list[i])*
8: *mutation_seeds* ← *append(mutation_seeds,S')*
9: *mapping_dict[S']* ← *strategy_list[i]*
10: **end for**
11: *f* ← *writeFile(mutation_seeds)*

The specific mutation strategies are shown in Table II. The decimal part can have infinite digits when floats are converted to binary and this causes a loss of precision. Thus, the bit flip strategy focuses on the integers and the integer part of floats. It realizes mutation via transforming integer to binary and setting the step size and flip amount. The second mutation strategy is arithmetic and achieves some operations by setting the upper and lower limits of addition and subtraction. The final mutation strategy, presets some "interesting values" to substitute data and achieve mutation. As shown in Fig. 3, the "interesting values" are generally numbers that may cause overflow.

Using the aforementioned three mutation strategies, this study generates abundant test cases. In addition, four combined

```
-128, -1, 0, 1, 16, 32, 64, 100, 127
-32768, -129, 128, 255, 256, 512, 1024, 4096, 32767
-2147483648LL, -100663046, -32769, 32768, 65535, 65536, 100663045, 2147483647
```

Fig. 3. Examples of "interesting values".

strategies to further improve the randomness and diversity of mutation in each iteration are defined.

- Comb1: Combination of the bit flip and arithmetic.
- Comb2: Combination of the bit flip and interesting.
- Comb3: Combination of the interesting and arithmetic.
- Comb4: Combination of the interesting, bit flip and arithmetic.

TABLE II
MUTATION STRATEGIES

| Name | Object Type | Description Of Mutation Strategy |
|---|---|---|
| bit flip | float,integer | transform integer to binary<br>set the step size and flip amount |
| arithmetic | float,integer | perform addition and subtraction<br>on integer and float |
| interesting | string,time<br>float,integer | set different interesting values<br>different types of data |

### D. Unified Execution and Difference Computation

After mutating the seeds, the metric difference in the ROS packages' execution of seed backtracking and mutation in the next iteration is compared. The RROSfuzz execution provides a unified runtime environment for the proposed target program.

The basic execution unit of ROS is a node. ROS also has a tool for managing nodes, namely the master, which is equivalent to the management center of the entire network communication architecture. The node is first registered at the master, which then incorporates the node into the entire ROS program. The first step is to get the message data that can directly feed into target programs implemented by ROS packages with different versions. Executing command *roscore* turns the master on. Then, one needs to enter into the workspace and compile the packages via executing *catkin_make* command. Finally, the nodes are started through the *rosrun* command, and RVIZ is opened through the command line or the launch file is executed to realize the graphical display of external information. The second step is to analyze the output results and calculate the difference in the next iteration.

### E. Evaluation Indicators

The main purpose of ROS is to facilitate the writing of robot programs. Based on the reliability and availability of data involved in robot development, this study specifies two indicators: early input filtering and reasonable difference.

- **Early Input Filtering.** When the fuzzer generates test files according to the input type of the target program, it does not consider the rationality of the data in the actual development and application. The purpose of establishing the input filtering indicator is to remove data that do not meet the actual application of the test module, such as

nan, inf, -inf and other data that are too large or too small, keep some meaningful experimental data to achieve the purpose of data optimization.

- **Reasonable Difference.** Different programs may present a difference in processing data accuracy. To prevent this difference impacting the results, a reasonable difference indicator is set. Data analysis generally includes qualitative analysis and engineering calculation. Qualitative analysis usually keeps two decimals for data whereas engineering calculation keeps significant decimals between four and six. To ensure the validity and actual meaning of the data, the range is set between $10^{-4}$ and $10^{-6}$. In the specific experiment, the function of the program and application are combined to make a reasonable selection within this range.

Based on the aforementioned two indicators, the evaluation system are further defined. First, RROSFuzz completes the generation of test cases through predefined generation and mutation strategies. Next, the seeds that undergo the early input filtering process will be retained. Then, the difference information is automatically calculated by executing the program. This information will be assessed using reasonable difference. The quality of the seeds after the evaluation will be much higher, further improving the efficiency of the experiment and the accuracy of the experimental results.

**Algorithm 3** Seed Selection

1: **Input:** mutated seeds *S'*, maximum difference *record*
2: **Output:** a list *candidate_seeds*,
seed difference priority *diff_pri*
3: *diff* ← *run(S')*
4: candidate_seeds ← *sort(candidate_seeds,cmp=diff_pri)*
5: */* Keep and sort seeds based on difference */*
6: **if** *isEvaluated(diff)* **then**
7: *diff_pri[S']* ← *diff*
8: **if** *diff>reord* **then**
9: *record* ← *diff /* update maximum difference */*
10: *candidate_seeds* ← *append(candidate_seeds,S')*
11: **else**
12: *candidate_seeds* ← *insert(candidate_seeds,S',diff)*
13: **end if**
14: **end if**

### F. Seed Selection

Difference can be used to measure the seeds' ability of inducing platforms to make differential decisions. Candidate seeds are selected based on the metric difference. As Algorithm 3 shows, the algorithm first obtains the difference after executing seed (Algorithm 3 line 3). The initial *candidate_seeds* are sorted based on the difference priority of each seed (Algorithm 3 line 4). If the difference after executing a mutated seed meets the reasonable difference indicator, the algorithm will save the difference of the seed in *diff_pri* (Algorithm 3 line 6-7). If a mutated seed enlarges the difference after executing, the maximum difference is updated and the seed is appended directly in *candidate_seeds* (Algorithm 3 line 8-10). Otherwise, the seed is inserted into the *candidate_list* based on the difference priority (Algorithm 3 line 12). Seeds with high priority will be considered as high-quality seeds and to have a higher probability of triggering vulnerabilities. These seeds will be preferentially selected in the next mutation iteration. When the final execution output is inconsistent, an exception is recorded and the cause is analyzed.

## IV. Evaluation

In this study, the proposed method is applied to the MoveIt! package, which realizes the control of the robot arm. For robots, the key challenge is to define a path for the robot arm to pick up an object, especially when there are obstacles within the environment. ROS provides MoveIt!, including *moveit_commander* implemented by Python and *move_group_interface* implemented by C++, which, together, can achieve the same function and help users to realize the movement of the robot arm. For developers, the accuracy and reliability of both the interface implementation will directly affect the development process. The experiment details are presented here and the following two questions are answered: (i) Could RROSFuzz mutate high-quality seeds? (ii) Could RROSFuzz find issues between packages through differential fuzz testing?

### A. Data and Environment Setup

ROS has released multiple versions for different Ubuntu versions, such as Kinetic Kame, Melodic Morenia and Lunar Loggerhead. Considering the fairness, to provide a stable operating execution environment, the officially recommended version, Kinetic Kame, was uniformly used. After choosing the ROS Kinetic, all experiments were performed atop the machine with 4 cores (Intel i5-8250U @1.60GHz), 8 GB of memory, and Ubuntu 16.04 as the host operating system.

### B. Experiment Process

This study established the execution flow of the tested module. In Fig. 4, the six-axis robot arm description file *probot_anno.xacro* was prepared, after which the *moveit setup assistant* was started by running the *setup_assistant.launch* to perform a series of configurations on the robot arm. After the configuration was complete, a ROS package was generated. By running *demo.launch* in the ROS package, the visualization tool RVIZ could be started, the robot model loaded and the *move_group* node executed. In the testing process, RVIZ needed to use some packages to display robot information, such as *effort*, *grid* and *robot_model*. Moreover, RVIZ integrated *motionPlanning* to choose the path planning algorithm. Finally, the nodes *moveit_fk_c* written by *move_group_interface* and *moveit_fk_demo* written by *moveit_commander* were run. The function realized by forward kinematics was to set the angles of six joints as the target pose. After the movement of the robot arm was complete, the pose of the end effector of the robot arm could be obtained. Therefore, the input data for the tested module were the angles of the six joints. Array was used to store the values of the joint angles of the six-axis robot arm. The specific experiment process is as follows:

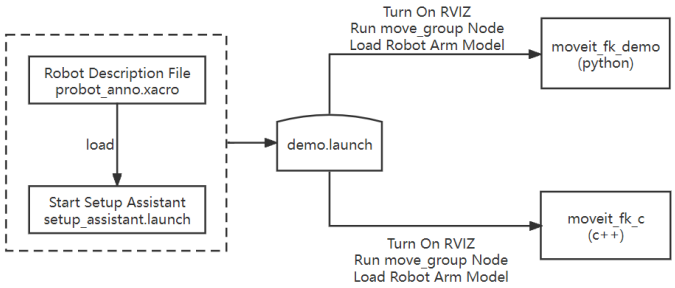



Fig. 4. The execution flow of the tested module.

- **Generate test case.** First, the test case generation module was executed and array type message data were generated. The CLI interface received parameters and transmitted them to the basic module for seed handling. The basic module created a type dictionary of the array and loaded the message file. Next, the strategy processing module called the generation and mutation strategies to complete the initial seed generation and mutation. Finally, a test case file was sent to the next module. For example, Fig. 5 shows a test case file with JSON format. The test case is to represent the joint angle value of the six-axis robot arm, which can be provided to two programs to achieve the movement of robot arm.


```
{"angle1":-0.43947295, "angle2":-0.34223611, "angle3":1.48416344,
"angle4":1.08492838,"angle5":-1.91057891,"angle6":-0.34223611,"seq":1}
{"angle1":0.5107178, "angle2": -0.3204373, "angle3":0.5107178,
"angle4":0.5107178,"angle5":-1.11639336,"angle6":0.5107178,"seq":2}
{"angle1":-1.49674411, "angle2": -0.12400328, "angle3":-1.23686938,
"angle4": 0.17239194, "angle5":0.21743168, "angle6": 0.17239194,"seq":3}
{"angle1":-0.31747301, "angle2": -0.45323972, "angle3":-1.60922462,
"angle4": 0.96381249, "angle5":0.60302567,"angle6": -0.06039972,"seq":4}
{"angle1":-1.97172096, "angle2": 1.71577175, "angle3":-0.93043369,
"angle4":-0.93043369, "angle5":-0.93043369,"angle6": -0.93043369,"seq":5}
{"angle1":0.47859137, "angle2": 0.47859137, "angle3":-0.97741399,
 "angle4":0.47859137,  "angle5":0.47859137, "angle6": 0.47859137,"seq":6}
```


Fig. 5. Example of the generated test case.

- **Perform differential fuzzing.** First, *moveit_fk_c* and *moveit_fk_demo* continuously accepted test cases as inputs. The robot arm used test cases as target positions. After completing the movement, two programs output poses. Next, the difference information about two sets of poses was calculated and the evaluation module was entered. The seeds that met the evaluation indicators were retained, after which we backtracked to the seed pool to continuously generate more high-quality seeds. Finally, the recorded difference information was combined to analyze the reasons for the inconsistent outputs.

### C. Could RROSFuzz generate high-quality seeds?

Within four days, RROSFuzz generated and executed 15246 non-redundant seeds. Among them, 60.78% of the seeds successfully triggered the differential outputs. In the process of mutation, RROSFuzz mutates seeds via basic mutation strategies and combination strategies. If a seed was evaluated as a high-quality seed, we would backtrack and obtain the name of the mutation strategy it used. Then, we applied the same mutation strategy to mutate the seed to generate more high-quality seeds. Furthermore, to evaluate the efficiency and quality of different mutation strategies, this study classified 2646 seeds based on the mutation strategies used.

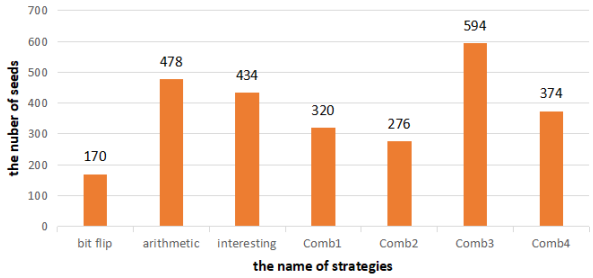



Fig. 6. Statistical data of mutation.

As mentioned in Section III.C, this study designed three basic and four combination mutation strategies, the performance statistics of which are presented in Fig. 6. For the basic strategies, compared with bit flip, arithmetic and "interesting" strategies can mutate more seeds that can trigger inconsistencies. The type of test case required by the target program is an array of floating point numbers. The array stores the six joints' angles which has a relatively small range, such as [-3.9212,3.9212]. Considering that floating-point numbers are converted to binary, a loss of precision in the fractional part is expected. Therefore, this study only focused on the integer part in performing bit flip. While performing bit flip for integers, some data exceeded the joint range, resulting in invalid test cases. Arithmetic and interesting mutation strategies are used for the fractional part and many values can be set, which do not cause the data to exceed the valid range. Therefore, they can mutate more high-quality seeds. For the combination strategies, the third combination strategy, arithmetic and interesting, can mutate 594 seeds to trigger inconsistencies; the proportion is close to a quarter (22.4%). This does not mean that the bit flip mutation strategy was invalid, but in the process of combining with other strategies, it may have filtered some inputs, which had a greater impact on the value of the data,. Hence, it is reasonable to conclude that RROSFuzz can mutate high-quality seeds from the statistics.

### D. Could RROSFuzz find issues between packages through differential fuzz testing?

In the experiment, 15246 test cases were successful executed. Each test case was an array of six joints' angles; the angle of each joint should be within the range of the robot arm description file. If it exceeded the range, the proposed target programs would catch exceptions instead of interrupting the execution of the programs. The distribution of test cases is shown in Fig. 7. 80% of the test cases had this feature: the number of joints' angles close to zero was less than three. The main reason is that after the evaluation process, we backtracked the seeds that could trigger inconsistencies and called the mutation strategies to generate more high-quality seeds. In other words, in the test cases, 70% of the data was more likely to trigger inconsistent outputs. Next, statistical analysis was conducted on the inconsistent results.

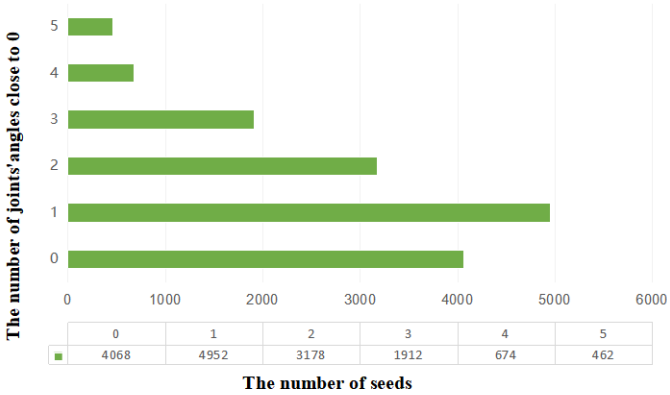



Fig. 7. The distribution of test cases.

As can be seen in Fig. 8 and Tab. III,through analyzing the data, approximately 60.78% of the test cases could trigger inconsistent outputs. The inconsistency outputs consist of the data difference and positive and negative difference. Approximately 48.9% of the data showed inconsistencies between positive and negative. The outputs with inconsistent positive and negative included two aspects, the coordinate values of the end position of the robotic arm and the rotation values of the posture. The proportion of the former was 17.6% and the latter 31.3%. The data for the difference range of $10^{-2}$-$10^{-1}$ accounts for 11.8%; however, approximately 11.9% of the data difference range was less than $10^{-2}$. To a certain extent, these inconsistencies indicate that the inaccuracy in the process of realizing the forward kinematics of the robot arm affected the results.

TABLE III
STATICTICAL DIFFERENCE

| Difference | | Numbers | Proportion |
|---|---|---|---|
| range | $<10^{-2}$ | 5980 | 39.3% |
| | $10^{-2}$-$10^{-1}$ | 1796 | 11.8% |
| positive and negative | coordinate values | 2670 | 17.6% |
| | rotation values | 4760 | 31.3% |

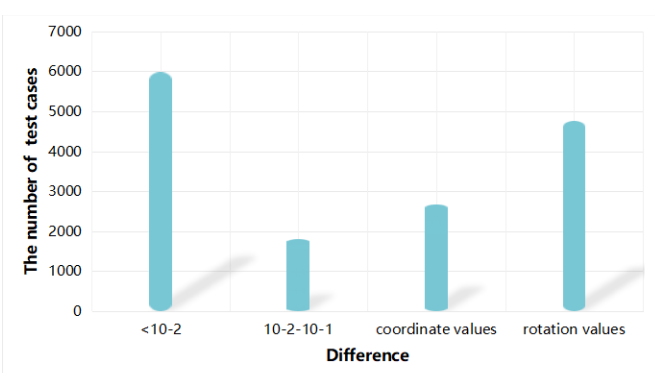



Fig. 8. The statistical results of difference information.

The reason for these differences is further analyzed here. At the beginning, it should be noted that the functions implemented by *moveit_commander* and *move_group_interface* were to control the robot arm movement. The following three steps were required to complete the movement.

First, the robot arm moved to the initial position, which was the robot pose set in the setup assistant.

Second, the target position needed to be set. In this experiment, the array was used as our test case.

Finally, the robot arm moved to the target position, and the difference was calculated by obtaining the pose of the end effector of the robot arm.

In the whole aforementioned process, we verify them one by one. The first step involved moving the robot arm to the initial pose. After the robotic arm returns to the initial position, the function that outputs the current position in two programs was called. The result was that both reached the set initial position. Next, the process of the setting target pose was verified, which involved reading the test case file. Considering whether the data read will be different due to rounding. Therefore, after the test case file was read, all the data was output to compare consistency. By analyzing the results, all the data were consistent. The aforementioned step-by-step verification showed that there was no problem in the first two processes of the experiment. The only difference, here, was the last step, which involved realizing the movement of the robotic arm.

TABLE IV
THE RESULTS OF THE CALCULATION

| | Close to move_group_interface | Close to moveit_commander |
|---|---|---|
| seed numbers | 6909 | 2357 |
| proportion | 74.56% | 25.44% |

Therefore, this study reviewed the source code of the robot arm movement to further analyze the reasons. In *moveit_commander* and *move_group_interface*, *arm.go()* and *arm.move()* respectively was used to realize the movement of the robot arm. During moving, it first called the kinematics planning algorithm and then planned an accessible path based on the target pose. The kinematics planning algorithm used by two functions was uniformly set in the setup assistant. As a result, the reason for the inconsistencies was that, due to the rotation of the joints, deviations were generated when the robot arm was moving. The data for the 9266 sets of inconsistent results generated was analyzed and verified. The statistical results are shown in Table IV. Among the calculation results of each set, 74.56% of the outputs are closer to *move_group_interface*. As shown in Fig. 9, it is found that, due to the overflow of the joint angle, the *moveit_fk_demo* implemented by *moveit_commander* caused the robotic arm to not accurately reach the specified position. Furthermore, because of overflow, the kinematic planning algorithm may not be able to plan the path for the robotic arm. Next, we verify the results and discuss the rationality of using this method.

**Validation Method.** The accuracy of *moviet_commander* and *moveit_group_interface* is verified by obtaining the final joints' angles.

**Rationality of Using this Method.** This study analyzed the whole process. First, the inputs were given, the six joints'

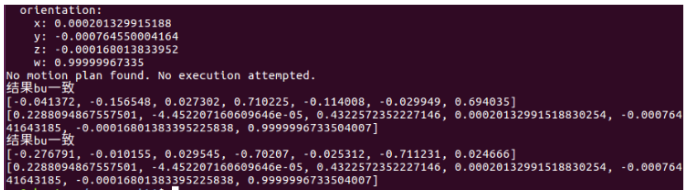



Fig. 9. The results of experiments.

angles being stored in the array. The robot arm moved to the final position, which can be expressed in the form of joints' angles. This can be expressed in other ways, such as pose. By giving the inputs, the robot arm returned to the initial position, and then moved to the proposed target position. After the end of the robot arm movement, the pose of the end effector of the robot arm was used as an evaluation indicator. However, because of DH parameters (the transformation relationship from the end of the robot arm to the base coordinate system) cannot be obtained, it was not feasible to verify the accuracy of *moveit_commander* and *moveit_group_interface* based on pose. This study also obtained the final joints' angles after the robot arm movement. Then, the results were compared with the set joints' angles one by one. Finally, the difference was calculated. The smaller the difference, the more accurate the output pose. Whether it was the current joints' angles or the pose of the end effector of the robot arm that were output, the overall meaning they represent was the same. Although they used different representation methods, they all represented the state of the robot arm after moving.

```
def compare(py_joint_values,c_joint_values,data):
    py_sum=0
    c_sum=0
    for i in range(6):
        a=decimal.Decimal(str(data[i]))
        a1=decimal.Decimal(str(py_joint_values[i]))
        a2=decimal.Decimal(str(c_joint_values[i]))
        py_sum=decimal.Decimal(str(py_sum))+a1-a
        c_sum=decimal.Decimal(str(c_sum))+a2-a
        if py_sum>c_sum:
            return True
        else:
            return False
```

Fig. 10. the *compare()* function

**Validation Process.** This study defined and initialized the container in the C++ program. After executing a test case, the function *getCurrentJointValues()* was called and the results were saved in the container. When all the test cases were executed, a file $f$ was generated. The python program also executed a test case, called the function *get_current_joint_values()* and saved the results in a list. Then, as shown in Fig. 10, the function *compare()* was defined to realize automatic calculations. The function used the *decimal module* in python to ensure accuracy. Finally, the function was called to perform calculations during comparing differences. By outputting the calculated results, most of the results were found to be closer to the outputs of *move_group_interface*.

MoveIt! is of great significance to the development and application of the robot. RROSFuzz found that the function of *moveit_commander* has vulnerability in achieving robot arms movement. This vulnerability threatens the security and trustworthiness of robot. If this vulnerability is not resolved, it is possible to cause destruction during manipulating the robotic arm, such as colliding with obstacles. Feedback has been given to the vendor. As an integrated development platform, ROS also contains many other important software packages. The proposed method can be applied to the whole robot system. In the specific application, the software packages are only needed to provide the CLI interface parameters, that is, the message type. Then, this method can be used to generate initial seeds and mutate seeds. Finally, differential fuzz testing can be performed. Hence, this study also answered the second question raised at the beginning of this section. Thus, inconsistencies in robot software packages can be found using this method.

## V. Discussion

This study proposes a differential fuzz testing framework and finds some issues. However, certain deficiencies remain. These are to be improved upon in future work.

- **More mutation strategies design.** To generate more abundant and random seeds, this study designs three mutation strategies; however, more strategies are required. Currently, this study just implements mutation for some data types. In the future, more mutation strategies, such as dictionary (that replaces/inserts tokens that automatically generated or user-provided files), havoc (makes considerable mutations to the original file) and splice (joins two files together to get a new file), will be developed.
- **Support Instrumentation for Coverage.** Program instrumentation involves the insertion of some probes into the program on the basis of ensuring the original logic integrity of the target program. These probes are essentially code segments of information collection and can be assignment statements and function to collect coverage information. Through executing the program with probes, the running characteristic data can be output into the program. Future work will aim to achieve instrumentation in the proposed method to obtain path coverage.
- **More software packages in robot.** The main idea is to continuously generate seeds for different versions of robot software packages, in order to find as many inconsistencies among results as possible, and eventually discover vulnerabilities. We first applied RROSFuzz to *moveit_commander* and *move_group_interface*. However, robots have a considerable open source community. There are many different versions implemented by different languages or different organizations. These packages that are also widely used may have some fatal vulnerability in implementation. In the future, we will find more software packages for testing.

## VI. RELATED WORK

**Trustworthiness of Robotics.** In 2017, the number of vulnerabilities related to the robot system disclosed was 1493, which is 1/10 of the number of vulnerabilities announced in the whole year. These vulnerabilities directly threaten the trustworthiness of the robot. In terms of robotics trustworthiness, it has probably gone through three periods. Early on, the research objects were the threat analysis of specific robot application scenarios, such as household robots and unmanned aerial vehicle. The focus was on remote communication and control trustworthiness. Then, researchers explored the robot framework. Mcclean et. al [19] verified the known trustworthiness risks of the robot system (such as no authentication and clear text communication) and the complexity of the Cyber-Physical System [28]. Sean Rivera also proposed ROS-defender [26], a comprehensive security architecture for ROS-based robotic systems to defend against a large number of attacks on ROS. Nowadays, researchers mainly investigate the formal verification of protocols or communication between nodes, establishment of runtime verification framework, and encryption of publish subscribe model module. Jia JuanJuan [20] proposed a formal verification method to verify the functional correctness of the communication in robot programs. By using a combination of model checking and theorem proving, she verified the XML-RPC protocol code in the robot system, including 205 functions of 63 program files. Debjyoti Bera used a formal verification method to study the weak termination of ROS systems [27].

**Fuzzing Technique.** As a software testing technology, the core idea of fuzzing is to automatically or semi-automatically input random data into the program and monitor program exceptions (crash and assertion failure) to detect potential program errors. By using fuzzing technology, the robustness and security of the application can be ensured. Godefroid et. al used the fuzzing tool SAGE [23] to find more than 20 unknown vulnerabilities in large Windows applications. Jiang Bo et. al proposed ContractFuzzer [24], a fuzzing tool applied to smart contracts. The tool performs fuzzing and runtime monitoring to detect vulnerabilities that occur during execution, which can generate fewer false positives. Aitel [25] successfully discovered multiple unknown vulnerabilities through the fuzzing tool SPIKE.

**Differential Testing.** Differential testing is a kind of random test, which is a component of mature technology for large-scale software and systems. In general, there are two cases of differential testing. The first analyzes the difference between executing different inputs on the same programs while the other analyzes the difference between executing the same inputs on multiple programs or variants. When using the second differential testing, two or more comparable systems must be available. These systems provide many detailed mechanically generated test cases. If the results are different, or one of the systems loops or crashes indefinitely, the bug-exposuring test is performed. For example, DLFuzz [21] continuously mutates the inputs to maximize the neuron coverage and prediction difference between original inputs and mutated inputs to guide the DL system to expose incorrect behaviors. DeepXplore [22] is a white box differential testing framework of system testing for real world DL.

## VII. CONCLUSION

This paper proposes RROSFuzz, the differential fuzz testing method, to efficiently discover vulnerabilities of robot software packages implemented by different programming languages and versions. In this method, a fuzzer that generates abundant seeds within a short time is designed. The fuzzer includes four generation strategies and three mutation strategies. Then, two evaluation metrics are introduced: early input filtering and reasonable difference, which improves the quality of seeds. Finally, taking guidance from difference information, seeds are preserved and selected. In addition, this method is applied to an experiment involving robotic arm control. By executing 15246 inputs on *moveit_commander* implemented by Python and *move_group_interface* implemented by C++, it is found that 60.78% showed differential performance. Analysis showed that *move_group_interface* is more accurate than *moveit_commander* on some function implementations. The function of *moveit_commander* has the joint'angle overflow vulnerability in realizing robot arm movement. Although the differential fuzz testing method is used to find the problems with robot software packages, some points can be optimized in the design of the method, such as the introduction of an instrument to capture the path coverage and the design of richer generation and mutation strategies for test cases.

## REFERENCES

[1] M. Basheer and A. Varol, "An Overview of Robot Operating System Forensics," 2019 1st International Informatics and Software Engineering Conference (UBMYK), 2019, pp. 1-4.

[2] Y. Saito, F. Sato, T. Azumi, S. Kato and N. Nishio, "ROSCH:Real-Time Scheduling Framework for ROS," 2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA), 2018, pp. 52-58.

[3] Huang J, Erdogan C, Zhang Y, et al. ROSRV: Runtime Verification for Robots[C]//International Conference on Runtime Verification. Springer, Cham, 2014: 247-254.

[4] S. Rivera, S. Lagraa and R. State, "ROSploit: Cybersecurity Tool for ROS," 2019 Third IEEE International Conference on Robotic Computing (IRC), 2019, pp. 415-416.

[5] Bai CC, Cheng WY, Guo HT. Brief Analysis of open source ROS robot Operating System [J].Science and Information Technology, 2019,000(036):2.

[6] Z. Ma, L. Zhu, P. Wang and Y. Zhao, "ROS-Based Multi-Robot System Simulator," 2019 Chinese Automation Congress (CAC), 2019, pp. 4228-4232.

[7] S. M. D. Almeida and L. V, "Design and Simulation of Micro Servo Robot in Robot Operating System," 2020 IEEE Recent Advances in Intelligent Computational Systems (RAICS), 2020, pp. 91-95.

[8] Liu RJ, Wang F, Zhang Q. ROS based trajectory planning of robotic arm [J].Navigation, positioning and timing.2016(6).

[9] Lian-Lian Sun, Yi-Na Shao, Mei-Xiang You, Cheng-Hua Li.ROS-mediated BNIP3-dependent Mitophagy Promotes Coelomocyte Survival in Apostichopus Japonicus in Response to Vibrio Splendidus infection[J].Zoological Research,2022,43(02):285-300.

[10] S. Kuzin and G. Sziebig, "SROS: Educational, Low-Cost Autonomous Mobile Robot Design Based on ROS," 2020 IEEE/SICE International Symposium on System Integration (SII), 2020, pp. 1052-1057.

[11] Wang Ying, Wang Bingqing, Guan Yong, Li Xiaojuan, Wang Rui. Differential Fuzzing Method for ROS [J]. Journal of Software, 2021,32(06):1867-1881.

[12] J. Wang, B. Chen, L. Wei and Y. Liu, "Skyfire: Data-Driven Seed Generation for Fuzzing," 2017 IEEE Symposium on Security and Privacy (SP), 2017, pp. 579-594.

[13] Dai LC, Zhang YR, Li FZ. A Brief Analysis of traditional Software Testing Methods [J].Science and Technology Wind, 2011(16):136-137.

[14] Ying Fu, Meng Ren, Fuchen Ma, Heyuan Shi, Xin Yang, Yu Jiang, Huizhong Li, and Xiang Shi. 2019. EVMFuzzer: Detect EVM Vulnerabilities via Fuzz Testing. In Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. Association for Computing Machinery, New York, NY, USA, 1110–1114.

[15] S. Pani, H. V. Nallagonda, S. Prakash, V. R, R. K. Medicherla and R. M. A, "Smart Contract Fuzzing for Enterprises: The Language Agnostic Way," 2022 14th International Conference on COMmunication Systems & NETworkS (COMSNETS), 2022, pp. 1-6.

[16] Yuanliang Chen, Yu Jiang, Jie Liang, Mingzhe Wang, and Xun Jiao. Enfuzz: From ensemble learning to ensemble fuzzing. arXiv preprint arXiv:1807.00182, 2018.

[17] Shuitao Gan, Chao Zhang, Xiaojun Qin, Xuwen Tu, Kang Li,Zhongyu Pei, and Zuoning Chen. Collafl: Path sensitive fuzzing.In CollAFL:Path Sensitive Fuzzing, page 0. IEEE.

[18] Google. honggfuzz. http://honggfuzz.com/, 2016.

[19] J. McClean, C. Stull, C. Farrar, D. Mascareas, A preliminary cyber-physical security assessment of the robot operating system (ros), in: Proc. SPIE, vol. 8741, 2013, pp. 874110–874110–8.

[20] Jia JJ, Shi ZP, Guan Yong, et al. Formal Validation of XML-RPC protocol implementation in ROS [J]. Minicomputer Systems, 2015, 36(12): 2629-2633.

[21] Guo J, Jiang Y, Zhao Y, Chen Q, Sun JG. DLFuzz: Differential Fuzzing Testing of Deep Learning Systems. In: Proc. of the 26th ACM SIGSOFT Intĺ Symp. on Foundations of Software Engineering (FSE). New York: Association for Computing Machinery, 2018. 4-9.

[22] Zhengxiong Luo, Feilong Zuo, Yuheng Shen, Xun Jiao, Wanli Chang, Yu Jiang: ICS Protocol Fuzzing: Coverage Guided Packet Crack and Generation. DAC 2020: 1-6.

[23] Godefroid P, Levin M Y, Molnar D. SAGE: whitebox fuzzing for security testing [J]. Communications of the ACM, 2012, 55(3): 40-44.

[24] B. Jiang, Y. Liu and W. K. Chan, "ContractFuzzer: Fuzzing Smart Contracts for Vulnerability Detection," 2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE), 2018, pp. 259-269.

[25] Aitel D. The advantages of block-based protocol analysis for security testing [J]. Immunity Inc., February, 2002, 105: 106.

[26] S. Rivera, S. Lagraa, C. Nita-Rotaru, S. Becker and R. State, "ROS-Defender: SDN-Based Security Policy Enforcement for Robotic Applications," 2019 IEEE Security and Privacy Workshops (SPW), 2019, pp. 114-119.

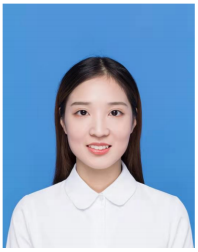
[27] Bera D, van Hee K M, van der Werf J M. Designing weakly terminating ROS systems[C]//International Conference on Application and Theory of Petri Nets and Concurrency. Springer, Berlin, Heidelberg, 2012: 328-347.

[28] Mansour Alali, Ahmad Almogren, Mohammad Mehedi Hassan, Iehab Al Rassan, M. Bhuiyan, Improving risk assessment model of cyber security using fuzzy logic inference system. Comput. Secur. 74: 323-339 (2018).

[29] Haoyue Wang, Yangyang Zhang, Jianxin Li, Richong Zhang, Zakirul Alam Bhuiyan, RPS-TSM: A Robot Perception System Based on Temporal Semantic Map. Proc. of SpaCCS Workshops 2017: 524-533.

[30] Hao Sun, Yuheng Shen, Cong Wang, Jianzhong Liu, Yu Jiang, Ting Chen, Aiguo Cui:HEALER: Relation Learning Guided Kernel Fuzzing. SOSP 2021: 344-358.

[31] Chijin Zhou, Mingzhe Wang, Jie Liang, Zhe Liu, Yu Jiang: Zeror: Speed Up Fuzzing with Coverage-sensitive Tracing and Scheduling. ASE 2020: 858-870.

**BingQing Wang** was born in 1997. She received her M.S. degree from the College of information engineering of Capital Normal University, Beijing, China in 2022. She participated in the National Natural Science Foundation of China under Grants 61877040. Her main research direction was robot software security. She and her team proposed using the fuzzing and differential fuzzing methods to find vulnerabilities in ROS (Robot Operating System).She has participated in the preparation of the paper. What's more, she has published a paper in the Journal of Software and also got a patent. She applied for a CVE number.

**Rui Wang** is currently a Professor of Capital Normal University, China. She received the B.S. degree in computer science from Xi'an Jiaotong University in 2004 and received the Ph.D. degrees in Computer Science and Technology from Tsinghua University in 2012. Her research interests include the safety and security of robot system, formal verification and their applications in cyber-physical systems. Dr. Wang has published more than 50 papers in international conferences and Journals. She presided over 2 National Science Foundation of China and other projects. Dr. Wang received the Best Paper Awards from CPSCom-2019, ICII 2019.

**Houbing Song** is a Tenured Associate Professor of AI and the Director of the Security and Optimization for Networked Globe Laboratory (SONG Lab, www.SONGLab.us), University of Maryland, Baltimore County, Baltimore, MD 21250 USA. He received the Ph.D. degree in electrical engineering from the University of Virginia, Charlottesville, VA, in August 2012. His research interests include cyber-physical systems/internet of things, cybersecurity and privacy, and AI/machine learning/big data analytics. His research has been featured by popular news media outlets, including IEEE GlobalSpec's Engineering360. Dr. Song is a senior member of ACM, and an ACM Distinguished Speaker. Dr. Song is a Highly Cited Researcher identified by Clarivate (2021) and a Top 1000 Computer Scientist identified by Research.com. Dr. Song was a recipient of the Best Paper Awards from CPSCom-2019, ICII 2019, ICNS 2019, CBDCom 2020, WASA 2020, DASC 2021, GLOBECOM 2021 and IEEE INFOCOM 2022.