How to Secure Autonomous Mobile Robots? An Approach with Fuzzing, Detection and Mitigation

Chundong Wang[a,1], Yee Ching Tok[b], Rohini Poolat[c,1], Sudipta Chattopadhyay[b,*] and Mohan Rajesh Elara[b]

[a]School of Information Science and Technology, ShanghaiTech University, China
[b]Singapore University of Technology and Design, Singapore
[c]National University of Singapore, Singapore

ARTICLE INFO

Keywords:
Fuzzing, Autonomous Mobile Robot, Attack Detection and Mitigation, Embedded Systems

ABSTRACT

Autonomous mobile robots share social spaces with humans, usually working together for domestic or professional tasks. Cyber security breaches in such robots undermine the trust between humans and robots. In this paper, we investigate how to apprehend and inflict security threats at the design and implementation stage of an autonomous mobile robot. To this end, we leverage the idea of directed fuzzing and design RoboFuzz that systematically tests an autonomous mobile robot in line with the robot's states and the surrounding environment. The methodology of RoboFuzz is to study critical environmental parameters affecting the robot's state transitions and subject the robot control program with rational but harmful sensor values so as to compromise the robot. Furthermore, we develop detection and mitigation algorithms to counteract the impact of RoboFuzz. The difficulties mainly lie in the trade-off among limited computation resources, timely detection and the retention of work efficiency in mitigation. In particular, we propose detection and mitigation methods that take advantage of historical records of obstacles to detect inconsistent obstacle appearances regarding untrustworthy sensor values and navigate the movable robot to continue moving so as to carry on a planned task. By doing so, we manage to maintain a low cost for detection and mitigation but also retain the robot's work efficacy. We have prototyped the bundle of RoboFuzz, detection and mitigation algorithms in a real-world movable robot. Experimental results confirm that RoboFuzz makes a success rate of up to 93.3% in imposing concrete threats to the robot while the overall loss of work efficacy is merely 4.1% at the mitigation mode.

1. Introduction

Autonomous mobile robots are widely used to relieve people from dirty, monotonous, and dull tasks, and also reduce economic costs [16, 13, 19, 14, 40]. For example, cleaning robots gain wide popularity in tidying private apartments and public places. Some airports have deployed cleaning robots to replace human cleaners save housekeeping manpowers [26, 28]. Since autonomous mobile robots are sharing social spaces with humans at home, in the offices and even in critical infrastructures like airports and banks, their security and safety are of paramount importance, especially concerning they are autonomous without human attendance.

Robotics are generally categorized as cyber-physical systems (CPS). A robot typically has 1) a digital controller, e.g., Raspberry Pi, to manage the system, 2) physical components, such as sensors and actuators, to sense the surrounding environment (e.g., distance) and to manipulate physical entities (e.g., wheels and robotic arms), respectively, and 3) cyber components that connect the robot to networks (e.g., for remote control via smartphones). The robot control program is critical to the security and safety of a robot as it decides how to manoeuvre actuators of the robot on reading sensor values. A number of studies have revealed that it is possible to compromise a CPS through fraudulent sensor values, while mitigating such attacks usually requires the involvement of a cloud server for remote computation or attestation [30, 38, 9, 15, 10]. However, such methods are not applicable to autonomous mobile robot. Firstly, the computational resource and battery capacity are relatively limited for an economical autonomous mobile robot compared to large CPS, say, a power grid. Secondly, the vast popularity of autonomous mobile robots imposes overwhelming difficulty on security patches or remote attestation from time to time. Thirdly, many autonomous mobile robots move themselves to complete planned tasks, which differentiates them from stationary CPS like power grid or 3D printer and necessitates a mitigation method that replenishes the movement of autonomous mobile robot. As a result, it is preferable and practical to secure an autonomous mobile robot as early as at its design and implementation stage.

In this paper, we aim to enhance the security and safety of autonomous mobile robots, particularly ones that are movable because they would be physically detrimental to human beings once compromised. On one hand, we attempt to systematically scrutinize the security threats to autonomous mobile robot through investigating the values of critical sensors, since these sensor values, as inputs to the robot control program, determine the next states of robot. On the other

*Corresponding author: S. Chattopadhyay.
cd_wang@outlook.com (C. Wang); yeeching_tok@mymail.sutd.edu.sg (Y.C. Tok); rohini_poolat@yahoo.com (R. Poolat); sudipta_chattopadhyay@sutd.edu.sg (S. Chattopadhyay); rajeshelara@sutd.edu.sg (M.R. Elara)
ORCID(s): 0000-0001-9069-2650 (C. Wang); 0000-0002-4843-5391 (S. Chattopadhyay); 0000-0001-6504-1530 (M.R. Elara)

[1]This work was partially done when Chundong Wang and Rohini Poolat worked in Singapore University of Technology and Design.

hand, with regard to the uncovered threats, we develop an efficient algorithm to mitigate their impacts while retaining most of the robot's work efficacy.

Without loss of generality, we illustrate with an autonomous mobile robot cruising by means of an ultrasonic distance sensor to avoid obstacles. Once the distance sensor indicates a close obstacle ahead, the robot control program ought to direct the robot to turn left or right. Otherwise, the robot would crash into the obstacle. As a result, altering the sensor value to be malicious for the robot control program would inflict serious threats to the robot. We hence employ the idea of software *fuzzing* to test the robot control program. In software testing, fuzzing is used to identify security vulnerabilities or bugs in a program by subjecting the program to various kinds of input, and the program may crash or yield absurd outputs [37, 20, 7]. By fuzzing the robot control program, we aim to discover as many flaws as possible in the robot control program and secure the robot.

We use a state-of-the-art fuzzing tool, i.e., Radamsa [20], to generate and feed a series of distance sensor values to the robot control program replacing real-world distances when the robot is moving. The robot trembles because the fuzzed sensor values, as intended to maximally uncover bugs of a program, fluctuate significantly. We can easily patch the robot control program with a filter to rule out such volatile sensor values, since distance sensor values fall in a reasonable range in line with the environment and the state of robot. For instance, a robot moving towards a wall continuously receive decreasing distance sensor values.

The analysis over arbitrary and irregular sensor values, however, implies us to test the control program with rational and regular sensor values in a *directed fuzzing* way. The distance is a critical *environmental parameter* for a moving mobile robot as it triggers state transitions for the robot. For example, a moving robot receiving decreasing distance sensor values would turn left or right if the distance gradually drops below a threshold. Given a dynamic obstacle, say, an automatic sliding door, it may move out of the robot's path and the distance sensor value suddenly increases to a large value, after which the robot keeps moving forward. Whereas the robot control program is unable to ascertain if an obstacle is truly dynamic or static solely depending on distance sensor values, because the scenarios where the distance either monotonically decreases or abruptly increases are both possible in the real world. Assume that the robot is moving towards a hard wall, but we instead replace the distance sensor values with ones that resemble the getaway of a dynamic obstacle. The robot shall collide with the wall.

The aforementioned example addresses the essence of our directed fuzzing strategy, namely RoboFuzz. In a nutshell, by investigating the state transitions and environmental parameters an autonomous mobile robot, RoboFuzz generates rational but harmful sensor values so as to mislead the robot for concrete threats.

Adversaries can implement RoboFuzz with realistic attack models, like suspending or fabricating sensor values, to compromise an autonomous mobile robot. As developers, we move forward and defend against the attacks entailed via RoboFuzz by detecting and mitigating them. There are two concerns in doing so. First, the detection and mitigation should not be heavyweight regarding the limited computational resources of an autonomous mobile robot. Secondly, once an attack is detected, the mitigation cannot barely shut down the robot but maximally retain the robot's work efficacy. Nevertheless, as mentioned, the robot control program alone cannot rule out rational but anomalous sensor values. We need further information that can be used to counteract RoboFuzz. We note that, for a mobile robot, such as a cleaning robot, it is supposed to repeatedly cruise in a certain and steady place. Consequently, the robot is able to make and maintain a historical record of obstacles for the place [39]. Such a historical record is an exploitable resource for us to detect the attacks initiated through RoboFuzz. Concerning that RoboFuzz that fuzzs distance sensor values to emulate fake obstacles, a historical record helps the robot control program to cross-check if the obstacle is really dynamic or not to protect the robot.

The historical record is also effectual for us to mitigate the impact caused by RoboFuzz. A movable autonomous robot must keep moving to complete the task planned for it even in the presence of an untrustworthy distance sensor. However, it can utilize the historical record to circumvent obstacles and navigate the robot. Doing so not only retains the robot's work efficacy, but also gains high cost efficiency in mitigating for an economical robot.

The main ideas of this paper are summarized as follows.

- We propose RoboFuzz which tests an autonomous mobile robot by fuzzing rational but harmful sensor values so as to mislead the robot's control program;
- To defend against the attacks initiated by RoboFuzz, we develop detection and mitigation methods which leverage historical records to maximally protect the robot and efficiently accomplish planned tasks.

RoboFuzz and strategies of detection and mitigation to contract RoboFuzz form a self-contained and systematic scheme that help to develop a secure autonomous mobile robot. We have prototyped them with a real-world movable robot, i.e., iRobot Create 2 with an HC-SR04 distance sensor. Experimental results confirm that RoboFuzz attains up to 93.3% success rate in imposing threats onto a moving iRobot Create 2. Our detection and mitigation methods also efficiently detect attacks at a very high rate and make the robot being under attack accomplish scheduled tasks with an insignificant loss of work efficacy, i.e., 4.1% overall.

The rest of this paper is organized as follows. In Section 2, we present the background of autonomous mobile robot. We conduct a motivational study to incur concrete threats to a mobile robot in Section 3. In Section 4, we detail the design of RoboFuzz. In Section 5 and 6, respectively, we show our algorithms for detecting and mitigating threats incurred by RoboFuzz. We present experiments with a prototype built with iRobot Create 2 in Section 7. We brief related works in Section 8. We discuss threats to validity in Section 9 and conclude the paper in Section 10.

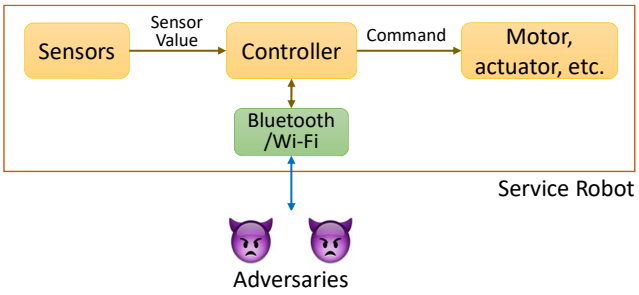
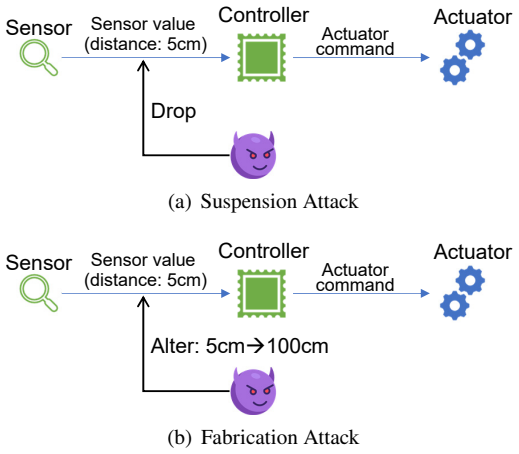


Figure 1: A Illustration of Mobile Robot and Adversaries



(a) Suspension Attack

(b) Fabrication Attack

Figure 2: An Illustration of Typical Attacks Models

2. Background

In contrast to robots used by manufacturers or specialists, mobile robots are close to people and easy to operate, providing a variety of mobiles, such as housekeeping and entertainment [19, 14]. According to ISO standard [1], a mobile robot is a class of robots that "perform useful tasks for humans or equipment excluding industrial automation applications". An autonomous mobile robot, such as the typical cleaning robot, has following components: 1) a digital controller such as Raspberry Pi or Arduino Mega where a control program runs, 2) numerous sensors to sense surroundings, 3) wheels to move the robot around, and 4) cyber accessories for network connection. Autonomous mobile robot puts reliance on the control program to decide the next move of it in accordance with sensor values obtained from time to time. Regarding sensors installed in a mobile robot, they quantitatively measure and report the environmental parameters the robot is encountering. For example, a distance sensor tells whether the robot is too close to any obstacle. Sensors may work in different modes. A sensor working in the proactive mode alerts the robot control program periodically or in case of emergency while a sensor working in the passive mode pends the robot control program to ask for sensor value.

Robots fall into the broad category of CPS. One outstanding characteristic of CPS is the vast heterogeneity of building blocks in different CPS for different usages [9, 21, 32, 4]. An autonomous mobile robot is significantly distinct from typical CPS such as power grids, handheld smartphones or 3D printer [43, 41, 42]. First, an autonomous mobile robot is generally a simple system with an economical micro-controller and a few hardware components including sensors, actuators, and network modules. Figure 1 shows a classic architecture of autonomous mobile robot. Secondly, autonomous mobile robots gain worldwide popularity in our daily life. For example, iRobot has sold more than 20 million cleaning robots since its foundation [22] while the sales volume of Xiaomi Mi robots has reached one million in 18 months since its release date [6]. Assuming that a critical flaw of cleaning robot is uncovered, a large population of users would be affected. Thirdly, unlike CPS that undergo frequent maintenance mobiles in subways, hospitals, and power stations [8, 42, 46], many mobile robots are unlikely to be promptly upgraded with security patches. To update a large number of robots or do remote attestation for each of them is also challenging and costly for a manufacturer. Finally, a mobile robot is not stationary like 3D printer or handheld smartphone. Once compromised, it might be manipluated to incur physical damages to surrounding people.

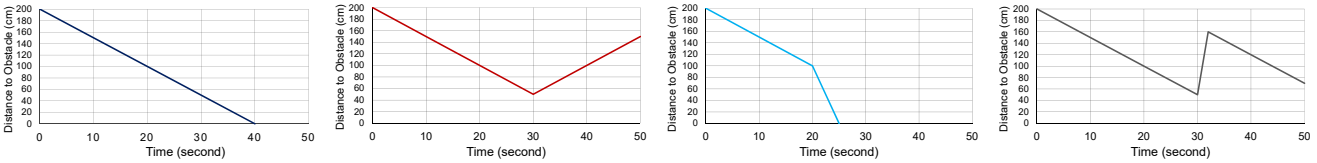
In summary, the demand to study security-related issues for autonomous mobile robots is actual and critical. Recently, researchers have looked into the cyber security of robots [35, 25, 34, 24]. The security issues should be considered in the design phase of a mobile robot due to the ever-increasing popularity of mobile robots and the ever-growing strengths of adversaries. In this paper, we first proceed at the standpoint of developers to explore how to reveal as many flaws as possible for an autonomous mobile robot. Then we continue to contemplate cost-efficient methods for detection and mitigation while retaining the robot's work efficacy.

3. Motivation and Overview

3.1. Security Treats for Autonomous Mobile Robot

Recently, Bonaci et al. [8] investigated the vulnerabilities of teleoperated surgical robots and Quarta et al. [33] performed an empirical analysis on the security issues of industrial robots. These works draw the attention of research community to the security of robots found in factories, operating rooms, and so on. Nevertheless, such awareness should be extended to the security of autonomous mobile robots. In practice, Giese and Wegemer have managed to hack a Xiaomi Mi cleaning robot [17]. Their success should not only alert robot manufacturers, but also the users of such robots.

As mentioned, the robot control program maneuvers an autonomous mobile robot by reading sensor values. On the other hand, the network interface of robot widely provides adversaries an exploitable attack surface, because many users still use default or weak passwords today, especially for domestic robots. As a result, adversaries are bound to manip-



(a) A static obstacle (e.g., a wall)

(b) A dynamic obstacle suddenly moves in the same direction as robot

(c) A dynamic obstacle suddenly moves towards the robot

(d) A dynamic obstacle suddenly moves out of the path of robot

Figure 3: An Illustration of State Transitions for a Cleaning Robot upon Obstacles

ulate the robot's sensor values through unauthorized remote access so as to misguide the robot control program. In the meantime, there are multiple attack models for adversaries to follow. In this paper we consider two harmful and representative ones that have been manifested recently [12, 34], i.e., *suspension attack* and *fabrication attack*.

Suspension Attack As shown in Figure 2(a), attackers suspend sensors from sending out information. A sensor at the passive mode, once suspended, would leave a null response to the robot control program, which misleads the control program to conclude that the sensor malfunctions. If the sensor works at the proactive mode, the impact of suspension attack should be even worse. Consider a distance sensor that alerts the robot control program only in case of a very close obstacle. After a successful suspension attack, the control program would no longer receive any alert. As a consequence, the robot might crash into an obstacle.

Fabrication Attack With a fabrication attack, adversaries fabricate harmful sensor values and feed them to the robot control program. As shown by Figure 2(b), when the robot is in motion, the control program asks for a distance sensor value to decide whether an obstacle is nearby to the robot. Noticing such a request, adversaries replace the normal sensor value with an anomalous one. The control program would accordingly make a wrong decision and put the robot into a concrete danger.

3.2. A Motivational Study

Without loss of generality, we choose a programmable cleaning robot, i.e., iRobot Create 2 [23], for case study. We run a control program in a Raspberry Pi 3 to maneuver the robot and install an ultrasonic distance sensor (HC-SR04) to enable the robot to avoid obstacles. As developers, we use the WiFi interface shown in Figure 1 as the port to communicate with the robot controller for monitoring and debugging.

The distance to obstacles is a crucial environmental parameter for a cleaning robot. The control program depends on the distance sensor values received at runtime to decide whether the robot moves forward or turns. As these sensor values are the input to the control program, the first attempt we did is leverage the idea of software fuzzing, which generates various kinds of input values to a program so as to inflict disorder or even crash to the program. We used Radamsa [20], a state-of-the-art fuzzing tool, to make a series of 1,006 values within the distance range supported by HC-SR04 (2cm to 400cm). A segment of the values fuzzed by Radamsa are as follows:

$$\{..., 26, 128, 5, 16, 3, 241, 107, 6, 255, 45, 240, 4, 18, ...\}$$

We supposed that such distance sensor values, when fed to the control program, should have compromised the robot. However, after we delivered them to satisfy the requests raised by the control program, the control program would refuse them as anomalies. We then analyzed the failure of fuzzing control program in isolation. The reason is mainly due to the concept of software fuzzing and the mechanism of mobile robot. Fuzzing a program is used to reveal bugs and security vulnerabilities of a program. Hence the fuzzed inputs, as shown in the above segment, fluctuate significantly so as to traverse different code paths and generate as many corner cases as possible. Therefore, fuzzing the control program is a good approach to test the program alone but ignores the mechanism of mobile robot. As mentioned, the control program transits a cleaning robot among states depending on sensor values it receives. A cleaning robot moving towards a wall will receive decreasing sensor values and in the end it should turn or stop, so the robot transits from a state of moving forward to the next state of turning or stopping. Given sensor values fuzzed by Radamsa that change strikingly and continually, they are easy to be distinguished since they obviously deviate from what the control program expects in an ordinary environment.

We thoroughly investigate the states of cleaning robot and environmental parameters that drive the robot to do state transitions. We find that, for a cleaning robot moving at a stable velocity (e.g., 5cm/s), its state transitions are affected by the distance to obstacles in four scenarios, as ideally illustrated by Figure 3. In the four diagrams of Figure 3, the Y axis is the distance to obstacles measured over time (cf. X axis). In Figure 3(a), the robot is moving towards a fixed obstacle (e.g., a wall), so the distance gradually decreases to zero. The remaining three diagrams show a robot meets three types of dynamic obstacles. In Figure 3(b), at a time, a dynamic obstacle (e.g., a pet) suddenly moves away at a higher velocity and in the same direction as the robot, so the distance stops dropping but increases abruptly. In Figure 3(c), after 20s, the dynamic obstacle moves towards the robot, which makes their distance decrease faster than before. Figure 3(d) represents another kind of obstacle that has been on the path of the robot but, at one moment, moves out of the robot's path, like the prompt open of an automatic sliding door. The distance thus migrates to another decreas-

ing linear curve.

The four cases capture normal scenarios where the distance to obstacles, as a critical environmental parameter, affects a cleaning robot in transiting its states at runtime, say, to keep moving forward or turn/stop. The four curves in Figure 3 help the control program rule out anomalous sensor values like ones generated by Radamsa. More important, they inspire us with the opportunities to *mislead* the control program. Note that the control program relies on the distance sensor values to learn the distance to obstacles. Consider a cleaning robot is steadily moving to a wall. We are monitoring the robot's state and the real distance by reading sensor values. When the robot is close to the wall, we fuzz increasing distance sensor values to emulate that the obstacle is dynamic and moving away. If the control program asks for distance sensor values, we will feed fuzzed values to it. From the viewpoint of control program, such increasing sensor values are absolutely rational regarding Figure 3(b). So the robot is misled from the curve in Figure 3(a) to the one in Figure 3(b). In the end, the robot shall crash into the wall.

We note that sensing modalities other than distance sensors can also be used to detect various environmental parameters. For example, radar and lidar are used to detect the presence, distance, and velocity of surrounding obstacles. Given a radar that detects a moving obstacle at a constant velocity, we can intentionally fuzz the velocity values of the obstacles to pretend that the obstacle stays static. Concretely, the autonomous mobile robot would react to a seemingly static obstacle. This is also likely to impair the robot.

3.3. Overview

Figure 4 illustrates an overview of the three schemes proposed in this paper. The preceding motivating example indicates the essence of RoboFuzz (at the leftmost corner of Figure 4). By closely monitoring the state of a robot and its environmental parameters (① in Figure 4), RoboFuzz starts to deceive the robot's control program at an appropriate occasion with faked but rational sensor values (② in Figure 4) so as to inflict concrete harm to the movable robot.

The sensor values fuzzed by RoboFuzz should impose concrete security breaches to autonomous mobile robots. Because our intention is to enhance the security and safety of autonomous mobile robots at their design and implementation stage, we need to defend against RoboFuzz. We subsequently develop detection and mitigation schemes, i.e., Shade and Remit at the top of Figure 4, to counteract RoboFuzz. The detection and mitigation reside within the robot control program. As a result, they can learn the robot's states and historical records of the environment in which the robot is working. Using such information (③ in Figure 4), the detection module would report whether the sensor values are compromised or not (④ in Figure 4). Upon an alert of detected attacks, the robot control program cannot rely on the sensor values to proceed moving. Instead, the mitigation module would be activated to leverage historical records (⑤ in Figure 4) of obstacles in the environment so as to navigate the robot to complete planned tasks (⑥ in Figure 4).

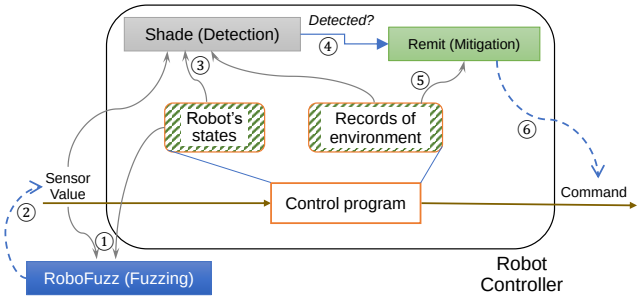


Figure 4: An Overview of RoboFuzz, Detection & Mitigation

4. RoboFuzz for Autonomous Mobile Robot

In this section, we first model the state transitions of autonomous mobile robot and explain the feasibility and procedure of RoboFuzz through state composition (cf. Section 4.1). Then we model RoboFuzz, a systematic scheme that effectively damages autonomous mobile robot by fuzzing sensor values (cf. Section 4.2).

4.1. State Compositions of RoboFuzz

An autonomous mobile robot can be modeled as a finite state machine (FSM). The upper-left part of Figure 5 captures a segment of a simplified FSM for a cleaning robot. This segment applies to all four scenarios mentioned in Section 3 as it shows how the cleaning robot proceeds on meeting an obstacle that can be either fixed or movable. Meanwhile, as developers of the robot, we maintain the FSM (cf. Figure 5) and continuously observe the environmental parameters from time to time. The outcome of RoboFuzz hence can be viewed as a *composition* of two FSMs (① in Figure 5). In particular, once RoboFuzz notices a significant change of an environmental parameter that is to incur a state transition, like the distance to an obstacle decreasing to be very small, RoboFuzz will fabricate a series of rational sensor values and feed them to the robot control program to make an illusion (② in Figure 5), e.g., the obstacle moving away. By doing so, RoboFuzz misleads the robot into the FSM intended by RoboFuzz, which, however, the robot control program will not be aware of. Eventually the robot is supposed to be wrecked because of hitting the obstacle (③ in Figure 5).

We note that the main purpose of RoboFuzz is to unveil the vulnerability of robot control program and in turn compromise the robot through fuzzing sensor values. RoboFuzz is an automated procedure. It keeps monitoring the states of robot and environmental parameters. At a proper occasion, it activates the state composition with faked but rational sensor values to deceive the robot control program.

4.2. Fuzzing Autonomous Mobile Robots with RoboFuzz

How RoboFuzz compromises an autonomous mobile robot is modeled as follows. Because RoboFuzz works in

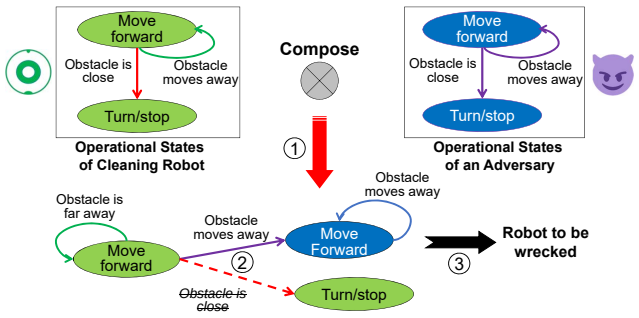


Figure 5: An Illustration of State Composition of RoboFuzz

line with the state of an autonomous mobile robot and the environment, it falls into the category of *directed fuzzing*. Directed fuzzing starts off with a given target, such as damaging the robot or reducing the robot's work efficacy. Let these targets form a set,

$$\boldsymbol{T} = \{\tau_0, \tau_1, ..., \tau_i, ..., \tau_{n-1}\},$$

where τ_i $(0 \leq i < n)$ is one independent target, e.g., to damage the robot, and the value of n depends on the intention of adversaries. Before fuzzing, we, at the standpoint of adversaries, assume that the physical states of the robot monitored at runtime form a set, i.e.,

$$\boldsymbol{Z} = \{\zeta_0, \zeta_1, ..., \zeta_k, ..., \zeta_{p-1}\}.$$

We also assume a thorough understanding of the robot, particularly all the components embodied in the robot, say,

$$\boldsymbol{C} = \{s_0, s_1, ..., s_{l-1}, a_0, a_1, ..., a_{m-1}\},$$

in which there exist all l sensors and m actuators. RoboFuzz relies on the l sensors to spot the environment. In addition, RoboFuzz can also utilize actuators for a target although we use sensors for illustration in preceding sections, e.g., by driving wheels faster than usual towards an obstacle.

To attain a specific target, RoboFuzz must formulate 1) what states and environmental parameters should be monitored, 2) which sensors and actuators in $\boldsymbol{C}$ are useful for the target, and 3) when (i.e., the aforementioned 'appropriate' occasion) and how to alter sensor values or actuator commands for a detrimental state transition (e.g., transiting between different curves shown in Figure 3).

Hence, for a target τ_i, we need 1) a subset of $\boldsymbol{Z}$, i.e., $\boldsymbol{Z}_i$, which subsumes states that are useful for τ_i, 2) a subset of $\boldsymbol{C}$, say, $\boldsymbol{C}_i$, which is a list of essential sensors and actuators for τ_i, and 3) a set $\boldsymbol{V}_i$ in which each element includes a tuple for the j-th $(0 \leq j < |\boldsymbol{C}_i|)$ sensor or actuator in $\boldsymbol{C}_i$, i.e.,

$$\left\langle v_j^{(i)}, \gamma_j^{(i)}, f_j^{(i)} \right\rangle.$$

$v_j^{(i)}$ is a normal sensor value/actuator command while $\gamma_j^{(i)}$ is a fuzzed sensor value/actuator command. For instance, $v_j^{(i)}$ and $\gamma_j^{(i)}$ fall into the range of $[2, 400]$ (cm) for an HC-SR04 ultrasonic distance sensor. Note that both of them can also be a special value $\varnothing$ which stands for the non-existence of sensor value/actuator command. $\varnothing$ is useful when there ought to be no sensor value/actuator command or adversaries intentionally drop a sensor value/actuator command. The third element in the tuple, i.e., $f_j^{(i)}$, is a function,

$$f_j^{(i)} : \boldsymbol{Z}_i \times Dom\left(v_j^{(i)}\right) \rightarrow Dom\left(\gamma_j^{(i)}\right), \tag{1}$$

where $Dom(x)$ means the domain of x. Assuming that the robot is at a state $\zeta \in \boldsymbol{Z}_i$ (e.g., moving forward) and one or multiple environmental parameters are to change, like when the distance to obstacles, i.e., $v_j^{(i)}$, is going to decrease to be 6cm, $f_j^{(i)}$ alters $v_j^{(i)}$ to $\gamma_j^{(i)}$, say, from 6cm to 60cm (i.e., making a fixed obstacle 'move'). $f_j^{(i)}$ hence converts a normal sensor value/actuator command or $\varnothing$ to be a still rational but harmful value. Also it may replace a sensor value/actuator command with $\varnothing$ to hinder the robot control process from interacting with corresponding sensors/actuators. $f_j^{(i)}$ keeps affecting the robot control process until the achievement of target τ_i.

Finally, we capture a successful fuzzing procedure for target τ_i as:

$$\boldsymbol{G}_i \vDash \tau_i, \tag{2}$$

in which $\boldsymbol{G}_i$ is defined as

$$\boldsymbol{G}_i = \bigcup_{\zeta \in \boldsymbol{Z}_i} \left\{ \langle v^{(i)}, \gamma^{(i)}, f^{(i)} \rangle \mid \langle v^{(i)}, \gamma^{(i)}, f^{(i)} \rangle \in \boldsymbol{V}_i \right. \\ \left. \wedge\ Dom\left(f^{(i)}\right) = \zeta \times Dom\left(v^{(i)}\right) \right\}. \tag{3}$$

$\boldsymbol{G}_i$ means that, for every state $\zeta \in \boldsymbol{Z}_i$, RoboFuzz discovers all tuples related to ζ and calls the respective function $f^{(i)}$ to fabricate and/or drop one or multiple sensor values and/or actuator commands for the success of τ_i.

Implementing $\boldsymbol{G}_i$ The implementation of $\boldsymbol{G}_i$ is based on the rationale discussed in the preceding section (cf. Section 3). Algorithm 1 shows the implementation of $\boldsymbol{G}_i$ for a distance sensor while the target τ_i is either to crash the robot or reduce the robot's work efficacy. RoboFuzz continuously tracks running states of an autonomous mobile robot and waits for a proper time to fuzz the robot ((Lines 1 to 2 in Algorithm 1)). For instance, when the sensor value $v_j^{(i)}$ is gradually decreasing (Line 4), RoboFuzz realizes that there is a fixed obstacle ahead. Therefore, to crash the robot (τ_i at Line 3), the $\boldsymbol{G}_i$ function would generate sensor values, i.e., $\gamma_j^{(i)}$, which continue increasing to resemble a leaving obstacle (Line 7). By doing so, RoboFuzz aims to use faked sensor values to change the scenario shown by Figure 3(a) to the one in Figure 3(b). Algorithms 1 also shows how to convert scenarios for other types of obstacles (Lines 8 to 10, Lines 12 to 15, and Lines 16 to 18).

5. Attack Detection with Shade

RoboFuzz provides a way to initiate successful attacks to an autonomous mobile robot. In this section we will consider how to efficiently detect attacks. We first investigate possible attack models which, once integrated with RoboFuzz, would carry the robot into misbehaving states. Accordingly we look into three classic detection methods, and develop a hybrid one with wider coverage, higher accuracy and less overhead.

Algorithm 1 The G_i for a Distance Sensor

Input: The target τ_i for fuzzing;
Ensure: $\gamma_j^{(i)}$ for the distance sensor s_i;

while (the robot is working) **do**
  Get the current state ζ_k, and sensor value $v_j^{(i)}$;
  if (τ_i is to crash the robot) **then**
    if ($v_j^{(i)}$ is decreasing) **then** //Approaching an obstacle
      if ($v_j^{(i)}$ gradually decreasing) **then**
        //Figure 3(a) ⇒ Figure 3(b)
        When $v_j^{(i)}$ is small enough, e.g., $v_j^{(i)} < 20$cm, $v_j^{(i)} \xrightarrow{f^{(i)}} \gamma_j^{(i)}$ ($\gamma_j^{(i)}$ continues to increase);
      else if ($v_j^{(i)}$ decreasing more sharply) **then**
        //Figure 3(c) ⇒ Figure 3(b)
        $v_j^{(i)} \xrightarrow{f^{(i)}} \gamma_j^{(i)}$ ($\gamma_j^{(i)}$ no longer decreases but gradually increases);
      end if
    else if (τ_i is to reduce the robot's work efficacy) **then**
      if ($v_j^{(i)}$ increases and continue increasing) **then**
        //Figure 3(b) ⇒ Figure 3(a)
        $v_j^{(i)} \xrightarrow{f^{(i)}} \gamma_j^{(i)}$ ($\gamma_j^{(i)}$ continues to decrease);
      else if ($v_j^{(i)}$ suddenly increases but then drops) **then**
        //Figure 3(d) ⇒ Figure 3(a)
        When $v_j^{(i)}$ suddenly increase, $v_j^{(i)} \xrightarrow{f^{(i)}} \gamma_j^{(i)}$ ($\gamma_j^{(i)}$ continues to decrease);
      end if
    end if
  end if
end while
Return $\gamma_j^{(i)}$ to replace $v_j^{(i)}$ for τ_i;

5.1. Classic Detection Methods

Fingerprinting Hardware devices have their unique physical characteristics [12], i.e., *fingerprints*, such as the sensor latency (i.e., response time). Assuming that attackers fabricate and send fake sensor values via Wi-Fi, the sensor latency observed by the control process should be extraordinary as network latencies are generally one or two orders of magnitude longer than typical sensor latencies. Take the ultrasonic distance sensor (HC-SR04) for example. Its sensor latency mostly falls in a range of 2ms to 12ms. By contrast, the network latency under TCP and UDP protocols varies between 200ms and 250ms. If the robot control process has learned a sensor's normal latency, it is able to detect an attack that delivers sensor values through the network.

Fingerprinting is advantageous with its simplicity and low overhead. But it has limited usages. Given a sensor working in a periodical or proactive mode, the robot control process cannot measure its sensor latency for validation.

Cross-reference Validation (CRV) CRV leverages information from two or more sources to cross-check for verification. The challenge in using CRV for an autonomous mobile robot is that every sensor might be compromised and using different sensors for cross-checking is unreliable. Also, not many sensors are installed in a small mobile robot for similar purposes. CRV must use some information that attackers are unaware of. Let us still use the distance sensor for example. A cleaning robot can make historical records of the positions of stationary obstacles in a normal working routine. In fact, some iRobot Roomba robots draw a map of the space they have cleaned [39]. Such historical records can be secured and used as the norm to validate distance sensor values. If the distance sensor gives a value that badly violates historical records, CRV can indicate the occurrence of an attack.

Compared to fingerprinting, CRV can detect attacks that compromise sensors working in the proactive mode since CRV cross-checks by exploiting extra historical records. However, CRV requires continually tracking the robot's motion so as to refer to the correct record. Also, the accuracy of CRV is not very high because records have been approximately made [18].

Figure 6: An Illustration of Shade with Robot Controller

Network intrusion detection (NID) NID performs an analysis over the behavior, payload and contents of inbound and outbound network packets [36]. As attackers remotely attack the robot via network, NID should be practical. Given a mobile robot working in a normal routine, packets exchanged between it and a legitimate user must follow a regular pattern and the network payload should not change largely. But when attackers undertake to obtain and alter sensor values, they would bring about unusual network packets, either in a large quantity or with abnormal contents. An independent process monitoring the network traffic should detect such breaches.

A major drawback of NID is its high cost in computation and energy. Therefore, in an autonomous mobile robot powered by a battery, NID should be periodically called for energy-efficiency [44]. Also, NID cannot capture all attacks although they go through the network interface. Consider suspension attacks. If attackers manage to compromise a sensor just at the first try with few packets, NID might neglect such an attack.

5.2. The Design of Shade

Each of the aforementioned methods has its strengths and limitations. We have developed a hybrid method called the *shadow detector* (Shade). Shade acts as a shadow process of the robot control process and closely communicates with the latter to avoid missing any attack imprint. Figure 6 illustrates how the Shade process collaborates with robot control process through inter-process communication. The control process provides runtime information to Shade, such as the motion trace, sensor values, sensor latencies, etc. On the other side, Shade swiftly informs the control process in case of attacks.

Shade is a hybrid mechanism of fingerprinting, CRV and

Algorithm 2 The Shade Process (`Shade()`)

Input: A request for attack detection p with runtime information;
// p may contain the sensor mode μ, the sensor latency λ, the current location of robot ζ, etc.
Ensure: An attack alert γ // γ will be either **True** or **False**

```
1:  if (p is with sensor information) then
2:      if (μ is PASSIVE) then //Robot actively demands sensor value
3:          // Shade calls fingerprinting method with sensor latency
4:          γ := fingerprinting_check(λ);
5:      else // The sensor reports to robot periodically or in emergency
6:          // Shade calls CRV first with robot location, then NID
7:          γ := CRV_check(ζ) Or NID_check();
8:      end if
9:  else // Robot controller queries without sensor information
10:     // Shade calls NID method
11:     γ := NID_check();
12: end if
13: Return γ to the robot controller process;
```

NID so as to achieve wide coverage, high accuracy and low overhead. Algorithm 2 describes the main procedure of Shade. The robot control process sends a request for attack detection either in an on-demand or periodical way and the Shade process returns whether an attack is happening or not. If Shade receives a request with sensor information (Lines 1 to 8 in Algorithm 2), it first determines the working mode of the sensor. Given a sensor working at a passive mode with a measurable latency, Shade prefers the fingerprinting method that comes with low cost but high accuracy (Lines 2 to 4). However, as to a sensor working in a proactive or periodical mode, Shade calls CRV to validate the sensor value against historical records (Lines 5 to 8); nevertheless, due to the accuracy of CRV, Shade may use NID for double check with a short-circuiting logical **Or** operator (Line 7). Moreover, the robot control process may ask Shade without any sensor information. For example, the control process can consult Shade every five seconds. In this case, Shade needs to execute NID that finds out abnormal network traffics (Lines 9 to 11). In the end, Shade timely notifies the robot control process with a detection result (Line 13).

Shade can detect various attacks and it is beyond just integrating three methods in one process. First, Shade explores the *context* provided by the robot control process for attack detection. Generic NID can also detect the most attacks but with high cost for self-learning and frequent computations. Shade, however, gains legitimate network behaviors shared by the robot control process, which surely entails higher accuracy and less overhead. Second, Shade considers the pros and cons of three methods and complement them for wider coverage. Like at Line 7 of Algorithm 2, Shade makes NID recheck if CRV generates a false result because of the latter's accuracy.

6. Mitigation with Remit

Once Shade detects any attack affecting an autonomous mobile robot, we must mitigate the attack's impact. A straightforward solution is to halt the robot immediately. However,

Algorithm 3 The Mitigation Process (`Remit()`)

Input: A switch from normal mode to safe mode for mitigation.
Ensure: A completion of task, or a switch back to normal mode.

```
1:  repeat
2:      Change/keep the robot moving at a lower speed;
3:      // Navigate the robot with historical records used by CRV_check()
4:      Call Navigate_with_historical_records();
5:      Try to reset corresponding sensor;
6:      Call Network_block_attackers() to block attackers;
7:      if (attackers are successfully blocked) then
8:          Return back to normal mode;
9:      end if
10:     // A dynamic obstacle (e.g., a pet) might appear
11:     if (Robot cannot move with no obstacle recorded) then
12:         Play sound to drive the person/pet, and wait 1 second;
13:     end if
14:     Continue moving with historical records;
15:     if (the scheduled task is completed) then
16:         Return a completion to the robot control process;
17:     end if
18: until (Shade() returns False); // No attack any longer
19: Switch back to the normal mode of robot control process;
```

a shutdown of the robot badly loses its work efficacy because the robot is supposed to have a scheduled task, like tidying a room. Thus, we need a mitigation algorithm that retains as much work efficacy as possible for the robot being attacked. In particular, the mitigation algorithm ought to take into account two issues. First, an autonomous mobile robot significantly differs from stationary CPS and handheld smartphones as the robot needs to move itself to work. Since the distance sensor is not reliable due to attacks, how to navigate the robot to continue its motion must be resolved. Second, due to the limited resources of a small mobile robot, including the computation capability and energy supply, the mitigation algorithm should be lightweight and cost-efficient.

Regarding the two challenges, we have designed a mitigation algorithm, namely *retaining-oriented mitigation* (*Remit*), to achieve the least loss of work efficacy for an autonomous mobile robot. One noteworthy point of Remit is that, it reuses the historical records used by Shade in detecting attacks with CRV, which not only preserves the motion of robot, but also avoids any extra cost for enabling the navigation. Algorithm 3 captures the procedure of Remit. We define the robot without being attacked is in the *normal* mode. Remit switches the robot to *mitigation* mode once Shade detects an attack. On entering the safe mode, the robot first decelerates its speed (Line 2 of Algorithm 3). This helps it have more time to respond to an emergent object, say, an obstacle. Then, Remit leverages the historical records to navigate the robot (Lines 3 to 4). Since these records are not very accurate, Remit tries to repair the compromised sensor through resetting (Line 5) and calls the network module to block attackers (Line 6). If the attackers are successfully blocked, Remit will switch the robot back to the normal mode (Lines 7 to 8). Remit also needs to deal with a dynamic obstacle (e.g., a pet or person) if the robot cannot move at a time but no obstacle was recorded (Lines 10 to 13). Remit alerts the pet or person by playing a sound (Line 12) and continues

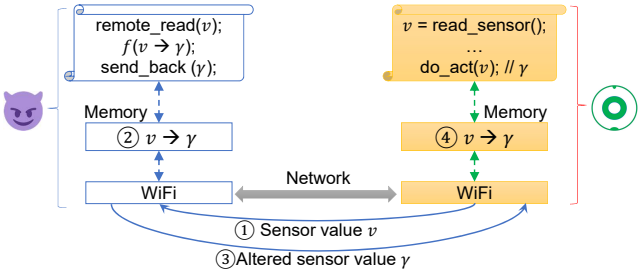


Figure 7: An Illustration of Initiating an Attack

moving (Line 14). If the robot completes the scheduled task, Remit stops the robot (Lines 15 to 17). Otherwise, Remit repeats the aforementioned steps until Shade detects no attack any more (Line 18).

Remit attempts to guarantee the work efficacy of the robot. Since the robot needs to move at a lower velocity, the time needed to complete a planned task might become longer. However, with regard to the robot being under attack, such additional time cost is insignificant and acceptable.

7. Evaluation

In this section, we would evaluate RoboFuzz, Shade, and Remit to answer following questions.

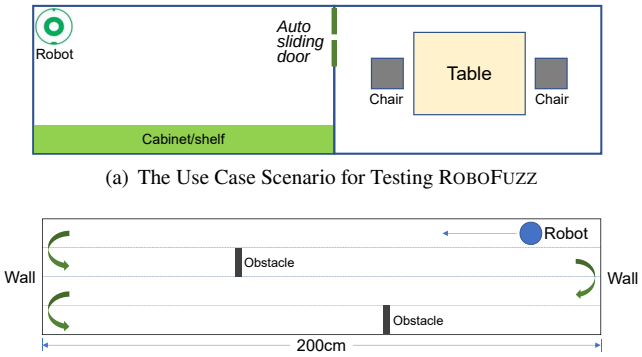
1) Does RoboFuzz manage to compromise an autonomous mobile robot? Compared to other fuzzing approaches, does RoboFuzz embrace a higher success rate?
2) Is Shade able to detect most of the attacks initiated through RoboFuzz?
3) Can Remit retain the work efficacy of mobile robot when mitigating the attacks detected by Shade?

We first present experimental setup and evaluation results regarding the competence of RoboFuzz in compromising a real-world cleaning robot with two attack targets. Then we test Shade and Remit to show their effectiveness in detecting two attack models and retaining the work efficacy of robot.

7.1. Evaluation Setup

Platform We use the aforementioned iRobot Create 2 [23] as the platform for evaluation. We have prepared a control program in Python 3 that runs in the Raspberry Pi 3 Model B+. The default velocity of the robot is set to be 50mm/s. The path planning of the robot follows the classic zigzag fashion. The main sensor used for the path planning is an ultrasonic distance sensor (HC-SR04) installed in front of the robot. The sensor can detect an obstacle from 2cm to 400cm. As mentioned in Section 2, in the robot control program, we configure the sensor to work in different modes to suit different attack models. In the passive sensor mode, the control program asks for the sensor value. In the proactive sensor mode, the sensor warns the control process if an obstacle is nearby or periodically.

As to the attacker side,. we have implemented RoboFuzz with two attack models (cf. Section 3.1). In light of



(a) The Use Case Scenario for Testing RoboFuzz

(b) The Use Case Scenario for Testing Shade and Remit

Figure 8: An Illustration of Use Case Scenarios for Evaluation

the analyses in Section 3.1, we set the sensor mode to be passive for fabrication attack model. For the suspension attack model, we choose the proactive mode.

Implementation In order to manipulate the iRobot Create 2, we make attack programs in a computer with Ubuntu 18.04. We first exploit the attack vector of WiFi interface of Raspberry Pi as the attack surface to invade the robot. Today many users still use default or simple passwords or their credentials are stored in plain text [33, 17]. For a Raspberry Pi with Raspbian, its default username/password are 'pi/raspberry'. After we successfully access the robot, we start to compromise it. Figure 7 exemplifies the process of altering one sensor value. As adversaries, we fetch the sensor value v through network (① in Figure 7), and then alter it to be γ via the function f (② in Figure 7). After sending back γ and replacing v (③ and ④ in Figure 7), the robot control program would proceed with γ instead of v. At runtime, RoboFuzz frequently reads v, but only when it perceives an appropriate opportunity, like the robot approaching a wall, will it call f and send faked γ back to mislead the robot control program.

Configuration We have made two scenarios to test RoboFuzz, Shade and Remit, respectively. Figure 8(a) shows the scenario we have used for testing RoboFuzz. It has two rooms that are connected by an automatic sliding door. The cleaning robot needs to clean both rooms starting from the top-left corner. When the robot is working, we try to compromise it using three fuzzing methods with two attack targets: 1) to damage the robot by crashing it to a hard obstacle (e.g., wall or cabinet), and 2) to reduce the robot's work efficacy by preventing it from entering and cleaning the right room. As to three fuzzing methods, the first one is Radamsa fuzzing sensor values for the control program, the second one is random fuzzing that initiates an attack at a random time with a hazardous alteration of sensor values (e.g., changing v of 10cm to γ of 60cm), and the third one is RoboFuzz. For both attack targets, we conducted 30 trials for a fuzzing method. We define the success rate as the fraction of the number of successful attacks to 30 trials in percentage. We note that besides Shade, the robot control program can rule out anomalous distance sensor values, e.g., ones that fluctuate greatly, and subsequently reset the sensor.

Table 1
The number of successful trials and success rates of three fuzzing methods to achieve the 1st target

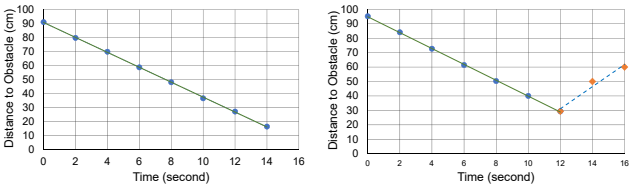
Fuzzing Method	Radamsa	Random fuzzing	RoboFuzz
The number of successful trials	0	5	30
Success rate	0%	16.7%	100%

Figure 8(b) captures the scenario we would use to test Shade and Remit. The reason why we evaluate them in a scenario different from Figure 8(a) is that we need a quantitative presentation to measure the work efficacy of robot in case of attacks. As mentioned, we use the quantitative success rate to show the effectiveness of RoboFuzz. On the other hand, the width of the room in Figure 8(b) is 200cm that falls into the range of HC-SR04 (≤400cm); the cleaning robot would cruise in the room, so we can record the exact distance a robot cleans with and without attacks. To avoid physically crashing the robot into walls or obstacles due to attacks, we set a safe distance to be 20cm. In other words, without any attack, when the distance to a wall or obstacle drops below 20cm, the robot should stop moving and turn left or right; however, on a successful attack, the robot spins itself in front of an obstacle to indicate that it is being attacked instead of really colliding with the obstacle. Concerning the safe distance and the diameter of robot, the robot would clean an estimate of 570cm overall in the room of Figure 8(b). In addition, for the use of fingerprinting and CRV, we have run the robot without any attack to collect sensor latencies and historical records of obstacles.

7.2. Target 1 for RoboFuzz: Damaging the Robot

In order to damage the robot, fuzzing methods must let the robot crash into a fixed obstacle in Figure 8(a). Note that the robot was not really damaged in trials but would play a special sound to indicate it was enforced to be within 5cm to an obstacle. Table 1 shows the number of successful attacks out of overall 30 trials for three fuzzing methods when they tried to achieve the target of damaging the robot. Radamsa failed in all trials because the robot control process certainly refused sensor values fuzzed by it as they evidently deviate from normal sensor values expected in the environment shown in Figure 8(a). As to random fuzzing, with regards to multiple stationary walls and furnitures in Figure 8(a), if it launched an attack at a moment when, though being randomly picked, the robot was approaching closely to any wall or furniture, the fuzzed sensor values might make the robot hit the obstacle and in turn attain the attack target. Whereas, since random fuzzing acts based on randomization, the success rate is low as confirmed by the experimental results (5 out of 30 trials).

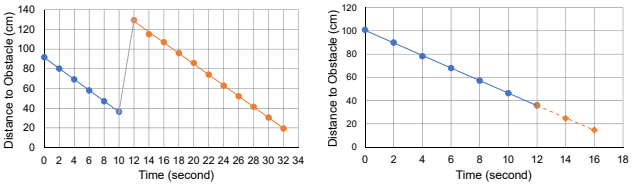
On the other hand, RoboFuzz successfully damaged the robot in all 30 trials. Because RoboFuzz continued to observe the environment and monitor the state of robot, at a proper occasion, it would generate sensor values that brought the robot from the curve in Figure 3(a) to the one in Figure 3(b). For a thorough comparison, we have collected distance sensor values in a normal routine without any attack



(a) Robot moving towards a wall without any attack

(b) Robot deceived with the appearance of dynamic obstacle

Figure 9: A Comparison between Distance Sensor Values with and without RoboFuzz when damaging the robot



(a) Robot moving towards automatic sliding door

(b) Robot stopped with auto sliding door changed to be an immovable wall

Figure 10: A Comparison between Distance Sensor Values with and without RoboFuzz when prematurely stopping the robot

and when RoboFuzz took effect in one trial. Figure 9(a) indicates that the sensor values from the normal routine well fit in a decreasing linear curve. On the other hand, in Figure 9(b), the solid linear curve links genuine sensor values before the attack initiated by RoboFuzz and the dashed line fits sensor values that impaired the robot. The two diagrams in Figure 9 clearly verify the capability of RoboFuzz.

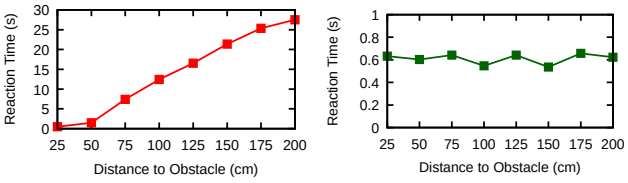
7.3. Target 2 for RoboFuzz: Reducing the Work Efficacy of Robot

To reduce the robot's work efficacy, we called three fuzzing methods to hinder the robot from tidying the right room. After the robot finished cleaning up the left room, the robot should not cross the automatic sliding door due to attacks. Table 2 shows that RoboFuzz achieves a success rate of 93.3% while the rates for Radamsa and random fuzzing are still low. Note that the success rates for both random fuzzing and RoboFuzz drop compared to that with the first target. The reason is, on damaging the robot, both fuzzing methods could find a number of static obstacles to leverage, but there is only one automatic sliding door connecting two rooms. Even so, RoboFuzz managed to sense the existence of automatic sliding door, and successfully changed sensor values in the most trials (28 out of 30) to be decreasing ones that emulated the door as an immovable wall.

We again tracked sensor values when the robot was going through the sliding door without attack (cf. Figure 10(a)). Also in one successful trial, we recorded sensor values the control process received before and after RoboFuzz launched the attack (cf. Figure 10(b)). As observed in Figure 10, after 12s, RoboFuzz effectively deceived the robot which subsequently stopped in front of the automatic sliding door.

Table 2
The number of successful trials and success rates of three fuzzing methods to achieve the 2nd target

Fuzzing Method	Radamsa	Random fuzzing	RoboFuzz
The number of successful trials	0	3	28
Success rate	0%	10.0%	93.3%



(a) Suspension Attack Model (b) Fabrication Attack Model

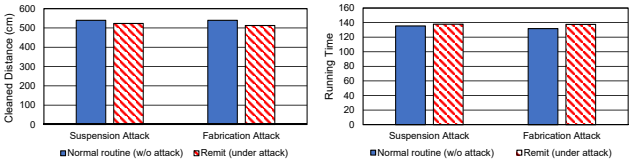
Figure 11: The Reaction Time of Shade to Attacks at Different Distances to Obstacle with Two Attack Models

7.4. Detection Results of Shade

We compared Shade to fingerprinting, CRV and NID methods. We used RoboFuzz to initiate attacks in line with two aforementioned attack models, i.e., suspension and fabrication attacks. For each attack model, a detection method underwent ten trials of attacks. So in all we performed $2 \times 4 \times 10 = 80$ trials regarding the composition of detection methods and attack models. Every trial was triggered at the startup of the robot, which means the robot is at the top-right corner as shown in Figure 8(b). We did so because an attack at the very beginning may incur the most challenges for a detection method, especially when the sensor works at a proactive mode reporting boolean values. We use two metrics to evaluate the effect of detection. One is the number of trials that a detection method successfully detected under an attack model. The other one is the average reaction time of ten trials for a detection method under each attack model.

Table 3 summarizes the results collected in 80 trials. Shade has successfully detected all trials while the limitations of other three methods are evident. For example, fingerprinting is competent only when the sensor works in the passive mode because the sensor latency is measurable. NID is not suitable for a suspension attack as such an attack model manages to suspend the sensor at the first attempt, which hardly leaves any hint for NID to take effect. Comparatively, Shade, as a hybrid detection method that closely collaborates with the robot control program, is not hindered by the working mode of sensors or attack models.

A notable observation revealed by Table 3 is that the average reaction time of Shade is much shorter or comparable than other detection methods. For suspension attacks, CRV could detect them as well. Given a suspension attack initiated at the startup of robot, only when the robot reached the safe distance (20cm) would CRV find that the sensor did not raise a 'True' warning. This is why the average reaction time for CRV and Shade is about 27s. For fabrication attacks, fingerprinting could instantly detect them. Meanwhile, the reaction time of CRV is much shorter for fabrication attack model than two preceding attack models. It is because of the passive sensor mode with fabrication attacks. Once the robot control program obtains a sensor value, it asks CRV to check the numeric distance, which facilitates CRV compared to boolean values used in the preceding two attack models.



(a) Cleaned Distance (b) Running time

Figure 12: A Comparison between Remit with Attacks and Normal Routine

The default position to initiate an attack is when the robot starts up, so the attack is issued at a distance of 200cm to the obstacle. To further verify the efficiency of Shade, we did more tests when RoboFuzz triggered attacks at eight different distances to the left wall. Figure 11 captures four curves of reaction time for Shade. In particular, given an attack occurring at a very short distance to the wall, say 25cm in Figure 11, Shade manages to detect it at 0.6s to 3s, which efficiently protects the robot from security threats.

7.5. Mitigation Results of Remit

We have also done experiments to evaluate Remit. The measurement of its effectiveness is the distance cleaned by the robot while its efficiency is measured in terms of running time to clean the use case in Figure 8(b). We first made the robot clean the use case in a normal routine without any attack and recorded the cleaned distance as well as running time. Then, we ran Remit with the robot under attacks. Figure 12(a) and Figure 12(b) present the results of cleaned distance and running time, respectively, for the normal routine and Remit. Since Remit leverages the historical records maintained by Shade for cross-checking, it can navigate the robot although the distance sensor is no longer reliable. Owing to the accuracy limitation of records in navigation, Remit made losses of 3.3% and 4.9%, respectively, with two attack models. The overall loss is 4.1%. Such insignificant losses confirm the effectiveness of Remit. On the other hand, after the robot entered the mitigation mode, Remit reduced the velocity of robot by 10%. Though, as the robot cleaned 4.1% less distances under attacks, the total running time at the mitigation mode is eventually 5.0% more than that of the normal routine. To sum up, Remit not only accomplishes scheduled tasks but also restricts extra time cost to an acceptable extent.

8. Related Work

CPS must be highly secure, especially for autonomous robotics [34, 25, 9, 2]. Researchers investigated the cyber threats to teleoperated surgical robots [8, 3]. For the cybersecurity of industrial robots, Quarta et al. [33] performed a thorough analysis. Comparatively, mobile robots are close to human beings, usually working together for service tasks [14]. Recently, Lera et al. [25] looked into the security threats with a survey on the cyber-attacks associated to mobile service robots as well as a taxonomy that classifies the risks

Table 3
Detection Results of Four Detection Methods under Two Attack Models

Attack Model	Number of Trials Detected				Average Reaction Time for Detection (unit: second)			
	Fingerprinting	CRV	NID	Shade	Fingerprinting	CRV	NID	Shade
Suspension Attack Model	0	10	0	10	Nil	27.1	Nil	27.2
Fabrication Attack Model	10	10	10	10	0.6	0.8	10.1	0.6

in using them. However, not much work has been done to compromise a mobile robot with rational but harmful sensor values as RoboFuzz does. In particular, Sabaliauskaite et al. [34] comprehensively developed methods to conduct cyber-attacks to a specific mobile robot. Whereas, their methods were significantly different from RoboFuzz since they tried to use irrational sensor values to crash the robot.

On the other hand, how to detect and mitigate attacks for various CPS has been investigated [31, 45, 9, 12, 5]. For example, Liu et al. [29] used partially observable Markov decision process to monitor and protect a smart home against pricing attacks. Dutta et al. [15] utilized the challenge response authentication to detect attacks for active sensors and the recursive least square algorithm to mitigate the impact of attacks. Chhetri et al. [11] studied how to detect an attack that could happen at various points of the digital process chain of analog emissions in CPS like a 3D printer.

Researchers have also looked into security issues of mobile robots in other aspects. For instance, Guerrero-Higueras et al. [18] attended attacks to real time location systems for autonomous mobile robots. Li et al. [27] proposed to upload the analysis of attack detection and mitigation to a cloud server in the improved deep belief networks. Our Shade and Remit differ from aforementioned approaches in that they detect attacks within the computational resources of an autonomous mobile robot and, furthermore, mitigate attacks without badly losing the robot's work efficacy.

9. Threats to Validity

We focus on protecting autonomous mobile robots. We use RoboFuzz to fuzz sensor values that would impact the physical movement of robot. We leverage the historical records of obstacles to detect fuzzed sensor values and navigate the robot to retain work efficacy. The limits of our proposals are twofold. First, they are not directly applicable to non-movable autonomous robots. Second, RoboFuzz fuzzes sensor values which are related to the movement of a robot; therefore, RoboFuzz does not cover how to fuzz values for other types of sensors, e.g., the detectors for dust and water.

The two attack models considered in this paper, i.e., suspension attack and fabrication attack, are comprehensive and representative. Adversaries have managed to conduct such attacks to CPS [12]. These two attack models target compromising the sensor values and subsequently misguide the robot control program. However, there exist other attack models that are not discussed in this paper. For example, a strong attacker may inject a malware in the control program; consequently, the attacker does not rely on forging or suspending sensor values to manipulate the robot.

The Shade and Remit schemes which detect and mitigate attacks launched by RoboFuzz demand the support of historical records of the environment. Thus, if a movable robot is placed in a fresh environment, or new furnitures are installed in the original environment, Shade and Remit might not function effectively as the records of such changed environments have not been fully obtained yet.

10. Conclusion

We have considered security threats for autonomous mobile robots in order to protect them. We propose RoboFuzz that automatically performs directed fuzzing in line with the normal state transitions of robot and the environment where the robot works. By fuzzing sensor values at appropriate occasions, RoboFuzz misleads the robot to a rational but dangerous state so as to compromise it.

Moving even further, we develop Shade and Remit to detect and mitigate attacks initiated through RoboFuzz, respectively. Shade and Remit take advantage of historical records of obstacles to detect inconsistent obstacle appearances regarding untrustworthy sensor values and navigate the mobile robot to continue working in motion. As a result, we are able to efficiently detect and mitigate attacks but also retain the robot's work efficacy, which in turn enhances the security and stability of autonomous mobile robot. Experiments with a real-world cleaning robot show that, 1) RoboFuzz dramatically outperforms fuzzing robot control program than state-of-the-art fuzzing tools, with much higher success rates of compromising the robot, and 2) Shade and Remit maintain a high work efficacy at the mitigation mode with an insignificant loss.

Declaration of competing interest

We declare that we do not have any commercial or associative interest that represents a conflict of interest in connection with the work submitted.

Acknowledgements

This work was jointly supported by grants RGAST1702 and MOE2018-T2-01-098, Singapore. C. Wang's work was partially supported by the Startup Funding of ShanghaiTech University.

References

[1] ISO 8373:2012. Robots and robotic devices – vocabulary, March 2012. https://www.iso.org/standard/55890.html.

[2] K. Ahmad Yousef, A. AlMajali, S. Ghalyon, W. Dweik, and B. Mohd. Analyzing cyber-physical threats on robotic platforms. *Sensors*, 18(5):1643, 2018.

[3] H. Alemzadeh, D. Chen, X. Li, T. Kesavadas, Z. T. Kalbarczyk, and R. K. Iyer. Targeted attacks on teleoperated surgical robots: Dynamic model-based detection and mitigation. In *2016 46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 395–406, June 2016.

[4] E. Bartocci et al. *Specification-Based Monitoring of Cyber-Physical Systems: A Survey on Theory, Tools and Applications*, pages 135–175. Springer International Publishing, Cham, 2018.

[5] S. Belikovetsky et al. dr0wned – cyber-physical attack with additive manufacturing. In *11th USENIX Workshop on Offensive Technologies (WOOT 17)*, Vancouver, BC, 2017. USENIX.

[6] bogdan-chub. Robot vacuum cleaner Xiaomi Mi robot vacuum stepped over the milestone, February 2018. http://gagadget.com/en/32219-robot-vacuum-cleaner-xiaomi-mi-robot-vacuum-stepped-over-the-milestone/.

[7] M. Böhme, V.-T. Pham, M.-D. Nguyen, and A. Roychoudhury. Directed greybox fuzzing. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pages 2329–2344, New York, NY, USA, 2017. ACM.

[8] T. Bonaci, J. Herron, T. Yusuf, J. Yan, T. Kohno, and H. J. Chizeck. To make a robot secure: An experimental analysis of cyber security threats against teleoperated surgical robots. *CoRR*, abs/1504.04339, 2015.

[9] A. Chattopadhyay, A. Prakash, and M. Shafique. Secure cyber-physical systems: Current trends, tools and open research problems. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, pages 1104–1109, March 2017.

[10] L. Cheng, K. Tian, and D. (D.) Yao. Orpheus: Enforcing cyber-physical execution semantics to defend against data-oriented attacks. In *Proceedings of the 33rd Annual Computer Security Applications Conference*, ACSAC 2017, pages 315–326, New York, NY, USA, 2017. ACM.

[11] S. R. Chhetri, A. Canedo, and M. A. Al Faruque. KCAD: Kinetic cyber-attack detection method for cyber-physical additive manufacturing systems. In *Proceedings of the 35th International Conference on Computer-Aided Design*, ICCAD '16, pages 74:1–74:8, New York, NY, USA, 2016. ACM.

[12] K.-T. Cho and K. G. Shin. Fingerprinting electronic control units for vehicle intrusion detection. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 911–927, Austin, TX, 2016. USENIX.

[13] D. Lee, W. Chung, and M. Kim. A reliable position estimation method of the service robot by map matching. In *2003 IEEE International Conference on Robotics and Automation (ICRA)*, volume 2, pages 2830–2835 vol.2, Sep. 2003.

[14] M. Decker, M. Fischer, and I. Ott. Service robotics and human labor: A first technology assessment of substitution and cooperation. *Robotics and Autonomous Systems*, 87:348 – 354, 2017.

[15] R. G. Dutta, X. Guo, T. Zhang, K. Kwiat, C. Kamhoua, L. Njilla, and Y. Jin. Estimation of safe sensor measurements of autonomous system under attack. In *Proceedings of the 54th Annual Design Automation Conference 2017*, DAC '17, pages 46:1–46:6, New York, NY, USA, 2017. ACM.

[16] P. Fiorini and E. Prassler. Cleaning and household robots: A technology survey. *Autonomous Robots*, 9(3):227–235, Dec 2000.

[17] D. Giese and D. Wegemer. Xiaomi smart home device reverse engineering and hacking, January 2018. https://github.com/dgiese/dustcloud.

[18] Á. M. Guerrero-Higueras, N. DeCastro-García, and V. Matellán. Detection of cyber-attacks to indoor real time localization systems for autonomous robots. *Robotics and Autonomous Systems*, 99:75 – 83, 2018.

[19] T. Haidegger, M. Barreto, P. Gonçalves, M. K. Habib, S. K. V. Ragavan, H. Li, A. Vaccarella, R. Perrone, and E. Prestes. Applied ontologies and standards for service robots. *Robotics and Autonomous Systems*, 61(11):1215 – 1223, 2013. Ubiquitous Robotics.

[20] A. Helin. Radamsa: a general-purpose fuzzer. https://gitlab.com/akihe/radamsa, June 2018.

[21] A. Humayed, J. Lin, F. Li, and B. Luo. Cyber-physical systems security – a survey. *IEEE Internet of Things Journal*, 4(6):1802–1831, 2017.

[22] iRobot Corporation. Company information of iRobot, September 2018. http://www.irobot.com/About-iRobot/Company-Information.aspx.

[23] iRobot Corporation. iRobot Create$_\wedge$ 2 programmable robot, August 2018. http://www.irobot.com/About-iRobot/STEM/Create-2.aspx.

[24] L. A. Kirschgens, I. Z. Ugarte, E. Gil-Uriarte, A. M. Rosas, and V. M. Vilches. Robot hazards: from safety to security. *CoRR*, abs/1806.06681, 2018.

[25] F. J. R. Lera, C. F. Llamas, Á. M. Guerrero, and V. M. Olivera. Cybersecurity of robotics and autonomous systems: Privacy and safety. In George Dekoulis, editor, *Robotics*, chapter 5. IntechOpen, Rijeka, 2017.

[26] LG Electronics. LG airport robots take over korea's largest airport, July 2017. https://www.lg.com/sg/press-release/lg-airport-robots-take-over-koreas-largest-airport.

[27] L. Li, L. Xie, W. Li, Z. Liu, and Z. Wang. Improved deep belief networks (IDBN) dynamic model-based detection and mitigation for targeted attacks on heavy-duty robots. *Applied Sciences*, 8(5), 2018.

[28] K. Lim. Changi airport turns to robots to keep T4 clean, August 2017. https://www.channelnewsasia.com/news/singapore/changi-airport-turns-to-robots-to-keep-t4-clean-9112610.

[29] Y. Liu, S. Hu, and T. Ho. Leveraging strategic detection techniques for smart home pricing cyberattacks. *IEEE Transactions on Dependable and Secure Computing*, 13(2):220–235, March 2016.

[30] Y. Liu, Pl Ning, and M. K. Reiter. False data injection attacks against state estimation in electric power grids. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, pages 21–32, New York, NY, USA, 2009. ACM.

[31] R. Mitchell and I.-R. Chen. A survey of intrusion detection techniques for cyber-physical systems. *ACM Comput. Surv.*, 46(4):55:1–55:29, March 2014.

[32] P. Moosbrugger, K. Y. Rozier, and J. Schumann. R2U2: monitoring and diagnosis of security threats for unmanned aerial systems. *Formal Methods in System Design*, 51(1):31–61, Aug 2017.

[33] D. Quarta, M. Pogliani, M. Polino, F. Maggi, A. M. Zanchettin, and S. Zanero. An experimental security analysis of an industrial robot controller. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 268–286, May 2017.

[34] G. Sabaliauskaite, G. S. Ng, J. Ruths, and A. Mathur. A comprehensive approach, and a case study, for conducting attack detection experiments in cyber–physical systems. *Robotics and Autonomous Systems*, 98:174 – 191, 2017.

[35] P. Salvini, G. Ciaravella, W. Yu, G. Ferri, A. Manzi, B. Mazzolai, C. Laschi, S. R. Oh, and P. Dario. How safe are service robots in urban environments? bullying a robot. In *19th International Symposium in Robot and Human Interactive Communication*, pages 1–7, Sep. 2010.

[36] R. Sommer and V. Paxson. Outside the closed world: On using machine learning for network intrusion detection. In *2010 IEEE Symposium on Security and Privacy*, pages 305–316, May 2010.

[37] A. Takanen, J. D. Demott, and C. Miller. *Fuzzing for Software Security Testing and Quality Assurance*. Artech House, Inc., Norwood, MA, USA, 1st edition, 2008.

[38] D. I. Urbina, J. A. Giraldo, A. A. Cardenas, N. O. Tippenhauer, J. Valente, M. Faisal, J. Ruths, R. Candell, and H. Sandberg. Limiting the impact of stealthy attacks on industrial control systems. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 1092–1105, New York, NY, USA, 2016. ACM.

[39] J. Vincent. iRobot's latest roomba remembers your home's layout and empties itself. https:

//www.theverge.com/circuitbreaker/2018/9/6/17817220/irobot-roomba-i7-robot-vacuum-empties-itself-maps-house, September 2018.

[40] S. Wang, X. Liu, J. Zhao, and H. I. Christensen. Rorg: Service robot software management with linux containers. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 584–590, May 2019.

[41] A. Wasicek, P. Derler, and E. A. Lee. Aspect-oriented modeling of attacks in automotive cyber-physical systems. In *Proceedings of the 51st Annual Design Automation Conference*, DAC '14, pages 21:1–21:6, New York, NY, USA, 2014. ACM.

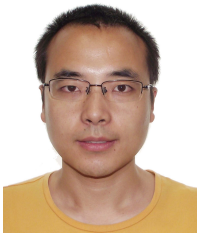
[42] T. Wei, B. Zheng, Q. Zhu, and S. Hu. Security analysis of proactive participation of smart buildings in smart grid. In *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 465–472, Nov 2015.

[43] Y. Liu and S. Hu and T. Ho. Vulnerability assessment and defense technology for smart home cybersecurity considering pricing cyber-attacks. In *2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 183–190, Nov 2014.

[44] K. Yan, L. Peng, M. Chen, and X. Fu. Exploring energy-efficient cache design in emerging mobile platforms. *ACM Trans. Des. Autom. Electron. Syst.*, 22(4):58:1–58:20, July 2017.

[45] B. Zheng, P. Deng, R. Anguluri, Q. Zhu, and F. Pasqualetti. Cross-layer codesign for secure cyber-physical systems. *Trans. Comp.-Aided Des. Integ. Cir. Sys.*, 35(5):699–711, May 2016.

[46] B. Zheng, W. Li, P. Deng, L. Gérardy, Q. Zhu, and N. Shankar. Design and verification for transportation system security. In *Proceedings of the 52nd Annual Design Automation Conference*, DAC '15, pages 96:1–96:6, New York, NY, USA, 2015. ACM.



Chundong Wang received the Bachelor's degree in computer science from Xi'an Jiaotong University (2004-2008), and the Ph.D. degree in computer science from National University of Singapore (2008-2013). Currently he is a tenure-track assistant professor in ShanghaiTech University, China. Previously, he worked in Singapore University of Technology and Design and Data Storage Institute, A*STAR. Chundong has published a number of papers in IEEE TC, ACM TOS, DAC, DATE, USENIX FAST, etc. His research interests include cyber-physical systems, data storage, non-volatile memory and computer architecture.

Yee Ching Tok received the Master's degree in information security from Royal Holloway, University of London, United Kingdom, in 2017. He is currently a PhD Student with the Information Systems Technology and Design Pillar, Singapore University of Technology and Design, Singapore. His current research interests are assessment of security in cyber-physical Systems, attack detection, and malicious software. Before pursuing his PhD degree, he worked as a Threat Hunter at Countercept where he helped to detect, respond and reduce impacts caused by malicious attackers to clients' critical assets. He has also carried out responsible disclosure of vulnerabilities to device manufacturers (CVE-2017-13663 and CVE-2017-13664) in professional and academic research activities.

Y. C. Tok serves as an executive committee member in the Association of Information Security Professionals in Singapore.

Rohini Poolat received the Master of Technology in Software Engineering from the National University of Singapore, Singapore in 2009. She is a research assistant in the cyber security research team at Singapore University of Technology and Design (SUTD), Singapore. She has worked on all phases of the software development cycle in various industry projects before moving into the research area. Her research interests include cyber-attack detection, mitigation and solutions to prevent attacks.

Sudipta Chattopadhyay received the Ph.D. degree in computer science from the National University of Singapore, Singapore, in 2013. He is an Assistant Professor with the Information Systems Technology and Design Pillar, Singapore University of Technology and Design, Singapore. In his doctoral dissertation, he researched on Execution-Time Predictability, focusing on Multicore Platforms. He seeks to understand the influence of execution platform on critical software properties, such as performance, energy, robustness, and security. His research interests include program analysis, embedded systems, and compilers.

Mr. Chattopadhyay serves in the review board of the IEEE Transactions on Software Engineering.

Mohan Rajesh Elara received the B.E. degree from the Bharathiar University, India, in 2003, and the M.Sc. and Ph.D. degrees from Nanyang Technological University in 2005 and 2012, respectively. He is currently an Assistant Professor with the Engineering Product Development Pillar, Singapore University of Technology and Design. He is also a Visiting Faculty Member with the International Design Institute, Zhejiang University, China. He has published over 80 papers in leading journals, books, and conferences. His research interests are in robotics with an emphasis on self-reconfigurable platforms as well as research problems related to robot ergonomics and autonomous systems. He was a recipient of the SG Mark Design Award in 2016 and 2017, the ASEE Best of Design in Engineering Award in 2012, and the Tan Kah Kee Young Inventors' Award in 2010.