INVITED PAPER

HoneyBot: A Honeypot for Robotic Systems

This paper describes a novel honeypot for robotic systems. Honeypots are internet computers that are set up as lures for attackers.

By Celine Irvene, *Member IEEE*, David Formby, *Member IEEE*, Samuel Litchfield, *Member IEEE*, and Raheem Beyah, *Senior Member IEEE*

ABSTRACT | Historically, robotics systems have not been built with an emphasis on security. Their main purpose has been to complete a specific objective, such as to deliver the correct dosage of a drug to a patient, perform a swarm algorithm, or safely and autonomously drive humans from point A to point B. As more and more robotic systems become remotely accessible through networks, such as the Internet, they are more vulnerable to various attackers than ever before. To investigate remote attacks on networked robotic systems we have leveraged HoneyPhy, a physics-aware honeypot framework, to create the HoneyBot. The HoneyBot is the first software hybrid interaction honeypot specifically designed for networked robotic systems. By simulating unsafe actions and physically performing safe actions on the HoneyBot we seek to fool attackers into believing their exploits are successful, while logging all the communication to be used for attacker attribution and threat model creation. In this paper, we present the HoneyBot and discuss our proof of concept implementation. Our HoneyBot prototype swaps between physical actuation and using prebuilt models of sensor behavior for simulation at runtime given user input commands.

KEYWORDS | Communication system security; computer simulation; cyber–physical systems; robot programming; robot sensing systems; systems simulation

I. INTRODUCTION

A robot is a device or mechanism guided by automated controls that can do the work of a person or perform complicated repetitive tasks [1]. Robots are used for a variety of different reasons, the most prominent of them being the elimination of the use of human operators. There are three main reasons for this; the first is to save on labor, which ultimately reduces costs. For instance, in the car manufacturing industry robots on the plant floor cost about $5.67 an hour over the course of their lifetime (including maintenance and energy costs), while the comparable human labor costs over $40 an hour (including wages, pension, and healthcare) [2]. The second reason to eliminate human operators is because human interaction might be bad for the product. We see this in the semiconductor manufacturing process, which consists of hundreds of steps. If at any point there is contamination (usually due to human error), the whole wafer could be ruined and the process must be restarted with a new one. The final, and arguably, most important reason for using robots is that the product or job may be too dangerous for humans. Imagine an encampment near a war zone, hours after a covert enemy attack has wiped out a squadron. The grounds are littered with corpses and debris and immediately after the assault not much can be done to rectify the situation. Rather than sending more troops to risk their lives, military robots can be sent to identify and collect the remains. Now let us imagine that same encampment if military robots had been used for battle instead of soldiers, the loss of life would have been minimized and even though there would have been equipment loss, the underlying intellectual property would still be available and the machinery rebuilt.

The current measures we have for avoiding scenarios like those previously mentioned, though limited, are growing and being deployed each day. These measures include technologies such as drones and unmanned ground vehicles. To reap the benefits of these robots they must be autonomous, remotely controlled, or a combination of the two. Given the variability of most missions, it is usually more desirable to have remote access through a network. While remotely accessible robots allow for more convenience and easier operational use, it opens them up to similar vulnerabilities

Manuscript received March 15, 2017; revised June 20, 2017 and August 28, 2017; accepted August 29, 2017. Date of publication September 26, 2017; date of current version December 20, 2017. This work was supported by the National Science Foundation under Grant 1544332. *(Corresponding author: Celine Irvene.)*
The authors are with the Department of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332 USA (e-mail: cirvene3@gatech.edu; djformby@gatech.edu; slitchfield3@gatech.edu; rbeyah@ece.gatech.edu).

Digital Object Identifier: 10.1109/JPROC.2017.2748421

as the ones faced in traditional computer networks. A classic example of this is Stuxnet, commonly known as the world's first digital weapon. Stuxnet was a 500KB computer worm that infected the software of at least 14 industrial sites in Iran, including a uranium-enrichment plant [3]. It worked by compromising the Siemens Step7 software used to program the programmable logic controllers (PLCs) driving the Iranian centrifuges. Once Stuxnet had access to the PLCs, the worm's creators were then able to take control of the industrial systems and caused the fast-spinning centrifuges to tear themselves apart, unbeknownst to the plant operators. The Stuxnet attacks were speculated to have been done over the course of several months by manipulating process pressure and centrifuge rotor speeds in an attempt to damage them and delay the Iranian nuclear program [4]. Stuxnet, if nothing else, has demonstrated that injecting malicious code on cyber–physical systems (CPSs), often embedded and real time, is possible and that every networked or remotely accessible system is at risk.

A CPS attack that occurred more recently than Stuxnet took place at the University of Washington where security researchers were able to hack a teleoperated surgical robot [5]. The lack of trained surgeons has been a major obstacle preventing life-saving surgeries from being performed in many parts of the world, and one solution to this involves leveraging the Internet and robotics. Teleoperated surgical robots enable expert surgeons to be in one country or region controlling a robot that physically performs a surgery in another country or region. The sale of these types of medical robots and others has been increasing at a rate of 20% per year and is expected to keep growing [6]. The rapid rise in popularity of teleoperated surgical robots has drawn the attention of several security experts at the University of Washington, leading them to discover that malicious attackers can disrupt the behavior of telerobots during surgery or take them over completely. The attack was performed on Raven II, a robot designed to reduce the size of medical robots and to improve their durability for use in extreme environments. The robot is built on a Linux-based computer running the robot operating system [39] and communicates via the Interoperable Telesurgery Protocol [38] over a public network with support for wireless and low link quality connections. This connectivity which is so helpful for saving lives is also what makes it vulnerable. "Due to the open and uncontrollable nature of communication networks, it becomes easy for malicious entities to jam, disrupt, or take over the communication between a robot and a surgeon," according to the researchers [6]. The attack's only requirement was that the malicious user be connected to the same network as the control console of the telerobot. The researchers performed three kinds of attacks on an experimental operation where the operator must move blocks from one peg board to another. They then measured the rate of task completion and the operator reported difficulty while under attack. Of the three types of attacks performed, the most appalling was how they were able to hijack a connection over the Interoperable Telesurgery Protocol due to insecure sequence number processing. This allowed them to take control of the teleoperated procedure and inject packets that triggered the built-in automatic stop mechanism, resulting in a denial-of-service (DOS) attack on the robot [5].

The prevalence of robotics is growing in all facets of everyday life and robots are becoming a crucial part of our ecosystem. We rely on them for military purposes on the war front, we rely on them for assisting doctors in the healthcare industry, and even first responders and the police use them. If we do not take steps to secure robotic devices they will become serious safety threats. In computer security, the first step to securing a resource is traditionally the development of a threat model. A threat model can help assess the probability and the potential harm, which can be useful in minimizing or eradicating the threat. Historically, security in robotics has not been an eminent concern, so to determine a valid threat model these systems must be studied and monitored to learn the scope of attacks they could face. We propose that this should be accomplished with a honeypot specially designed for robotic systems, which we call HoneyBot. The HoneyBot is the first software hybrid interaction honeypot specifically designed for networked robotic systems. By simulating unsafe actions and physically performing safe actions on the HoneyBot we seek to fool attackers into believing their exploits are successful, while logging all the communication to be used for attribution and threat model creation.

The rest of this paper is presented as follows. Section II describes related work in Honeypots and CPS specific Honeypots, as well as honeypot evasion techniques, alluding to how HoneyBot avoids these techniques. Section III provides background information on robotic systems. Section IV describes a proposed architecture for a honeypot for robotic systems, HoneyBot. Section V describes experiments performed to prove feasibility and build the required models. Section VI concludes the paper.

II. RELATED WORK

Since their inception, honeypots have primarily focused on the traditional IT computing domain, seeking to monitor attackers that aim to compromise company and government workstations/servers. Several different classes of honeypots were created. Honeypots were classified based on their levels of interaction and simulation. In addition to these previously defined classes, our previous work [18] defines a "hybrid" interaction honeypot, which is one that dynamically swaps between being low-interaction and high-interaction given certain triggering conditions. Table 1 breaks down the differences between these classes of honeypots.

The first honeynet (a network of honeypots) for CPSs/SCADA (supervisory control and data acquisition) was created by Pothamsetty and Franz of the Cisco Infrastructure Assurance Group (CIAG) in 2004 [7]. The goal was to simulate a few popular PLC services to help researchers better

Table 1 Breakdown of Key Features for Each Interaction Level Honeypot

Levels of interaction	How does it work?	Ease of deployment	Risk	Detection
Low interaction	Simulates services and applications	Simple	Low risk - do not run in production system	Easier to detect
High interaction	Utilizes real OS and applications	Complex	High risk - runs in production system	Difficult to detect
Hybrid interaction	Dynamically switches between real system and simulation	Simple	Medium risk - runs within production system	Difficult to detect

understand the risks of exposed control system devices. This work has laid the foundation for many other CPS honeypots [8], [9], none of which, though, are directly applicable to the robotics domain. With the prevalence of robotic systems on the rise, it is critical that advanced monitoring techniques, such as honeypots, be extended to defend them.

Honeypot evasion is, as with most security problems, a cat and mouse game. There almost always exist configuration fingerprints for any honeypot, and they are corrected as attackers discover them and defenders must devise appropriate improvements. An early example is the high-interaction honeynet, Sebek [25], which was discovered and trivialized (exposed as a honeynet) by techniques described in [26]. The techniques revealed how to detect, circumvent, and blind Sebek. Consequently, the fingerprints Sebek exposed were remedied by leveraging a similar system called Qebek that utilized virtualized high-interaction systems [27]. For many non-high-interaction honeypots, evasion boils down to finding the edges of emulation for the presented services, but timing approaches have also been used [28]. For example, Kippo is a popular medium-interaction honeypot for the SSH (Secure Shell) service [29]. Kippo is easily detected by sending a number of carriage returns, and noting the output difference from production SSH servers [30]. While high-interaction honeypots are harder to detect, many rely on virtualization. Virtualization technologies usually leave their own fingerprints, such as device names, device driver names, file-system hallmarks, and loaded kernel modules [31]. Even though these fingerprints can be altered, there exist a rich set of techniques for detecting virtualization (and defeating detection attempts) from the malware-analysis field [32].

Other, more honeypot-technology agnostic, detection techniques have been proposed. Some of these techniques rely on the liability issues inherent in hosting deliberately compromised machines. A botnet architecture proposed in [34] leverages the honeypot owner's desire to restrict outgoing malicious traffic to authenticate new hosts before integrating them into the botnet. Specifically, the new host is directed to send apparently malicious traffic to an already compromised "sensor." Most honeypot systems will attempt to identify and block or modify this malicious traffic, so whether the sensor receives the traffic unaltered can be used to determine if the new host is genuine. This work was built upon in [35], where multiple pieces of evidence can be formally combined to derive a metric of likelihood that a host is a honeypot. This evidence could be the virtualization status of the host, the diversity of software on the host, the level of activity of the host, or the difficulty in compromising the host. This newer technique is presented in the context of a botnet, but the generalized belief metric is equally applicable to any honeypot technology, depending on the evidence used. There are also techniques, described in two more recent surveys [36], [37], which elaborate on the themes mentioned above. Themes such as finding edges of emulation, finding subtle discrepancies that indicate virtualization, or analyzing the results of communication with an already compromised sensor.

The HoneyBot attempts to address these concerns for robotic systems. First, the robotic system exists and is in use, which remedies the traditional lack of context and provides convincing evidence correlated with a real system. The HoneyBot is not virtualized, but implemented on actual hardware. The robotic system fully implements the presented services, so all system responses are in line with the HoneyBot devices. Furthermore, because the system is a very specific robot, the system is not likely to try and propagate a compromise. The HoneyBot is the first software hybrid interaction honeypot specifically designed for networked robotic systems. By simulating unsafe actions and physically performing safe actions on the HoneyBot we seek to fool attackers into believing their exploits are successful, while logging all the communication to be used for attribution and threat model creation.

III. ROBOTICS

The field of robotics is constantly changing, but the components that unite almost every class of robots are sensors, actuators, and controllers. Fig. 1 shows a labeled diagram of the GoPiGo robot (the platform which the proof of concept HoneyBot was built on) [10] denoting these components. Some background information on the basic building blocks of robotics is required to describe how HoneyBot presents a honeypot for a full robotic system and is presented in Sections III-A–III-C.

A. Sensors

Sensors are the robot's eyes and ears; they enable the robot to understand the world around it and judge several features of the environment. Some examples of common sensors on a robot include: cameras, thermometers, IR (infrared) and sonar, accelerometers, and magnetometers. The HoneyBot must be able to "spoof," through simulation, all sensor values outputted by the robot such that the attackers are incognizant of the change. To do this we developed device models that provide realistic system responses given an "unsafe/indeterminate" input from an attacker. One such model that we have developed is for the Grove SEN10737P Ultrasonic Sonar. This device model was crafted through

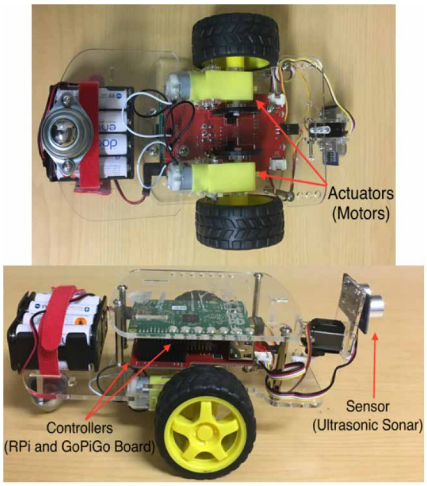


Fig. 1. ***Robot with labeled components.***

lab experimentation and empirical observation, such that for a given speed and distance the model mimics the true response time of an ultrasonic pulse, and "spoofs" the distance measured. These data are used to provide a reasonably timed response to the user such that there is no distinguishable difference when comparing the time taken for a simulated response versus an actual response. More details about experimentation and device modeling can be found in Section V.

In addition to spoofing individual sensor values, the HoneyBot must also spoof the correlation between sensors values. For instance, if the purpose of a certain robot is to navigate efficiently through an indoor–outdoor obstacle course, an attacker who sends commands to continually ram the robot into a wall in the outdoor portion of the course would not only expect to see the distance sensors reflecting these actions, but also the temperature sensor should reflect the outdoor temperature during a hot summer day. In this scenario, the HoneyBot would be triggered to perform simulations once the bad commands are received and would not actually allow the robot to crash into the wall, but instead output alternating distance sensor readings reflecting wall proximity and sensible temperature readings based on the known context of the robot's state. Given the sporadic nature of electronic components when subjected to conditions they are not built to support, one response (when destructive/unsafe commands aimed at destroying the robot are received) of the HoneyBot will be to disallow the occurrence of the actions and take down the connection to the attacker in a manner that suggests the robot has failed/died and was abruptly taken offline.

B. Actuators

The next component common to every class of robots is the actuators. Actuators enable the robot to modify the environment and move (or actuate). Some examples of these components include motors, speakers, and tools that go on the end of robotic arms, known as manipulators, like hammers, shovels, or even scalpels in the case of surgical robots. Since our proof-of-concept HoneyBot implementation lives in a robot that operates in networked environments and users/attackers have no physical or visual access to it, this work does not focus on modeling the underlying control system of the actuators. Instead, we emphasize on accurately simulating the timing of system responses and how user commands will change the state of the overarching system. In addition to this, we generate realistic device fingerprints and responses, which are sent back to remote users over the network. To do this, the HoneyBot must know what commands are safe and can be performed, and what commands are unsafe and must be simulated. We recognize that determining the "safety" of user commands is a hard problem to solve, and many researchers have worked on formal methods for verifying the safe navigation of ground robots [11]–[14]. Though it is not the focus of this work, these works could be leveraged and incorporated into the HoneyBot as the input verification module is built for plug and play. As shown in Fig. 2 all the commands received by the robotic system are passed through the input verification module to determine their fitness to be performed. Given the command and the current system state, the module must output whether the action is safe or unsafe. If deemed safe, the HoneyBot will allow the command to be performed. Otherwise, the command is not sent to the actuators but instead is simulated.

C. Controller

The third, and arguably most important, component of a robot is the control system or controller. The controller, also known as the brain of the robot, enables the robot to parse commands, send signals to different devices, and communicate with other robots as well as with the user. The HoneyBot is software that will live in the robot's controller so that it can easily access all data commands and signals to and from the robot's brain allowing it to make the necessary decisions.

Fig. 2 shows the flow of the HoneyBot system architecture. A user/attacker remotely connects to the networked robotic system through the Internet and can send

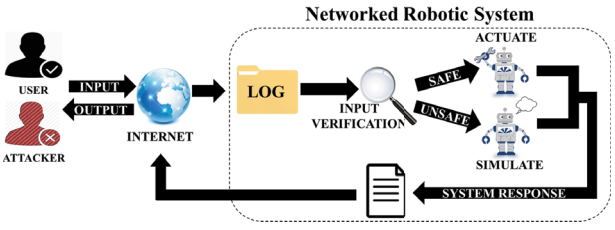


Fig. 2. ***HoneyBot system architecture.***

commands. All the commands received by the robot will be logged and passed to the input verification module, which as stated earlier is flexible in structure and can be very comprehensive in the evaluation of commands or relaxed depending on application needs. If the input verification module deems a command safe, the action will be performed as usual and the system response will be returned to the user/attacker. If, on the other hand, the input verification module deems a command unsafe the command will be simulated in real time and the "spoofed" system response will be generated and returned.

IV. HONEYPOT FOR ROBOTIC SYSTEMS

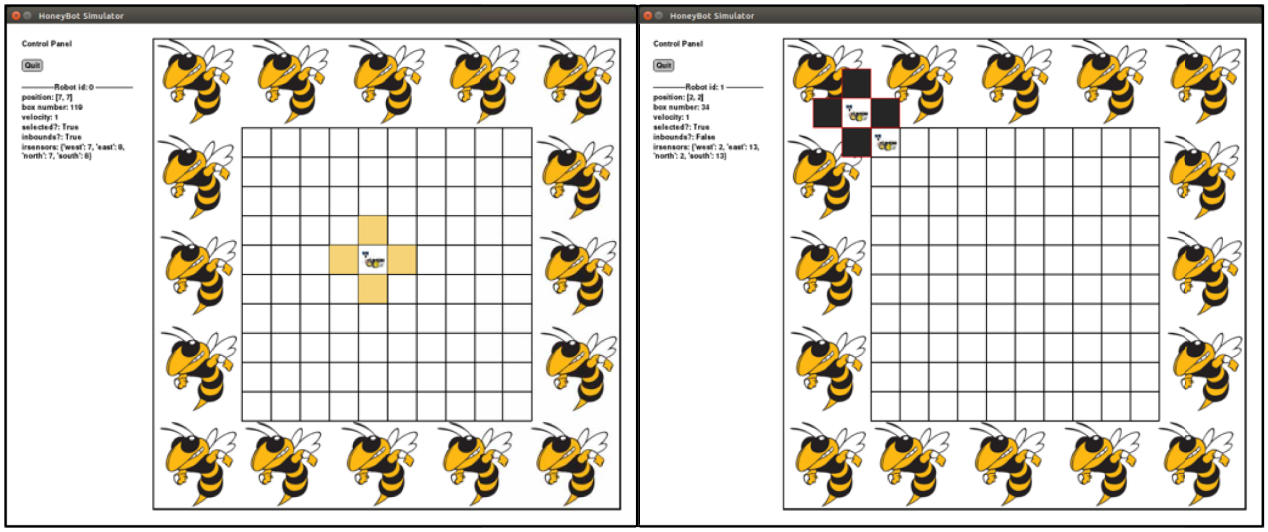
Most existing CPS honeypots neglect certain aspects of these systems that can alert an attacker to the nature of the honeypot, namely the physics of the devices that interact with the process [18]. That is why the HoneyBot is based on our previous work, HoneyPhy. HoneyPhy is a generic physics-aware honeypot framework that accurately models software and protocol fingerprints which are then used to simulate the CPS and fool attackers who access the honeypot [18]. HoneyBot then applies this generic framework, and tailors it to the specifics of a robotics system. The HoneyPhy framework is composed of three main modules: the internet interface, the process model, and the device model modules. As applied to the HoneyBot, the internet interface module is used for opening ports or interfaces on the robotic system so that it can connect to a network. It can also be used to support multiple connections to other networked robots and verify that state ensuring commands are always received by the correct robot at the correct port/address. In other words, the internet interface is the user facing interface, the front end that the attacker can see. The process model is triggered by an "unsafe" or "indeterminate" command, and the action will be simulated in real time by querying the appropriate device model rather than sent to be deterministically performed on the robot. The device model is different from the other modules in that it contains a model representative of each device found within a robot. These models are built using real data gathered from a given device. We have built several proof-of-concept device models, one of which is for the Sharp GP2Y0A21YK IR proximity sensor.

Device models are used for "spoofing" response times of sensors when the HoneyBot receives malicious commands. The details of the experimental setup and infrared distance sensor model building can be found in Section V. The device model may model a robot's motor. When the HoneyBot is sent a command to normally power the motor or any otherwise safe/benign command, the HoneyBot will allow the action to be performed. However, when the HoneyBot is sent a command to overpower the robot's motor the process model will query the motor device model and simulate the overpowering of the motor. The HoneyBot then changes the system state and responds to the attacker with data reflecting the sudden change in power to the motors. In general, the procedure for calling the process model will entail simulating an incoming malicious command and responding with the data obtained through simulation. Due to the seamless switches between allowing the robot to physically perform "safe" commands and simulating "unsafe" or "indeterminate" commands, the HoneyBot is classified as a hybrid interaction honeypot.

A. HoneyBot Simulator

As a proof-of-concept model for the HoneyBot, we have developed a simulator. The HoneyBot simulator is a graphical user interface (GUI) tool that provides a graphical and interactive resource for users to envision an attacker scenario in a networked robotic system (where the HoneyBot has been implemented). Screenshots of the simulator can be seen in Fig. 3.

The simulator displays a control panel to the left of a 16×16 bounded area, where the center 10×10 grid is the legal boundary for the robots to roam and perform tasks. Normal and benign activity can be performed within the 10×10 grid; this is representative of the traditional user privilege zone. Outside the legal boundary is the "honey," an extra boundary which is representative of resources that are attractive to an attacker, like open ports or unpatched software in a traditional computer network. In the simulator, a robot is represented by a GoPiGo image and has the following attributes: unique ID, velocity, position, box number, IR sensor values, legal moves, a value corresponding to whether the robot is within the legal bounds, and a value denoting if it is currently selected. Position is the robot's location in coordinate form and box number is the location in terms of the number of boxes from the first legal box (each of these location indicators can be derived from each other). The velocity is the number of boxes a robot can move at a time and the default value is 1. The IR sensor readings indicate how far in boxes the robot is from the legal boundary to the north, south, east, and west at any given moment. Given its velocity, a robot will have n number of possible next boxes that it can move to and this knowledge is always maintained by the bot. Two Boolean values tell the robot whether it is within the legal boundaries of the grid and if it is currently selected. Only one robot on the grid can move at a time and that robot must be "selected." This is indicated in the GUI by highlighting that robot. There is one attribute of the robot that is not exposed to the user, and this is the robot's true position. The true position is the system administrators' way of keeping tabs on the actual location of the robot. It is the only sensor that will not "lie." In the simulator, when a robot is out of bounds, all the information sent back to the user reflects this, from the IR sensor readings to the position updates, and even to the next moves it can make. This falsified information,



(a) (b)

Fig. 3. ***Screenshots of the HoneyBot simulator. (a) HoneyBot simulator with single robot inbounds. (b) HoneyBot simulator with single robot out of bounds. True position shown inbounds.***

shown in the control panel, is what the user has access to, while the true position, as represented in Fig. 3(b), where the robot is still inbounds, is only known to the system administrator. In this way, the attacker will be unaware he/she is being monitored and will continue performing the malicious actions. The malicious actions, in the case of a ground robot, could include the running of the robot into the edge of the true boundary to cause physical damage or robot–robot collisions.

V. EXPERIMENTATION AND MODEL BUILDING

For model development and to simulate the device behavior, we used a combination of techniques including experimentation and physical process modeling to simulate device behavior. The models built are queried at runtime to generate "spoofed" responses which are sent back to attackers when they perform malicious or otherwise unsafe actions. Suppose that the HoneyBot was implemented in a military drone used for finding improvised explosive devices (IEDs) and that it received a command directing it to go through a no-fly zone. Clearly, there is something suspicious so the input verification module will flag the action as unsafe. Then, the drone queries its GPS device model and sends back false coordinates, while maintaining its position in an unrestricted area, but leading the user to believe it is elsewhere. Device models must not only provide realistic state-aware data, but they must also reflect the correlation between sensors. For example, the distance sensors must corroborate the reported velocity data, and the velocity data must be in line with the encoder readings.

A. Proof-of-Concept Model Development

We have developed a proof-of-concept HoneyBot deployed on a GoPiGo [10] (a programmable robot which uses a Raspberry Pi 2 [19] as the robot's brain) to show the feasibility of the approaches discussed in this paper. The GoPiGo used in conjunction with the GrovePi (a sensor board for RPi's) [23] allowed us to connect many sensors to the robot. We purchased several of these Grove sensors from Seeed Studio, each costing less than $20 and we built the corresponding device models. Of the many device models developed, we will discuss three in detail, the models for the GrovePi SEN10737P Ultrasonic Sonar, the Sharp GP2Y0A21YK IR proximity sensor, and the Sensolute MVS0608.02 Collision sensor. Each of these sensors can be seen in Fig. 4.

B. Ultrasonic Sonar

The GrovePi SEN10737P Ultrasonic Sonar [shown in Fig. 4(a)] is a noncontact medium range distance measurement module which works at 42 kHz. This sonar is rated for measuring objects between 3 and 400 cm away, but we

(a) (b) (c)

Fig. 4. ***Images of the sensors. (a) Ultrasonic sonar. (b) Collision sensor. (c) Infrared distance sensor.***

found that it was only accurate up to approximately 200 cm. This lack of reliability is likely due to its inexpensive manufacturing, as it was not built for high fidelity sensing. An experimental test environment was set up in a laboratory to measure the time taken for the ultrasonic sensor attached to the HoneyBot driving at different speeds toward a static target. The lab set up consisted of placing the robot 183 cm away from a target and then driving it toward the target while taking distance and ultrasonic pulse time measurements. Ten trials per each of the seven chosen speeds (50–200 by increments of 25) of the robot were performed, 70 measurements in total. The speed on the GoPiGo robotic platform is unitless and represents a percentage of the full power the motors can get. For example, 0 is for no movement at all (the motors are stopped) and 255 is for setting the duty cycle at 100%. The objective was to succinctly determine the time taken to perform an ultrasonic pulse for a specific speed and distance from a target. The times were all recorded in a data frame along with the corresponding distance and speed and are queried at runtime when a malicious command is given and the HoneyBot must simulate a sonar reading. We created plots from seven lab experiments (where the ten trials have been averaged) that were run in order to map ultrasonic sensor obtained distances to the time taken for the ultrasonic pulse to be received as the HoneyBot approached a target at various speeds. Fig. 5 shows the results of each speed plotted together. From Fig. 5 (and underlying the sonar theory) we determined that at these low speeds and short distances the robot speed does not influence the time for an ultrasonic pulse to occur.

C. Collision Sensor

The Sensolute MVS0608.02 Collision sensor [shown in Fig. 4(b)] is a microvibration sensor that records motion and vibrations through a gold-plated moving microball. Due to its small size, it is perfect for mounting on the HoneyBot and detecting obstacle collisions. The digital sensor outputs a high signal when a collision is detected and a low signal if no collision is detected. To develop the device model for simulating the collision sensor, the sensors sensitivity to collisions had to first be tuned to ignore the rumblings of the robotic wheels and to only identify major crashes as triggers. The sensitivity was tuned by introducing delay to the querying process and by verifying the collision status after the delay. For instance, instead of querying the device every quarter of a second we might query every half second. If the sensor reads high (implying there was a collision) we wait for approximately 50 ms, then query it again. If the sensor still reads high, then we conclude that a collision has occurred. Due to the binary nature of this device, spoofing the output is as simple as returning a trigger event to the user when a simulated crash occurs.

The collision sensor device model was built and tested through a series of trials where the HoneyBot was placed on the ground varying distances from an obstacle and driven toward it. Each time a false positive trigger event occurred we modified the delay. We eventually determined that a delay of 50 ms produced the most accurate results on our carpeted lab surface. The Sensolute MVS0608.02 Collision sensor costs about $10 and at this price point reliability tends to be inconsistent. We found that the delay factor had to be tuned for different ground surfaces, but once properly calibrated will be accurate nine out of ten times. The key to simulating a collision is in the corroboration of sensors. The collision sensor device can easily trigger a collision event, but it must first read from the distance sensors that it has closed in on a boundary and they must reflect this by outputting a "spoofed" position.

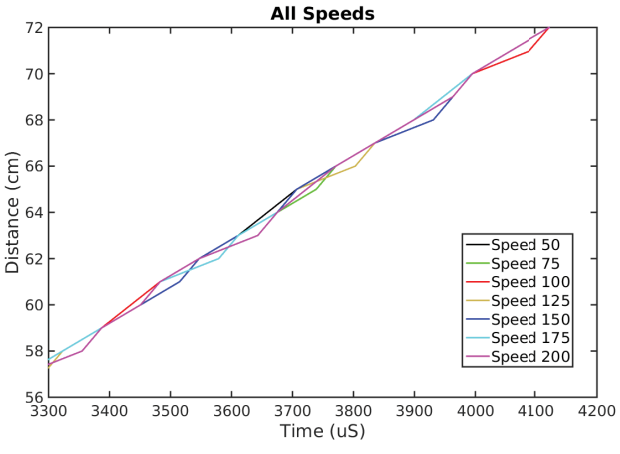


Fig. 5. ***Zoomed in distance versus time for all seven speeds on one plot.***

D. Infrared Distance Sensor

The Sharp GP2Y0A21YK IR proximity sensor [shown in Fig. 4(c)] is a general-purpose short range distance measuring sensor. This sensor is rated for measuring distances between 10 and 80 cm. It works by using triangulation and a small linear charge-coupled device (CCD) array to output an analog voltage, which can be used to compute the distance of objects in its field of view. The measuring process starts with a pulse of IR light being emitted from the emitter, which travels out and either hits an object or does not. If there is no object, the light is never reflected and the voltage reading is low. If the light reflects off an object, it returns to the detector and creates a triangle between the point of reflection, the emitter, and the detector. Because of this triangulation the outputted analog values do not have a linear relationship to the distance from the object. Given this, a common method of determining distance is to build a plot of voltage outputted from the sensor versus distance (measured manually) and derive a function from the fit of the line. The following equation was derived from data collected by a

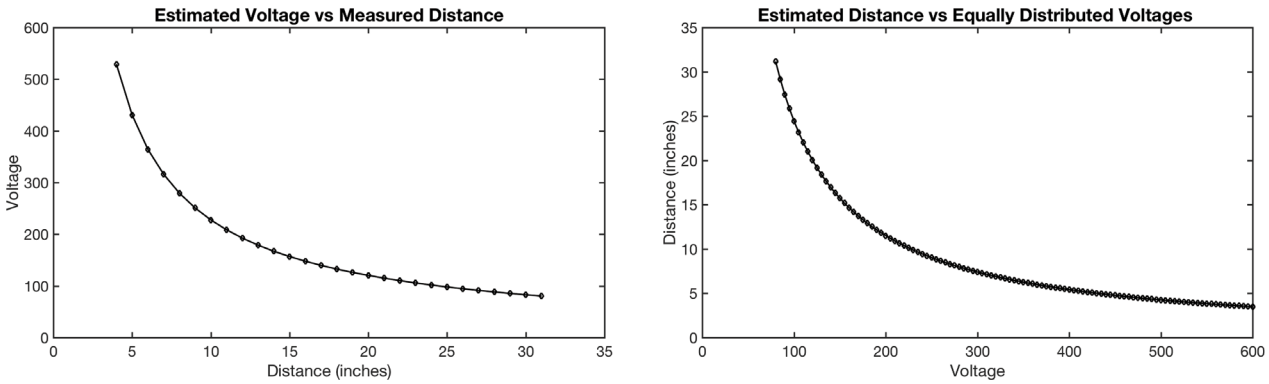


Fig. 6. ***(a) Plot of the estimated voltage versus measured distance from the experimentally determined (1). (b) Plot of the estimated distance versus equally distributed voltages from (2).***

user who performed the experiment described above, plotted, and fit a line [20]:

$$\text{observedVoltage} = 1893.9 * \text{distance}^{-0.92}. \quad (1)$$

This user's data are verified by the Sharp datasheet [24] and thus (1) is accepted as valid. In Fig. 6(a), we can see a plot created from (1). The following equation was computed by simply solving (1) for distance:

$$\text{distance} = 3650.4/\text{observedVoltage}^{1.087}. \quad (2)$$

A plot of (2), where a vector of voltages equally distributed from 80 to 600 by steps of 5, can be seen in Fig. 6(b). Given these two equations and an understanding that the voltage increases as objects become closer, "spoofing" the IR sensors was fairly straightforward. When a malicious navigation command is received by the HoneyBot, both the IR and the ultrasonic sonar device models will be queried and provided with a precalculated distance. The IR device model will plug the "spoofed" distance into (1) and return the voltage in the system response. The precalculated distance is determined by the HoneyBot through position estimation based on the malicious command given by the user and knowledge of the true and "spoofed" state of the robot.

E. Sensor Correlation

For a robotic system or any machine that performs tasks and reports back to users, sensors (in addition to supplying critical information for understanding an environment) provide vital insight to the equipment's operational status. Moreover, they can relay information about the task as well, like whether it was performed to completion. For instance, with the HoneyBot, the battery is a tell-tale sign of how much work the robot has done as well as how much more work it can do on that charge. Sensors can also conflict with each other. Consider an IR proximity sensor; it is a very effective means of measuring distance, but it is often susceptible to interference from outside light sources and changes in material reflectivity. Now consider ultrasonic sonar; it is also efficient at range detection (and for longer distances), but can be fooled by textures and echoes. Individually, if faced with either of their weaknesses these sensors will fail to perform accurately, but when used in conjunction they prove almost infallible.

Formally, let S represent the set of all n sensors on a specific robot, with some sensors being correlated and others completely independent. An intelligent attacker wishing to determine whether his malicious commands are being sent to a honeypot or to a real system can first build a ground truth of the sensor correlation behavior by collecting data during "normal" operation and performing pairwise Pearson correlation calculations on the sensor data $r_{xy} = (\text{cov}(S_x, S_y))/(\sigma_{Sx}\sigma_{Sy})$ to build an $n \times n$ correlation matrix R_{true}. Now, as he begins to send malicious commands to the robot, he can continually monitor the sensor data streams, calculating new correlation matrices R_{test}, and monitoring for changes in the correlation behavior. Mathematically, to make a decision D_i deciding if sensor S_i is real or simulated, the attacker first chooses some distance metric d, such as the $L2$-norm, and some threshold ϵ. The attacker then determines the $L2$-norm of the difference between row i of R_{test} with row i of R_{true}. If the result is greater than the threshold, this tells the attacker that the sensor correlation has changed and he is most likely in a honeypot, as illustrated in

$$D_i = \begin{cases} \text{real}, & d(R_{\text{true}}[i], R_{\text{test}}[i]) \leq \epsilon \\ \text{simulated}, & \text{else.} \end{cases} \quad (3)$$

This is why sensor correlation within the HoneyBot is so important. Under normal operation many sensors in a system will be related and, thus, under simulation these relationships must still hold and the sensor values must dynamically update with each user command. In order to accomplish this, we have built each device model for the HoneyBot sensors to mirror true operation as closely as possible in an effort to avert attacker detection for as long as possible.

VI. CONCLUSION

The need for security in the field of robotics will only grow in the coming years as robots are used for extensive tasks across various facets of life. Vulnerability and susceptibility to exploits in networked systems is unavoidable, and precautions must be taken to ensure the following:

- the system can notify administrators if it is compromised;
- there is a method for gathering intelligence on system intruders.

To help solve these problems, this paper introduced HoneyBot, the first honeypot specifically designed for robotic systems. Existing honeypots fail to deceive intelligent attackers because they do not accurately model device physics. Because HoneyBot leverages HoneyPhy and techniques from traditional honeypots, device models were built for common robotic sensors and are queried at runtime to provide convincing system state updates and responses. By simulating unsafe actions and physically performing safe actions on the HoneyBot we can fool attackers into believing their exploits are successful, while logging all the communication to be used for attribution and threat model creation. We have assumed a networked setup similar to the Georgia Tech Robotarium [33] in which users have no physical or visual access to the robotic system and send all commands remotely. Under this network configuration the users' only channel for keeping abreast of system changes are the real-time system responses sent over the network. The HoneyBot is a hybrid interaction honeypot specifically designed for robot systems and it could potentially be the *de facto* standard for networked robot security as the prevalence of robots grow in society. ■

REFERENCES

[1] Merriam-Webster (2016). *Robot*. [Online]. Available: http://merriam-webster.com/dictionary/robot

[2] S. Webzell. *The Rise of Robotics*, accessed: Feb. 3, 2015. [Online]. Available: http://www.automotivemanufacturingsolutions.com/technology/the-rise-of-robotics

[3] D. Kushner. *The Real Story of Stuxnet*, accessed: Feb. 26, 2013. [Online]. Available: http://spectrum.ieee.org/telecom/security/the-real-story-of-stuxnet

[4] R. Langner, "To kill a centrifuge," Langer Group, Tech. Rep., 2013.

[5] T. Bonaci, J. Herron, T. Yusuf, J. Yan, T. Kohno, and H. J. Chizeck, "Experimental analysis of denial-of-service attacks on teleoperated robotic systems," in *Proc. ACM/IEEE Int. Conf. Cyber-Phys. Syst.*, Seattle, WA, USA, Apr. 2015, pp. 11–20.

[6] E. T F. T. *Security Experts Hack Teleoperated Surgical Robot*. Accessed: Apr. 24, 2015. [Online]. Available: https://www.technologyreview.com/s/537001/security-experts-hack-teleoperated-surgical-robot/

[7] V. Pothamsetty and M. Franz *SCADA HoneyNet Project: Building Honeypots for Industrial Networks*, accessed: Mar. 20, 2004. [Online]. Available: http://scadahoneynet.sourceforge.net/

[8] L. Rist, J. Vestergaard, D. Haslinger, A. Pasquale, and J. Smith. Conpot *Conpot Development Team*, accessed: May 11, 2013. [Online]. Available: http://conpot.org

[9] K. Wilhoit and S. Hilt, "The GasPot experiment: Unexamined perils in using gas-tank-monitoring systems," Trend Micro, Tokyo, Japan, Tech. Rep., 2015.

[10] D. Industries. Dexter Industries, (2016). *GoPiGo*. [Online]. Available: http://www.dexterindustries.com/gopigo/

[11] M. Zhang and X. Zhang, "Formally verifying navigation safety for ground robots," in *Proc. IEEE Int. Conf. Mechatron. Autom.*, Heilongjiang, Chinese province, Aug. 2016, pp. 1000–1005.

[12] A. M. Zanchettin, N. M. Ceriani, P. Rocco, H. Ding, and B. Matthias, "Safety in human-robot collaborative manufacturing environments: Metrics and control," in *IEEE Trans. Autom. Sci. Eng.*, vol. 13, no. 2, pp. 882–893, Feb. 2016.

[13] L. Wang, A. Ames, and M. Egerstedt, "Safety barrier certificates for heterogeneous multi-robot systems," in *Proc. Amer. Control Conf. (ACC)*, Boston, MA, USA, 2016, pp. 5213–5218.

[14] N. Hacene and B. Mendil, "Toward safety navigation in cluttered dynamic environment: A robot neural-based hybrid autonomous navigation and obstacle avoidance with moving target tracking," in *Proc. Control, Eng. Inf. Technol. (CEIT)*, Tlemcen, Algeria, 2015, pp. 1–6.

[15] R. Lee. *Potential Sample of Malware from the Ukrainian Cyber Attack Uncovered*, accessed: Jan. 1, 2016. [Online]. Available: https://ics.sans.org/blog/2016/01/01/potential-sample-of-malware-from-the-ukrainian-cyber-attack-uncovered

[16] *Cyber-Attack Against Ukrainian Critical Infrastructure*. Dept. Homeland Security ICS-CERT, ICS-CERT, 2016.

[17] K. Trahan. *Industrial Control Systems: Next Frontier for Cyber Attacks?* accessed: Jun. 22, 2016. [Online]. Available: http://www.tripwire.com/state-of-security/featured/ics-next-frontier-for-cyber-attacks/

[18] S. Litchfield, D. Formby, J. Rogers, S. Meliopoulos, and R. Beyah, "Re-thinking the honeypot for cyber-physical systems," *IEEE Internet Comput.*, vol. 20, no. 5, pp. 9–17, Sep./Oct., 2016.

[19] R. P. Foundation, Raspberry Pi Foundation. (2016). *Raspberry Pi*. [Online]. Available: https://www.raspberrypi.org

[20] Matthew. (Dec. 27, 2014). *Linearizing the Sharp IR Ranger With an Arduino*. Accessed: Jan. 2017. [Online]. Available: http://projectsfromtech.blogspot.com/2014/10/linearizing-sharp-ir-ranger-2yoa21-with.html

[21] R. Goodrich. Live Science. (Oct. 1, 2013). *Accelerometers: What They Are & How They Work*. Accessed: Feb. 8, 2017. [Online]. Available: http://www.livescience.com/40102-accelerometers.html

[22] F. Cohen. The Risks Digest. (Mar. 9, 1998). *The Deception ToolKit*. Accessed: Feb. 22, 2017. [Online]. Available: http://catless.ncl.ac.uk/Risks/19.62.html#subj11

[23] D. Industries. Dexter Industries. (2016). *GrovePi*. [Online]. Available: http://www.dexterindustries.com/grovepi/

[24] *Wide-Angle Distance Measuring Sensor*. Sharp. GP2Y0A21YK datasheet, Jan. 2005.

[25] Honeynet. The Honeynet Project. (2017). *Sebek*. [Online]. Available: https://projects.honeynet.org/sebek/

[26] M. Dornseif, T. Holz, and C. N. Klein, "NoSEBrEaK—Attacking honeynets," in *Proc. 5th Annu. IEEE SMC Inf. Assurance Workshop*, Jun. 2004, pp. 123–129.

[27] Honeynet. The Honeynet Project. (2017). *Qebek*. [Online]. Available: https://projects.honeynet.org/sebek/wiki/Qebek

[28] P. Defibaugh-Chavez, R. Veeraghattam, M. Kannappa, S. Mukkamala, and A. H. Sung, "Network based detection of virtual environments and low interaction honeypots," in *Proc. IEEE Inf. Assurance Workshop*, West Point, NY, USA, Jun. 2006, pp. 283–289.

[29] Desaster. Github. (2017). *Kippo–SSH Honeypot*. [Online]. Available: https://github.com/desaster/kippo

[30] A. Morris (Dec. 2014). *Detecting Kippo SSH Honeypots, Bypassing Patches, and All That Jazz*. [Online]. Available: http://morris.sc/detecting-kippo-ssh-honeypots/

[31] T. Holz and F. Raynal, "Detecting honeypots and other suspicious environments," in *Proc. 6th Annu. IEEE SMC Inf. Assurance Workshop*, Jun. 2005, pp. 29–36.

[32] X. Chen, J. Andersen, Z. M. Mao, M. Bailey, and J. Nazario, "Towards an understanding of anti-virtualization and anti-debugging behavior in modern malware," in *Proc. IEEE Int. Conf. Dependable Syst. Netw. FTCS DCC (DSN)*, Anchorage, AK, USA, Jun. 2008, pp. 177–186.

[33] Georgia Tech. Robotarium. (2017). [Online]. Available: https://www.robotarium.gatech.edu

[34] C. C. Zou and R. Cunningham, "Honeypot-aware advanced Botnet construction and maintenance," in *Proc. Int. Conf. Dependable Syst. Netw. (DSN)*, Philadelphia, PA, USA, 2006, pp. 199–208, doi: 10.1109/DSN.2006.38.

[35] O. Hayatle, A. Youssef, and H. Otrok, "Dempster-shafer evidence combining for (anti)-honeypot technologies," *Inf. Secur. J. Global Perspect.*, vol. 21, no. 6, pp. 306–316, 2012.

[36] M. L. Bringer, C. A. Chelmecki, and H. Fujinoki, "A survey: Recent advances and future trends in honeypot research," *Int. J. Comput. Netw. Inf. Secur.*, vol. 4, no. 10, p. 63, 2012.

[37] M. Nawrocki, M. Wählisch, T. C. Schmidt, C. Keil, and J. Schönfelder (2016). "*A survey on honeypot software and data analysis*." [Online]. Available: https://arxiv.org/pdf/1608.06249.pdf

[38] H. H. King *et al.*, "Preliminary protocol for interoperable telesurgery," in *Proc. Int. Conf. Adv. Robot.*, Munich, Germany, 2009, pp. 1–6.

[39] M. Quigley *et al.*, "ROS: An open-source robot operating system," in *Proc. Open-Source Softw. Workshop Int. Conf. Robot. Autom. (ICRA)*, vol. 3. 2009, p. 5.

ABOUT THE AUTHORS

Celine Irvene (Member, IEEE) is currently working toward the M.S. degree in the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, USA.

She is a member of the Communications Assurance and Performance (CAP) Group. Currently, her research emphasizes network security for robotic systems and IoT devices.

David Formby (Member, IEEE) received the M.S. and Ph.D. degrees in electrical and computer engineering from Georgia Institute of Technology (Georgia Tech), Atlanta, GA, USA, in 2014 and 2017, respectively.

He is a Postdoctoral Researcher in the School of Electrical and Computer Engineering at Georgia Tech, and a member of the Communications Assurance and Performance (CAP) group. His research primarily focuses on network security for industrial control system networks.

Samuel Litchfield (Member, IEEE) is currently working toward the M.S. degree in the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, USA.

He is a member of the Communications Assurance and Performance Group (CAP). His current research interests lie primarily in network and control systems cybersecurity.

Raheem Beyah (Senior Member, IEEE) is the Motorola Foundation Professor and Associate Chair for Strategic Initiatives and Innovation in the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, USA, where he leads the Communications Assurance and Performance Group (CAP).

Dr. Beyah received the National Science Foundation CAREER award in 2009 and was selected for DARPA's Computer Science Study Panel in 2010. He is a member of the American Association for the Advancement of Science (AAAS), the American Society for Engineering Education (ASEE), a lifetime member of the National Society of Black Engineers (NSBE), and a Distinguished Member of the Association for Computing Machinery (ACM).