



# Rethinking the Honeypot for Cyber-Physical Systems

Honeypots derive much of their value from their ability to fool attackers into believing they're authentic machines. Current cyber-physical system (CPS) honeypots fail to sufficiently capture and simulate behavior that's necessary to project this authenticity. In response, the HoneyPhy framework was developed for CPS honeypots that takes into account behavior originating in the CPS process and devices that make up the CPS itself. HoneyPhy aims to make it possible to construct honeypots for complex CPSs. Here, the authors discuss their implementation of a proof-of-concept for this framework, and show that it's possible to simulate these behaviors in real time.

Since the early 1990s, the idea of entrapping and deceiving computer attackers to study their behavior and misdirect them has been used with great success in the computer security field. This practice, traditionally done through a computing system configured to emulate critical resources, called a honeypot, has revealed attacker strategies and kept more critical computing resources safe. As attackers learned of honeypots, they developed techniques to detect their presence.

As the cyber-physical system (CPS) space grows and becomes increasingly networked, attackers have been more interested in compromising the resources controlling these CPSs. Honeypots have

been designed to emulate CPS-specific components in response. However, all existing CPS honeypots neglect certain aspects of these systems that can alert an attacker to the nature of the honeypot – namely, the simulation of the attached physical process and the physics of the devices that interact with the process. We propose a new CPS-specific honeypot framework, called HoneyPhy: A Physics-aware Honeypot Framework, that addresses these problems and aims to be extensible to all CPSs.

## Traditional Honeypots

Traditionally, network-layer honeypots are broken into two broad classifications: low and high interaction. Low-interaction

**Samuel Litchfield, David Formby, Jonathan Rogers, Sakis Meliopoulos, and Raheem Beyah**  
*Georgia Institute of Technology*

honeypots work to accurately emulate a set of services and some system behaviors. No effort is given to other services, and the emulated services might not implement the service's full feature set. The limited usefulness of these honeypots motivated the movement to high-interaction honeypots.

High-interaction honeypots, in contrast, don't emulate. They're real resources, instrumented to log attacker's behavior, and deployed to be unused for any other purpose. Consequently, high-interaction honeypots offer great benefits in logging all of an attacker's behavior, but also expose great risk in allowing an attacker to potentially compromise these real resources.

### Fitting a CPS Honeypot into this Classification

Pure high-interaction honeypots are fundamentally unsuited to CPS, because they rely on either deploying another physical copy of the resource in question, or somehow virtualizing the resource. Deploying a copy of an entire CPS with the purpose of being compromised exposes the same safety risks as deploying the original system, and imposes large costs. A pure high-interaction honeypot can be deployed for a single component within a CPS, but without the physical portion of the system to interact with, the usefulness of these honeypots is limited. One solution, proposed in more detail in the following, is to create a hybrid-interaction honeypot, where real CPS devices and interfaces interact with process and device simulations that can effectively replicate the behavior of the CPS process.

### CPS Honeypots

The first low-interaction CPS targeted honeypot was released in March 2004 by Cisco Systems, and was called the Supervisory Control and Data Acquisition (SCADA) HoneyNet Project.<sup>1</sup> It leveraged Honeyd,<sup>2</sup> Arpd, Snort, and Tripwire to emulate many hosts on a network, and it aimed at emulating services for a Schneider programmable logic controller (PLC) and for a Siemens PLC. The honeypot made no attempt to simulate process behavior.

Released shortly after, Digital Bond's HoneyNet<sup>3</sup> had a similar goal of providing a low-interaction honeypot simulating a single Modicon Quantum PLC. The system can be configured to either have a virtual machine as a target, with simulated applications and Honeyd monitoring

interactions, or be deployed as a high-interaction honeypot with a real device. Both targets are set behind a Honeywall<sup>4</sup> designed to separate the target machine from production networks and to filter outgoing traffic.

GasPot is a low-interaction honeypot motivated by attacks observed on gas station control devices. This approach was based on research done at TrendMicro.<sup>5</sup>

Conpot is an actively maintained low-interaction CPS honeypot.<sup>6</sup> Conpot provides a range of protocols, and the system is extensible to multiple services and devices. While inherently extensible, Conpot isn't aimed at modeling processes or devices. System definition occurs through XML files, and protocol emulation takes place using Python.

The Crys PLC Honeypot (CryPLH)<sup>7</sup> is an actively developed low-interaction honeypot designed to emulate a Siemens Simatic 300 PLC. It simulates the exposed interfaces on a minimal Ubuntu Linux virtual machine, and uses a central configuration file to simplify the end user's configuration.

### Why Existing CPS Honeypots are Insufficient

In traditional network focused honeypots, as well as in existing CPS honeypots, the main goal was to emulate the kinds of protocol quirks that fingerprinting utilities such as Nmap and p0f look for. However, CPS honeypots should provide auxiliary information arising from the attached physical system. This auxiliary information enables the ability to compare the moment-to-moment state of the CPS for consistency (for example, leveraging the physics of the process and sensors), as well as observing the individual connected devices for unreasonable actuation times. If either the process physics or device actuation time are unrealistic, an attacker can easily determine whether they're in a honeypot.

A simple example can illustrate why process and device simulation are important to the design of a CPS honeypot. A consumer home heating, ventilation, and air conditioning (HVAC) system represents a familiar and intuitive CPS, where networked thermostats control physical devices such as heaters, compressors, and fans. In reality, if a command is issued by a thermostat to begin heating, a heater turns on. If temperatures are read in succession, the home temperature will slowly rise in response. Imagine that an attacker

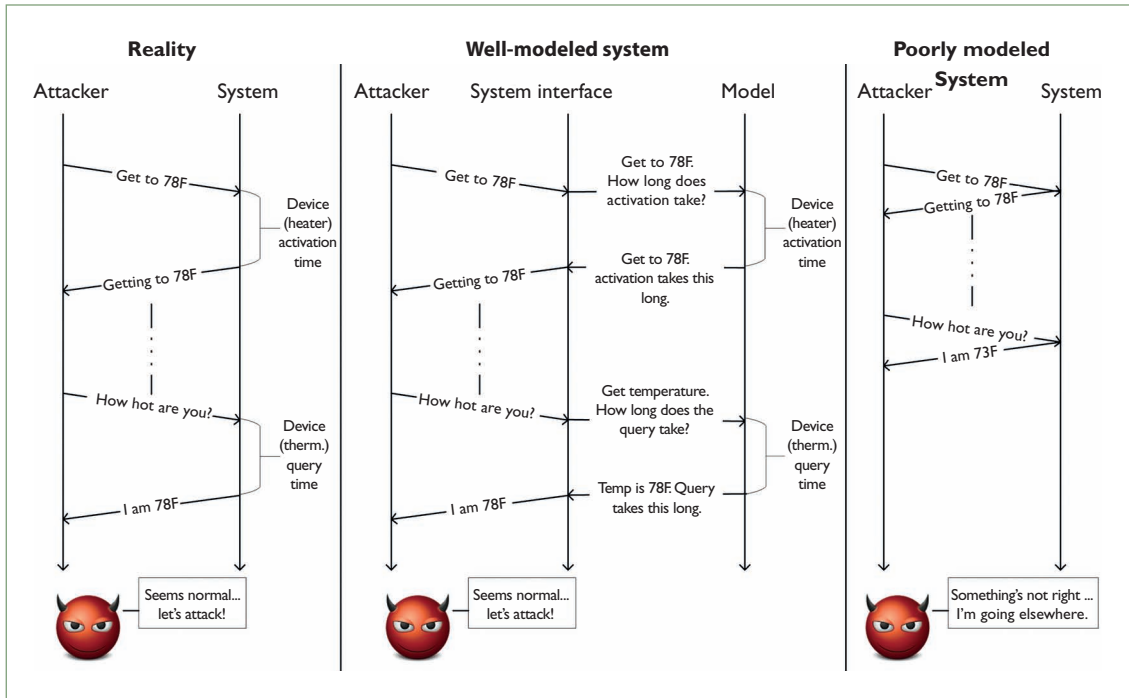


Figure 1. Outcomes of an attacker interacting with different levels of modeling. The left side shows an attacker interacting with a real system; the middle scenario shows a well-modeled system with an attacker interacting with a cyber-physical system (CPS) honeypot that's capable of modeling process behavior and device delay; and the right side shows a poorly modeled system with an attacker interacting with a CPS honeypot that doesn't model process behaviors and device delay.

is interacting with a honeypot designed to emulate this system. First, the attacker turns on the heater, and then he closely monitors the home's temperature sensor. If this honeypot makes no attempt to simulate the process it claims to control, and instead returns random responses or doesn't respond to the heating, an attacker will see temperatures over time that don't reflect the heater's activation. Alternatively, the temperature sensor could instantly show the final temperature. In either scenario, the attacker knows the system he's interacting with is either faulty, or doesn't control the system it claims to.

A similar example can illustrate the need to simulate the physical/mechanical delay of a device (for example, the actuator). If the thermostat controls a heater through the use of a mechanical relay, activation of the heater requires changing the state of that relay. This state change requires an amount of time determined by the relay's electromechanical characteristics.<sup>8</sup> In some devices, this delay can be on the order of milliseconds. If, as is the case in many other kinds of CPSs, the thermostat is

instrumented to confirm the state change, this device delay is now exposed to the attacker. An attacker can look for this device delay, and use it to test whether the system is a honeypot. Figure 1 illustrates this: the left scenario illustrates an attacker interacting with a real system, so all delays and responses result from process and device behavior. The right scenario illustrates an attacker interacting with a CPS honeypot that doesn't attempt to model process behaviors and device delay, and the lack of this delay and deviations from expected process behavior will alert the attacker to the honeypot. The middle scenario illustrates an attacker interacting with a CPS honeypot that's capable of modeling both process behavior and device delay. Responses provided to the attacker are thus realistic, and the attacker proceeds to perform observed malicious actions.

Table 1 shows the state of emulation provided by the five previously noted CPS honeypots. None of them attempt to model process behavior or device delays. In the table, the term "proven" either indicates an attacker was fooled by the honeypot at this level in the honeypot's

Table I. Modeling capabilities of existing CPS honeypots.

Capability	Supervisory Control and Data Acquisition (SCADA) HoneyNet	Digital Bond HoneyNet	GasPot	Conpot	Crysys Programmable Logic Controller (PLC) Honeypot (CryPLH)
Human interaction	Proven	Proven	Proven through experimental results	Proven	Mostly indistinguishable <sup>7</sup>
Network stack emulation	Honeyd provides stack emulation	Capable of instrumenting a real PLC, so a full stack	Relies on Python's network stack, so fingerprintable	Proven through testing by Nmap	Nmap can discern between honeypot and real PLC
System/process simulation	Single device emulation, so none	Single device emulation, so none	None	None by default, but extensible	None
Device delay modeling	None	None	None	None	None

history, or that the honeypot was tested by an associated tool by the honeypot's creator.

Our Vision for Future CPS Honeypots

We address the limitations of current CPS honeypots by providing HoneyPhy's extensible framework that accounts for the physics of the physical process and the mechanical delays of the physical actuators.

Future honeypots should correctly model software and protocol fingerprints, as with all other honeypots. This interface layer of the honeypot could be either high or low interaction, depending on access to equipment such as human-machine interfaces (HMIs).

In addition to this, future honeypots should correctly model the physical system's behavior. This primarily ensures that physical parameters behave in a way consistent with attacker expectations. A CPS process model will require simulation.

Finally, future honeypots should correctly model the time delay introduced by the constituent devices within the CPS. These delays could either originate from the operation of real devices used in the CPS honeypot, or be generated by modeling the devices. Attackers measuring the physical actuation time for these devices could detect a honeypot environment if the measurements lay outside the known operation times for each device. These operation times can be modeled, and these models can be generated in one of two ways – white or black box modeling.

Black box modeling is the method of generating a model for the device's behavior based on physical access to it and a set of true

measurements of its behavior. By comparison, white box modeling requires no physical access to the device and involves mathematical modeling of the device based on estimates of the device parameters and standard physical models. The primary advantage black box modeling holds over white box modeling is that it results in the most accurate representation of the device behavior, because it's based on empirical measurements. In our previous work, we showed that not only is the construction of these models feasible, the models are convincing.<sup>8</sup>

The New CPS Honeypot Architecture

To satisfy our vision of future honeypots, HoneyPhy offers a CPS hybrid-interaction honeypot framework. This framework contains three major modules: the Internet interface(s) module, process model(s) module, and device model(s) module. Each module's contents, permissions, interfaces, controllable and sensible process variables, and metadata are configured by a central XML file. Figure 2 shows a framework overview.

Internet Interface(s) Module

The Internet interface module exposes the declared interfaces at the declared addresses. It will maintain connections and multiplex them to their destinations, while ensuring that outgoing packets reflect the network fingerprint for each device, and modifying them if necessary.

Process Model(s) Module

The process model exists to simulate the physical process in question. It can be interrogated and acted upon by the other devices modeling

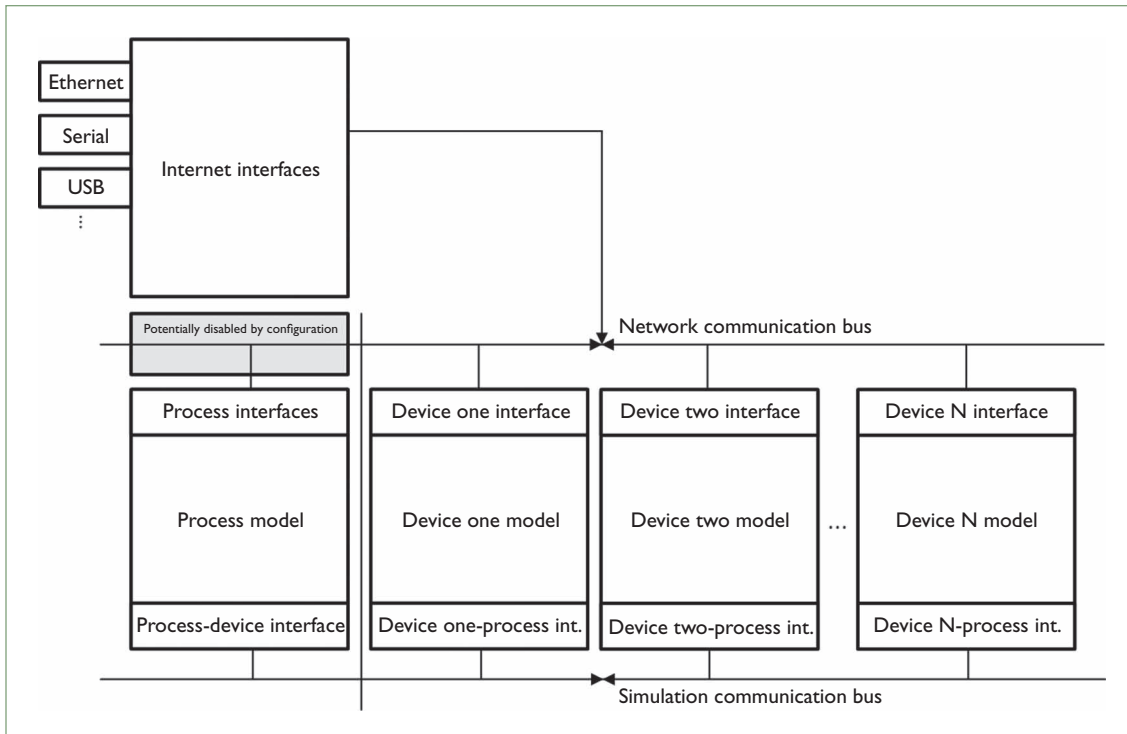


Figure 2. HoneyPhy, the proposed CPS honeypot architecture. The framework contains three major modules: Internet interface(s) module, process model(s) module, and device model(s) module.

sensors and actuators, and should simulate the process in real time.

The process model communicates with the various devices and models over a separate databus. If desired, a secure Internet-facing interface can be opened to remotely interact with the process model directly.

Process models could consist of National Instruments LabVIEW simulations, replays of empirically observed responses (as done in our proof-of-concept presented later), or more traditional controls system models such as a linear dynamical system or auto-regressive models.<sup>9,10</sup>

### Device Model(s) Module

The device models encompass all devices found within a CPS. Where they model computing devices such as PLCs, the model should implement logic to simulate those devices. This can include interpreting incoming queries and responding with the value they obtain by querying the process model, or executing incoming commands by modifying the process model. Where device models simulate mechanical devices such as relays or valves, the model should introduce a suitable delay corresponding to the time necessary to change the device's state.

Device models can range from very simple low-level black box timing models to real devices, sufficiently instrumented to interact with the process model.

### Inter-module Communication

While our framework doesn't specify a required inter-module communication method, it does specify where communication must be available. The Internet interface module must be able to route incoming and outgoing communications to all applicable devices, and optionally expose the Internet-facing interface for the process model. Additionally, all devices must be able to talk not only to each other, but also to the process model. This necessitates a separate process/device databus.

### Our Proof-of-Concept Implementation

Leveraging an early version of HoneyPhy, we constructed a simple heating system, modeled the process and devices involved, and set up a real-time simulation that uses existing honey-pot technology as the system interface.

### Physical System Architecture

The simple heating system consists of an insulated and heated volume, a set of components

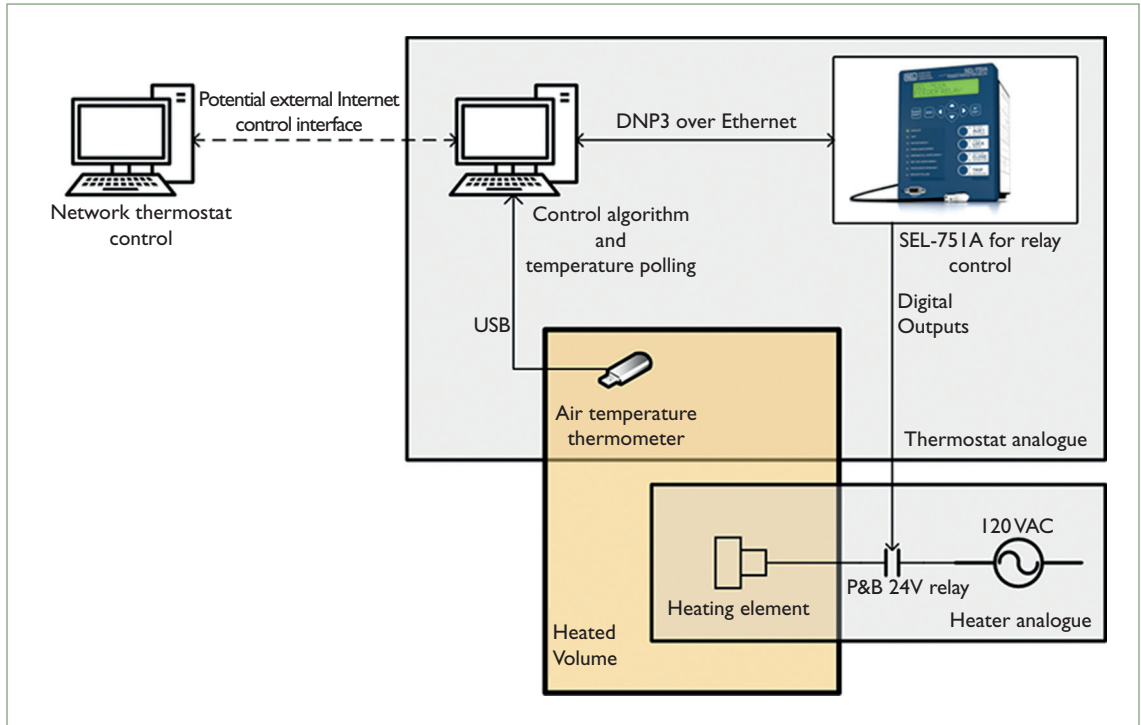


Figure 3. Logical architecture of a proof-of-concept heating system. The system consists of an insulated and heated volume, a set of components acting as a thermostat, and a set of components acting as a heater.

acting as a thermostat, and a set of components acting as a heater. Figure 3 shows the system's logical architecture.

The thermostat analogue labeled in Figure 3 consists partially of a computer executing temperature control logic based on the temperature read from a USB thermometer located inside the heated area. If this temperature violates the programmed limit set points, the PC sends a Distributed Network Protocol (DNP3) command, using the OpenDNP3 library, to a connected SEL-751A Relay to either turn the heater on or off. The SEL-751A Relay responds to that command by closing or opening a physical relay (a Potter and Brumfield KUL-11D15D-24), for which a black box model was previously obtained.

This physical relay, along with the ceramic heater bulb it controlled, formed our heater analogue (see Figure 3).

While not implemented in the system, in a real Internet of Things (IoT) thermostat the current temperature, heater status, and set points could be interrogated and controlled through a PC's external network interface.

### Models and Simulation

To simulate this system, we developed an analytic model of how the enclosed volume was heated

and cooled. This was based on empirical results gathered from the internal USB thermometer, and made up of closed-form equations seen in Equations 1 and 2:

$$T_{\text{bulb}}(t) = T_{\text{bulb}}(t - \Delta t) + \Delta t \cdot [(S_{\text{heater}}) \cdot (0.00775 \cdot T_{\text{bulb}}(t - \Delta t)^{1.04}) - 0.00084 \cdot (T_{\text{bulb}}(t - \Delta t) - T_{\text{air}}(t - \Delta t))^{1.48775}] \quad (1)$$

$$T_{\text{air}}(t) = T_{\text{air}}(t - \Delta t) + \Delta t \cdot [0.00084 \cdot (T_{\text{bulb}}(t - \Delta t) - T_{\text{air}}(t - \Delta t))^{1.085} - 0.00041 \cdot (T_{\text{air}}(t - \Delta t) - T_{\text{env}}(t - \Delta t))^{1.225}] \quad (2)$$

where  $T_{\text{bulb}}$ ,  $T_{\text{air}}$ , and  $T_{\text{env}}$  represent the temperatures in degrees Celsius of the bulb, air, and environment, respectively;  $t$  indicates time; and  $S_{\text{heater}}$  represents the heater's state. The coefficients resulted from creating the general form of the equations from Newton's Law of Cooling, then fitting the equations to the observed data. We used this analytic model for the process model portion of the simulation.

Figure 4a shows an observed heating/cooling curve from the physical system, and a similar curve generated by the process model for the same control actions. The heater is energized at the beginning of the dataset and de-energized at the vertical red line. Assum-



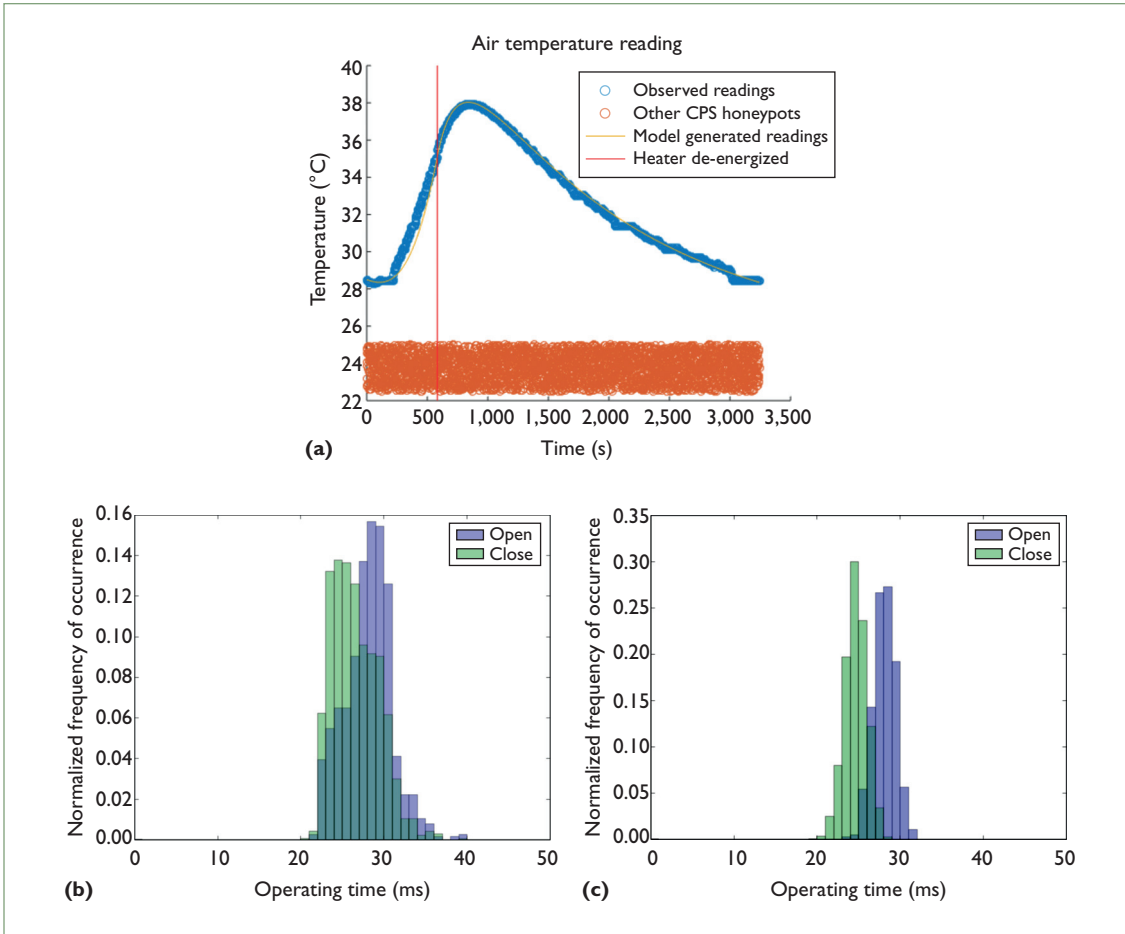


Figure 4. Simulation results. (a) Process model comparisons. (b) Black box model used in simulation. (c) White box model used for the same relay.

ing these control commands originate from the attacker, if they energize the heater at  $t = 0$  s, at  $t = 500$  s they can compare the reported system temperature against either their intuition of the system or a separate process model they've created.

Data points have been superimposed below the real data to illustrate what SCADA Honeynet and GasPot would return when asked for the air temperature, if used to simulate a similar system. We generated this data by inspecting relevant sources. This clearly doesn't capture process behavior, and when the attacker checks the system's reported behavior at  $t = 500$  s, it will be clear that something is wrong.

To capture device behavior, we leveraged black box models developed in previous work for the same relay,<sup>8</sup> randomly sampling the black box data gathered. We also developed a white box model for this relay. The simulation is convincing with both models, as can be seen

in Figures 4b and 4c, which shows histograms of device operation times from both models. Machine learning tools leveraging the white box model differentiated two relays with 80 percent accuracy.<sup>8</sup>

### Extending to Real CPSs

While the proof-of-concept provided here applies to a scaled-down version of an HVAC setting, HoneyPhy will extend to any CPS, of any size, given that models are created at the right level of abstraction.

One complete example could be to simulate an electric power substation. Constructing a honeypot that models an entire substation would require the ability to simulate the effects that various control actions would have on the substation's state.

Power transmission systems commonly use system-level state estimation to combine individual sensor readings from substations into a

representation of the state of the entire system. Our previous work involved the development of a substation-level distributed state estimation (DSE) for the purpose of detecting malicious control actions within power substations.<sup>11</sup> This DSE system uses a high-fidelity dynamic model (three-phased) and accurately simulates the results of control actions on an individual substation's systems faster than real time. For a prospective honeypot modeling a substation, this DSE could be used as the process model for an entire substation. While it has faster-than-real-time capability, it will respond to an attacker's commands with the actual timing of system evolution so that the attacker will think he's interacting with the real system.

With the DSE in hand, all that would be required to implement the proposed framework is developing models for the individual power devices that make up the substation. In Appendix E of our previous work,<sup>8</sup> we constructed a white box model for a common Siemens GMSG vacuum circuit breaker, proving the feasibility and value of constructing these device models.

High-fidelity power system models tend to be complex. For a practical HoneyPhy system, it will be necessary to automatically create the high-fidelity model. The model's automated generation is described elsewhere.<sup>12</sup> With a real-time process model – proven possible by the DSE, and sufficiently convincing device models, which are shown to be obtainable – a convincing honeypot for a system as complex as a substation could be constructed using the proposed CPS honeypot framework.

**M**ore details and future updates to HoneyPhy can be found at [www.honeyphy.gatech.edu](http://www.honeyphy.gatech.edu). □

### References

1. V. Pothamsetty and M. Franz, *SCADA HoneyNet Project: Building Honeypots for Industrial Networks*, tech. report, Cisco Systems, 2005; <http://scadahoneynet.sourceforge.net>.
2. N. Provos, *Developments of the Honeyd Virtual Honeypot*, user forum, 2008; <http://honeyd.org>.
3. Digital Bond, *SCADA HoneyNet*, 2016; <http://digitalbond.com/tools/scada-honeynet>.
4. The HoneyNet Project, *Honeywall*, 2016; <https://projects.honeynet.org/honeywall>.
5. K. Wilhoit and S. Hilt, *The GasPot Experiment: Unexamined Perils in Using Gas-Tank-Monitoring Systems*, white paper, 2015; [www.blackhat.com/docs/us-15/materials/us-15-Wilhoit-The-Little-Pump-Gauge-That-Could-Attacks-Against-Gas-Pump-Monitoring-Systems-wp.pdf](http://www.blackhat.com/docs/us-15/materials/us-15-Wilhoit-The-Little-Pump-Gauge-That-Could-Attacks-Against-Gas-Pump-Monitoring-Systems-wp.pdf).
6. L. Rist et al., "Conpot," 2016; <http://conpot.org>.
7. I.D. Buza et al., "CryPLH: Protecting Smart Energy Systems from Targeted Attacks with a PLC Honeypot," *Smart Grid Security*, LNCS 8448, Springer, 2014; pp. 181–192.
8. D. Formby et al., "Who's in Control of Your Control System? Device Fingerprinting for Cyber-Physical Systems," *Proc. Network and Distributed Security Symp.*, 2016; [www.internetsociety.org/sites/default/files/blogs-media/who-control-your-control-system-device-fingerprinting-cyber-physical-systems.pdf](http://www.internetsociety.org/sites/default/files/blogs-media/who-control-your-control-system-device-fingerprinting-cyber-physical-systems.pdf).
9. Y. Shoukry et al., "Py-CRA: Physical Challenge-Response Authentication for Active Sensors Under Spoofing Attacks," *Proc. 22nd ACM Sigsac Conf. Computer and Comm. Security*, 2015, pp. 1004–1015.
10. D. Hadziosmanovic et al., "Through the Eye of the PLC: Semantic Security Monitoring for Industrial Processes," *Proc. 30th Ann. Computer Security Applications Conf.*, 2014; [www.icir.org/robin/papers/acsac14-ics.pdf](http://www.icir.org/robin/papers/acsac14-ics.pdf).
11. S. Meliopoulos et al., "Distributed Dynamic State Estimation: Fundamental Building Block for the Smart Grid," *Proc. 2015 IEEE Power Energy Society General Meeting*, 2015; doi:10.1109/PESGM.2015.7285790.
12. S. Choi et al., "Feasibility Study: Autonomous State Estimation in Distribution Systems," *IEEE Trans. Power Systems*, vol. 26, no. 4, 2011, pp. 2109–2117.

**Samuel Litchfield** is a master's student in the School of Electrical and Computer Engineering at the Georgia Institute of Technology, and a member of the Communications Assurance and Performance (CAP) group. His research interests include network and control systems' cybersecurity. Litchfield has a BS in computer engineering from the Georgia Institute of Technology. Contact him at [slitchfield3@gatech.edu](mailto:slitchfield3@gatech.edu).

**David Formby** is a PhD student in the School of Electrical and Computer Engineering at the Georgia Institute of Technology, and a member of the CAP group. His research primarily focuses on network security for industrial control system networks. Formby has an MS in electrical and computer engineering from the Georgia Institute of Technology. Contact him at [djformby@gatech.edu](mailto:djformby@gatech.edu).



**Jonathan Rogers** is an assistant professor in the Woodruff School of Mechanical Engineering at the Georgia Institute of Technology, and the director of the Intelligent Robotics and Emergent Automation Lab. His research interests include nonlinear dynamics, control, and state estimation with a focus on modeling and simulation of multibody systems and complex dynamics. Rogers has a PhD in aerospace engineering from the Georgia Institute of Technology. Contact him at [jonathan.rogers@me.gatech.edu](mailto:jonathan.rogers@me.gatech.edu).

**Sakis Meliopoulos** holds the Georgia Power Distinguished Professorship in the School of Electrical and Computer Engineering at the Georgia Institute of Technology. His research interests include power system protection, automation, grounding, harmonics, and reliability assessment of power systems. Meliopoulos has a PhD in electrical and computer engineering from the Georgia Institute of Technology. He's a recipient of the IEEE Richard Kaufman Award and the George Montefiore Award, Belgium. He's also a Fellow of IEEE and a member of Sigma Xi. Contact him at [sakis.m@gatech.edu](mailto:sakis.m@gatech.edu).

**Raheem Beyah** is the Motorola Foundation Professor in the School of Electrical and Computer Engineering at the Georgia Institute of Technology, where he leads the CAP group. His research interests include network security, wireless networks, network traffic characterization and performance, and critical infrastructure security. Beyah has a PhD in electrical and computer engineering from the Georgia Institute of Technology. He's a recipient of the US National Science Foundation CAREER award and was selected for DARPA's 2010 Computer Science Study Panel. He's also a senior member of ACM and IEEE. Contact him at [rbeyah@ece.gatech.edu](mailto:rbeyah@ece.gatech.edu).



Selected CS articles and columns are also available for free at <http://ComputingNow.computer.org>.

### Take the CS Library wherever you go!




IEEE Computer Society magazines and Transactions are now available to subscribers in the portable ePub format.

Just download the articles from the IEEE Computer Society Digital Library, and you can read them on any device that supports ePub. For more information, including a list of compatible devices, visit

[www.computer.org/epub](http://www.computer.org/epub)



IEEE  computer society