Re-thinking the Honeypot for Cyber-Physical Systems

Samuel Litchfield[1], David Formby[1], Jonathan Rogers[2], Sakis Meliopoulos[1], Raheem Beyah[1]

[1]School of Electrical and Computer Engineering, Georgia Institute of Technology

[2]School of Mechanical Engineering, Georgia Institute of Technology

Abstract

Honeypots derive much of their value from their ability to fool attackers into believing they are authentic machines. Current Cyber-Physical Systems (CPS) honeypots fail to sufficiently capture and simulate behavior that is necessary to project this authenticity. In response, the proposed framework, HoneyPhy, was developed for CPS honeypots that takes into account both behavior originating in the CPS process and the devices that make up the CPS itself. We implemented a proof of concept for this framework, and showed that it is possible to simulate these behaviors in real-time. Using HoneyPhy, it will be possible to construct honeypots for complex CPS.

Keywords: Honeypot, Honeynet, Cyber Physical Systems, CPS Security, CPS Modeling, HoneyPhy

Introduction

Since the early 1990s, the idea of entrapping and deceiving computer attackers in order to study their behavior and misdirect them has been used with great success in the computer security field. This practice, traditionally done through a deliberately unused computing system configured to emulate critical resources, called a Honeypot, has revealed many different attacker strategies, allowed researchers to gather malware binaries, and kept more critical computing resources safe. As networking evolved, many honeypots were linked together to form Honeynets, in order to emulate full deployed networks. As attackers learned of honeypots, they improved their techniques to detect whether a resource is being faked, and this is now the primary means of defeating honeynets. If an attacker can realize that a resource is being faked, he can move on to more critical resources, or feed the defender false information in turn.

As the Cyber-Physical Systems (CPS) space grows and becomes increasingly networked, attackers have been more interested in compromising the resources controlling these CPS. Honeypots/Honeynets 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 cyber-physical systems.

Background: Traditional Honeypots

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

Low Interaction

Low interaction 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. Low interaction honeypots get their name from the low level of interaction available between the attacker and the machine. Consequently, low interaction honeypots can behave in unexpected fashions when they encounter unexpected behavior. This unexpected behavior can also alert the attacker to the fact that the machine he is interacting with is a honeypot, which will likely cause the attacker to either disconnect or continue the interaction with the intent to mislead. Additionally, because attackers are limited in how they can interact with the machine, the information that can be gained from the honeypot is also limited. Both of these cases severely limit the usefulness of a low interaction honeypot, and motivated the movement to high interaction honeypots in their place. One very well-known example of a low interaction honeypot is HoneyD [1], which can be configured to emulate a variety of different OS, offering a variety of services.

High Interaction

High interaction honeypots, in contrast, do not emulate. They are real resources, instrumented to log an attacker's behavior, and are deployed to be unused for any other purpose. This allows the attacker to interact with real operating systems and applications. 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.

High Interaction honeypots have gained popularity since virtualization has become more prevalent. Virtualized environments allow many victim machines to be hosted on the same physical resource, and the networking conditions to be tightly controlled.

How should a CPS honeypot fit 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 it. Deploying a copy of an entire CPS with the express purpose of being compromised exposes the same safety risks as the original system, and imposes large costs. A pure high interaction honeypot can deployed for a single component or small group of components 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 below, is to create a *hybrid-interaction honeypot*, where real devices (e.g., programmable logic controllers (PLCs), intelligent electronic devices (IEDs), and remote terminal units (RTUs)) and interfaces interact with process and device simulations that can effectively fully replicate the behavior of the CPS process.

Background: CPS Honeypots

Since 2004, a variety of CPS targeted honeypots have been released and deployed.

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 [2]. It leveraged HoneyD [1], Arpd, Snort, and Tripwire to emulate many hosts on a network. Specifically, it aimed at emulating FTP, HTTP, Telnet, and Modbus for a Schneider PLC, and FTP, HTTP, SNMP, and S7comm for a Siemens PLC. The honeypot makes no attempt to simulate process behavior. The project is no longer maintained.

Released shortly after, Digital Bond's Honeynet [3] has 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 [4] designed to separate the target machine from production networks and filter outgoing traffic.

Gaspot, presented at Blackhat 2015 [5], was based on research done at TrendMicro. Motivated by attacks observed on gas station control devices, the low interaction honeypot simulates basic services provided by these devices, and logs all interactions. The honeypot is relatively simple, and responds to queries with randomized values within plausible ranges. After deployment in a variety of countries, attacker interactions and origins were analyzed.

Conpot is an actively maintained "low interactive server side Industrial Control Systems honeypot designed to be easy to deploy, modify and extend" [6]. While inherently extensible, Conpot is not aimed at modeling either processes or devices. System definition is done through xml files, and protocol emulation is done using Python. Out of the box examples simulate device memory, leveraging the Modbus_tk library.

CryPLH, the Crysys PLC Honeypot [7] is an actively developed low-interaction honeypot designed to emulate a Siemens Simatic 300 PLC. It simulates the exposed HTTP, HTTPS, SNMP, and Siemens SIMATIC STEP7 (carried out over the ISOTSAP protocol) configuration interfaces on a minimal Ubuntu Linux VM, and uses a central configuration file to simplify and end user's configuration. The HTTP/S and ISOTSAP interfaces both have logins where no username/password combination will successfully log in, and the visible web portal does not change to reflect the PLC's environment.

Why Existing CPS Honeypots Are Not Sufficient

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 like Nmap and p0f look for. However, CPS honeypots should provide auxiliary information arising from the attached physical system. This auxiliary information is both the ability to compare the moment to moment state of the CPS for consistency (i.e., 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 if they are 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 like 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 can be seen to slowly rise in response. Imagine that an attacker 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 does not respond to the heating, an attacker will see temperatures over time that do not reflect the activation of the heater. Alternately, the temperature sensor could instantly show the final temperature, which would completely neglect the physics of the system. In either scenario, the attacker

knows the system he is interacting with is either faulty, or does not control the system it claims to. Accordingly, they are likely to not continue interacting with the system, and the honeypot loses utility.

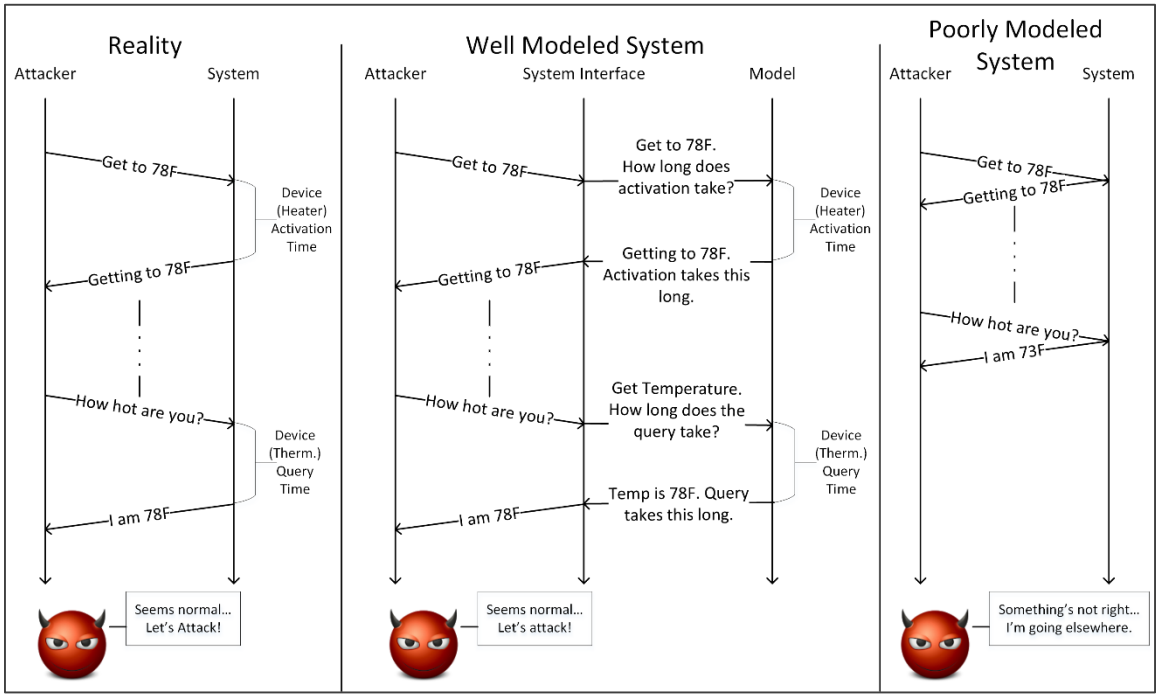


Figure 1 Outcomes of an attacker interacting with different levels of simulation

A similar example can illustrate the need to simulate the physical/mechanical delay of a device (i.e., 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 some amount of time, determined by the electro-mechanical characteristics of the relay [8]. In some devices, this delay can be on the order of milliseconds. If, as is the case in many other kinds of CPS, the thermostat is instrumented to confirm the state change, this 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. This is illustrated in Figure 1. 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 does not 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 is 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 history, or that the honeypot was tested by an associated tool by the honeypot's creator.

Table 1 Modeling Capabilities of Existing CPS Honeypots

	SCADA HoneyNet	DigitalBond Honeynet	GasPot	Conpot	CryPLH

Human Interaction	Proven	Proven	Proved through experimental results	Proven	"Mostly Indistinguishable" [7]
Network Stack Emulation	Honeyd provides stack emulation	Capable of instrumenting real PLC, so 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

Our Vision for Future CPS Honeypots

Our vision for future CPS honeypots addresses the limitations of current CPS honeypots by providing an extensible framework, HoneyPhy: A Physics-aware honeypot framework, for accounting 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 is the well understood portion, and the one all previous CPS honeypots aimed to address. 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 behavior of the physical system. This primarily ensures that physical parameters, when queried, behave in a way consistent with attacker expectations. Without this, future attackers could conduct simple tests to check the authenticity of the machine or machines they are interacting with. 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. Returning to the HVAC example, these delays would originate from the time it takes for the electromechanical relays to physically open or close, energizing or de-energizing the fan, heater, or compressor to move the system to the desired state. 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 box modeling or black box modeling.

Black box modeling is the method of generating a model for the behavior of the device 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, since it is based on empirical measurements. However, attackers will not always have physical access to a target device type to make empirical measurements on to form the black box

model. In this scenario intelligent attackers can then resort to white box modeling to generate an estimate of the device behavior and still make educated guesses about whether the device is a honeypot or a true target. In our previous work, we show that not only is the construction of these models feasible, the models are convincing [8].

Proposed New CPS Honeypot Framework

To satisfy this vision, HoneyPhy, a new CPS hybrid-interaction honeypot framework is proposed. The new framework is composed of three major components: the Internet Interface(s) Module, the Process Model(s) Module, and the Device Model(s) Module. Each module's contents, permissions interfaces, controllable and sensible process variables, and metadata are configured by a central xml file. A framework overview can be seen in Fig. 2.

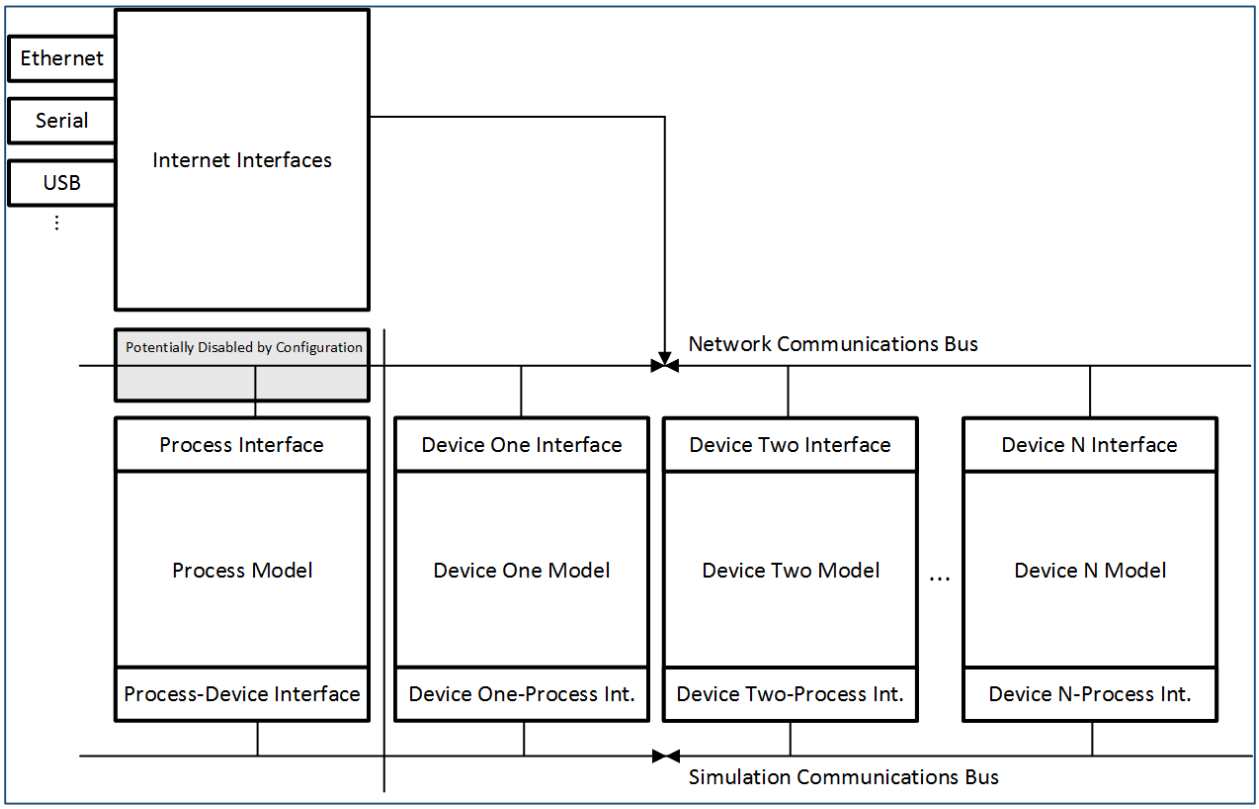


Figure 2 Proposed CPS Honeypot Architecture

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 outgoing packets to reflect the network fingerprint for each device and modifying them if necessary. The interfaces that are exposed to the Internet are specified through the central xml configuration file.

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 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 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 [9, 10].

The process variables that can be sensed or acted upon, along with individual device model permissions, are specified in the central xml configuration file.

Device Model(s) Module

The Device Models encompass all devices found within a CPS, from PLCs to relays. 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.

For each model, the central configuration file specifies metadata such as manufacturer and part number, as well as process parameters the device can sense and act upon. This ensures that devices only act upon appropriate physical quantities.

Inter-module Communication

While our framework does not 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/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.

Putting the Pieces Together

We propose this framework in order to provide a means to identify the key components of a process, identify how those components map to models, and provide a common language to configure and interconnect those models. In the future, process models could be simulations in LabView, communicating with physical PLCs located behind a production HMI. Existing honeypot technology such as HoneyD could be configured to expose realistic Internet interfaces for black box device models communicating to a process model that is empirically generated (as will be shown in the next section).

Physical boundaries do not have to dictate the boundaries of the models either. For example, substation protection relays and their attached physical relays could be modeled as a single unit. The Internet interface unit could be implemented within the process model. The most logical implementation is left up to the honeypot creator, but if there exists a common way to interface these different levels and types of simulations, constructing CPS honeypots will become much more accessible and possible for complex systems.

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 honeypot technology as the system interface.

Physical System Architecture

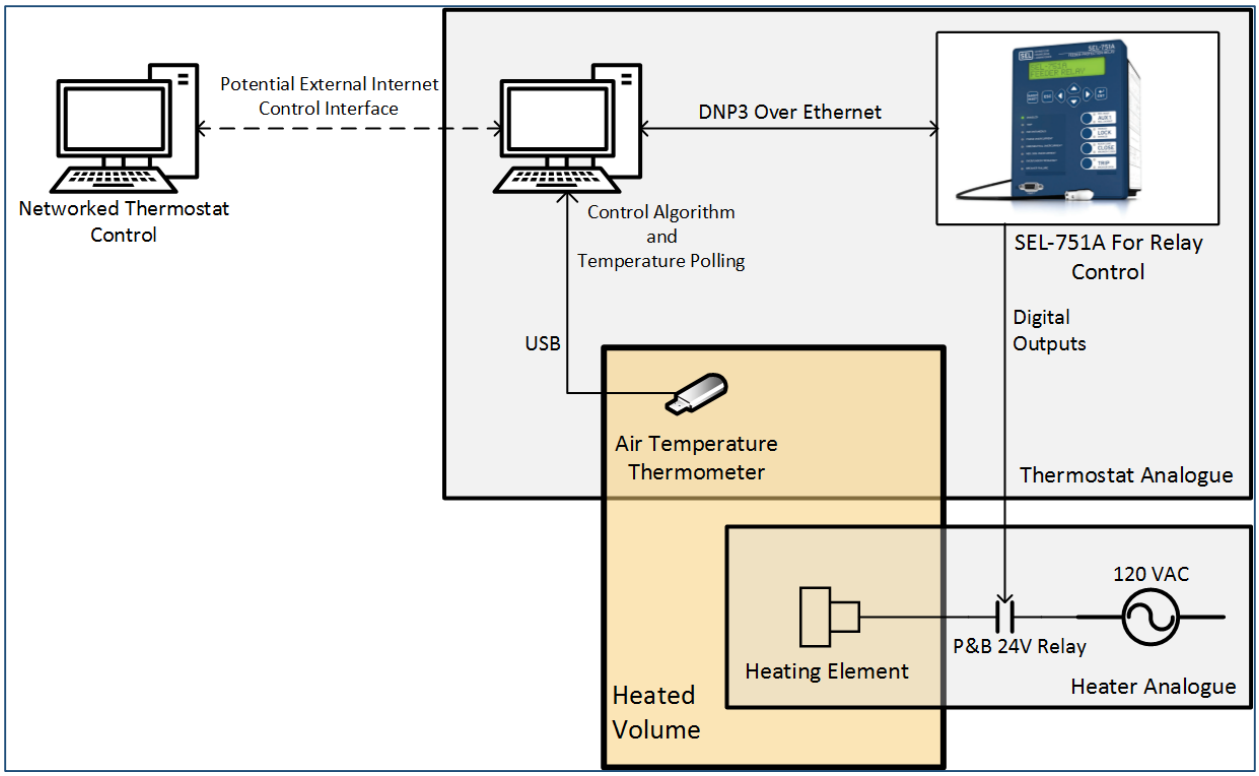


Figure 3 Logical Architecture of the Proof of Concept HVAC System

The simple heating 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. The logical architecture of this system is presented in Fig. 3.

The thermostat analogue labeled in Fig. 3 consists partially of a personal 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 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 (Potter and Brumfield (P&B) 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, as labeled in Fig. 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 an external network interface of the PC.

Models and Simulation

In order to simulate this system, we developed an analytic model of how the enclosed volume was heated by the heating element and was cooled by the environment. 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 where T_{bulb} , T_{air} and T_{env} represent the temperatures in degrees Celsius of the bulb, air, and environment respectively, t indicates time, and S_{heater} represents the state of the heater. As the model looped through the simulation, the time between the last state update and the current state update are used to calculate the update in observed temperature for both the internal heater and the air temperature. The specific coefficients in the equations resulted from creating the general form of the equations from Newton's Law of Cooling, then fitting the resulting equations to the observed data. This analytic model was used for the process model portion of the simulation.

$$Bulb\ Temperature\ (℃) = Bulb\ Temperature + \Delta t \cdot [(Heater\ State) \cdot (0.00775 \cdot Bulb\ Temperature^{1.04}) - 0.00084 \cdot (Bulb\ Temperature - Air\ Temperature)^{1.48775}]$$

$$Air\ Temperature\ (℃) = Air\ Temperature + \ \Delta t \cdot [0.00084 \cdot (Bulb\ Temperature - Air\ Temperature)^{1.085} - 0.00041 \cdot (Air\ Temperature - Environment\ Temperature)^{1.225}]$$

Fig. 4 shows an observed heating/cooling curve from the physical system, and a similar curve generated by the process model for the same control actions. This curve occurs above room temperature, energizing the heater at the beginning of the data set at 28.5° C and turning it off at the vertical read line. Assuming these control commands originate from the attacker, if they energize the heater at t = 0s, at t = 500s they can compare the reported temperature of the system against either their intuition of the system or a separate process model they have created.

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

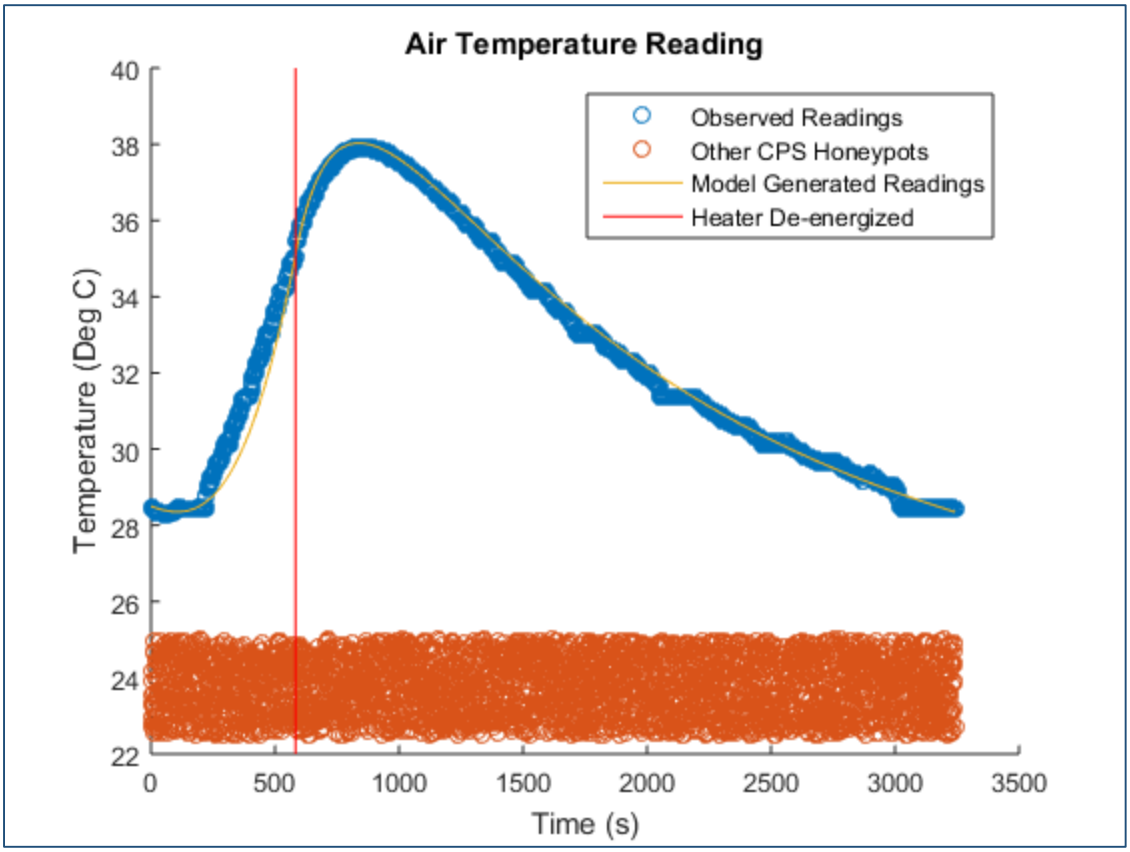


Figure 4 Empirical Heating Results vs. Generated Process Model vs. What Some Other CPS Honeypots Responds With

To capture device behavior, we leveraged black box models developed in previous work for the same relay [8]. For our device model, we randomly sampled the black box data gathered. We also developed a white box model for the same relay. The simulation is convincing with both models, as can be seen in Fig. 5, which shows histograms of device operation times from both models. The black box model was generated by empirically observing the device, while the white box was generated by simulating mechanical properties based on specifications. While the standard deviation of both distributions differs slightly due to simplifications made in the white box modeling, the means of both models are clearly similar. The present differences should not be apparent to any shorter term attackers. Machine learning tools in [8], trained using this white box model, successfully differentiated two physical relays with 80% accuracy.

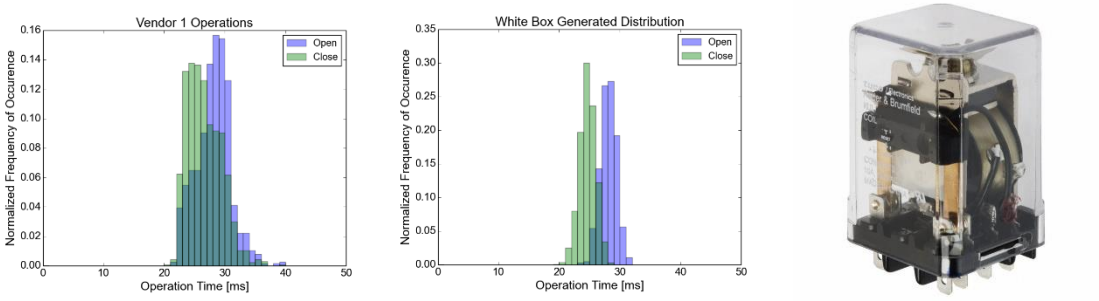


Figure 5 Black and White Box Models for P&B Relay, Seen On Right

The modules used in this HoneyPhy instance were all multithreaded and written in Python, using standard system libraries. When complete, the system is capable of real-time simulation, at a variety of time-step granularities.

Getting Started with HoneyPhy

When approaching construction of a CPS honeypot with HoneyPhy, the first step is to identify what physical devices within the system need to be included in the Device Models Module, and which physical devices, if any, can be combined within a single Device Model. Identify how those models interact with each other and the underlying process, and what each model can sense and control. Additionally, identify which models will need to be accessible from the Internet.

With the identified devices and controls known, one could then apply existing technologies or real devices to simplify the system. For example, existing HMIs can make good Internet Interfaces and require no simulation. If a device needs to be simulated, HoneyD can conform a simulation's outgoing packets to a desired network fingerprint. It is possible that when adding real devices, instrumentation will also need to be added to convert raw control signals into something the relevant process model can implement.

Once it is decided *how* all devices will interact with the process model, the process model itself must be developed while taking into account the necessary interaction with the devices or device models. This can be done in a variety of ways, but explicitly defining input and output signals from the model early will simplify the process.

In the example instance of HoneyPhy provided earlier, physically distinct entities mapped well onto device models. Once the device model boundaries were decided upon, we determined the control actions each device could exert on each other and the system, as well as possible delays encountered in each device's actuation. In this case, the only device that added significant actuation delay was the physical relay, for which a previously developed black box model was used.

With model boundaries and control actions known, it was decided that models would communicate by local network sockets, using a simple string-based message format.

Once all device models, control actions, and communication methods were decided upon, the process model was developed to respond to those control actions, with the chosen communication method.

Extending to Real CPSs

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

One complete example could be to simulate an electric power substation. Substations are prime targets for attackers due to their complexity, the number of devices located within the substation, and the potential to inject false data from the substation. Constructing a honeynet that models an entire substation would require the ability to simulate the effects 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, so this kind of high-

level state estimation from sensor readings is a well-known problem. The model associated with system wide state estimation is simplified and corresponds to balanced operating conditions. This model will not be useful to the HoneyPhy concepts, as it may provide unrealistic responses to attacker's commands. This system level state estimation is then used for both economic functions, such as load forecasting and pricing, as well as security functions, such as managing line congestion and automatic generator control. System level state estimation is traditionally centralized, depending on data returned from individual substations.

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 [11]. This DSE system utilizes a high fidelity dynamic model (3-phase) 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 attacker's commands with the actual timing of system evolution so that the attacker will think he is 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 [8], we have 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 automated generation of the model is described in [12]. 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.

Conclusion

Honeypots derive much of their value from their ability to fool attackers into believing they are authentic machines. Current CPS honeypots fail to sufficiently capture and simulate behavior that is necessary to project this authenticity. In response, the proposed framework, HoneyPhy, was developed for CPS honeypots that takes into account both behavior originating in the CPS process and the devices that make up the CPS itself. We implemented a proof of concept for this framework, and proved it is possible to simulate these behaviors in real-time. Using HoneyPhy, it will be possible to construct honeypots for complex CPS. More details and future updates to HoneyPhy can be found at www.honeyphy.gatech.edu

References

[1] N. Provos, "Developments of the Honeyd Virtual Honeypot," 2008. [Online]. Available: http://honeyd.org/.

[2] V. Pothamsetty and M. Franz, "SCADA HoneyNet Project: Building Honeypots for Industrial Networks," 2005. [Online]. Available: http://scadahoneynet.sourceforge.net/.

[3] DigitalBond, "SCADA Honeynet," 2016. [Online]. Available: http://digitalbond.com/tools/scada-honeynet/.

[4] The Honeynet Project, "Honeywall," 2016. [Online]. Available: https://projects.honeynet.org/honeywall/.

[5] K. Wilhoit and S. Hilt, "The GasPot Experiment: Unexamined Perils in Using Gas-Tank-Monitoring Systems," 2015. [Online]. Available: https://www.trendmicro.com/cloud-content/us/pdfs/security-intelligence/whitepapers/wp_the_gaspot_experiment.pdf.

[6] L. Rist, J. Vestergaard, D. Haslinger and A. Pasquale, "Conpot," 2016. [Online]. Available: http://conpot.org.

[7] I. D. Buza, F. Juhasz, G. Miru, M. Felegyhazi and T. Holczer, "CryPLH: Protecting Smart Energy Systems from Targeted Attacks with a PLC Honeypot," in *Smart Grid Security: Second International Workshop*, 2014.

[8] D. Formby, P. Srinivasan, A. Leonard, J. Rogers and R. Beyah, "Who's in Control of Your Control System? Device Fingerprinting for Cyber-Physical Systems," *NDSS,* February 2016.

[9] Y. Shoukry, P. Martin, Y. Yona, S. Diggavi and M. Srivastava, "Py-CRA: Physical Challenge-Response Authentication for Active Sensors Under Spoofing Attacks," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, New York, NY, 2015.

[10] D. Hadziosmanovic, R. Sommer, E. Zambon and P. H. Hartel, "Through the Eye of the PLC: Semantic Security Monitoring for Industrial Processes," in *Proceedings of the 30th Annual Computer Security Applications Conference*, 2014.

[11] S. Meliopoulos, R. Huang, E. Polymeneas and G. Cokkinides, "Distributed dynamic state estimation: Fundamental building block for the smart grid," in *2015 IEEE Power Energy Society General Meeting*, 2015.

[12] S. Choi, B. Kim, G. J. Cokkinides and S. Meliopoulos, "Feasibility Study: Autonomous State Estimation in Distribution Systems," *IEEE Transactions on Power Systems,* pp. 2109-2117, 2011.

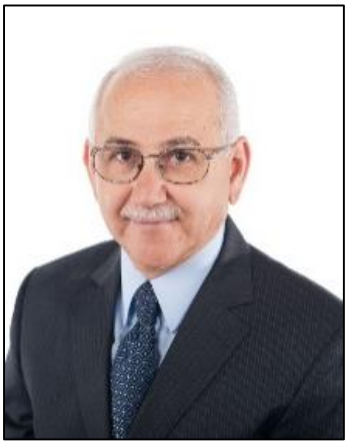
Authors

Samuel Litchfield is a Master's Student in the School of Electrical and Computer Engineering at Georgia Tech, and a member of the Communications Assurance and Performance Group (CAP). His current research interests lie in network and control systems cybersecurity.

David Formby is a PhD student 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.

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. Dr. Rogers research interests are in the areas of nonlinear dynamics, control,

and state estimation with particular expertise in modeling and simulation of multi-body systems and complex dynamics.



A. P. Sakis Meliopoulos (M ’76, SM ’83, F ’93) received the M.E. and E.E. diploma from NTUA, Greece, in 1972; the M.S.E.E. and Ph.D. degrees from GIT in 1974 and 1976, respectively. In 1976. He joined the Faculty of ECE, GIT, and he hold the Georgia Power Distinguished Professorship. He has made significant contributions to power system protection, automation, grounding, harmonics, and reliability assessment of power systems. He is the author of three books, he holds three patents and he has published over 330 technical papers. Recent honors: IEEE Richard Kaufman Award (2005), George Montefiore Award, Belgium (2010). He is the Chairman of the Georgia Tech PRC, a Fellow of the IEEE and a member of Sigma Xi.

Raheem Beyah is the Motorola Foundation Professor in the School of Electrical and Computer Engineering at Georgia Tech where he leads the Communications Assurance and Performance Group (CAP). He 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 AAAS, ASEE, a lifetime member of NSBE, and a senior member of ACM and IEEE.