Model-driven Deception for Control System Environments

William Hofer
Cyber Security Engineer
Pacific Northwest National Laboratory
Richland WA, U.S.A
william.hofer@pnnl.gov

Thomas Edgar
Cyber Security Researcher
Pacific Northwest National Laboratory
Richland WA, U.S.A
thomas.edgar@pnnl.gov

Draguna Vrabie
Electrical Engineer
Pacific Northwest National Laboratory
Richland WA, U.S.A
draguna.vrabie@pnnl.gov

Kathleen Nowak
Mathematician
Pacific Northwest National Laboratory
Richland WA, U.S.A
kathleen.nowak@pnnl.gov

***Abstract*—Deception techniques are a useful defensive mechanism that could be very valuable in control systems where updates and patches can be difficult. However, to deceive a motivated and targeted attacker it is necessary to enhance deception platforms with capabilities to model and simulate physical processes to achieve necessary fidelity. In this paper, attributes of cyber-physical decoys are discussed and a design of a system with these attributes is presented. A boiler model use case is provided to demonstrate how this novel deception capability can be integrated into and used to defend a system.**

***Index Terms*—cyber-physical systems, process modeling, deception, network security**

I. Introduction

Operation technology environments, including energy delivery and oil, gas, and water distribution, have increasingly become targets for cyber attacks. The digitization of the sensing and control and the commoditization of the communication infrastructure has exposed a cyber attack surface to these critical services. The impact of successful attacks on these environments can have costly [14] and physical [6] ramifications. Cyber security solutions that are tailored to and effective for these unique systems is greatly needed.

It is understood that OT and IT systems have different operational and cyber security requirements [19]. The best practice of rolling updates and patch cycles that are now common place in enterprise operations are much more difficult to follow in OT environments due to the high availability requirement and subsequent rigorous and slow testing and validation processes. As such, additional cyber security techniques are needed to provide OT defenders the ability to protect themselves from cyber threats.

Deception frameworks are one such solution. Cyber deception has been a concept that has existed for many years and is historically utilized for honeypots [17] to gather threat intelligence. Honeypot technology includes real, emulated, and/or simulated cyber capabilities to mimic services from operational environments. Collections of honeypot decoys are called honeynets [7]. Honeypots have been extensively used to model general system configurations to ensnare and study threats and their vectors on the internet.

Only more recently have deception techniques been commercially utilized for cyber defense. Capabilities using deception techniques, including honeynets, in coordinated mechanisms for operational cyber defense are called deception platforms [11]. Deception platforms for traditional IT environments lack the integration and coordination with real systems to sufficiently deceive targeted threats. Current approaches to cyber deception within cyber-physical environments model the device independently of the physical process of which it would be tied. Deceptions that are disconnected from physical process and not communicating with the cyber controllers limits the fidelity and its deceptive qualities.

In this paper we contribute a methodology for creating deception decoys that are integrated into real OT systems to make them harder to detect and more appealing as targets. We discuss the deficiencies of current deception platforms for deploying realistic decoys in OT. We present the various models of deception. We then go into depth coverage of the requirements for an integrated OT decoy. Finally, we present a prototype implementation in a boiler system use case to demonstrate these concepts.

II. Related Work

Cyber deception has been an academic concept since the late 1980s [18] and was first developed into a security tool by Cohen et. al[3]. Since then a community of researchers have been developing and leveraging honeypot technology extensively for threat intelligence gathering. The honeynet.org [5] project has become a major community organization for collecting honeypot contributions. Extensive research has been done to develop and use honeypot technology within traditional IT environments. The increasing interest in cyber-physical environments has led to more recent research into honeynet technology and use cases for operational environments.

978-1-7281-5092-5/19/$31.00 ©2019 IEEE

The initial effort to develop an OT honeypot was the SCADA Honeynet Project [12]. This effort leveraged the honeyd low interaction honeypot daemon that provides the ability to create simulated hosts with network services on a system. The SCADA Honeynet project extended the honeyd to model a common OT programmable logic controller (PLC). The networking interfaces of common PLC services, including Modbus/TCP with a few supported functions, FTP, Telnet, and web servers, were developed to provide a simulation of a PLC. The objective of this was to provide a honeypot to detect scans looking for PLC devices.

Since the initial SCADA Honeynet project, a few more specific honeypots have been developed. The Conpot project [13] is a docker based low interaction honeypot designed to mimic common industrial communication protocols to appear as process control devices. The GasPot[21] project created from scratch a special purpose python based honeypot to emulate a Veeder Root Guardian AST. Both MiniCPS[1] and HoneyPhy [8] are research into frameworks to enable model backed ICS honeypots. MiniCPS utilizes the miniNET network emulator with an integrated database to enable the definition and execution of physical process algorithms to feed data to honeypots. HoneyPhy defines an architecture where honeypots can query a simulator to respond with realistic data.

With the strong progression of cloud technologies, more recently there has been a resurgence of using deception techniques to provide cyber network defense [16]. Distributed deception platforms provide centralized control of resources to define, deploy, and manage decoys in operational environments for threat detection and defense [11]. A key defining difference between distributed deception platforms and honeypots is that they are meant to be integrated into operational systems for defending against active threats and attacks where honeypots are traditionally used for threat intelligence gathering. In the rest of this paper we present our research into turning concepts from ICS honeypots into a distributed deception platform for integration of decoys into and defense of operational technology systems.

III. Deception with a Physical Process

A cyber-physical system, by definition, integrates cyber components to monitor and control physical processes. The cyber components are, in general, controllers, sensors, communication infrastructure, and the software applications that use the data from the physical process to perform different functions such as optimization [22], delegation of human agency[20], or safe process management[4]. The physical systems include the mechanisms that interact with the physical world like actuators and instrumentation that are driven by physical processes.

Due to the strong integration of real world physics, deception platforms must operate differently than traditional IT deceptions. For instance, turning off a valve will be detected downstream by other sensors because the flow will reduce and stop. Additionally, controllers and applications leverage data from sensors to send control commands to each other. A believable deception must be integrated with the system to project the effects of events. An attack will likely attempt to control the physical process in a negative manner. To make the attacker believe they are achieving their objective, it must predict the effects of these actions, to a reasonable degree.

Traditional cyber deception platforms integrate into a system through the generation of realistic looking data, access/identity control accounts, and/or honey tokens and the deployment of decoy services such as web applications, file shares, or remote access. Each of these components can be made to appear as part of the system with realistic banners, directory, and filenames related to the business or corporate branding. This data can be distributed onto real workstations and servers to be found by attackers and direct them to the decoy services. These capabilities are useful and could be used within cyber-physical deceptions, but they alone are insufficient.

The threat model driving this research is an adversary knowledgeable in cyber-physical systems with a targeted objective to effect the operation of the physical process. Under this threat model the attacker will be aware of, and searching for, specific categories of equipment to understand how they interface with the physical process and how they can be controlled or manipulated. The objective of this research is to delay the attack from successful completion while increasing the probability of detecting the threats presence and actions. Achieving these goals under this threat model requires sufficient fidelity.

To provide sufficient fidelity there are additional requirements that must be met by cyber-physical deception platforms. First, the deception platform must provide the ability to simulate a model of the physics of a real system process. This includes supporting the ability to generate realistic variable data from decoy device responses. Second, a deception platform must provide the ability to define new devices and connect them to the physics model being simulated. We will go into more depth about the attributes of these decoy devices later. Finally, the decoys should appear as tempting, easy to exploit targets, that are part of the real operating cyber-physical system. We propose that these three requirements are necessary for an adequate deception of cyber-physical decoys in a real system.

IV. Types of Deception

There are multiple ways in which a model driven deception can be deployed. These include cloning, copies, and integrated deceptions. Each type of deception is useful under different circumstances. Their use and utility are a function of the location of the deception in relation to the real system and what type of threat is being countered. A brief overview of each type of deception and microgrid examples of how they could be utilized are provided. The focus of the discussion and contributions in this paper is largely centered on integrated deceptions.

978-1-7281-5092-5/19/$31.00 ©2019 IEEE

A. Clone

A clone deception is when an exact replica is presented as the real system to deceive an attacker that they are interacting with the real system components. This type of deception traps the attacker into a fictional world that is directly related to the real system. The model for this type of deception can be driven directly from observed data of the real system. Only upon control or other altering interactions by the attacker is the projection of effect necessary. In a microgrid, a deceptive clone could be developed that mimics an entire building HVAC cyber-physical system as shown in "Fig. 1". The clone is posing as the adjacent building in this deception. In this type of deception there would be some mechanism, like a firewall, IPS, or VPN, that would determine when to send connections to the cloned system instead of the real system.

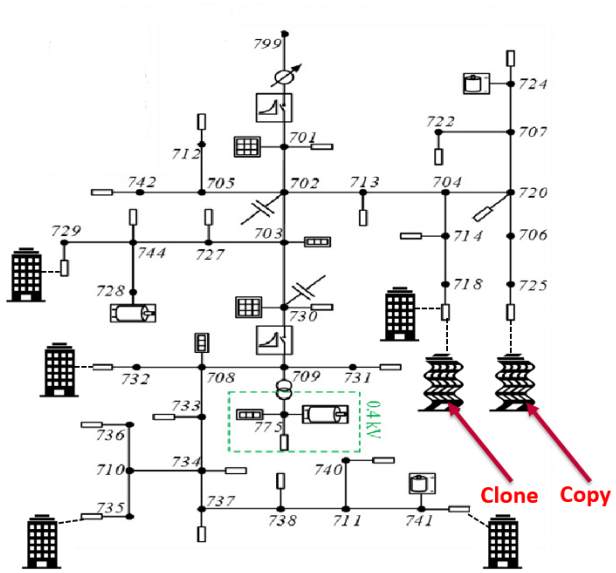

Fig. 1. Deception Logical Model

B. Copy

A copy deception, also shown in "Fig. 1", is similar to the clone where replicas of a real system are presented. However, the difference with this deception is that one or more replicas are presented within the same network perspective as the real system. This type of deception makes it appear to observers that there are multiple running cyber-physical systems and provides an obfuscation style defense where an attacker must determine which system is the real system. Each decoy copy interacts with a simulation of the real system model and can be driven at offset times or with some form of data fuzzing such that the data does not appear to be exactly the same, further obfuscating which system is the correct system. Each copy can respond and react to interactions independently. This type of deception could be utilized in coordination with moving target defense techniques like IP address hopping [15] to further confuse an attacker. An example of this type of deception in our microgrid example would be to create a copy, or multiple copies of a building on the microgrid and deploy deceptive ones in different locations. The copied building could have the same components as the original or differences to add to the confusion.

C. Integrated Decoys

Integrated deceptions places newly conceived decoys within the real system where the decoy modeled data is defined such that it logically relates to real data within the system. For example, in a chemical process, a decoy could be made that controls a fictional valve downstream that controls the flow to a decoy sensor. The modeled values output from these decoys are extrapolated or derived as some function of the real system values. Through these extrapolations it appears that the decoy produces data related through physics to the real system and thereby providing a high fidelity target of value to an attacker that has already bypassed other defenses and infiltrated the OT network. An example of constructed integrated decoys is provided in the use case presented in section VIII.

To further increase the fidelity of integrated decoys honeyshills can be configured. A honeyshill is are real systems that are configured to interact with decoys to make them further appear as part of the real cyber-physical system. Most components of cyber-physical systems are embedded sensors and controllers that continually interact to monitor and control a physical process. As such, the devices are configured to for machine-to-machine interaction. A decoy that is quietly sitting on a network appears either as a decoy or a non-valuable test system. In addition, existing threat tactics including searching data on systems like human-machine interfaces (HMIs) and data historians to locate devices to target /citehacker-machine-interface. Configuring honeyshills like real remote terminal units (RTUs) or HMIs to communicate with decoys the same fashion as real components completes the fidelity picture for decoys.

V. Distributed Decoy Architecture

To enable the definition and deployment of a deceptive campaign across, and integrated within, a real system, it is necessary to utilize a distributed system with centralized control. OT systems are often distributed across multiple networks or network segments where each contains a part of the physical process operation. Decoys could potentially be deployed to any of those networks and therefore requires distributed platforms for running decoys.

"Fig. 2" depicts an architecture of a system that would allow the central deployment and management of decoys across a distributed OT environment. A central server hosts the model and simulation of the physical process to drive the behavior and interaction of the decoys in a deception campaign. This central server provides the capability to define decoy configurations and allocations across the distributed execution platforms. Logging and alerting for the entire system would be collected and reported through this central control host.

978-1-7281-5092-5/19/$31.00 ©2019 IEEE

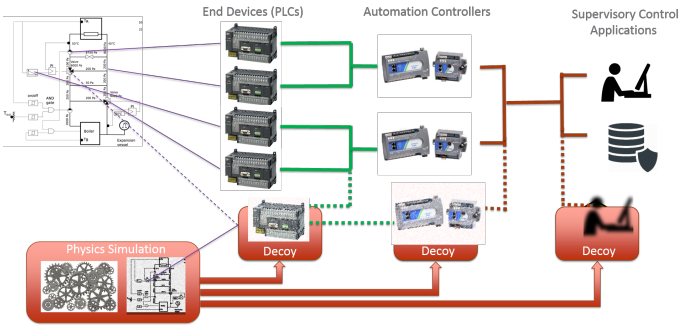

Fig. 2. Distributed Decoy Architecture

One or more distributed execution platforms are deployed across all networks where decoys would be desired for detection and defense. Ideally, a special security network segment would be created to connect the execution platforms and the central configuration host, such that the deception platform communication isn't easily observable breaking the camouflage of the decoys. Each execution platform locally caches up-to-date simulation data to support decoy execution, definitions and configurations of decoys, and a reporting capability to inform the central server of any interactions. The distributed platform is collocated in any network segments where deception is desired. The platform bridges decoy devices onto the network segment such that they are visible to an attacker via scans and device discovery tools.

VI. Attributes of an OT Decoy

The three main attributes to our OT Decoys include protocol, variables, and logic. Each of these attributes defines the characteristics and behavior of the decoy acting as a device. Breaking down each decoy into these three attributes allows for an abstraction of complex system architectures. A device in a real environment would speak one or more network protocols, control or monitor some set of variables, and perform actions based on a set of logic. To be effective, a decoy should also do the same. Each of these attributes are user defined to construct decoys that appear as a realistic device in their system.

OT devices speak a multitude of different protocols depending on the domain in which they are performing control operations, a small sample of which includes DNP3, BACnet, and Modbus protocols. For instance, a decoy in a deception based on building automation and control would most likely communicate via BACnet in a way that relates to the physical process. Devices also support traditional protocols, like HTTP, SSH, Telnet, and FTP, for configuration management. A set of protocols should be associated with each decoy to provide interfaces for threat interaction.

Decoys should expose through protocols one or more variables that correspond to the physical process they are emulating. Decoy variables can include things like temperature, voltage, or flow rate. Any number of variables can be assigned to a decoy and the variables can be newly defined from extrapolations of other existing variables learned from the physical system. A decoy variable might be an input or an output. For instance, if we want a decoy that monitors pressure and triggers a control valve, we could extrapolate the pressure value by using two input variables from the values for temperature and flow rate from real sensors. A variable that is an output of one decoy device can be read by another device using the network protocol associated with those devices. This interaction generates traffic on the network and adds to the realism of the deception.

The final attribute for an OT Decoy is a set of logic that directs the operation of the device. A simple example of such logic would be the decision to turn on or off a fan in an HVAC system to allow cool air to flow into a room. A decoy device could act as the controller that monitors the rooms air temperature and triggers a fan to rotate if the temperature reaches a certain threshold. The logic of a device can be as simple as an if-then statement. For example, "if temp ¡ X then turn off fan". The logic of decoy devices effects the operation of the simulation which in turn drives the values of the decoy.

VII. Prototype Implementation

To implement the intended Distributed Decoy Architecture, the MiniCPS [1] framework has been leveraged. Nodes in MiniCPS are Mininet containers with an assigned module of Python executing their logic. We extended MiniCPS with additional functionality to prototype the discussed deception platform.

The underlying Mininet command line interface was expanded to allow for dynamic decoy, variable, and logic creation. In addition, MiniCPS was augmented to support the BACnet protocol via the bacpypes Python library. Also, a dynamic abstract class was created for decoys that executes user defined logic. This abstract class runs the logic associated with a decoy device, stored in a database, and queries and updates the database housing the present value of all of the variables in the deception. The abstract decoy Python code also inherits a send and receive functionality from the chosen protocol for the decoy.

When a deception campaign is executed, two special features are also started; a virtual network is created that bridges decoys to a user selected network interface and a special container is started for the simulation of the physical process. The virtual network creates the virtual interfaces that provide the networking configuration like MAC and IP address so the system can communicate and interact on the bridged physical network. The simulation container runs the simulated model of the real physical process to generate realistic data for the decoys to use during execution. The simulation container communicates through a database in shared file system space with the decoys which hides it from external observation.

Controllers execute logic to monitor and control a process for safety and optimization. To enable user input of mathematical logic functions in our prototype, we leveraged Sympy [9]. Sympy is a symbolic mathematics Python library. Sympy provides the ability to validate and extract variables from

978-1-7281-5092-5/19/$31.00 ©2019 IEEE

mathematical expressions. Extracted variables can then be filled with data from the simulation and evaluated to calculate new values and determine if decoy control actions should be executed.

VIII. Constructing an OT Deception

In this section a simple boiler system decoy use case is presented. For our use case we used an existing model of a boiler taken from the Modelica Buildings Library [2]. "Fig. 3" is a logical depiction of the boiler and radiator heating a room. The Modelica simulation was used, with four software PLCs, to represent the real physical system. Dissecting the boiler model yields variables relating to temperature measurements, actuator values, and one supervisory input. The variables are listed in "Table I" below. "Fig. 4" is a logical representation of inputs and outputs from the model mapped to each software PLC device.

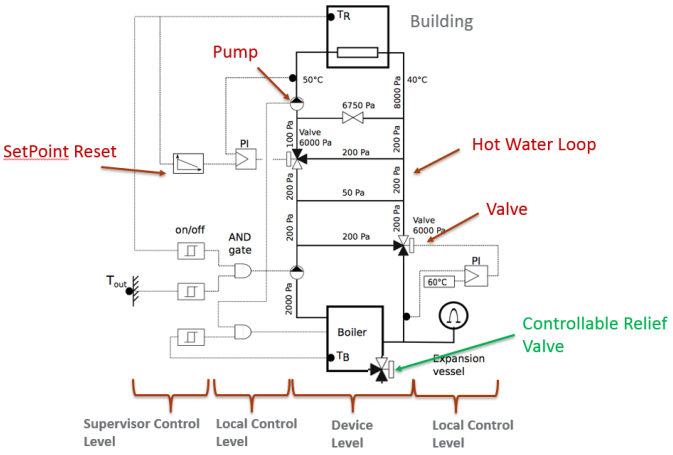

Fig. 3. Logical Diagram of Boiler Model

The first step in constructing the deception is to generate the physics model for simulating the real system. The expected real world approach for generating this model would be to utilize archived data in a data historian or to capture live network traffic and learn in real time. However, for this test case we simulated a year long dataset to train a recurrent neural network (RNN). RNN models were chosen due to their defining feature of drawing information from temporal data produced from input data. The RNN considers data inputs from concurrent connections representing short term memory [10]. The implementation also allows for the examination of broad contextual features of the data, forming long term memory. Training the RNN was done by framing a forecasting problem where given temperature readings and subsequent controller decisions made at previous time steps enables the forecasting of temperature readings at the next time-step. A set of variables, the behavior of the PLCs, and the ability to project values based on system changes are learned in the RNN model. Executing the RNN model predicts the future states of the physical system and updates a database.

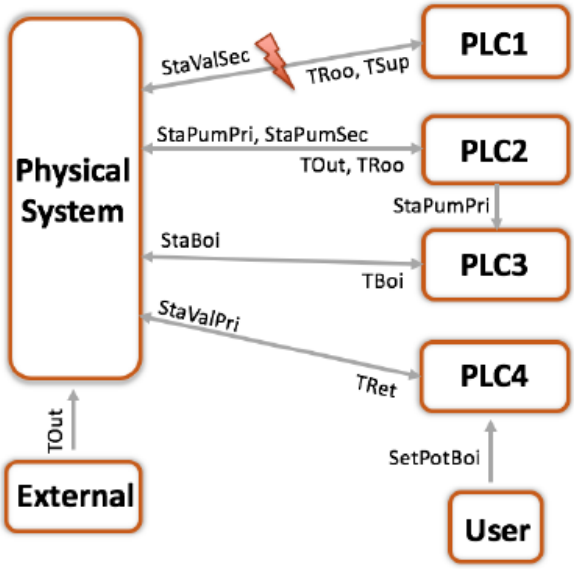

Fig. 4. CPS Variable Relationships

TABLE I
Boiler Model Variables

Variable	Description	Units
TOut	Outdoor air temperature	°K
TRoo	Room air temperature	°K
TSup	Supply water temperature	°K
TRet	Return water temperature	°K
TBoi	Boiler water temperature	°K
StaPumPri	Primary pump flow rate	kg/s
StaPumSec	Secondary pump flow rate	kg/s
StaValPri	Primary valve position	%
StaValSec	Secondary valve position	%
StaBoi	Boiler Status	On/Off
SetPotBoi	Set point for the boiler	°K
BoiOverride	Override switch for the boiler	On/Off

For our use case, we want to add a decoy that is integrated into the system and looks appealing. As such, we selected a decoy safety valve controller that calculates the pressure at certain points in the boiler system, shuts-off the boiler, and open a control valve if it reaches a threshold beyond acceptable values. If an attacker wanted to cause a damaging failure to this boiler system, they would first have to shutoff this safety controller. The placement of our decoy valve is shown with green text in 3.

Creating a decoy for this safety valve constructs a miniCPS container with specified attributes. The first attribute is logic operations to determine whether the boiler should shutdown. For this example, our logic is extrapolated from the boiler model. Decoys need only approximate reality and do not require perfect models of physics. We leverage Gay-Lussac's law of proportional relation of pressure and temperature to approximate the pressure readings of this system where only

978-1-7281-5092-5/19/$31.00 ©2019 IEEE

temperature is available.

For our logic to operate, creation of new decoy variables; boiler pressure, control valve state, and an upper threshold of boiler pressure set point; was necessary. The default values for the control valve is set closed, pressure to 0 (as it will be calculated as the decoy starts), and finally the pressure set point to 30 PSI. Each of these new variables, and the existing boiler override and temperature reading variables, are mapped to the decoy to control the status of the boiler and calculate a temperature based pressure.

A typical boiler systems pressure falls somewhere between 12 and 30 PSI. To approximate the pressure value, the temperature reading is normalized by feature scaling it to a value between 0 and 1. Then, we can multiply by the count of values in the range of PSI readings we want to produce, in this case 19, the count of values between 12 and 30, inclusive. Finally, by adding that value to the base value of 12 PSI our boiler would sit at when cold, we get an extrapolated pressure reading for the boiler. The formula, given that we know the minimum and maximum temperatures produced by the model, for this mathematic operation is:

$$(((Tboi - 308.35)/60.911) * 19) + 12 \quad (1)$$

‘ The next step is to add conditional logic that looks at the output of the equations and makes a decision. The logic for our safety valve decoy is:

```
If PresBoi > 30 PSI, then BoiOverride = 1
```

If the extrapolated pressure is greater than 30 PSI then the boiler is set to off in the simulation which predicts the resultant system behavior and physics; which would be lowered temperature of the water. Using our modified miniCPS CLI we can add conditional logic to the decoy safety valve that compares the calculated pressure to the pressure set point and make a decision on whether or not to trigger the boiler's override switch. With the variables associated and the logic in place, the deception can be executed.

Once we run the decoys we can observe them in this case using a tool called Yet Another BACnet Explorer installed on a host on the same network segment as the decoy valve and the ”real” boiler Modelica simulation. In “Fig. 5”, the x-axis represents the temperature in the boiler and the y-axis represents time. The red line represents the temperature of the boiler in the real system and the blue line is the decoy node getting its reading from the learned model. At around 15:25 we changed the value of the pressure set point in the boiler to 12 PSI, thus causing the logic to turn the boiler off. You can see that the blue line trended downward as if the boiler turned off, while the red line continued its pattern.

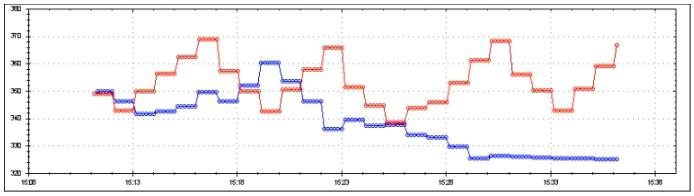

Fig. 5. Boiler Model Vs. Decoy

IX. Conclusion

In order to combat the cyber-physical attacks targeting OT systems which are often slow to patch and upgrade, new and innovative security solutions must be utilized. Deception defense is a perfectly aligned solution that minimally impacts the physical system while providing enhanced detection and attack disruption. Additional fidelity is a necessary component for deception to work in cyber-physical environments, due to the physical process monitoring and control associated with these types of networks. By adding a model of the physical system it is possible to increase the realism of decoy devices. A distributed platform as described in this paper is capable of deploying decoys in multiple network segments and managing their perspective of the physical world. The prototype implementation of this solution and the use case for a boiler model is only one example of how this novel methodology could be applied. Future work could be performed to expand this approach for more test cases and other systems. In addition, there is more research to be done to study and enhance the fidelity of decoys by creating vendor/product specific profiles that include items like protocols supported, ports used, and configuration of register points.

Acknowledgment

The research described in this paper is part of the Agile Adaptive Cyber Initiative at Pacific Northwest National Laboratory. It was conducted under the Laboratory Directed Research and Development Program at PNNL, a multiprogram national laboratory operated by Battelle for the U.S. Department of Energy.

References

[1] Daniele Antonioli and Nils Ole Tippenhauer. “MiniCPS: A Toolkit for Security Research on CPS Networks”. In: *Proceedings of the First ACM Workshop on Cyber-Physical Systems-Security and/or PrivaCy*. CPS-SPC ’15. Denver, Colorado, USA: ACM, 2015, pp. 91–100. ISBN: 978-1-4503-3827-1. DOI: 10.1145/2808705.2808715. URL: http://doi.acm.org/10.1145/2808705.2808715.

[2] The University of California through Lawrence Berkeley National Laboratory. URL: http://simulationresearch.lbl.gov/modelica/.

[3] Fred Cohen et al. *A Framework for Deception*. Tech. rep. 2001. URL: http://all.net/journal/deception/Framework/Framework.html.

[4] *Generator Protection Theory & Application*. https://www.eiseverywhere.com/file_uploads/529211ae18f0798b5a92f1ced9ac3673_WSU_GENTheory_1PageperSlide_160114.pdf. Accessed: 2019-02-15. Pullman, WA, USA.

[5] Wayne Hartmann. *The Honeynet Project*. http://web.archive.org/web/20080207010024/http://www.808multimedia.com/winnt/kernel.htm. Accessed: 2019-3-30.

[6] R M Lee, M J Assante, and T Conway. *Analysis of the Cyber Attack on the Ukrainian Power Grid*. Mar. 2016.

[7] J. G. Levine, J. B. Grizzard, and H. L. Owen. "Using honeynets to protect large enterprise networks". In: *IEEE Security Privacy* 2.6 (Nov. 2004), pp. 73–75. ISSN: 1540-7993. DOI: 10.1109/MSP.2004.115.

[8] S. Litchfield et al. "Rethinking the Honeypot for Cyber-Physical Systems". In: *IEEE Internet Computing* 20.5 (Sept. 2016), pp. 9–17. ISSN: 1089-7801. DOI: 10.1109/MIC.2016.103.

[9] A Meurer et al. *Sympy: symbolic computing in Python*. Jan. 2017. URL: https://doi.org/10.7717/peerj-cs.103.

[10] T. Mikolov et al. "Extensions of recurrent neural network language model". In: *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. May 2011, pp. 5528–5531. DOI: 10.1109/ICASSP.2011.5947611.

[11] Lawrence Pingree. *Competitive Landscape: Distributed Deception Platforms*. Tech. rep. 2015. URL: https://www.gartner.com/en/documents/3402317.

[12] Venkat Pothamsetty and Matthew Franz. *SCADA HoneyNet Project: Building Honeypots for Industrial Networks*. URL: http://scadahoneynet.sourceforge.net/.

[13] Lukas Rist et al. *Conpot: ICS/SCADA Honeypot*. http://conpot.org/. Accessed: 2018-09-19.

[14] Wolfgang Schwab and Mathieu Poujal. *The State of Industrial Cybersecurity 2018*. June 2018.

[15] M. Sifalakis, S. Schmid, and D. Hutchison. "Network address hopping: a mechanism to enhance data protection for packet communications". In: *IEEE International Conference on Communications, 2005. ICC 2005. 2005*. Vol. 3. May 2005, 1518–1523 Vol. 3. DOI: 10.1109/ICC.2005.1494598.

[16] Paulo Simoes et al. "On the use of Honeypots for Detecting Cyber Attacks on Industrial Control Networks". In: July 2013.

[17] Lance Spitzner. *Honeypots: tracking hackers*. Addison-Wesley, 2003. URL: http://www.it-docs.net/ddata/792.pdf.

[18] Clifford Stoll. *The Cuckoo's Egg: Tracking a Spy Through the Maze of Computer Espionage*. New York, NY, USA: Doubleday, 1989. ISBN: 0-385-24946-2.

[19] Keith A. Stouffer et al. *SP 800-82 Rev 2. Guide to Industrial Control Systems (ICS) Security: Supervisory Control and Data Acquisition (SCADA) Systems, Distributed Control Systems (DCS), and Other Control System Configurations Such As Programmable Logic Controllers (PLC)*. Tech. rep. Gaithersburg, MD, United States, 2011.

[20] Kris Subbarao et al. *Transactive Control and Coordination of Distributed Assets for Ancillary Services*. Tech. rep. 2013. URL: https://www.pnnl.gov/main/publications/external/technical_reports/PNNL-22942.pdf.

[21] Kyle Wilhoit and Stephen Hilt. *The GasPot Experiment: Unexamined Perils in Using Gas-Tank-Monitoring Systems*. http://conpot.org/. 2015.

[22] O. Younis and N. Moayeri. "Employing Cyber-Physical Systems: Dynamic Traffic Light Control at Road Intersections". In: *IEEE Internet of Things Journal* 4.6 (Dec. 2017), pp. 2286–2296. ISSN: 2327-4662. DOI: 10.1109/JIOT.2017.2765243.

978-1-7281-5092-5/19/$31.00 ©2019 IEEE