Chapter 11

INTEGRATING SIMULATED PHYSICS AND DEVICE VIRTUALIZATION IN CONTROL SYSTEM TESTBEDS

Owen Redwood, Jason Reynolds and Mike Burmester

Abstract Malware and forensic analyses of embedded cyber-physical systems are tedious, manual processes that testbeds are commonly not designed to support. Additionally, attesting the physics impact of embedded cyber-physical system malware has no formal methodologies and is currently an art. This chapter describes a novel testbed design methodology that integrates virtualized embedded industrial control systems and physics simulators, thereby supporting malware and forensic analyses of embedded cyber-physical systems without risks. Unlike existing hardware-based testbeds, the resulting soft industrial control system testbeds are portable, distributable and expandable by design. However, embedded system virtualization is non-trivial, especially at the firmware level, and solutions vary widely depending on the embedded system architectures and operating systems. This chapter discusses how the proposed methodology overcomes the challenges to virtualizing embedded systems and explores the benefits via a proof-of-concept implementation involving a Siemens MJ-XL variable step voltage regulator control panel.

Keywords: Cyber-physical systems, control systems, testbeds, virtualization

1. Introduction

Cyber-physical systems are computational systems that monitor and control physical systems; they encompass control systems, sensor-based systems, autonomous systems, robotic systems as well as higher-level supervisory, control and human-in-the-loop systems. These complex, specialized and diverse systems exist at the core of industrial control systems, critical infrastructure assets, operational technology networks and other utility networks. Hardware-based cyber-physical system testbeds often are expensive to design and maintain, especially in the case of critical infrastructure assets. This primarily limits

M. Rice and S. Shenoi (Eds.): Critical Infrastructure Protection X, IFIP AICT 485, pp. 185–202, 2016.
DOI: 10.1007/978-3-319-48737-3_11

entire classes of vulnerability analyses, malware analyses, forensics and other defensive research.

Analyzing memory corruption vulnerabilities in an embedded industrial control system often runs the risk of damaging or destroying (i.e., "bricking") the testbed hardware. Additionally, there are several techniques that attackers can use to brick, disable or destroy embedded industrial control systems. These factors limit the ability of industrial control system testbeds to support malware analysis and forensics research on embedded systems when the malware utilizes such techniques. In turn, this limits the defender's ability to analyze the physics impact of sabotage-based embedded cyber-physical system malware. These defensive challenges are becoming very significant as attacker sophistication increases. Indeed, attacker sophistication is growing rapidly as a result of the availability of commercial penetration testing "exploit packs" from vendors such as Core Impact, Metasploit and GLEG.

Despite decades of advocacy by experts and government authorities, "air-gaps" are not utilized properly, supported by vendors or relied upon to safeguard industrial control systems and operational technology and utility networks. In 2011, the Director of the U.S. Department of Homeland Security's National Cybersecurity and Communications Integration Center (NCCIC) testified that: "In our experience in conducting hundreds of vulnerability assessments in the private sector, in no case have we ever found the operations network, the SCADA system or energy management system separated from the enterprise network" [16]. This situation is not unique to the U.S. power grid and there is compelling evidence that industrial control system vendors have been intentionally moving away from the traditional airgap advice [2].

Standardized, smart grid automation protocols drive down operational costs and are the new norm; however, this paradigm also drives down the difficulty bar for industrial control system attackers [14]. Even if the protocols and standards were perfect, vendor implementations would naturally have bugs and vulnerabilities [9]. Attackers are also targeting utility networks through Trojanized signed vendor firmware updates by directly targeting the supply chain [11].

As cyber-physical systems become increasingly interconnected and linked to the Internet and industrial control system operators and vendors adopt traditional information technology advancements for automation, it is imperative that the defensive benefits of information technology somehow be leveraged to secure cyber-physical systems. Virtualization, for instance, has given way to malware sandboxing, dynamic malware and forensic analysis, honeypots, cloud technologies, and more. However, the variety of processor architectures and operating systems prevalent in cyber-physical systems is the primary barrier to advancements of defensive information technology solutions for malware sandboxing, static and dynamic analysis, and infection and attack remediation. The second barrier is knowledge about and experience with the complex ways operational technology systems interact with, measure and control physical processes (e.g., electrical grids and oil, gas and water pipelines). Simply

removing malicious files or payloads from affected operational technology systems may not restore their overall functionality. Advances in physics simulation and microprocessor emulators and simulators can address these barriers.

This research has two main contributions. The first is a detailed simulated physics and embedded virtualization integration (SPAEVI) methodology that marries physics simulation advances with microprocessor virtualization, emulation and simulation to enhance industrial control system testbed capabilities. The second is the detailed implementation of the SPAEVI methodology that integrates a virtualized Siemens MJ-XL voltage regulator control panel within a medium-fidelity physics simulation of arbitrary electric grids with GridLAB-D.

2. Related Work

This work builds on previous research [10]. Also, it leverages the testbed taxonomy presented in [13], where operational technology networks and systems are broken down into four layers:

- **Layer I:** Sensors and actuators.
- **Layer II:** Distributed controllers, which include programmable logic controllers (PLCs), intelligent electronic devices (IEDs) and other forms of programmable automation controllers (PACs).
- **Layer III:** Supervisory and control systems, which encompass systems that store process data and implement control schemes that manage the lower levels.
- **Layer IV:** Human-machine interfaces (HMIs), which enable human operators to manage physical processes.

The methodology presented in this chapter requires the virtualization of real firmware and software at Layers II, III and IV, although fully virtualized grid networks are not necessary to realize the benefits of the methodology. Physics simulation is integrated in Layer I. The remainder of this section discusses previous work related to the four layers.

2.1 Testbeds

A number of SCADA testbeds have been developed by academic, government and private entities [4, 5, 8]. The testbeds are used to find new vulnerabilities; train engineers, incident responders and researchers; analyze attack patterns, footprints and impacts; and develop innovative defenses. Testbeds commonly fall into two categories: (i) real grid testbeds; and (ii) simulated grid testbeds that commonly engage a real-time digital simulator for physics simulations and digital-to-analog signal generators to support cyber-physical hardware integration. Generally, both groups focus on hardware-in-the-loop testing in Layers II and III, and, thus, inherit all the aforementioned limits related to vulnerability analysis, malware analysis, etc.

The most notable testbed is the National SCADA Test Bed (NSTB). This testbed, which comprises 17 test and research facilities, incorporates realistic-scale control systems ranging across 61 miles of 138 kV transmission lines, seven substations and various modeling tools. It connects other testbeds such as the Critical Infrastructure Test Range at Idaho National Laboratory, Center for SCADA Security at Sandia National Laboratories, Energy Infrastructure Operations Center at Pacific Northwest National Laboratory and facilities at Oak Ridge and Argonne National Laboratories. These testbeds largely rely on real hardware for Layers I and II, and real software for Layer IV. Layer III incorporates a mix of real and virtualized systems.

Joint academic and industry testbeds include ExoGENI-WAMS-DETER at North Carolina State University [3], DETERLab at the University of Southern California and the Trustworthy Cyber Infrastructure for the Power Grid facility at the University of Illinois.

The ExoGENI-WAMS-DETER testbed has two layers. The first layer is a hardware-in-the-loop setup comprising a real-time digital simulator integrated with phasor measurement units from multiple vendors. The second is a cloud-based virtual network based on ExoGENI+DETER to enable the simulation, observation and management of arbitrary network topologies of phasor measurement units using a real-time digital simulator to generate physics inputs. The ExoGENI and DETER laboratories provide virtual networking of arbitrary topologies to simulate power grid networks at any scale.

The Trustworthy Cyber Infrastructure for the Power Grid (TCIPG) is a large, diverse initiative that focuses on all areas of grid security research from power generation and transmission to distribution and metering. It incorporates a real-time digital simulator for hardware-in-the-loop testing. Also, it has a virtual power system testbed discussed below, which is a notable exception to the categorization above.

Thornton and Morris [13] maintain that the testbeds described above are not portable, expandable or distributable. Furthermore, only researchers with hands-on access to the testbeds are able to perform research using the resources.

2.2 Physics Simulations

Commonly-used simulation software for electric power grid security research include GridLAB-D, MATLAB Simulink and PowerWorld.

The Trustworthy Cyber Infrastructure for the Power Grid's Virtual Power System Testbed (VPST) utilizes PowerWorld software for its physics simulations, real components for hardware-in-the-loop testing and the VPST-C simulator to model computer hardware, software and communications infrastructures. The VPST approach simplifies grids by fitting all devices, software and processes to the ISO model. However, it does not appear to support embedded system virtualization at Layer II [1].

Thornton and Morris [13] present a software-based approach for virtual SCADA laboratory design utilizing Simulink. However, this work does not involve the full virtualization of Layer II systems; instead, it only provides a

simulation of the ladder logic code of programmable logic controllers. Therefore, the approach may not enable the testing and analysis of real zero-days, exploits and malware designed for programmable logic controllers, such as the infamous, still-unpatched, ladder logic remote code execution (RCE) vulnerabilities [9]. Nevertheless, Thornton and Morris provide valuable discussions of process simulation, programmable logic controller emulation and analysis that are relevant to this work.

Redwood et al. [10] have developed a physics simulation integrated with Layer III and IV devices for real-time anomaly detection based on changes to the physics. GridLAB-D was utilized to simulate the physics and real software was used for the human-machine interface and smart IEC 68150 switch implementation. The advantage of physics-based intrusion detection is that attacks can be detected regardless of their stage, properties or vectors, and sabotage-based attacks can be identified regardless of whether they are delivered by sophisticated zero-day exploits or script kiddies who manipulate human-machine interfaces. However, GridLAB-D suffers from poor fidelity in terms of physics simulations and cannot model phasors, harmonics and other transients in real time.

Approaches that utilize real-time digital simulators for physics simulations to support hardware-in-the-loop testing are not modular, portable or distributable. However, they do offer physics simulations at the highest levels of fidelity.

2.3 Emdedded Virtualization

Embedded virtualization is a relatively small, but rapidly growing field. Its primary applications are smartphones and tablets, aerospace avionics (with the PikeOS hypervisor) and automobiles (Automotive Open System Architecture).

Operational technology systems vary significantly in hardware and software compared with information technology systems. Operating systems that are common in operational technology deployments include variants of Linux, VxWorks, Windows Embedded, dozens of propietary operating systems and real-time operating systems, raw firmware level binaries, and so on.

Hardware options span processor families from Intel, ARM, Atmel, Texas Instruments, IBM PowerPC, Motorola Freescale, MIPS, Siemens C166, Hitachi/Renesas and numerous peripherals and analog devices. With the notable exception of SATCOM's space plug-and-play architecture (SPA) driver model, embedded cyber-physical system peripherals, sensors and actuators do not follow a plug-and-play driver model and, therefore, require individual attention. As a result, embedded system virtualization solutions for industrial control and SCADA systems are not widespread.

Adopting and/or modifying an architecture emulator or simulator may be necessary to virtualize a target device. An emulator is designed to recreate the original functionality of target hardware. Software executing on an emulator should perform exactly as if it were running on the target hardware. Simulators are designed to recreate the original functionality of the software or hardware, and are, thus, imitations. The end results are often very similar, but the im-

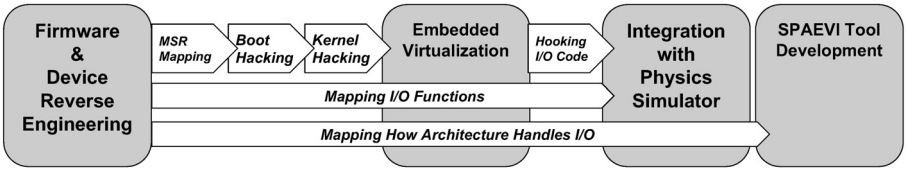

Figure 1. SPAEVI methodology workflow.

plementations of emulators and simulators are very different. Emulators often meet the requirements for virtualizing embedded systems, but many commercial simulators are considered to be compatible if they simulate enough of the underlying hardware.

Embedded emulation is historically associated with in-circuit emulation, which involves the use of a hardware device (e.g., JTAG debugger) or in-circuit emulator to debug the software on the embedded hardware. For the purpose of this work, it is considered to be a pure software-based emulation, because it involves no hardware. Several embedded system emulation options exist; the most notable is the open-source Multi Emulator Super System (MESS), which is the basis for the Multiple Arcade Machine Emulation (MAME) Project.

While embedded system virtualization is not new, the simulated physics and embedded virtualization integration (SPAEVI) methodology presented in this chapter is novel in that it combines embedded virtualization and simulated physics. It builds on previous work [10] and offers several benefits beyond traditional virtualization technologies.

3. SPAEVI Methodology

The simulated physics and embedded virtualization integration (SPAEVI) methodology is designed to produce portable, expandable and distributable software-based testbeds, primarily for embedded systems. Thus far, SPAEVI efforts have focused on single-core embedded systems; multi-core embedded systems impose additional challenges that are outside the scope of this work. Embedded virtualization is not new; however, the novel, methodical integration of a physics simulator with virtualized embedded systems is the core of the SPAEVI methodology. This section provides details about the methodology and presents a proof-of-concept implementation and experimental results.

3.1 Overview

Figure 1 presents the SPAEVI methodology workflow. The first step in the methodology is to acquire the firmware for the target device and know the processor(s) on which it executes. Vendors typically release firmware patches online; however, most vendors do not disclose any details about the processors that execute the firmware. In the event customer service or online research do

not reveal processor details, the target hardware can be inspected to read the identifying numbers on the processors.

3.2 Reverse Engineering Requirements

This section outlines the SPAEVI tasks that satisfy the reverse engineering requirements for each device.

Understanding Machine-Specific Register Usage. Machine-specific registers are significant to the SPAEVI methodology because they are used to locate the lowest levels of the embedded firmware at which sensor inputs and actuator outputs occur. Processors have unique registers, dubbed machine-specific registers, outside the general purpose register set that are used for a variety of internal and external tasks, including timing, I/O, exceptions, chip selects, addressing, ports, watchdogs and more. By studying the processor user manual, it is possible to identify the machine-specific registers that facilitate I/O.

Accessing the machine-specific registers usually involves: (i) special instructions; or (ii) a fixed region of addressable memory that is reserved for the internal functionality of the processor. In the latter case, the high end (`0xFFF...`) region of memory is typically reserved for machine-specific registers in most systems. Furthermore, these registers are typically given unique titles for each processor. However, all the non-general-purpose registers are typically identified as machine-specific registers.

In the case of special instructions, machine-specific registers may appear in disassembled code with unique mnemonics, but are often executed only in the privileged or supervisory modes. Malware analyses of x86 systems generally do not focus on the machine-specific registers because malware typically resides outside low-level drivers, the BIOS and kernel code.

In the case of fixed memory machine-specific registers, the registers typically do not appear in a disassembler with unique mnemonics, but instead can be identified by the region of addressable memory they occupy.

Understanding how firmware uses the machine-specific registers for I/O requires the enumeration of the machine-specific registers of the processor that facilitate I/O, followed by their enumeration in the disassembled firmware. This enables the machine-specific registers that are not utilized by the device firmware to be excluded from further consideration. After enumerating the machine-specific registers that facilitate I/O, it is necessary to reverse engineer "up" the cross-reference chain from the drivers and subroutines in order to map and understand how the registers are actually used.

Understanding BIOS, Bootup and Kernel Space. Before thoroughly reverse engineering and mapping the use of I/O machine-specific registers, it is necessary to reverse engineer the BIOS code, which is usually at or near the entry point to the firmware binary. The BIOS code typically configures, initializes, pings and/or tests the system I/O using the I/O machine-

specific registers. It also configures chip selects (i.e., direct memory addressing), I/O direction pins and purposes, as well as interrupts, exceptions, timeouts and the stack. These details are essential and further narrow down how the I/O machine-specific registers are utilized by the device.

Additionally, the BIOS and bootup code in most embedded firmware perform various integrity, timing and system checks during the boot process, which may impede the full boot of a virtualized embedded system. For example, a device often checks if its sensors, actuators, modules and/or peripherals are properly connected and operating; this is typically documented in the installation manual of the device and the processor user manual. It may be necessary to use breakpoints, hooks or binary patching to bypass the checks in order to force the firmware to fully boot up in the virtualized environment.

The bootstrap or bootloader may unpack and boot to a backup driver if startup checks fail; this may require an operator to insert a serial or USB cable to configure or flash the firmware. Operating systems likewise perform configuration, initialization, timing and integrity checks. Default settings and configurations are usually detailed in the device manual, although such checks may need to be bypassed as well.

Finally, a fully booted kernel usually ends up in a main loop that handles I/O. The I/O model can provide valuable guidance – these models usually fall into three main categories: (i) polling; (ii) interrupts; and (iii) direct memory access driven I/O. Note that it is not necessary to fully reverse engineer the kernel nor is it necessary to fully boot into a stable kernel to utilize the benefits of the SPAEVI methodology.

Mapping I/O Interactions and Handling. After understanding how the BIOS initializes and configures the I/O for the architecture, and enumerating the machine-specific registers in the disassembled code, it is necessary to reverse engineer each enumerated subroutine. Reverse engineering "up" the cross-references to subroutines reveals the context in which each machine-specific register in question is utilized. The chain of functions may span across drivers, kernel system calls, library function code and userland application code.

This process helps discover the purpose of the I/O machine-specific registers because it is common to find logging, protocol and even leftover ASCII debugging strings. Fortuitously, these strings were in the exact same subroutines in the proof-of-concept system developed in this research. The strings helped immediately identify the purpose of the I/O machine-specific registers and, thus, no cross-referencing was required.

3.3 Virtualization Requirements

Cyber-physical system virtualization requires accurate temporal and instruction set simulation or emulation of the processor, its I/O handling, machine-specific-registers and peripherals. Furthermore, the virtualization platform should execute the raw firmware of the target device. Additionally, it must be able to handle firmware extraction and memory region initialization.

Processor families often have multiple variants, all of them founded on a base instruction set and processor features (e.g., pipeline size, I/O model, timing module and peripheral modules). If no simulator or emulator exists for the processor of the target hardware, then a simulator or emulator for a processor in the same family may serve as a suitable starting point.

3.4 Integration Requirements

Integration requires an accurate virtualization platform and the mapping of the I/O machine-specific registers. Hooking or breakpointing the code that performs I/O enables the integration of relevant inputs and control outputs with the physics simulator. For a Layer II device, the inputs come from sensors and may be pre-processed by various FPGAs; the outputs are signals to actuators. The only thing virtualized by the SPAEVI methodology is the main processor. It is not necessary to reverse engineer or virtualize digital signal processors, FPGAs and microprocessors that handle analog I/O because they can be digitally integrated via the SPAEVI methodology.

3.5 Benefits

The SPAEVI methodology provides some novel benefits:

- **Dynamic Analysis of Embedded Cyber-Physical Systems:** Dynamic analysis is the foundation of modern malware analysis methodologies. The ability to provide dynamic analysis capabilities for embedded cyber-physical systems is a novel contribution of the SPAEVI methodology. This aspect is discussed in detail later in this chapter.

- **Physics Impact Analysis:** Integrating the interaction of a virtualized embedded cyber-physical system with a simulated electric grid is the core of the SPAEVI methodology and it provides novel physics impact analysis capabilities. However, the fidelity of a chosen physics simulation directly affects the benefits provided by the SPAEVI methodology. A SPAEVI implementation with a physics simulator that does not simulate down to the harmonics cannot detect or analyze an attack that maliciously triggers harmonics or similar line transients in a power grid.

- **Physics-Based Intrusion Detection:** Physics-based intrusion detection for symbolic cyber-physical honeynets is described in [10]. The proof-of-concept implementation involved a smart IEC 68150 distribution substation switch integrated with GridLAB-D. The approach requires the accurate integration of networking in the virtualization platform, otherwise the target cannot communicate using the necessary protocols. In theory, the SPAEVI methodology can provide this benefit. However, network handling for the MJ-XL in Trace32 has to be implemented in order to realize physics-based intrusion detection for embedded cyber-physical systems.

4. Proof-of-Concept System

The SPAEVI proof-of-concept system comprises the Siemens MJ-XL voltage regulator control panel, selected because it controls the electric grid in a complex manner. The device technically could be categorized as a programmable logic controller; however, it predates the IEC 61131 Standard published in December 1993. It was built using the Diablo C compiler (with timestamps of 1990, 1992 and 1993). It uses the legacy RTXC real-time operating system, which is no longer supported and may be considered abandonware. Also, it uses the DNP3 protocol for remote monitoring and control.

Exploitation and access control bypass details are not presented in this chapter. Additionally, the reverse engineering details of the target outside of the I/O integration for the SPAEVI implementation are not presented. In any case, these details would be unique to each device and would not serve to clarify the methodology. Determining the processors that run the MJ-XL was a difficult task because the vendor meticulously omitted all mention from the manuals, brochures and documentation. However, this problem was solved after purchasing a used unit on eBay and inspecting the hardware. The device firmware, which was found on the vendor website, is in the Motorola S-record format.

4.1 Reverse Engineering

The SIEMENs MJ-XL control panel main board processor is a Motorola MC68332. The MC68332 is part of the 68 K (i.e., CPU32) family of processors. This processor handles several aspects of I/O, including the front key pads, configuration/update front serial port, as well as the 48 screw-in connectors on the back for interfacing with a voltage regulator. The communications module daughterboard facilitates the device's remote control via DNP3 over three network options: (i) fiber; (ii) legacy serial; and (iii) GSM. The focus in the proof-of-concept development was to discover which I/O machine-specific registers in the MC68332 read the sensor inputs and control the actuators in the voltage regulator.

The machine-specific registers in the MC68332 occupy a fixed 2 K region of memory spanning from `FFFA00` to `FFFFFF`. They belong to three main modules (i.e., on-chip peripherals): (i) system integration module (SIM); (ii) queued serial module (QSM); and (iii) timing processor unit (TPU).

The system integration module handles key internal functionality, including chip selects, interrupts, system protection logic, watchdogs and external bus support.

The queued serial module handles the device's front data port via a serial communications interface (SCI) for configuring and flashing the firmware, and the connection to the communications module daughterboard via the queued serial peripheral interface (QSPI). The serial communications interface is configured by SCCR0 and SCRR1, which set the baud clock and bits per frame, respectively. In theory, virtualizing the serial port of the serial communications interface would enable the Siemens configuration software to be used to config-

ure the virtual device, although the stability and baud clock synchronization could be problematic.

The timing processor unit provides several channels of I/O to a separate 2 KB block of RAM.

Table 1 enumerates some of the I/O machine-specific registers. Counting the number of times the machine-specific registers are used reveals the I/O functionalities used by a device. By analyzing the pin direction assignment registers, it is possible to determine the channels used for input and output.

The Motorola S-record binary format explicitly dictates the entry point into the unpacked firmware binary, which corresponds to the beginning of the BIOS. Reverse engineering the BIOS code quickly revealed the configuration of the chip selects; these are noted in Table 1 for CS1 to CS9.

Reverse engineering the cross-references to the code that uses the machine-specific registers for I/O reveals how the firmware handles sensor inputs, and more importantly, actuator control outputs to move the regulator taps. The MJ-XL patent [15] and installation manual provide the valid ranges of the inputs, which served as a reference for virtualization I/O integration and testing. Ultimately, the chip selects initialized in the BIOS were vital to determining the I/O machine-specific register mapping. Table 2 presents the final results.

4.2 Virtualization

Several platforms were considered for creating the virtualization of the target. Specifically, the Trace32 sim68k, MAME, easy68k, turbo68k and other simulators were examined. However, the Trace32 sim68k simulator has the most accurate and detailed model for the MC68332 processor and on-chip peripheral modules. Furthermore, it has the most feature-laden breakpoints, enabling the CPU to be frozen and external operating system commands to be executed.

4.3 Simulated Physics Integration

GridLAB-D currently has some fidelity limitations. For example, it cannot simulate sub-second events in the mode utilized for the SPAEVI integration (real-time mode). Nevertheless, it was possible to integrate the necessary sensor inputs to the virtualized device, primarily three-phase voltage and current. GridLAB-D allows the configuration of the regulator taps for each phase, band center and width of the desired voltage, time delays for taps to move, current and voltage transducer ratios, and much more. The top controls were the primary actuators considered in the SPAEVI proof-of-concept.

The MJ-XL processor uses a global variable at `0x100050` (initially set to `0xFFFF`) to craft the control signal to the tap control actuator. Depending on whether the tap actuator is raised or lowered, the signal is ANDed and ORed by a mask, and then moved to the actuator control chip select `0x82000` as follows:

Table 1. Enumeration of I/O machine-specific registers.

MSR	Address	Times Used	Summary
PORTQS	FFFC14	0	Not used by device
PQSPARS	FFFC16	4	Pins Select GPIO or QSPI. It is always 01111011.
DDRQS	FFFC17	4	PORTQS Data Direction Register. Always 11111110.
SPCR0	FFFC18	24	Frequently Used
SPCR1	FFFC1A	48	Frequently Used
SPCR2	FFFC1C	24	Frequently Used
SPCR3	FFFC1E	4	Rarely Used
SPSR	FFFC1F	29	Frequently Used
RR[0:F]	FFFD00-1F	6+	RX RAM
TR[0:F]	FFFD20-3F	18+	TX RAM
CR[0:F]	FFFD40-5F	17+	COMMAND RAM
Serial Port MSR Stats (Data port on front panel)			
SCCR0	FFFC08	10	
SCCR1	FFFC0A	17+	
SCSR	FFFC0C	20	Tells if SCI has data
SCDR	FFFC0E	19	SCI Data buffer
MC68332 System Integration Module (SIM) MSR Stats			
CS1	FFFA50	1	Maps to 0x100000
CS2	FFFA54	1	Maps to 0x80800
CS3	FFFA58	1	Maps to 0x80000
CS4	FFFA5C	0	
CS5	FFFA60	1	Maps to 0x84000
CS6	FFFA64	0	
CS7	FFFA68	1	Maps to 0x81000
CS8	FFFA6C	1	Maps to 0x81800
CS9	FFFA70	1	Maps to 0x82000
MC68332 Timing Processor Unit (TPU) MSR Stats			
TPUMCR	FFFE00	4	
TICR	FFFE08	1	
HSRR0	FFFE14	12	Used with CS 5&9
HSRR1	FFFE16	9	Used with CS 5&9

Table 2. MJ-XL chip select I/O map.

Chip Select	Base Address (DMA MAPPING)	Z	Config	Reverse engineering & Mapping notes
1	0x100000	256k	16bit	System RAM CHANNEL
2	0x80800	2k	8bit	DISPLAY SCREEN 2
3	0x80000	2k	8bit	DISPLAY SCREEN 1
5	0x84000	2k	8bit	TX/RX VRC SENSORS
7	0x81000	2k	16bit	KEYPRESS CONFIRM
8	0x81800	2k	16bit	KEYPRESS INPUT
9	0x82000	2k	16bit	VRC CONTROL TX

```
00023DD4    pea     (aRaise_2).l            ;"Raise" (Raise Tap)
00023DDA    jsr     print_to_display
...
00023E00    ori.w   #%0000011111111111,($100050).l
00023E08    andi.w  #%1111110111111111,($100050).l
00023E10    move.w  ($100050).l,($82000).l ;Send Control Signal to Tap
00023E1A    clr.w   (_ADC_CHANNEL_SELECTOR_0x100190).l
00023E20    bra.w   loc_24038
```

Similar code is used to command the actuator to lower the tap position:

```
00023E24    pea     (asc_2477C).            ;"Lower" (Lower Tap)
00023E2A    jsr     print_to_display
...
00023E52    ori.w   #%0000011111111111,($100050).l
00023E5A    andi.w  #%1111101111111111,($100050).l
00023E62    move.w  ($100050).l,($82000).l ;Send Control Signal to Tap
00023E6C    clr.w   (_ADC_CHANNEL_SELECTOR_0x100190).l
00023E72    bra.w   loc_24038
```

In order to integrate the tap actuator controls with the GridLAB-D simulation, breakpoints were placed at lines `0x00023E10` and `0x00023E62` as well as other lines of code that interact with `0x82000`. The breakpoint runs a script or command outside of the Trace32 sim68k that, in turn, passes the tap control signal to an intermediate wrapper. This wrapper parses the signal and controls the tap object in GridLAB-D.

Handling the semantics of how control signals are intended to control devices is the primary challenge during integration, mainly because the semantics are unique to each device. In the code above, it is apparent that bits 10 and 11 of the signal are flipped differently by the AND operation. Using these two cases as reference points, it is possible to reverse engineer and integrate other

variations of how 0x82000 is controlled. Handling the sensor inputs takes more care because the analog-to-digital channels are each pinged and then read from individually, as follows:

```
0002422A    moveq   #3,d0
0002422C    move.l  d0,-(sp)
0002422E    clr.    -(sp)                   ;Push 0 for channel #0
00024230    jsr     PING_AND_READ_SENSOR_BY_ID
00024236    addq.l  #8,sp
00024238    move.l   d0,d5                  ;Sensor value in d0
0002423A    move.l  d5,-(sp)                ;Push sensor value
0002423C    pea     (aAdcCh08d)             ;"ADC CH0=%8d"
00024242    pea     (_0x108F1C_ADC_CHANNEL_BUFFER).l
00024248    jsr     SENSOR_INPUT_DRIVER
```

The last function call to SENSOR_INPUT_DRIVER uses the arguments to look up the global settings table (not shown) in order to obtain the destination to store the corresponding sensor value. The destination is calculated based on a combination of the output of PING_AND_READ_SENSOR_BY_ID and the string identifying the sensor source (e.g., ADC CH0=%8d). It is beneficial, but not always necessary, to integrate the sensor inputs when implementing the SPAEVI methodology.

4.4 Verification of Benefits

Several sophisticated, sabotage-based, malicious test payloads were designed and analyzed in order to verify the defensive benefits of the SPAEVI methodology. No actual exploits were designed nor were any vulnerabilities or access control bypasses sought. Specifically, the efforts focused only on return-oriented programming (ROP) payloads, which for defensive analysis, require the highest fidelity in a virtualization platform. Additionally, embedded industrial control systems usually have no defenses for such attacks [6].

Return-oriented programming is an exploit development methodology that works across all computer architecture paradigms, from von Neumann to Harvard. Typically, payloads partially use return-oriented programming and partially use traditional code injection to handle complex operations (e.g., facilitating remote command and control for attackers). However, in the case of an embedded system, if a payload is simply designed to achieve a physical effect, then it can typically re-use existing code that handles the sensors and actuators of the system being attacked.

After thoroughly mapping the I/O subroutines, a return-oriented payload compiler [12] was used to generate specific actuator actions. Each actuator device, in turn, can be tagged with a meta-description that details how the physics is changed. The return-oriented payload compiler, dubbed the Physical Effect Payload Compiler (PEPC), was used in several experiments. It is

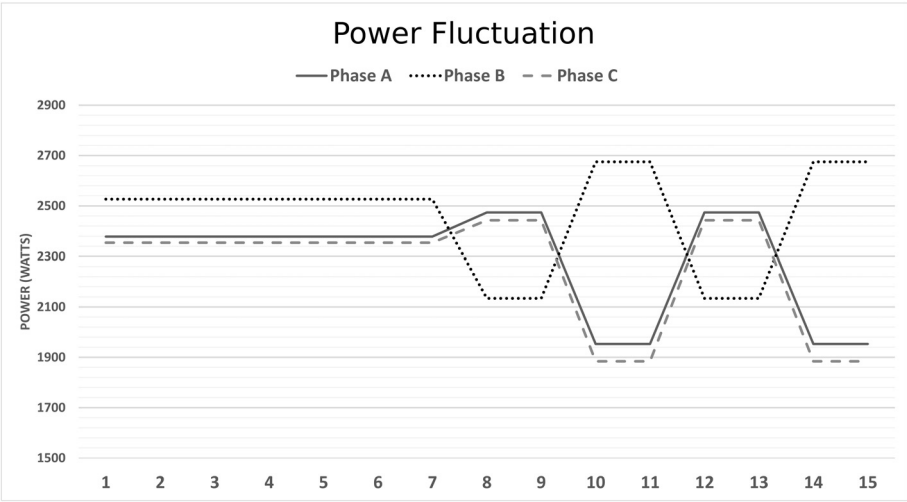

Figure 2. Payload experiment results.

important to note once again that a return-oriented payload is distinct from an actual exploit on a target.

4.5 Results

The Physical Effect Payload Compiler was scripted to maliciously and arbitrarily move the regulator taps. The IEEE 13 Node Test Feeder model [10] was used in the experiments and the MJ-XL was modeled as a 4.16 KV, 3.9 MW voltage regulator that feeds a distribution network. By default, the configured range of possible voltage regulation was ±10% and the per-tap change was dictated by this range divided by the number of tap positions. A tap in the maximum position (16) indicates a 10% increase and the minimum position (−16) indicates a −10% decrease.

Several payload experiments were conducted. Figure 2 presents the results of three experiments. The payload effects in Figure 2 were directly caused by a return-oriented programming payload executed by the virtualized firmware in the SPAEVI testbed. Note that the effects were designed to be delayed by a few seconds between each change using NOP gadgets in order to visually present the data.

Note that Figure 2 shows the power fluctuations in Watts. Payload 1 flips Taps A and C to the maximum position while Tap B is moved to the minimum position, following which the inversion begins. The malicious fluctuations can force up to ±10% maximum increase or decrease in the voltage level. The malicious regulation also affects the phase angle; the experiments cause up to ±1.20˚phase angle shift per phasor. Additionally, the other phasors do not change and can be forced into precise, minor phase angle imbalances. Theoret-

ically, these physical fluctuations, while dependent on the device configuration, may represent a "profile" for potential malicious effects.

After further reverse engineering, it was discovered how to place any tap in an arbitrary position with a 76-byte return-oriented programming chain per tap. It was possible to indefinitely hold a tap in the maximum or minimum position with a single return-oriented programming payload of eight bytes. These statistics are useful in terms of footprint and sophistication because single-purpose payloads require minimal footprints, but sophisticated, arbitrary control of an embedded cyber-physical system requires much more. With polymorphic return-oriented programming chains, the cost per arbitrary tap chain would be much less than 76 bytes per change. Malicious regulator fluctuations can induce other transients into a grid that GridLAB-D does not have the fidelity to model.

In some instances, as in the case of the IEEE model, the phase angle and voltage fluctuations may be problematic for an electric grid. However, infrastructure redundancies, constraints and implementations may be resilient to a single device misbehaving within the described "profile" of physics changes. Breakers, fuses, protective relays and even phase angle regulators may protect against such malicious effects. However, if a grid relies on homogenous redundancies, a single exploit can be trivially leveraged against all redundant systems simultaneously, amplifying the potential for harm instead of protecting against it. Indeed, heterogeneous, diverse technologies controlling an electric grid should be declared best practices as far as cyber security is concerned. Relying on a single vendor for a specific type of device across a power grid, exposes the grid to zero-day exploits against all susceptible targets simultaneously. This would also be true for n-day exploits due to poor patching practices in industrial control systems.

This research verifies that a SPAEVI testbed can support dynamic malware analysis and forensics for Layer II to IV systems. All the effects discussed above were caused by real return-oriented programming payloads that maliciously abuse device firmware and, thus, make the results tangible. Furthermore, the SPAEVI methodology offers incident responders with portable testbed systems for malware sandboxing, incident response, physics impact analysis and other defensive research in the area of industrial control systems.

5. Conclusions

The novel SPAEVI testbed design methodology for integrating virtualized embedded industrial control systems and physics simulators facilitates malware and forensic analyses of embedded cyber-physical systems without risks. Unlike existing hardware-based testbeds, the SPAEVI industrial control system testbeds are portable, distributable and expandable by design. The benefits of the SPAEVI methodology span the domains of incident response, forensics, attack characterization, vulnerability analysis, sandboxing and defensive applications. The case study involving a virtualized Siemens MJ-XL voltage regulator control panel integrated with a medium-fidelity physics simulation of arbitrary

electric grids using GridLAB-D conclusively demonstrates the benefits of the SPAEVI testbed design methodology.

References

[1] D. Bergman, D. Jin, D. Nicol and T. Yardley, The virtual power system testbed and inter-testbed integration, *Proceedings of the Second USENIX Conference on Cyber Security Experimentation and Test*, 2009.

[2] E. Byres, #1 ICS and SCADA Security Myth: Protection by Air Gap, Tofino Security, Lantzville, Canada, 2012.

[3] A. Chakrabortty, Y. Xin and A. Hussein, A U.S.-wide DETER-WAMS-ExoGENI testbed for wide-area monitoring and control of power systems using distributed synchrophasors, presented at *Cyber-Physical Systems Week*, 2015.

[4] C. Davis, J. Tate, H. Okhravi, C. Grier, T. Overbye and D. Nicol, SCADA cyber security testbed development, *Proceedings of the Thirty-Eighth North American Power Symposium,* pp. 483–488, 2006.

[5] G. Dondossola, F. Garrone and J. Szanto, Supporting cyber risk assessment of power control systems with experimental data, *Proceedings of the IEEE/PES Power Systems Conference and Exposition*, 2009.

[6] I. Evans, Analysis of Defenses Against Code Reuse Attacks on Modern and New Architectures, M.E. Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, Massachusetts, 2015.

[7] Freescale Semiconductor, MC68332 User's Manual, Chandler, Arizona (`cache.freescale.com/files/microcontrollers/doc/user_guide/MC68332UM.pdf`), 2004.

[8] A. Giani, G. Karsai, T. Roosta, A. Shah, B. Sinopoli and J. Wiley, A testbed for secure and robust SCADA systems, *ACM SIGBED Review*, vol. 5(2), article no. 4, 2008.

[9] E. Leverett and R. Wightman, Vulnerability inheritance in programmable logic controllers, *Proceedings of the Second International Symposium on Research in Grey-Hat Hacking*, 2013.

[10] O. Redwood, J. Lawrence and M. Burmester, A symbolic honeynet framework for SCADA system threat intelligence, in *Critical Infrastructure Protection IX*, M. Rice and S. Shenoi (Eds.), Springer, Heidelberg, Germany, pp. 103–118, 2015.

[11] P. Roberts, Industrial control vendors identified in Dragonfly attack, *The Security Ledger* (`securityledger.com/2014/07/industrial-control-vendors-identified-in-dragonfly-attack`), July 4, 2014.

[12] E. Schwartz, T. Avgerinos and D. Brumley, Q: Exploit hardening made easy, *Proceedings of the Twentieth USENIX Conference on Security*, 2011.

[13] Z. Thornton and T. Morris, Enhancing a virtual SCADA laboratory using Simulink, in *Critical Infrastructure Protection IX*, M. Rice and S. Shenoi (Eds.), Springer, Heidelberg, Germany, pp. 119–133, 2015.

[14] A. Timorin, SCADA Strangelove: SCADA deep inside, presented at the *Balkan Computer Congress*, 2014.

[15] J. Trainor, C. Laplace, M. Bellin and M. Hoffmann, Man-Machine Interface, United States Patent 5,844,550, 1998.

[16] Subcommittee on National Security, Homeland Defense and Foreign Operations of the Committee on Oversight and Government Reform, Cyber Security: Assessing the Immediate Threat to the United States, Serial No. 112–55, U.S. House of Representatives (112th Congress, First Session), Washington, DC, May 25, 2011.