Secure RTOS Architecture for Building Automation

Xiaolong Wang
Kansas State University
Manhattan, Kansas, USA
danielwang@ksu.edu

Masaaki Mizuno
Kansas State University
Manhattan, Kansas, USA
masaaki@ksu.edu

Mitch Neilsen
Kansas State University
Manhattan, Kansas, USA
neilsen@ksu.edu

Xinming Ou
Kansas State University
Manhattan, Kansas, USA
xou@ksu.edu

S. Raj Rajagopalan
Honeywell ACS Labs
siva.rajagopalan@honeywell.com

Will G. Baldwin
Biosecurity Research Institute
Manhattan, Kansas, USA
wbaldwin@k-state.edu

Bryan Phillips
Biosecurity Research Institute
Manhattan, Kansas, USA
bryanp@k-state.edu

ABSTRACT

Building Automation System (BAS) is a computer-based control system that is widely installed in office buildings and laboratories for monitoring and controlling mechanical/electrical equipment. With the advancements in Cyber-Physical System (CPS) and Internet of Things (IoTs), BAS is in the process of becoming more intelligent by merging computing resources and network communication with physical control. Along with potential benefits, it also brings tremendous risks of security breaches and safety violations, especially when it comes to Programmable Logic Controllers (PLCs). In this paper, we systematically analyze biocontainment laboratory control models based on real case scenarios from Biosecurity Research Institute (BRI) at Kansas State University. We present a vision for a new secure Real-Time Operating System (RTOS) architecture, which leverages various technologies, including microkernel structure, Trusted Platform Module (TPM), proxy-based policy enforcement, and formal verification. The secure RTOS architecture is designed specifically to work with embedded controllers which are widely used in BAS and other CPS to achieve a highly secure and trustworthy control system.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection

General Terms

Design; Security; Reliability

Keywords

Cyber-Physical System; Building Automation; RTOS; Microkernel; TPM; Trusted Computing

CPS-SPC'15, October 16 2015, Denver, Colorado USA
©2015 ACM. ISBN 978-1-4503-3827-1/15/10 ...$15.00
DOI: http://dx.doi.org/10.1145/2808705.2808709.

1. INTRODUCTION

Building Automation System (BAS) is a complex network-based distributed control system responsible for the communication and cooperation of different electrical/mechanical subsystems. A typical building automation is designed to automatically control a building's heating, ventilation, and air conditioning (HVAC). Along with advances in control systems, BAS has evolved into a gigantic system. It not only handles HVAC, lighting, air humidity, but also manages building security and safety subsystems by controlling and monitoring fire and flood safety, CCTV, elevator, power supply, and part of the process for room access authentication. Often, BAS provides a communication backbone which serves as an infrastructure that provides rules, policies, and integration medium for different subsystems.

Buildings, including commercial, government, and private, are a ubiquitous critical infrastructure [37] and yet the last to be considered as such. The Boston Marathon bombing revealed that the financial losses from the shutting down of large commercial buildings in the area were in the hundreds of thousands of dollars per day. Recent cyber attacks have targeted critical infrastructure control system [1, 31] raising safety and security concerns. The safety and security of BAS is at high risk. Unfortunately, there have been few systematic studies of cybersecurity for building controls, in part because the sector is completely dominated by vendors who mostly do not have much interest in academic research. A significant number of off-the-shelf BAS products in the market are currently based on outdated technologies, which have limited security features, and unpredictable vulnerabilities due to their backward-compatible designs and focus on ease of maintenance. Moreover, the building industry is highly fragmented and no single vendor has a controlling share of the market to force change, which makes the situation even more challenging.

In this paper, we conduct a systematic case study of biocontainment facility building automation system based on real case scenarios from Biosecurity Research Institute (BRI) at Kansas State University, and analyze specification of the requirements for each control task. We provide a new possible research direction to secure CPS infrastructure from an operating system's point of view and present a roadmap of our proposed secure operating system architecture. Our contributions are:

- We systematically study realistic bio-containment facility building scenarios and identify safety requirements therein.
- We provide a detailed analysis of potential security vulnerabilities in the BAS and how that may impact the safety requirements of biocontainment facilities.
- We propose an architecture and roadmap for ensuring security/safety properties of building environments by adopting a microkernel-based architecture, where the real-time operating systems (RTOS) on embedded controllers become the security anchors for such cyber physical systems. We point to directions where this architecture could be used to strengthen the security/safety properties of BAS.

2. BACKGROUND

Buildings worldwide account for 30% to 40% of global energy consumption [45]. There are increasing concerns of energy efficiency and high demand for more intelligent buildings. A number of advanced technologies are designed to incorporate into BAS to make it smarter. The concept of Smart Building has existed since the 1980's, and a definition given by the Intelligent Building Institution in Washington is "one which integrates various systems to effectively manage resources in a coordinated mode to maximize: technical performances; investment and operating cost savings; flexibility." [13]. The idea is to use computers and networks to automatically perform operations based on predefined rules. Today leveraging the advance of Internet and various digital technologies, researchers are proposing Cyber-Physical System (CPS) and Internet of Things (IoTs), which conjoin computational and physical resources through digital networks to achieve a more intelligent, automatic, and energy-efficient control system.

The benefits new technologies bring to BAS have also opened the gate for various safety and security risks. Those trends in the building industry such as convergence of control and IT networks, use of mobile devices to access building controls, and the increasing use of COTS is raising the risk of cybersecurity breaches dramatically. Even though the number of reported attacks on Industrial Control Systems are few [17, 46], one can expect them to increase in the coming years due to a number of issues. BAS are already connected to the Internet. As shown by Billy Rios at Black Hat USA 2014 [40], there are 21,000 Tridium Systems (one of the most popular platforms for building control) connected to the Internet. It has been identified in over 50,000 buildings which are exposed to the Internet either intentionally or due to misconfiguration and can be publicly searched using tools such as Shodan [28, 40]. Besides, BAS widely uses outdated low level protocols, such as BACnet, KNX, Modbus, which send data in plaintext and lack proper authentication mechanisms. Works [2, 32] show that attackers can easily sniff control packets, modify Programmable Logic Controller (PLC) arbitrarily, use carefully crafted low-level data gathered through PLCs to inject high-level control software. Moreover, many commercial BAS, like MetaSys and Niagara are based on outdated Windows operating systems. Stuxnet [47] suggests that specifically crafted malware can be easily launched against BAS to sabotage high security-concerned institutions. While the U.S. Department of Energy works together with alliances launching a new Net Zero Energy Installation (NZEI) initiative for achieving self-sustaining buildings, building infrastructure is integrating with different internal and external infrastructures [5]. It can be a stepping stone for attacking other critical infrastructures such as power, water, transportation, health service, etc. With the potential threats, attack surface, and the relatively low risk to impact ratio for attackers, buildings would be an attractive target for terrorist attack and cyber war.

2.1 PLC Basics

PLCs play a critical rule in control systems. A PLC is a digital embedded device used for automation of industrial processes. They perform real-time control of electromechanical processes. A PLC works by continually scanning a program, which is called the scan cycle. This involves reading sensor measurements repeatedly, executing control logic program to calculate output, and actuating output with electromechanical processes. PLCs are responsible for constantly adjusting physical machinery based on the sensor measurement as well as functions as a gateway between the machinery and human operators. PLCs translate continuous analog signals into digital values. Processing is performed using a cycle of input, processing and output. The control logic for processing is typically represented using relay ladder logic written in a graphical language. The most widely used PLC control structure is a Proportional Integral Derivative (PID) controller. This controller algorithm has three separate constant parameters: the proportional, the integral, and the derivative values. PID controllers are used to dynamically adjust output by comparing setpoint (desired value) and actual value of the process variable from the process under control.

2.2 PLC Security Issues

BAS security and safety has gradually raised concerns of both industry and academia. While many researchers are working to secure BAS protocols, and designing new detection algorithms [6, 8] we believe that the security of Programmable Logic Controller is overlooked. PLC is the controller that directly interacts with sensors and actuators, as a central anchor. The stability of PLCs is the weak link in a chain that directly influences the robustness of the BAS. Current PLCs, however, are relatively fragile. A majority of PLCs are limited electrical devices that run a control loop directly on bare hardware. They lack functionality for simple emergency situations such as power outage or network failures. Some PLCs run a simple real-time operating system (RTOS); e.g., FreeRTOS. All applications are compiled together with a real-time scheduler in a single binary file. Systems lack the capability for handling complex situations and the absence of protection rings makes it trivial for malware to gain system privileges. There are some PLCs that use advanced operating systems, e.g., customized Linux, QNX, VxWorks, etc. Although QNX and VxWorks are embedded real-time operating systems that are designed for industrial control with hard real-time capability, there are, however, certain well-known vulnerabilities in VxWorks and QNX [19, 41]. More importantly, those general-purpose OSes either are designed with legacy support for compatibility reasons, which have a large attack surface, or are proprietary software that lack technical details for evaluation. Therefore, it gives designers no confidence for functional correctness and are almost inevitable to have vulnerabilities [41]. Furthermore, embedded controllers has their special properties that demand different requirements from a general purpose real-time OS. For example, the work by Hernandez, *et al.* [16] shows how attackers tamper and own a Nest thermostat, which runs on a Linux based platform, with a reset attack.

3. RESEARCH METHODOLOGY

While the need is obvious, making building controls cyber-secure is a challenge for a variety of reasons. Any approach to securing building controls must accommodate the long field life of control hardware. But at the same time, we are seeing tremendous growth in the software-enabled functionality to support new business drivers such as energy efficiency, customized comfort, etc. This means that while control hardware will stay in place for a decade or more, the software will follow the pattern of IT in using highly insecure,

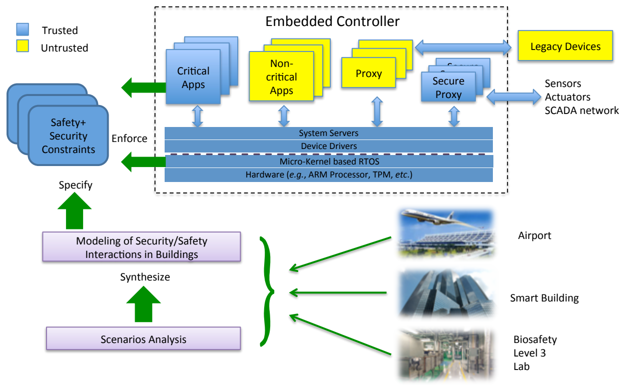

Figure 1: Building Security/Safety Control Framework

but quick-to-market technologies such as web- related services. The essential challenge then is: how can we persuade the building industry that effective cybersecurity is achievable at reasonable cost and without major disruption to the existing infrastructure? A related challenge is how to create a model architecture for control that permits the use of IT software components but avoids the attendant constant patching and upgrading. The goal of this research is to fulfill this vision by designing a security/safety modeling framework for building controls; the framework will enable decomposing the domain-specific security/safety properties into the various processes that run on embedded controllers, so that a microkernel based real-time operating system (RTOS) can guarantee that the critical tasks a controller must fulfill will never be jeopardized by the malicious environment. In other words, the attacks in the insecure cyberspace will be stopped at embedded controllers, which serve as the security anchors of BAS (and possibly other types of cyber physical systems).

Smart building is a unique environment that poses challenges to security and safety. Buildings are containers of many different types of human activities supported by various types of technologies. The type and utility of a building determines its unique security and safety needs. The very important first step is to research the multitude of security/safety issues that may arise and properly model them so each embedded controller in the environment has clearly defined security/safety requirements it has to uphold. In this paper, we conduct different investigations of BAS application scenarios, with special focus on a biocontainment laboratory. Figure 1 shows the envisioned solution framework. Based upon scenario analysis, we abstract the model of building automation system and the logic behind control policies. We extract safety/security constraints that should be enforced by different layers of system running on those embedded controllers. Through our investigations, we believe that microkernel-based real-time OS is appropriate for critical infrastructure control systems. Microkernel architecture is designed to address the increasing growth of kernel and the difficulty of managing and maintaining the code base. Different from monolithic kernel, microkernel breaks up functionalities and modularizes them into independent components. Nonessential kernel parts are moved into user space. The core functionality is isolated from system services and device drivers, which greatly increases the stability of OS. In a microkernel architecture, different components are loosely tied together through Interprocess Communication (IPC) similar to distributed system. Such a setup enables system designers to define and enforce fine-grained constraints separately.

4. BIOCONTAINMENT FACILITY

An important use case of BAS is one for controlling and monitoring a biocontainment facility. Biocontainment facilities are designed to do research on infectious diseases, pathogens, quarantined pests, invasive alien species and living organisms [24]. The nature of this kind of laboratory makes it highly hazardous. Since some samples might be contagious for human beings, oncereaching the public space it could cause tremendous danger that even jeopardize human lives. In this case, protecting just the worker is oftentimes not enough. Systems must also be in place to protect the environment and the facility from possible contamination. In the United States, the Centers of Disease Control and Prevention (CDC) have specified different biosafety levels of biocontainment precautions. They range from the lowest level 1 (BSL-1) to the highest at level 4 (BSL-4) for isolating dangerous biological agents in an enclosed laboratory facility [36]. Biosafety levels increase safety and security measures as the dangers associated with the agents increase. For example, BSL-3 is applicable to clinical, diagnostic, teaching, research, or production facilities in which work is done with indigenous or exotic agents. The agents may cause serious or potentially lethal disease after inhalation. BSL-4 is specifically for fatal diseases such as the Marburg or Ebola viruses [36]. For guiding the design and maintenance of laboratories, the CDC and the U.S. Department of Agriculture - Agricultural Research Service (USDA-ARS) have published Biosafety in MicroBiological and Biomedical Laboratories (BMBL) and USDA-ARS Facilities Design Standards respectively. They define various policies and standards for HVAC, air pressure, temperature, and the procedure of handling special situations [36, 44]. Moreover, The National Institutes of Health (NIH) details design requirements and guidance manuals for biomedical research facilities [35]. In all of them BAS is regarded as critical and responsible for enforcing biosafety level and personnel security. BRI is a biosafety level 3 and biosafety level 3 agriculture facility.

4.1 Scenario Analysis

One of the high-priority concerns of a biocontainment facility is airborne hazards exposures through air exchange. Pathogens can be transmitted through air. Modern laboratories are busy environments with personnel sharing equipment across overlapping workstations. Different laboratories usually dedicate to different tasks. Without proper isolation and aseptic counter measures, contaminants can easily be transmitted to different areas and pollute the purity of cell cultures as well as a safe lab environment. The solution for this is to constrain airflow. Only let air flow from the corridors inward ensuring that contaminated air cannot escape from the laboratory to other parts of the facility. Therefore, by requirement, all biocontainment laboratories must constantly maintain negative differential room pressure to prevent cross-contamination. In order to achieve this, multiple precautions are enforced by BAS to ensure uninterrupted safe operations. Those include directional airflow (air flows from areas of lower containment to areas of higher containment), ventilation, high-efficiency particulate (HEPA) filters that provide a very high filtration efficiency for the smallest as well as the largest particulate contaminants, door interlock control that enforces no more than one door opens in a zone at the same time, ID authentication and door access. Furthermore, chemical processes are temperature sensitive. Laboratories must maintain an appropriate temperature for chemical processes as well as for occupants. Last but not least, the use of chemicals and other potentially hazardous compounds separates laboratories from other parts of building spaces. How to properly isolate biocontainment laboratories from other building spaces, while maintaining a convenient, comfortable and energy-efficient environment is also important. With those concerns in mind,

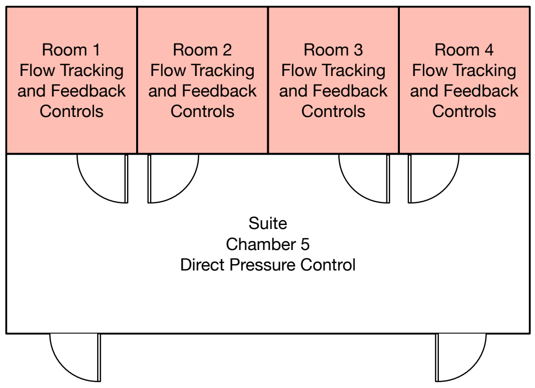

Figure 2: Biocontainment Laboratory Scenario Layout

our presented scenario is based on a real biocontainment facility. For safety consideration, all examples discussed in this paper are assumptive.

For the sake of simplicity, in this scenario we are only considering a suite with five rooms in a biocontainment facility. A biocontainment facility is usually separated into different isolated zones. Each zone is totally independent from the others and has its own ventilation system. In our scenario, the suite is composed of an isolated zone. It contains four laboratories for conducting different research and a chamber as a public area for researchers. As shown in Figure 2, rooms start from No. 1 to No. 4 are biocontainment laboratories and room No. 5 is the public chamber. The suite must comply with basic biocontainment laboratory standards. In this case, the suite should only allow certain authorized researchers to have access. Every authentication should be logged for audit purposes. Each laboratory within this suite has to constantly maintain a negative differential air pressure relative to the chamber room in order to avoid airborne hazards exposures. Meanwhile, different laboratories maintain independent air pressure among each other and airborne hazards exposures are prevented through controlling doors by using interlock system installed in chamber. Each laboratory has to keep the temperature in a certain range. When a laboratory is operational, the fume hood fan in the room has to start working steadily. Furthermore, aside from normal operations, laboratories need to be decontaminated when hazards have been detected as exposed. In order to handle this situation, laboratories usually have two modes: normal mode and decontamination mode (DECON mode). For this purpose, each room has a strobe that is used to indicate emergency situations including power failure, DECON mode, door-open-too-long alarm, etc. When in DECON mode, laboratories would only allow researchers to exit and reject any ordinary user's entering requests except requests from users with administrator privilege. No matter in what emergency situations, certain functionalities of a laboratory have to work – for example, user authentication and the door access system, directional airflow control, fire alarm, *etc*. Besides, in order to prevent cross-contamination, no two doors in an isolated zone should be allowed to open at the same time. This is enforced by the magnetic door interlock subsystem, except when the fire alarm is being triggered. When the fire is being detected, the fire control subsystem will override the interlock subsystem and unlock all doors at the same time.

The logic structure of the suite BAS is pictured in Figure 3. Each laboratory has a controller (the Laboratory Controller) that is used to conduct Flow Tracking and Feedback Controls. Flow tracking controls, a.k.a. offset controls, maintain a constant larger amount

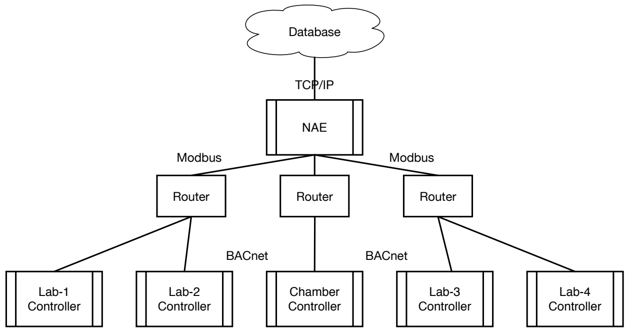

Figure 3: Biocontainment Laboratory Scenario Structure

of exhaust flow than supply flow. For example, some laboratories require to maintain 150 cfm more exhaust airflow than supply airflow of all time. Because more air is exhausted than supplied in a relatively airtight environment (such as a laboratory with door closed), the laboratory differential air pressure is negative. When the door is open, however, because the nature of air dynamics the differential pressure would drop to zero rapidly. Therefore, when the door is open, the differential pressure is monitored by a feedback loop using feedback controls. Feedback controls constantly measure the differential pressure comparing to the reference (in this case, the chamber). When the differential pressure drops below the threshold, the feedback loop would temporarily adjust exhaust and supply airflow within the allowed range in order to maintain negative differential pressure (by boosting the exhaust airflow and decreasing the supply airflow). If those two steps fail to maintain relative negative pressure of the laboratory, DECON mode will be triggered. Chamber Controller controls air pressure using Direct Pressure Controls, which modulate supply and exhaust dampers dynamically for maintaining negative differential pressure against outdoor air pressure. The Chamber Controller is also responsible for keeping its air pressure as stable as possible. All laboratories are relatively separate from each other and maintain their own access control, while Chamber Controller controls functionalities for the whole suite, e.g., the magnetic door interlock for all doors in this zone, the fire detector and alarm, etc. The Network Automation Engine (NAE) is a central control device that is designed to provide a web-based user interface, to create and push administrator defined policies and a database for each controller. On the very top of the figure is a database, which could be a cloud server or a local data center that is used to store policies, data, logs, and authorization information. Controllers are connected to a router through the BACnet protocol; routers communicate with the NAE through the Modbus protocol. The NAE communicates with database using Ethernet. Suite administration network is separated from Internet access using VLan. The detailed functionalities that are supported by Laboratory Controllers, Chamber Controllers, and the NAE are described below.

4.2 Laboratory Controller

1. Maintains 150 cfm exhaust airflow more than supply airflow.
2. Senses air pressure and sends air pressure status to the Chamber Controller.
3. Senses the laboratory door position; when one or more doors are open, receives the Chamber Controller overwriting command for the exhaust and supply airflows.
4. Flashes strobe: when differential pressure passes the threshold, when one or more doors open too long, when power failure, when the DECON mode is on, etc.

5. Senses the laboratory door position; when door is open for more than 20 seconds, twinkles strobe.
6. Reads access card reader, the access button statuses; authenticates with local database; controls the door lock.
7. Records all actions, including access authentication, current room temperature, current air pressure, laboratory status, etc.
8. Maintains temperature at 65 - 80 F.
9. Adjusts temperature according to user setup from the panel.
10. In DECON mode, only allows exit and turns on strobe.
11. Senses fire detector and notifies the NAE.
12. When fire alarm triggers, unlock door.
13. Maintains heartbeat signal with the NAE.

4.3 Chamber Controller

1. Senses outdoor air pressure and adjusts exhaust/supply airflows for No. 5 chamber room (Direct Pressure Controls).
2. Receives air pressure from different Laboratory Controllers in the same zone and calculates differential air pressure against itself respectively (Feedback Controls).
3. Flashes the chamber strobe when any laboratory strobe flashes.
4. Senses all door position sensors in this zone and maintains the magnetic interlock.
5. Reads access card reader, the access button statuses; authenticates with local database; and controls the door lock.
6. Records all actions, including access authentication, current room temperature, current air pressure, laboratory status, etc.
7. Maintains temperature at 65 - 80 F.
8. Adjusts temperature according to user setup from panel.
9. In DECON mode, only allows exit and turns on strobe.
10. Senses fire detector and notifies the NAE.
11. When fire alarm triggers, unlock all doors, and disable the magnetic interlock.
12. Maintains heartbeat signal with the NAE.

4.4 NAE

1. Monitors heartbeat signals with all devices, and records their statuses.
2. Receives user authentication requests from controllers and authorizes access control according to the queries from database.
3. Pushes/updates/creates database information and policies for controllers.
4. Provides web interface.
5. Sends out/cancels DECON mode signal.
6. Checks statuses of different fire detectors through room controllers and automatically turns on/off fire alarm by coordinating with controllers.
7. Automatically sends message/email to administrators based on policies.

5. VULNERABILITIES

In the scenario discussed above, there are certain vulnerabilities. First, as many researchers pointed out, both BACnet and Modbus data are transmitted in plaintext. It is trivial for attackers to figure out what each packet means and what control commands are, if they manage to access the network [42]. That a number of building control devices can be found through Shodan suggests that many control network is directly connected to Internet. Although some control network is well separated through VLan, like the one our scenario describes, it cannot prevent malicious attackers from attaching unauthorized devices and eavesdropping on control network or changing network configuration through compromised administration hosts.

While it is not hard to directly add encryption features to existing protocols, for this specific case, however, even there is a proper encryption method, it is still possible for attackers to inject packets, if it lacks robust infrastructure to support it. For example, SSL without a valid certificate issued by trusted third party is vulnerable for Man-in-the-Middle attack. Because BAS is designed for steady daily use and expected to reliably work for decades, attacker has plenty of opportunities to try different methods out. In DEF CON 19 Kennedy demonstrated how one can sniff encryption keys out of controllers that use ZWave protocol with AES encryption during initialization of devices [12]. The fundamental problem is that those authorized devices on the network are lacking a reliable mechanism to recognize each other and delegate trust. Second, because current BAS lacks identification, privilege separation is very limited and obscure. Devices either have all privileges or none. There are no effective ways to enforce fine-grained access control. Hence, if one device on the network is compromised it is very likely that it will open the gate for attackers to impersonate other devices and send out arbitrary control commands on their behalf. For example, if one laboratory controller is controlled by attackers, it is not hard for them to use it to impersonate NAE and send DECON mode signal or fire alarm signal to all controllers, therefore disrupting daily operations.

Threats not only come from network, but also from physical access. USB is a common built-in connector for microcontrollers. Many controllers contain USB connectors due to its flexibility and wide support from peripherals, but the convenience comes at a cost. Wide support makes the code base of USB driver bloated and complex and prone to vulnerabilities. USB is a dangerous port. USB devices include a microcontroller hidden from users. A device indicates its capabilities through a descriptor stored in the firmware. It is possible for attackers to reprogram firmware, use a USB flash drive to impersonate different devices for injecting malware, or for spying on the system [20]. Alternatively, some USB peripherals have the ability to use direct memory access (DMA) transferring data to main memory without operating system supervision. It potentially gives attackers a chance to modify main memory. Besides, the majority of OSes support booting from USB drives. When it comes to PLC, it is a common design for using USB to update system components, adding new policies, and debugging programs even loading new OSes. While it makes the administrator's job easier, attackers can take advantage of such a path to bypass system-level security precautions and load malicious programs and boot modified operating system as [16] has demonstrated. The most well-known cyber weapon, Stuxnet is believed to have spread through a USB thumb drive to the BAS [14]. An RS-232 serial port is even worse. As a legacy port it is still widely supported by PLCs. Serial ports do not support authentication. Systems often allow remote access through serial ports [33].

One of the important constraints of BAS is real time. Laboratory must constantly maintain negative differential pressure under any circumstance. PLC highly relies on the scheduler of real-time operating system (RTOS) to regularly execute programs, check sensors and adjust actuators before deadline. The cost of missing a deadline is potential hazards exposures, which is unacceptable. Current RTOS, however, is designed with the assumption that all programs work as specified. Only with this assumption the real-time scheduler can guarantee that all constraints be satisfied, which is hardly the case for control system like BAS. One reason is the lack of awareness. People tend to believe field devices such as PLCs cannot be compromised. Therefore security is not an integral part of most control systems but merely an after thought. "There is no public security certification process for control system devices and vendors are unwilling to share information on security incident," as mentioned at Black Hat 2014 by Stefan Luders, computer security officer at European scientific research center (CERN) [30]. Attackers who compromise a process (e.g., temperature control process) in laboratory controller can ceaselessly fork itself and consume the CPU time of airflow control process by fairly divide it's share, hence deliberately sabotage the real-time constraint. Cyber-Physical Systems involve multiple controllers cooperating through network. Compromised chamber controller can intentionally delay the override airflow response to laboratory controllers, postpone multiple parties' communication and make them miss deadline or it can overwhelm the receiving party with a large number of junk messages.

Although real-time constraint is critical, the highest priority is stability. BAS is designed to run for decades. Once a BAS is installed, the cost of upgrade is considerably high and usually unacceptable. For example, think about the cost and effort of disabling security systems of airport or subway stations. The stability requirement of BAS makes the whole system vulnerable to constantly trying attackers. Especially with the dawn of Internet of Things, home owner can access from anywhere in the world to remotely control lights, door locks, house temperature, electric appliances, water valves, alarm system, garage door, the ability to open and close shades and blinds, or even to turn on music and crank up the volume. While appealing, it also gives attackers a unprecedented open world. The incident of the control system at Google Australia office being compromised by researchers in 2013 shows how serious and realistic such a problem is [17]. The headquarter uses Tridium's Niagara Framework which contains numerous vulnerabilities that allow researchers to obtain administrative password and access control panel to disrupt the system; publicly available web interface also make it trivial to exploit such vulnerabilities.

Study of software reliability shows that industrial software system in general contains 2-75 bugs per 1000 lines of executable code [38], which is considerably higher than normal code. On the other hand, patching of security vulnerabilities in control systems is difficult and usually have to wait when maintenance is performed. These issues make control systems like BAS an attractive target for attackers. Now in light of the advance of CPS, control systems are highly automated and digitally interconnected with high-level applications and Internet. Considering the easy accessibility, poor protection, and potential benefit the hackers can gain, PLCs have become the low hanging fruits for cyber attacks. The environment and assumption of real-time operating system for BAS has changed. We can no longer confidently assume that all applications running on top of the OS are benign. For example, with the new setup, a program with buffer overflow vulnerabilities could possibly allow attackers to inject malicious code remotely, deceiving the scheduler to plunder CPU time slots, therefore intentionally starving critical applications, forcing BAS to violate real-time constraints.

It is well-known that device drivers have error rates 3 to 7 times higher than other code [10]. In general-purpose OSes, device drivers contribute to a large percentage of vulnerabilities. It is reasonable for general-purpose OSes since general-purpose OSes need to support as many peripherals as possible. Convenience and compatibility are higher concerns. Besides, for general-purpose OSes, peripherals are usually keyboards, monitors, hard disks which are relatively non-critical and device drivers come from various vendors which is impossible to guarantee the quality. It is the applications such as browser that casts a bigger threat for users rather than device drivers. When it comes to industrial control and BAS, however, the situation has changed. PLCs and NAE usually only need to support limited number of devices and peripherals. Although the number is small, they are critical for safety and security of the whole system since device drivers directly interact with sensors and actuators. If the device driver is compromised it can easily fool the OS, control actuators arbitrarily and stealthily. Malware like Stuxnet demonstrates this possibility [26]. While it is challenging for application programmers to make sure the applications are high-assurance by applying methods such as formal verification, it does place a high demand for such rigorous assurance on the underlying RTOS. BAS not only requires the OS of PLC to guarantee functional correctness but also have fault tolerance and attack resilience, which means the OS of controllers should not crash (guaranteed) and in case if critical system errors it has certain ways to recover the system. More importantly, the OS should strive to execute the most critical applications before the deadline even when malicious apps are running on the same devices trying to sabotage it. In the case of laboratory control, this means maintaining negative differential air pressure, and notifying BAS about failures as soon as possible.

6. SECURE RTOS ARCHITECTURE

To understand how we intend to model and specify the requirements that critical applications need to convey to the RTOS, and why we choose to use a microkernel architecture for the embedded controller RTOS, it is important to first introduce the basics of the microkernel architecture.

6.1 Microkernel Architecture

The unique characteristics of a biocontainment facility place a challenge on the real-time operating system for controllers. None of the current commercial monolithic systems can confidently guarantee to satisfy such high security and stability requirements. As Tanenbaum pointed out [43], current operating systems are complex software. The Linux kernel has over 2.5 million lines of code (LOC) and Windows systems are even larger, which makes it impossible to formally verify the functional correctness of a monolithic kernel.

By using a layered approach with a formally-verified microkernel (referred to as kernel hereafter), we can enforce security and temporal constraints that could not be enforced using a monolithic kernel. Since the MINIX implementation is based on one of the purest forms of the microkernel architecture, we will use MINIX as an example to explain the architecture.

Kernel's roles are to provide the process environment and inter-process communication/synchronization primitives to the processes (apps) above it. Kernel itself is not a process, but all other software modules residing on top of kernel are running as processes. Each process runs in its own independent address space and can only communicate with other processes using kernel's inter-process communication primitives. Kernel runs in the highest processor privilege level and all processes running above kernel run in the lowest privilege level.

In MINIX, the processes running on top of kernel are structured in three software layers. A process in one layer uses services provided by those in the layers below it. The top-most layer is the user process layer. The layer below it is the server layer, which provides system calls to the user processes. The server layer consists of the Process Manager (PM), File System (FS), Network server (INET), *etc*. PM provides process/memory related system calls such as fork and exec. FS provides file related system calls, such as read and write. INET provides TCP/IP services. The lowest software layer is the device driver layer.

We now compare behaviors of the microkernel OS architecture and the monolithic OS architecture. In a monolithic OS, such as Linux and Unix, all OS functions (including interrupt handlers, device drivers, and system call functions) are implemented as separate functions in one big OS program, all in the same address space and usually run at the highest privilege level. Therefore, potential vulnerabilities in any function in the OS can allow an attacker to issue any sensitive operations and could destroy other OS functions. Consider execution of a system call, which is assumed to be implemented by a sequence of function calls. When a user process issues a system call, a software interrupt transfers control to the OS program (in a monolithic OS) or kernel (in the microkernel). In a monolithic OS, all functions in the OS are executed on behalf of the user process and no process switching takes place, as a result no module can monitor these function calls.

In a microkernel, on the other hand, if the sequence of functions are implemented by different processes, each such function call/return will cause a process switching. We counted the number of process switchings required to complete a read system call onto a RAM disk drive. Compared with 0 in the monolithic OS, MINIX requires 12 process switchings. Since process switching is costly, most general purpose OSes use the monolithic architecture for performance reasons. However, our application of BAS is not CPU intensive (this will be explained later). In fact, from the security point of view, the multiple process switchings provide an advantage in the BAS system! It means that each inter-process communication and its frequency, data transfer operations across process boundaries, and executions of sensitive operations, such as I/O operations, all go through kernel and that kernel can monitor each of these operations. Furthermore, since kernel has access to the message buffer contents in the inter-process communication primitives, kernel can even control which requests (specified in the message) may be sent to the destination server.

How the functionality of each BAS component is divided into a separate process must be determined by two factors. From the performance point of view, it would be better to put more functionality into one process, since it would require less process switching. However, from the security point of view, it would be better to put less functionality into one process, since process must communicate with one another more often and the microkernel can control such traffic in more fine-grained manner.

Consider the following function in our BAS system, involving three components: a Fire Sensor driver and a Fire Alarm component on a chamber controller and NAE. Each chamber is equipped with one chamber controller, and there is only one NAE in a zone. Fire Sensor Driver periodically senses the fire sensor. Suppose that (1) Fire Sensor Driver of one chamber detects a fire, (2) it informs NAE of the fire, (3) then, NAE switches the mode from normal to DECON and sends requests to the Fire Alarm components of all chambers, and (4) upon a receipt of the request, the Fire Alarm component triggers alarm siren and unlocks all the doors to the chamber. In this scenario, we assume three processes to implement the above functionality in the following discussions; (1) fire sensor process which senses a fire and informs NAE of the fire, (2) NAE process that triggers the DECON mode, and (3) Fire Alarm process which receives a request from NAE and triggers the alarm siren and unlocks the doors of the chamber.

6.2 Access Control on Inter-Process Communication

We will specify the security policy in the form of access control on inter-process communication. A formal language is used to specify the following: (1) for each sender process, which processes it can communicate with (send messages to), and (2) for each communication, (2a) which functional services (operations) the sender process can request, and (2b) how often/seldom each communication must be issued (communication frequency; that is the upper- and lower-bounds of communication frequency). For

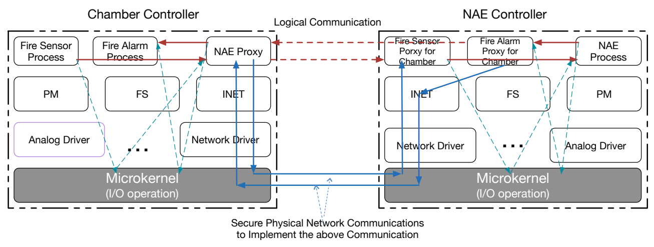
Chamber Controller
Logical Communication
NAE Controller
Fire Sensor Process
Fire Alarm Process
NAE Proxy
Fire Sensor Porxy for Chamber
Fire Alarm Proxy for Chamber
NAE Process
PM
FS
INET
INET
FS
PM
Analog Driver
...
Network Driver
Network Driver
...
Analog Driver
Microkernel (I/O operation)
Microkernel (I/O operation)
Secure Physical Network Communications to Implement the above Communication

Figure 4: Proxy Based Communication

example, consider the above fire sensor scenario. The fire sensor process can send a request only to NAE. The request must be either "no-operation" or "report fire." The frequency of the communication must be in the range between once every 45 seconds (lower bound) and once every 1 second (upper bound). NAE can send a request to the fire alarm process of each chamber, among other request messages since NAE is involved in many tasks, and its operation must be either "no-operation" or "trigger siren and lock doors." Its communication frequency must be in the range between once every 45 seconds and once every 1 second.

Such formal security policy specification is compiled into a table information (called access control table (ACT)), which is compiled with the microkernel to be stored in the kernel address space. The microkernel refers to the ACT to control each inter-process communication. If a process tries to send a request to another process, the request is granted only if the ACT has permission information for the request; otherwise, the request is rejected.

In addition to the access control specification on IPC, we allow to specify the characteristics of each process, such as the worst execution time. This information is also compiled into the kernel address space and the kernel uses the information to monitor the process behaviors.

6.3 Proxy Based Communication

When it comes to processes from different controllers that need to cooperate together for a task, the proposed RTOS architecture distributes security control using proxy-based communication (Figure 4) to enforce access control on the inter-process communication. Suppose that a chamber controller has a Fire Sensor process and a Fire Alarm process, and the NAE controller has a NAE process. Then, the Fire Sensor process on one chamber controller must have a proxy for the NAE process, and the NAE controller houses one Fire Sensor Proxy and one Fire Alarm proxy for each chamber controller. Each process does not directly communicate with a process on another controller. Instead, it communicates with the proxy on the same controller using the kernel's IPC service. Therefore, the security policy implemented in the form of ACT is enforced among processes within the same controller. For example, the fire sensor task sends a "report fire" request to the NAE proxy, which is allowed according to the ACT.

The proxy then communicates with the senders proxy on another controller using the inter-controller (network) communication. In our example, the NAE proxy on the chamber controller sends the "report-fire" message to the fire sensor (the sender of the original request) proxy on the NAE controller. This network communication goes through the INET server, the network driver, and the kernel (to actually issue sensitive I/O operation on the network device) and each such communication is also dictated by the kernel with ACT. The actual network communication will be secured using TPM (discussed later). When the Fire Sensor proxy on the NAE controller receives the "report fire" request, it forwards the request to the NAE using the kernel IPC. Therefore, the kernel enforces the security policy in this communication by referring to its ACT.

Once the NAE process receives the request, it sends a "trigger alarm and lock door" request to the Fire Alarm proxies for all the chambers on the NEA controller. Each Fire Alarm proxy forwards the request to the NAE proxy on the corresponding chamber controller. On each chamber controller, upon a receipt of the request, the NAE proxy forwards the "trigger alarm and lock door" request to the Fire Alarm process. Then, the Fire Alarm process triggers alarm and lock all the doors of the chamber. This way, by using proxies, we map global (inter-processor) security to local (in-processor) security issues to be controlled by each microkernel.

6.4 Temporal Requirements

Biocontainment facilities require the system to maintain negative differential air pressure between rooms. When the atmospheric pressure changes, the internal pressures must be modified in real time. Likewise, when a scientist opens a door, the differential pressure causes air to rush in from the outside. The door can only remain open for a bounded time before the pressure would equalize without intervention. Thus, even the opening and closing of doors must be carefully choreographed to ensure the safe operation of the facility.

In a real-time system, a unit of work to achieve a system function is refereed to as a *task*. In our Fire Alarm example, a sequence of executions from (1) detection of a fire by the fire sensor process to (2) triggering alarm siren and unlocking all the doors by the fire alarm process is a task. Each task has timing specifications called *release time* and *deadline*.

- release time is the time at which the task becomes eligible for execution. For example, the time at which (1) detection of a fire occurs in the above scenario is a release time.
- deadline is the time by which the execution of the task must complete. In most systems, a deadline is specified relative to a release time. For example, if (2) triggering siren and unlocking all doors in the task must have completed in 10 seconds after (1) occurs, the deadline is 10 seconds.

Virtually all modern real-time systems are implemented on a periodic task driver using fixed priority-based scheduling, commonly referred to as a Rate Monotonic Scheduler (RMS)[29]. In a periodic task driver, each task is scheduled and executed periodically. If an input that triggers a release of a task arrives at the system periodically, the period is defined to be the same length as the deadline. But in most systems, an input that would trigger a release of a task does not arrive periodically. For example, in the above system, fires would not occur periodically. In such a system, the sensor periodically senses an input (called polling). If an input is detected, it proceeds the task. If the sensor does not detect an input, it does not execute any further action in the period. In such a system, the period is defined to be a half the length of its deadline. For example, in the above system, if the deadline is 10 seconds, the period should be 5 seconds; that is, the fire sensor driver should sense the fire sensor once every 5 seconds. This means that if a task has a longer deadline, its period becomes longer; that is, the task is scheduled less frequently and consumes less CPU. Since the deadlines of BAS tasks are assumed to be longer (because the inertia of air is large) than those found in other applications such as automotive and aircraft systems, the BAS system is considered to be non-CPU intensive, and we have chosen the microkernel based architecture by focusing on the security rather than efficiency.

In virtually all systems using the periodic task driver with RMS, the task periods are defined to be harmonic (that is, in any pair of periods, one is multiple of the other) and in phase (that is, all the periods start at the same time). In such a system, for a given task set, if there exists a priority assignment to the tasks which would satisfy the deadlines of all tasks, the priority assignment based on RMS will satisfy all the deadlines (such a scheduler is called an optimal scheduler)[29]. Furthermore, let $U = \sum_{i=1}^{n}(Ex(T_i)/Pd(T_i))$, where $Ex(T_i)$ is the execution time of task T_i, and $Pd(T_i)$ is the period of T_i. Then, $U <= 1$ is the necessary and sufficient condition to satisfy all the deadlines[27]. If U becomes greater than 1, the system cannot satisfy all the deadlines

With our proxy based communication, we must carefully consider in determining a period for a given task deadline. Since network communications take place only between proxies (not between regular processes or between a regular process and a proxy) in our system, this fact may make real-time analysis easier. On the other hand, challenging factors would include (1) proxies are extra processes which will work as an overhead in the total execution time of the task, and (2) network communications between proxies are asynchronous and possibly two communicating proxies may run at different frequencies (periods), which may yield additional delays. Further studies will be required for temporal analysis of the proxy based communication.

6.5 Legacy Device Support

Backward compatibility is an important factor for BAS. Building control system are designed to last decades. There are so many legacy devices from different vendors in existing system that one can reasonably assume that those would not be replaced in a foreseeable future. However legacy devices usually lack security consideration and therefore are vulnerable to potential threats. The proposed architecture minimizes threats and provides backward compatibility through proxies. A controller installs a proxy for a legacy device it wants to control. The legacy device communicates the controller using the secure network service. The target of the communication is the proxy for the device. Then, the necessary security is enforced by the microkernel. This approach allows the proposed RTOS work together with old systems, and enforces security constraints on all communications involving the legacy device.

6.6 Trust Delegation

One of the unique characteristics of BAS is the requirement of identification and trust delegation. BAS after being set up is considerably stable, barely having external devices being attached on the network. Different controllers are responsible for different control tasks, and only communicate with a constant subset of devices. To prevent unauthorized malicious devices from attaching to the network and spoofing others, BAS demands non-forgeable identities of each device for communication and authorization. It is only achievable with OS level enforcement and hardware support. One of the well-known technologies for building hardware support identification is Trusted Platform Module (TPM). TPM is a cryptographic coprocessor chip designed by Trusted Computing Group (TCG) that leverages hard-to-alter characteristics of hardware cryptography capability designed to address the problem of lack of trust in electric devices and to provide evidence for system status. TPM has been used for different purposes, including providing uniquely non-forgeable identify for devices and applying measured boot. It provides various cryptographic functions and secure data vault for key storage that guarantees the private key would never leave the device without physical tampering. TCG defines specifications for TPM driver and key management.

TPM suits the nature of BAS for building a chain of trust for all components, from low level hardware BIOS, Unified Extensible Firmware Interface (UEFI) to high level applications, and providing verifiable evidence for remote attestation to prove that the device has been intact since last time it was booted. Besides, TPM offers different cryptographic algorithms that can be used to encrypt network sessions, build communication channels on top of BACnet, Modbus and other legacy protocols, verify application certificates, and provide evidence for secure application loader. This security reinforcement not only prevents devices from being spoofed, but also gives BAS applications the capability to refine access and communication privilege at system level.

6.7 Software Architecture

With our investigation and scenario analysis, we propose a micro-separation kernel based secure RTOS architecture Our proposed OS architecture is based on MINIX, a well-known, ever-evolving standard microkernel based OS. MINIX emphasizes on stability. The latest version is MINIX 3 which targets embedded devices. MINIX kernel only provides a tiny portion of OS critical functionalities such as real-time scheduler, process control blocks (PCBs), interrupt handler, IPC, and system clock. In our proposed architecture, besides the components of current MINIX kernel with built-in separation and self-verification enhancements, we include direct TPM support in the kernel space, which empowers the kernel with cryptographic computation capability and the strength of verifying other components before loading them. Between kernel and other components is the security kernel/user interface. Security kernel/user interface offers TPM's privileged APIs to user space processes. Typical usage includes but is not limited to encrypting/decrypting network sessions, querying device identity and status, answering remote attestation, and most importantly, helping Secure Process Manager (SPM) to achieve whitelist/blacklist binary loading and monitoring. Such an architecture makes sure that every component on top of kernel can be verified and authenticated by kernel, while the kernel itself is verified and measured by security loader or trusted loader and hardware (*e.g.*, UEFI, TPM) which can be audited by remote servers, therefore extending trust from bottom to the top.

On top of security kernel/user interface are isolated system servers including device drivers, the file server, inet server, secure process manager (SPM), and building automation server (BAS), which is responsible for communication with analog devices. Secure process manager is an enhanced version of MINIX 3 process manager. In MINIX, process manager is responsible for loading/forking/killing processes, message passing and synchronizing processes' status with Kernel. SPM is enhanced with separation and verification capability. With the help of security kernel/user interface, SPM verifies applications and reinforces application separation by setting up communication boundaries, virtual memory, restricting privileges, as well as limiting resource usage. BAS server guarantees to dispatch query and response of analog devices in real time at the system's best effort. BAS works tightly with analog device driver. It monitors and transfers analog signals into proper OS data structure and forwards messages between applications and device drivers. On top of the operating system are applications and corresponding proxies. They are scheduled according to their priority. Proxies implement network policies and filter out potential malicious traffic before passing to applications.

6.8 Case Study: Airflow Control in the Proposed Architecture

Airflow control task is a good example to demonstrate how the proposed RTOS architecture works in a biocontainment facility scenario. In a laboratory, airflow control is one of the most critical functions to prevent cross-contamination. Maintaining proper airflow control requires multiple devices cooperating together, which makes it vulnerability to potential cyber attacks.

Figure5 illustrates each step of the task. Three types of controllers are involved in the airflow task; the lab controller for each laboratory and the chamber controller in the chamber with in the same zone and the NAE controller. The figure only shows the processes involved in the airflow task, and does not show the microkernel, the device driver processes, and the OS server processes. The solid arrows represent IPCs that are controlled by the microkernel by referring to the ACTs, and the dotted arrows represent network communications which is encrypted with unique device identity provided through TPM and process information assigned by kernel. In the Airflow control task, two independent but related sub-tasks are executed concurrently; one sub-task monitors the differential pressure between each laboratory and the chamber, and another regulates the airflow into and from each laboratory.

In the first sub-task, the Differential Pressure Sensor process periodically sends the differential pressure between laboratory and the chamber to the Differential Pressure process (via IPC denoted by ***a***). The Differential Pressure process sends the differential pressure data to the NAE process for logging. This communication is done by the following sequence of IPC and Network communications; (1) the Differential Pressure process sends the data to the NAE proxy via an IPC ***b***, (2) the NAE proxy sends the data to the Airflow Control Proxy for this chamber on the NAE controller via the network communication ***c***, and (3) the Differential Pressure proxy for the chamber forwards the data to the NAE process ***d***. When the NAE process receives the data, it logs the data. The NAE process also monitors the received data and if it detects that the differential pressure has been below the predefined threshold value for a certain length of time (which is an unacceptable situation), the NAE process changes the operation mode of the zone from the normal to the DECON mode. The NAE process broadcasts a request to all the related processes in the zone, commanding to switch to the DECON mode. In our scenario of the Airflow task, the request command is sent to the Differential Pressure process of the chamber and the Airflow Control process of all labs in the zone via their corresponding proxies ***e, f, h, i, j***.

In the second sub-task, the Airflow Sensor/Actuator process periodically monitors the airflow to/from the laboratory and reports the airflow to the Airflow Control process (IPC denoted by ***1***). The Airflow Control process communicates with the door sensor driver in the device driver layer (not shown in the figure) to check if the door is open. If the door is closed (when a big change in the airflow is not expected), the Airflow Control process calculates the difference between the appropriate exhaust airflow and the supply airflow, and sends a command to the Airflow Sensor/Actuator process to adjust the airflow (denoted by ***2***). If the door is open (when a big change in the airflow may occur), the Airflow Control process alone may not be able to maintain the negative differential pressure relative to the chamber. Therefore, the Airflow Control process reports this to the Differential Pressure process in the Chamber Controller. This is done by the sequence of IPC and Network communications denoted by ***3, 4, 5***. As described above, since the Differential Pressure process receives the differential pressure data among the chamber and all the labs, the process calculates the necessary airflow of the Lab to maintain the required negative pressure and sends a command to the Airflow Control process in the Lab controller, commanding to adjust the airflow to the lab appropriately. This is done by a sequence of communications among the processes and proxies denoted by ***6, 7, 8***. Upon a receipt of the command, Airflow Control process sends

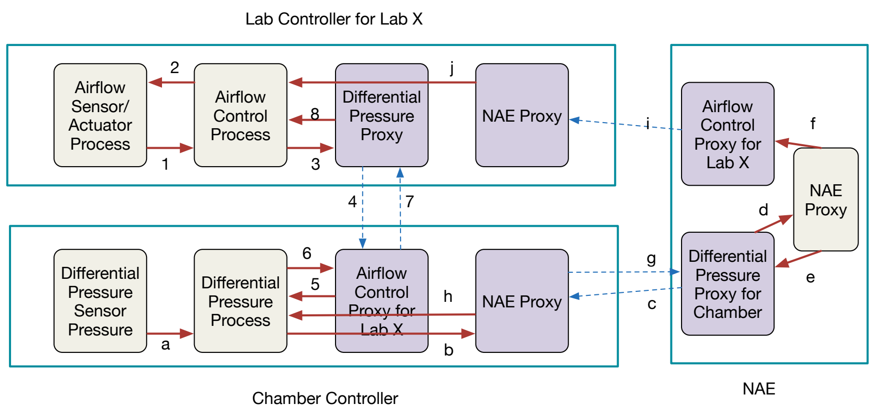
Lab Controller for Lab X
Airflow Sensor/ Actuator Process
Airflow Control Process
Differential Pressure Proxy
NAE Proxy
Differential Pressure Sensor Pressure
Differential Pressure Process
Airflow Control Proxy for Lab X
NAE Proxy
Airflow Control Proxy for Lab X
NAE Proxy
Differential Pressure Proxy for Chamber
Chamber Controller
NAE
1
2
3
4
5
6
7
8
a
b
c
d
e
f
g
h
i
j

Figure 5: Airflow Control with Secure RTOS

a command to the Airflow Sensor/Actuator process to adjust the airflow (denoted by ***2***).

In the above scenario, all IPC communications are regulated by the microkernels on the associated controllers using the ACTs, and all the network communications are assumed to satisfy real-time requirement.

7. POTENTIAL ATTACKS AND MITIGATIONS WITH SECURE RTOS

In this section we discuss various potential attacks of the scenario described above and explain how the proposed RTOS architecture can help mitigate these threats.

7.1 Attack on Software

As discussed above, controllers in BAS have quite a few control tasks that involve multiple processes. It is reasonable to assume that some of them might contain vulnerabilities such as buffer overflow, memory leakage, *etc*. that would allow attackers to gain remote access in the local controllers. For example, the attackers might be able to take total control of temperature control task and try to take the system down from there. The false data injection attack is a common method for trying to compromise peer processes and escalate privilege. Once attackers gain access to a controller by compromised processes, they might try to send message, request system call to access the local database, or to turn on the DECON flag, *etc*. However this type of attack can be effectively detected and prevented by using microkernel architecture. In microkernel architecture all the inter-process communications go through kernel message passing. Although it might involve many context switches and make the system slower than a monolithic kernel, the benefit is that it natively separates processes into independent modules. Kernel theoretically monitors all communication and can control the communication flow by policies. For example, the temperature control process cannot request conversation with database service and the DECON control process; the kernel will simply reject this communication attempt. On the other hand, microkernel architecture can help prevent attacks from spreading through the network. Each control task can only communicate with related processes on other controllers through a proxy. In the worst scenario, even the temperature control process is totally compromised through network, it is very unlikely that the threat will impact more critical control loop such as airflow control and differential control.

In a controller there are multiple processes sharing the same hardware resources simultaneously and they all have real-time constraints. Thus, one compromised process can potentially sabotage other processes on the same controller by conducting resource consumption attack that consumes memory space as much as it can or hogs CPU utility by forking multiple threads. The proposed architecture can help avoid this type of attack by modeling each process's running characteristics, and the kernel can detect deviation from the characteristics and stop the attack. Microkernel architecture has good separation. All processes are running in separate virtual memory spaces. Therefore even if a malicious process tries to consume memory it will only exhaust its own quota. On the other hand, the kernel prevents a specific process from exhausting CPU capacity by assigning each process a share of time slots based on the process's running characteristics. When the usage is more than it is assigned the specific process will be preempted, therefore only itself will miss its deadline.

Other threats for modern OS include virus, backdoor, and rootkit, *etc*. For viruses that take advantage of application process vulnerabilities, they might spread themselves through network. Due to the communication restriction between controllers enforced by the ACT, the impact will be limited. For viruses due to system process vulnerabilities, they might have a severe impact. However, in microkernel architecture kernel is the only thin layer running in the privilege mode. This minimizes the risk to as low as possible. Moreover, the capability of formally verifying the kernel and related modules dramatically reduces the number of such vulnerabilities. Even if a process in the controller is compromised by attackers, the attackers can only modify current process's context. To execute unauthorized binary (such as, plants backdoor and sensitive I/O operations) and communicate with other processes, it requires the kernel support, which is strictly regulated.

7.2 Network Attack

The biggest threat for industrial control system is from networks, either through unguarded Internet access or wireless interfaces. One popular attack is the deception attack, which represents a type of attack that try to gain control of controllers through unauthorized third devices. For the scenario described above, attackers might attach arbitrary devices on the control network and impersonate the chamber controller for deceiving the laboratory controllers into accepting fake override command. Similar strategy might be use to forge a fire alarm signal on behalf of the NAE for trying to trigger the amble strobe in a laboratory therefore conveniently lock down the whole zone. With the proposed RTOS architecture this type of attack would not be a problem. The proposed architecture enforces devices with unforgeable identities supported by TPM. Controllers would only be able to negotiate and initialize conversations with each other if they are authorized to communicate by presented cryptographic evidence. Thus prevent unauthorized conversion at the first place.

Another type of well-known attack is the replay attack. Similar to deception attacks, replay attacks intend to manipulate receiving controllers by using spoofing messages on behalf of others. The difference is that instead of creating arbitrary messages, replay attack collects legitimate control message on the network, tweaks it and send it later. This attack might happen, for example due to communication protocol deficiency. In the scenario described above, a malicious attacker might keep sending outdated override commands to the laboratory controller and deviate the laboratory controller to cause positive pressure. However, the proxy-based design should be able to prevent this type of attack. The benefit of proxy-to-proxy communication separate control loop with network. Because of the separation, proxies can be customized according to different requirements of the process it serves. In general proxies in different controllers can keep track of the session number, define the rate of communication between different devices, guarantee the freshness of each override command.

The same protection applies to denial of service (DoS) attack too. Attacker might simply sabotage the control system by applying denial of service attack or stressing control network with DoS attack to camouflage more stealthy attacks. Either the DoS attack is from an unauthorized device or through a compromised one, the effect will not create more damage than jamming the network therefore, delays cooperation. Although this is bad enough for a real-time control system, there is not much the embedded controllers with limited resources can do. The solution most likely requires network level detection and prevention mechanisms. However, all control processes that require network cooperation should enforce limited time for expecting responses. With the kernel monitoring the upper and lower bound of communication frequencies of processes, the controller can guarantee that even under DoS attack, it still can maintain local control loop independently, thus help minimize the damage.

8. RELATED WORK

CPS research is still in its early stage. As Madhukar, Eric *et al.* in [3] point out there are certain security challenges such as measuring confidentiality, trust management, *etc*. There are decent amount of researches on CPS control theory and analysis of different type of attacks, for example, [9] identifies and defines the problem of secure control; [48] assesses vulnerabilities and categorizes CPS attacks. However, there are limited number of researches propose systematically security solution for CPS. When it comes to Building Automation System, the research is mainly focus on security of SCADA system. In [21] Wolfgang, Georg *et al.* conducted a investigation of BAS communication system, in which they emphasized the important of formal specification. In [34] the author presented an approach of developing safety and security related application of BAS.

On the other hand, microkernel architecture has been a popular topic in recent years once again. Several companies and academic research groups are dedicated to designing more secure OS. The most outstanding microekernel is MINIX 3. MINIX 3 is a Unix-like operating system that focuses on stability and fault tolerance. MINIX 3 is built on top of a pure microkernel with under 4000 lines of executable code. The superior contribution of MINIX 3 is that it is the only operating system so far that has each server and each device driver running as a separate user-mode process, and its capability to recover failures that normally would be fatal [15]. Another progress on microkernel architecture is the Secure embedded L4 [23] developed at OKLabs and NICTA. SeL4 is the most advanced member of L4 microkernel family. It is known as the first formally verified microkernel that is verified using theorem prover Isabelle/HOL [22]. SeL4's implementation is mathematically proved functional correct against its specification. It is a milestone in system reliability research and it proves that formal verification of microkernel is practical. Although SeL4 is exciting but it is still an experimental research project and SeL4 is designed as a hypervisor rather than a full stack OS. Another related research project is Muen Separation Kernel [7]. Muen is developed in Switzerland by the Institute for Internet Technologies and Applications (ITA) at the University of Applied Sciences Rapperswil (HSR). It claims as the first open source microkernel that has been formally proven to contain no runtime errors at the source code level. Muen is a separation kernel for X86/64 platform that is developed in SPARK. However, Muen is still an early research concept project and it is not suitable for embedded system with real-time requirement.

Several companies are dedicated to designing more secure RTOS. One major player of embedded real-time operating system development company, Wind River Systems just has updated its well-known VxWorks embedded RTOS for the emerging network-connected embedded systems of Internet of Things [18]. It is based on modular design that allows adding and upgrading without modifying kernel. It features MMU-based memory protection and separate user mode and kernel mode [11]. One of the major competitors in RTOS of VxWorks is QNX [4] from BlackBerry. QNX is a commercial Unix-like RTOS that is mainly used on Internet routers, Remote Terminal Units (RTUs) and in-car infotainment systems. QNX Neutrino RTOS is microkernel based that supports MMU-based memory management with kernel-application and application-application protection and is certified to Common Criteria ISO/IEC 15408 Evaluation Assurance level (EAL) 4+ [39]. Since both VxWork and QNX are commercial close source RTOS, availability is limited. However according to Koscher, Karl, *et al.* [25] the protection mechanism seems still limited. On the other hand, these trend shows that RTOS with more built-in security mechanism is highly on demand and microkernel-based architecture for secure RTOS is a promising direction.

9. CONCLUSION AND FUTURE WORK

We conduct empirical research and investigation of real-life biosecurity laboratory, as an example of building automation system security and safety. Through our studies, we propose a novel secure real-time OS architecture that is specially designed to suit critical control devices for building automation systems, that is based on microkernel structure, with proxy-based policy enforcement, and TPM supports. We believe that the proposed architecture would help address fundamental issues of embedded operating system, such as lack of identity, and prevent various popular attacks by default.

The next step of this work is to design and experiment a new scheduling method that is applicable to the proposed architecture specifically address the co-shceduling of processes at different controllers for the same tasks to meet the real-time constraint, and implement the RTOS. The challenge of this work is to formally specify OS primitives and formally verify the implementation to make sure that it complies with the specification and can be guaranteed functional correctness. Eventually the RTOS would be formally verified in source code level and test the prototype OS in a practical BAS environment for proof-of-concept.

10. ACKNOWLEDGMENT

This research is partially supported by the National Science Foundation under Grant No. 1136040 and 1314925. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

11. REFERENCES

[1] M. Abrams and J. Weiss. Malicious control system cyber security attack case study–maroochy water services, australia. *McLean, VA: The MITRE Corporation*, 2008.

[2] G. C. Alexander Bolshev. Icscorsair: How i will pwn your erp through 4-20 ma current loop. *Black Hat USA*, 2014.

[3] M. Anand, E. Cronin, M. Sherr, M. Blaze, Z. Ives, and I. Lee. Security challenges in next generation cyber physical systems. *Beyond SCADA: Networked Embedded Control for Cyber Physical Systems*, 2006.

[4] R. V. Aroca, G. Caurin, and S. Carlos-SP-Brasil. A real time operating systems (rtos) comparison. In *XXIX Congresso da Sociedade Brasileira de Computação*, 2009.

[5] S. Booth, J. Barnett, K. Burman, J. Hambrick, and R. Westby. *Net zero energy military installations: A guide to assessment and planning*. National Renewable Energy Laboratory, 2010.

[6] H. Boyes. Cyber security of intelligent buildings: A review. In *System Safety Conference incorporating the Cyber Security Conference 2013, 8th IET International*, pages 1–7, Oct 2013.

[7] R. Buerki and A.-K. Rueegsegger. Muen-an x86/64 separation kernel for high assurance. Technical report, Tech. rep, 2013.

[8] A. A. Cardenas, S. Amin, Z.-S. Lin, Y.-L. Huang, C.-Y. Huang, and S. Sastry. Attacks against process control systems: Risk assessment, detection, and response. In *Proceedings of the 6th ACM symposium on information, computer and communications security*, pages 355–366. ACM, 2011.

[9] A. A. Cardenas, S. Amin, and S. Sastry. Secure control: Towards survivable cyber-physical systems. In *The 28th International Conference on Distributed Computing Systems Workshops*. IEEE, 2008.

[10] A. Chou, J. Yang, B. Chelf, S. Hallem, and D. Engler. *An Empirical Study of Operating Systems Errors*, volume 35. ACM, 2001.

[11] B. Cole. Wind river brings a 20 kbyte microkernel to the vxworks rtos. `http://www.embedded.com`. Accessed: 2015-06-20.

[12] R. S. Dave Kennedy. Hacking your victims over power lines. *Def Con 19*, 2011.

[13] T. Derek and J. Clements-Croome. What do we mean by intelligent buildings? *Automation in Construction*, 1997.

[14] N. Falliere, L. O. Murchu, and E. Chien. W32. stuxnet dossier. *White paper, Symantec Corp., Security Response*, 2011.

[15] J. N. Herder, H. Bos, B. Gras, P. Homburg, and A. S. Tanenbaum. Construction of a highly dependable operating system. In *Dependable Computing Conference, 2006. EDCC'06. Sixth European*, pages 3–12. IEEE, 2006.

[16] G. Hernandez, O. Arias, D. Buentello, and Y. Jin. Smart nest thermostat: a smart spy in your home. *Black Hat USA*, 2014.

[17] ISC-CERT. Ics-cert monitor newsletters. Technical report, ISC-CERT, Apirl/May/June 2013.

[18] J. Jackson. Wind river updates vxworks os to join 'internet of things. `http://www.pcworld.com`. Accessed: 2015-06-20.

[19] M. Kadrich. *Endpoint Security*. Addison-Wesley Professional, 2007.

[20] J. L. Karsten Nohl, Sascha KriÃ§ler. Badusb âĂŤ on accessories that turn evil. *Black Hat USA*, 2014.

[21] W. Kastner, G. Neugschwandtner, S. Soucek, and H. M. Newman. Communication systems for building automation and control. *Proceedings of the IEEE*, 2005.

[22] G. Klein, P. Derrin, and K. Elphinstone. Experience report: sel4: Formally verifying a high-performance microkernel. In *ACM Sigplan Notices*. ACM, 2009.

[23] G. Klein, K. Elphinstone, G. Heiser, J. Andronick, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, et al. sel4: Formal verification of an OS kernel. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pages 207–220. ACM, 2009.

[24] G. D. Koblentz. Biosecurity reconsidered: Calibrating biological threats and responses. *International security*, 2010.

[25] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, et al. Experimental security analysis of a modern automobile. In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 447–462. IEEE, 2010.

[26] R. Langner. To kill a centrifuge: a technical analysis of what stuxnet's creators tried to achieve. *Arlington, VA: Langner Group*, 2013.

[27] J. P. Lehoczky, L. Sha, J. Strosnider, and H. Tokuda. Fixed priority scheduling theory for hard real-time systems. In *Foundations of Real-Time Computing: Scheduling and Resource Management*, pages 1–30. Springer, 1991.

[28] E. P. Leverett. Quantitatively assessing and visualising industrial system attack surfaces. *University of Cambridge, Darwin College*, 2011.

[29] J. W. S. Liu. *Real-time systems*. Prentice Hall, 2000.

[30] S. Luders. Why control system cybersecurity sucks. *Black Hat USA*, 2014.

[31] B. Miller and D. Rowe. A survey scada of and critical infrastructure incidents. In *Proceedings of the 1st Annual conference on Research in information technology*, pages 51–56. ACM, 2012.

[32] J. Molina. Learn how to control every room at a luxury hotel remotely: The dangers of insecure home automation deployment. *Black Hat USA*, 2014.

[33] H. Moore. Serial offenders: Widespread flaws in serial port servers. *Security Street Rapid7*, 2013.

[34] T. Novak and A. Gerstinger. Safety-and security-critical services in building automation and control systems. *Industrial Electronics, IEEE Transactions on*, 2010.

[35] N. I. of Health. *National Institutes of Health (NIH) Design Requirements Manual for Biomedical Laboratories and Animal Research Facilities*. National Institutes of Health, 2008.

[36] U. D. of Health, C. f. D. C. Human Services, Public Health Service, and N. I. o. H. Prevention. *Biosafety in Microbiological and Biomedical Laboratories, 5th Edition*. HHS, 2009.

[37] D. of Homeland Security. Dhs daily open source infrastructure report. Technical report, Department of Homeland Security, 2014.

[38] T. J. Ostrand and E. J. Weyuker. The distribution of faults in a large industrial software system. In *ACM SIGSOFT Software Engineering Notes*, volume 27, pages 55–64. ACM, 2002.

[39] QNX. Qnx os for security. `http://www.qnx.com/products/certified_os/secure-kernel.html`. Accessed: 2015-06-20.

[40] B. Rios. Owning a building: Exploit access control and facility management systems. *Black Hat Asia*, 2014.

[41] A. K. Sood. Digging inside the vxworks os and firmware the holistic security. *SecNiche Security Labs*, 2011.

[42] S. Szlósarczyk, S. Wendzel, J. Kaur, M. Meier, and F. Schubert. Towards suppressing attacks on and improving resilience of building automation systems-an approach exemplified using bacnet. In *Sicherheit*, pages 407–418, 2014.

[43] A. S. Tanenbaum, J. N. Herder, and H. Bos. Can we make operating systems reliable and secure? *Computer*, 39(5):44–51, 2006.

[44] United States Department of Agriculture, ARS Offices in Headquarters, Areas, and Locations. *ARS Facilities Design Standards*, May, 2012.

[45] WBSCD. Transforming the market: Energy efficiency in buildings. Technical report, World Business Council for Sustainable Development (WBSCD), 2009.

[46] K. Zetter. Researchers hack building control system at google australia office. `http://www.wired.com`. Accessed: 2015-01-26.

[47] K. Zetter. *Countdown to Zero Day: Stuxnet and the Launch of the World's First Digital Weapon*. Crown Publishing Group, New York, NY, USA, 2014.

[48] B. Zhu, A. Joseph, and S. Sastry. A taxonomy of cyber attacks on scada systems. In *Internet of things (iThings/CPSCom), 2011 international conference on and 4th international conference on cyber, physical and social computing*. IEEE, 2011.