The SAP Link: A Controller Architecture for Secure Industrial Control Systems

Matthew Cody Wyman
Brigham Young University

Follow this and additional works at: https://scholarsarchive.byu.edu/etd

Part of the Electrical and Computer Engineering Commons

BYU ScholarsArchive Citation
https://scholarsarchive.byu.edu/etd/8815

This Thesis is brought to you for free and open access by BYU ScholarsArchive. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of BYU ScholarsArchive. For more information, please contact scholarsarchive@byu.edu, ellen_amatangelo@byu.edu.
The SAP Link: A Controller Architecture  
for Secure Industrial Control Systems

Matthew Cody Wyman

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

James K. Archibald, Chair
  Doran Wilde
  Jeffrey Goeders

Department of Electrical and Computer Engineering
Brigham Young University

Copyright © 2019 Matthew Cody Wyman
All Rights Reserved
ABSTRACT

The SAP Link: A Controller Architecture for Secure Industrial Control Systems

Matthew Cody Wyman
Department of Electrical and Computer Engineering, BYU
Master of Science

Industrial Control Systems are essential to modern life. They are utilized in hundreds of processes including power distribution, water treatment, manufacturing, traffic management, and amusement park ride control. These systems are an essential part of modern life and if compromised, could result in significant economic loss, safety impacts, damage to the environment, and even loss of life. Unfortunately, many of these systems are not properly secured from a cyber attack. It is likely that a well-funded and motivated attack from a nation-state will successfully compromise an industrial control system’s network. As cyber war becomes more prevalent, it is becoming more critical to find new and innovative ways to reduce the physical impacts from a cyber attack.

This thesis presents a new architecture for a secure industrial controller. This architecture protects the integrity of the controller logic, including the safety logic which is responsible for keeping the process in a safe condition. In particular, it would prevent malicious or accidental modification or bypassing of the controller logic.

This architecture divides the controller into three components; the logic controller, the interface controller and the SAP link. The logic controller is responsible for controlling the equipment and contains the safety logic. The interface controller communicates with the rest of the control system network. The Simple As Possible (SAP) link is a bridge between the logic and interface controllers that ensures the integrity of the logic controller by drastically limiting the external interface of the logic controller. We implement this new architecture on a physical controller to demonstrate the process of implementing the architecture and to demonstrate its feasibility.

Keywords: SCADA, ICS, security, controller architecture, industrial control system, cyber war, safety logic
ACKNOWLEDGMENTS

I would like to thank all those who have contributed to this, especially to my father, Richard Wyman, who inspired this research and provided many of the ideas that I explored. Additionally, I would like to thank my advisor Dr. Archibald who was flexible with my unique schedule as I worked and travelled while researching and writing my thesis. Also thanks to Alex Avila and John Shurtleff for helping with the physical aspects of the ride model. And thanks to my fellow graduate students who helped review my ideas, my writing and helped with \LaTeX\ questions.
# TABLE OF CONTENTS

List of Tables ........................................................................................................... vi

List of Figures ............................................................................................................. vii

List of Listings ............................................................................................................. viii

Nomenclature ............................................................................................................. ix

Chapter 1 Introduction ............................................................................................... 1
  1.1 Industrial Control Systems ................................................................................. 1
    1.1.1 System Topology ......................................................................................... 2
    1.1.2 Security ....................................................................................................... 4
    1.1.3 Attacks ........................................................................................................ 6
  1.2 Purpose of Controller Architecture ................................................................. 8
  1.3 Scope of Research ............................................................................................. 9
  1.4 Thesis Outline .................................................................................................. 9

Chapter 2 Related Work ............................................................................................. 10
  2.1 Reducing Vulnerabilities and Strengthening Defenses ..................................... 10
  2.2 Threat Detection .............................................................................................. 11
  2.3 Reducing Consequences .................................................................................. 13

Chapter 3 Threat Model ........................................................................................... 15
  3.1 Consequences .................................................................................................. 15
    3.1.1 HSE Impacts ............................................................................................... 15
    3.1.2 Operation Impacts ..................................................................................... 17
  3.2 Attacks ............................................................................................................ 18
    3.2.1 Level 2 Loss of Availability Attacks ......................................................... 19
    3.2.2 Level 2 Loss of Integrity Attacks .............................................................. 20
    3.2.3 Level 1 Loss of Availability Attacks ......................................................... 21
    3.2.4 Level 1 Loss of Integrity Attacks .............................................................. 21

Chapter 4 Proposed Controller Architecture and Design Methodology .................. 23
  4.1 SAP Link Controller Architecture Overview .................................................. 23
  4.2 SAP Link Controller Design Methodology ...................................................... 26
    4.2.1 Physical Consequences of an ICS Failure ............................................... 26
    4.2.2 Critical Level 1 Device Outputs ............................................................... 27
    4.2.3 Cyber Commands .................................................................................... 27
  4.3 Implement SAP Link Architecture ................................................................... 28
    4.3.1 Interface Accessible Logic Controller Functions ..................................... 28
    4.3.2 Logic Controller Functions ...................................................................... 28
    4.3.3 Interface Controller Functions ................................................................ 29
**LIST OF TABLES**

3.1 Consequences of Loss of Availability and Loss of Integrity Attacks in Level 2 and Level 1 Controllers .......................................................... 19

6.1 Latency of SAP Link Results ............................................................... 53
6.2 Raspberry Pi Load Results ................................................................. 55
LIST OF FIGURES

1.1 Purdue Model .................................................. 3
1.2 Overview of Crashoverride ................................. 8

3.1 Physical Impacts from Cyber Attack ....................... 16
3.2 Idaho National Laboratory - Operation Aurora .......... 20

4.1 Typical Level 1 Controller Diagram ....................... 24
4.2 SAP Link Controller Diagram .............................. 25
4.3 SAP Link Controller Architecture Diagram .............. 30

5.1 Motion Simulator .............................................. 34
5.2 Ride Simulator 3D Model ................................. 35
5.3 Photo of Completed Ride System ......................... 36
5.4 Ride System Block Diagram .............................. 37
5.5 Controller Schematic .................................... 38
5.6 Interface Controller GUI .................................. 40
5.7 Raspberry Pi/Arduino Interface Detail .................. 42
5.8 Normal SAP Link Operation ............................... 43
5.9 Abnormal SAP Link Operation ............................ 43

6.1 Arduino Loop Time Results ............................... 54
6.2 Raspberry Pi Usage Statistics ............................ 55
LISTINGS

5.1 Interface Controller SAP Code ............................................. 44
5.2 Logic Controller SAP Code ................................................. 45
5.3 Legal SAP Functions .......................................................... 46
6.1 Invalid Commands Test while Emergency Stopped Test Code ........ 48
6.2 Valid Commands Test while Emergency Stopped Test Code .......... 49
6.3 Stress Test Code ............................................................... 50
6.4 Manual Commands Test Code .............................................. 51
6.5 Open Restraints While Ride is Running Test Code .................... 52
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPS</td>
<td>Cyber Physical System</td>
</tr>
<tr>
<td>DCS</td>
<td>Distributed Control System</td>
</tr>
<tr>
<td>DOS</td>
<td>Denial of Service</td>
</tr>
<tr>
<td>EUC</td>
<td>Equipment Under Control</td>
</tr>
<tr>
<td>HMI</td>
<td>Human-Machine Interface</td>
</tr>
<tr>
<td>HSE</td>
<td>Health, Safety, and Environment</td>
</tr>
<tr>
<td>ICS</td>
<td>Industrial Control System</td>
</tr>
<tr>
<td>IDS</td>
<td>Intrusion Detection System</td>
</tr>
<tr>
<td>IIoT</td>
<td>Industrial Internet of Things</td>
</tr>
<tr>
<td>IoT</td>
<td>Internet of Things</td>
</tr>
<tr>
<td>PLC</td>
<td>Programmable Logic Controller</td>
</tr>
<tr>
<td>RTU</td>
<td>Remote Telemetry Unit</td>
</tr>
<tr>
<td>SAP</td>
<td>Simple As Possible</td>
</tr>
<tr>
<td>SCADA</td>
<td>Supervisory Control and Data Acquisition</td>
</tr>
</tbody>
</table>
CHAPTER 1. INTRODUCTION

1.1 Industrial Control Systems

In 2017 over 97 million vehicles were produced throughout the world, over 475 million guests were thrilled at theme parks worldwide and 4.01 trillion kilowatt-hours of electricity were produced in the US [1] [2] [3]. None of this would be possible without modern industrial control systems. An industrial control system (ICS) is used to control industrial processes such as manufacturing plants, water utilities, electric utilities, amusement park rides, and traffic controls. These systems can range from single installations, such as a manufacturing plant, to large distributed systems such as electric utilities. ICSs are essential to modern civilization and if compromised can cause many problems.

Early attempts to automate physical processes go as far back as 250 B.C. where early engineers used float-valves to create an automatic clock that kept accurate time [4]. Further advances throughout history, especially during the industrial revolution, led to the creation of modern control systems. The first modern ICSs consisted of discrete relays, timers and other components. These controls would be located near the equipment under control (EUC). Eventually they wired the controls to centralized control rooms. These were completely custom solutions that would require extensive rewiring and reconfiguration if changes were made to the process.

Computerized control systems were first created in the 1950s and 1960s. These systems relied on the vacuum tube based computers of the day to monitor inputs and make process changes automatically [5]. They were very expensive, but also limited in what they could do. With the introduction of relatively cheap microcontrollers in the late 1960s, the programmable logic controller (PLC) was developed. PLCs are industrial micro-controllers that are easily programmed using ladder logic, a programming language designed to model relay wiring which made it easier to learn and adopt by engineers already used to wiring relays [6]. It was originally developed for use in the automotive industry to facilitate the factory changeovers as car models changed yearly.
As these control systems grew more sophisticated, they started to be used for control systems that covered large geographic areas such as power grids and water distribution systems. These digital control systems were called *Supervisory Control and Data Acquisition* (SCADA) systems or *Distributed Control Systems* (DCSs). SCADA system usually refers to a control system that covers large distances where the overall control is centralized and the field contains devices such as *Remote Telemetry Units* (RTUs) that receive commands directly from the central server [4]. A DCS is very similar to a SCADA system but usually controls more localized physical processes such as a factory. We will use ICS as a generic term for SCADA systems and DCSs since the principles discussed here apply to both.

In order for the central servers to communicate with the field devices, several communication protocols were developed. These protocols were designed to be simple, fast, and reliable. Modbus and DNP3 are two common protocols found in many ICSs. These systems were designed with little to no security as it was assumed that they were operating in an isolated environment. As the control process grew more complicated, there was a greater desire to leverage the Internet and provide easy monitoring and control from anywhere around the world. But by connecting these previously isolated networks to the Internet, many potential security risks were exposed.

As we approach the end of the decade, we are entering the Internet of Things (IoT) age. While IoT is mostly known for consumer products such as smart appliances, IoT has recently been adapted to ICSs with examples ranging from remote sensors in roadways to smart electric meters. These IIoT (*Industrial Internet of Things*) devices will eventually be used in closed-loop control where the remote sensors and actuators will be critical in the performance of the ICS [7]. While allowing for greater flexibility and cost savings, this greater connectivity must be accompanied by better security measures.

### 1.1.1 System Topology

To understand the unique security requirements of an ICS, it is important to understand how the various control components and the physical process interact with each other. The Purdue Model, created by Theodore Williams of Purdue University around 1989 [8], has been used extensively to model industrial control systems. The Purdue model separates the different functions and devices of the ICS into five levels. These levels are shown in Figure 1.1.
Level 0 is referred to as the *process* or *field* level. This level contains the actuators and sensors that make up the manufacturing process, the ride system or other physical process. Actuators include devices such as motors, valves, lights and magnets. Sensors include devices such as water level sensors, temperature gauges, proximity sensors, light sensors and buttons. Sensors gather data about the system and report it back to the Level 1 controllers while the actuators are sent commands from the Level 1 controllers [9]. These devices often convert the physical sensor data into an analog signal which is attached directly to a Level 1 controller. Some Level 0 devices have more processing power and send a digital signal back to the Level 1 devices.

Level 1 is referred to as the *basic* or *direct* control level. This level includes devices such as PLCs, RTUs and distributed controllers. Level 1 devices are the bridge between the digital network and the physical system since the physical Level 0 devices are connected to and controlled by them.
They also receive sensor data and forward it to Level 2 devices. Level 1 devices are responsible for the start up, shut down and operation of the Level 0 devices they connect to. Additionally, they contain safety functions that are designed to guarantee the safe operation of the equipment regardless of the control signals the Level 1 devices receive. This research primarily focuses on a new controller architecture for Level 1 devices to protect the integrity of the safety logic.

Level 2 is referred to as the *supervisory control* level. This level collects information from the various Level 1 devices and makes process-wide system changes. This level includes devices such as *human-machine interfaces* (HMIs) and audible or visual alarms [9]. HMIs are machines (often Windows or Linux based desktop machines) that allow a human operator to see the current state of the entire process. From here the operator can make process-wide changes such as a change in tank level or production speed. This is also where an operator can be notified if there is an error in the process or a potential safety hazard. This information allows the operator to take necessary corrective action before a problem occurs.

Level 3 is referred to as the *site* or *operations management* level. This level manages the entire manufacturing network. Level 3 devices provide the necessary IT services that the ICS network needs to function. Additionally, this level supports high level operations that affect the entire ICS, such as data logging, energy management and scheduling. This level can send and receive information to and from the Level 2 supervisory devices [9].

Level 4 is referred to as the *enterprise* or *site logistics* level. This level provides information to the outside world for business activities. This includes inventory reports, email services, etc. While important to the successful operation of the industrial process, this level is not critical to the process control of an ICS. Since this level connects to the outside world, it is potentially a security risk if an attacker can infiltrate this level. For this reason, Level 4 devices typically access Level 3 devices through a firewall or demilitarized zone to prevent any direct access from the outside world to Levels 0-3 [9].

**1.1.2 Security**

Securing ICSs became a critical concern after the 9/11 terrorist attacks. Early systems were developed with little consideration about security since they were designed to operate in an isolated environment. They were built to be reliable and simple, but security was an afterthought. They
were also built using proprietary and obscure protocols (some of which have since become open standards) such as Modbus and DNP3 [10] that had no built-in security measures.

Modbus was developed in 1979 by PLC manufacturer Modicon. It was originally implemented over RS232 and RS485 serial lines. This protocol became the de facto standard because it was easy to implement and a large variety of controllers and sensors supported it [10]. It was later encapsulated in TCP/IP packets to communicate over standard networks but no security measures were added [11]. Because of the lack of security, any device that can be pinged can be controlled [12] [13].

The only security these early ICSs implemented was physical security, enforced by ensuring that only authorized personnel had access to the controllers and the network [14]. In the 1990s, ICSs moved to more standard TCP/IP topologies which allowed for easier integration into preexisting networks and the ability to use Windows/UNIX machines to monitor and control the system [15]. While this made it much easier and cheaper to implement complex systems, it also opened them to the same vulnerabilities found in traditional TCP/IP networks.

With the move to TCP/IP networks and more complex systems, there was naturally a greater desire to monitor and control these systems remotely. To do this, control networks had to be connected to the Internet. While this creates systems that are much more accessible and convenient, it also creates many security risks. In 2017 researchers with Positive Technologies found 175,632 ICS devices accessible online with 42% of them located in the USA [16]. They found these devices by using Google and search engines that specialized in ICSs, such as Shodan [17] and Censys [18].

These results are especially worrying since many ICS devices are put into production with their default settings, making it trivial for an attacker to compromise them. Bonney et al. analyzed a common PLC and found many security issues, especially if left with default settings. There were numerous ways an attacker could compromise the system since the default user “webguest” has the password “1”. This user has complete admin rights to the PLC. Even if the password is changed, the PLC does not block after incorrect password attempts, meaning that an attacker could launch a brute force attack to guess the password fairly easily [19].

Device manufacturers release patches as often as security holes are found but the devices are seldom actually updated due to the difficulty of shutting down the ICS to apply the patches.
This means there are many vulnerable PLCs in live control systems. In conjunction with IoT search engines, such as Shodan, attackers of average skill can use penetration testing tools such as Metasploit [20] to attack vulnerable devices. Metasploit is a framework that allows easy penetration testing using many known system exploits. If a system is properly patched, the vast majority of these attempted exploits will fail.

Luckily there is now a greater emphasis on security and widespread agreement that something has to be done to protect ICSs and the physical systems they control. Successful attacks such as those discussed in the next section have shown that attacks are feasible and that the results can be catastrophic.

1.1.3 Attacks

Attacks against ICSs have been on the rise. Dell Sonicwall reported in 2015 that attacks against ICSs increased from 163,228 in January 2013 to 675,186 in January 2014 [21], a four-fold increase in just one year. Most of these attacks targeted the Level 4 devices of the ICS and few affected the actual industrial process, but this increase in attacks is concerning. The attack frequency has surely only increased since 2014. The Repository of Industrial Security Incidents (RISI) Online Incident Database includes over 240 reported cyber incidents since the early 1980s that resulted or could have resulted in physical consequences in the industrial process [22]. Many successful breaches are not reported so it is impossible to know exactly how many attacks have been successful. Known attacks have been made against many targets, including nuclear power plants, oil companies, airports, and sewage processing plants [23].

One of the most well-known attacks was Stuxnet, an attack that specifically targeted an Iranian uranium enrichment plant. Their plant design was based on obsolete stolen plans and never performed reliably. They used a protection system to keep the centrifuges working as well as they could. Stuxnet attacked and attempted to bypass the safety logic provided through this protection system.

What is known as Stuxnet was actually two separate attacks. The first attack targeted the pressurization of the centrifuges while the second one attacked the centrifuge speed. The first attack had to be installed via a USB stick or some other infected hardware while the second attack was self-propagating. This meant that machines that were never connected to the Internet or a
compromised USB stick were now also at risk. Additionally, the first attack was much more careful about staying secret, while the second used many zero-day exploits and stolen digital certificates which made it easier to infect systems but also easier to detect. One thing that both attacks share is that they both went to great lengths to not cause immediate catastrophic destruction. Both of them were periodic attacks designed to wear down the equipment faster than normal but not in a way that would immediately be evident. This made it much harder for the engineers to determine why their centrifuges were failing at an increased rate. Experts have concluded that a powerful nation-state must have designed the attack with a very good understanding of the specific Iranian plant [24].

Another example of a successful cyber-physical attack occurred in December 2016, when an electric substation in Kiev, Ukraine was targeted in an attack that is now known as CrashOverride or Industroyer. This attack seemed to be a test for a potentially larger attack. The malware was created to be modular so that it could be easily updated to work on different power stations that use different protocols. It could potentially be ported to attack different physical processes such as oil and gas distribution or factories. It was clear that the attackers had a good knowledge of the Ukrainian power station and what would be needed to shut it down [25].

Dragos Inc, a cyber physical security company, investigated the attack and provided a comprehensive analysis of the malware. The actual malware was broken up into several parts as shown in Figure 1.2. The first part was a backdoor that allowed the malware to control the system. It then installed a launcher which executed one of several available payloads. The payload was chosen based on the desired result of the attack and the infected device. This is the part of the malware that could be adapted for other attacks fairly easily. After a certain amount of time, in this case 1-2 hours, the launcher activated a data wiper that rendered the system unusable, thus hindering system recovery and covering its tracks [25].

This attack underscores the need for more proactive security research as we enter into the era of cyber-warfare. State-supported and terrorist hackers have much to gain in rendering their enemies’ ICSs useless and usually have a large budget to support their activities [26]. It is safe to assume that there have been many more attacks like Stuxnet and CrashOverride; some reported, some unreported and others ongoing and not yet discovered. Few successful attacks are reported presumably because operating entities want to avoid public embarrassment and damage to their
reputation and brand, so it is hard to know how much damage has been inflicted on ICSs from cyber attacks.

1.2 Purpose of Controller Architecture

In a perfect world, ICS networks would be perfectly secure and no outside intruders could infiltrate the network. Unfortunately, it is likely that the current cyclical process will continue, in which attackers keep finding new vulnerabilities every time one is patched. Additionally, even in a perfectly secure system, an insider who has access to the system could bypass the outer security and attack the system from inside. Likewise, a well meaning operator could make a mistake which could potentially cause as much damage as an attack.

The motivation of the controller architecture, as proposed in this thesis, is to limit the physical consequences of a successful cyber attack or a mistake by an operator. We do this by protecting the safety logic of Level 1 devices from unauthorized or accidental modification. The safety logic should be designed in a way that, if unmodified, the system will limit the potential
physical damages of a failure. Thus, if the safety logic is preserved then an attacker will be limited to only predetermined commands whose actions will be limited by the safety logic.

1.3 Scope of Research

It is important to note that this research is not focused on stopping intruders from gaining control of the ICS network. Likewise, it is not designed to detect attackers. It is designed solely to mitigate the consequences of a successful attack or unintended control commands by protecting the safety logic of a Level 1 device. If this safety logic is protected, attackers will be severely limited in the damage that they can cause. This controller architecture should be implemented with other security solutions that repel attackers from the network and notify system administrators of infiltrations.

1.4 Thesis Outline

The remainder of this thesis is organized as follows. Chapter 2 will introduce related work regarding ICS security. Chapter 3 will outline the threat model to ICSs. Chapter 4 will introduce the SAP Link controller architecture and a design methodology to implement the architecture. Chapter 5 presents an example implementation of the architecture following the proposed design methodology. Chapter 6 will provide some results and analysis. Chapter 7 will provide a conclusion to the thesis.
CHAPTER 2. RELATED WORK

As ICSs have moved towards more extensive use of IT technologies such as TCP/IP networks, there has been a resurgence in security research. Due to the cyber-physical nature of ICSs, traditional IT security measures, while important, are not enough. Properly securing these systems will require defense-in-depth solutions that include traditional IT protections, such as firewalls, as well as cyber-physical specific protections [27].

This chapter introduces several related works that can be implemented to help secure an ICS. ICS security research focuses on three main topics listed below.

- Reducing vulnerabilities and strengthening defenses
- Threat detection
- Reducing the consequences of a successful attack

2.1 Reducing Vulnerabilities and Strengthening Defenses

Keeping an attacker out of the ICS network is the first line of defense against a successful attack. While it is likely that a dedicated attacker with enough resources can eventually break in, security solutions that reduce vulnerabilities and strengthen defenses are vital to the security of any ICS. This is similar to the physical protections that keep a castle safe, such as thick walls, deep moats, and hard to navigate entrances. Many attacks can be avoided just by raising the costs required to launch a successful attack.

Best Practices

One essential way to secure a system is to follow best practices as recommended by security organizations. The National Institute of Standards and Technology (NIST) has produced a set of standards to follow when developing a secure ICS [28]. They provide guidelines in terms of
network architecture, management control, operational control and technical control. Some of the best practices include securing the SCADA network with firewalls, demilitarized zones limiting physical access, hardening servers and patching software frequently. Additional recommendations can be found in the book, *Cyber Security for Cyber Physical Systems* [29]. ICS administrators that are familiar with these best practices will be able to protect their system from many common attacks.

**Encryption**

Since most ICS communication protocols were designed without security in mind, any attacker that has access to the system can read, write, and distort the network messages as they please. Some researchers have proposed encryption to keep an attacker from being able to manipulate the network packets [30]. One potential problem with encryption is that it often requires a dedicated hardware module or enough processing power to encrypt and decrypt the data in real time which is often not possible on low powered remote terminals or inexpensive microcontroller. Researchers continue to investigate implementations that will work well in SCADA systems [31] [32] [33].

**System Modeling**

Another methodology proposed by Kriaa et al. can help evaluate the specific risks inherent in a specific ICS [34]. Their approach, S-Cube modeling, is a statistical programming model that uses historical data based on specific equipment used within an ICS to provide an automatic and complete evaluation of both safety and security risks for the whole system. This model will allow designers to plan for potential threats and reduce the cyber risk in the ICS. This may come in the form of additional security measures or a change in the equipment used in the system.

**2.2 Threat Detection**

Even though an ICS administrator has reduced vulnerabilities and strengthened the network’s defenses, it is still possible for an attacker to infiltrate the network. Being able to detect and then defend against any malicious behavior as quickly as possible is vital to the successful defense
of the ICS. Many successful attacks, such as Stuxnet, occurred over many months. If these attacks were discovered earlier, the damage could have been dramatically reduced.

An Intrusion Detection System (IDS) is a security system that is specifically designed to detect intrusions in an ICS. There are three main categories of IDSS; anomaly-based IDS, signature-based IDS, and specification-based IDS. Anomaly-based IDSs use learning algorithms to distinguish between normal, permitted traffic and malicious traffic. Signature-based IDSs use pre-discovered traffic patterns from previous attacks to discover intrusions. Specification-based IDSs use a predefined specification of intended network traffic and system operations to find anything that violates this model [35].

Anomaly-based IDS

Anomaly-based IDSs are employed to protect against both known and unknown threats. Instead of relying on rules that determine what data is valid and what is not, anomaly-based IDSs learn from the network and decide what is normal traffic. When it detects traffic that seems unusual, the IDS will alert the operator. The problem with this methodology is that it can be subject to false positives especially when a new task is introduced [36]. Eventually it will learn what is normal and what is not, and the false positives should decrease in frequency the longer the system is run.

Signature-based IDS

Signature-based IDSs rely on recorded network patterns from previous attacks. These IDSs look for the same patterns from past attacks to find the same attacks in different systems. While this does not help find undiscovered attacks, it is useful to protect against attacks that are already known and helps prevent attackers from reusing past attacks. Waagsnes designed a SCADA specific signature-based IDS built off of Suricata and Snort, lightweight signature-based IDSs traditionally used in IT networks [37] [38].

Specification-based IDS

Specification-based IDSs look for network patterns that do not fit the normal traffic patterns. SCADA network traffic is predictable due to the nature of the industrial process and since it is
predictable, it is possible to model the traffic and detect undesired activity. Goldenberg et al. present a methodology that looks deep into Modbus TCP packets and records the details of what each packet is doing. Using this, it creates a model for each link between the HMI and PLC [39]. Other specification-based IDSs have been proposed in [40] [41]. These IDSs verify that all network packets conform to the correct standard and are sent between authorized devices.

### 2.3 Reducing Consequences

Detecting threats and reducing vulnerabilities and strengthening defenses have been the primary forms of reducing ICS risk from a cyber attack. This is largely due to extensive IT research that ICS security researchers have leveraged. However, traditional cyber attacks against IT networks have different consequences than attacks against cyber-physical systems. In a cyber-physical system, attackers only cause harm if they can cause physical damage. Even though this is true, little research has been conducted on mitigating the physical impact of a cyber attack on a process. If attackers are limited in their ability to cause physical damage, their attack will fail, even if they successfully breach the ICS network.

One proposal to mitigate an attack was created by Mclaughlin who proposes a policy enforcement scheme, C\(^2\), that only allows commands to be issued if they will keep the system in a valid state. In contrast, many policy enforcement schemes rely on restricting certain actions to certain users [13]. This is helpful but can still lead to undesired behavior if a user’s account is compromised, an insider attack is launched, or an operator makes a mistake. Using the state of the physical system to determine what commands are valid keeps the system in a safe state. This should limit an attacker who successfully breaches the system from causing too much damage.

Wyman proposes a holistic approach to designing industrial control systems by considering the physical impacts of a cyber attack [42]. He argues that understanding how a cyber attack can lead to a physical impact is crucial to deciding how best to protect the system. He shows that even when a successful cyber attack occurs, designers can still reduce ICS risk from the attack by implementing cyber and physical defense mechanisms that can limit the attack’s consequences.

In this approach, the most important line of defense is the built-in safety logic in the Level 1 controllers. This safety logic should keep the system in a safe state no matter what inputs the Level 1 device receives from the Level 2 controllers. However, if this safety logic is modified or bypassed,
safety cannot be guaranteed and catastrophic failure may result. This was seen in Stuxnet where
the attackers were able to modify the safety logic and were able to run the equipment under control
in unsafe ways. The research described in this thesis builds on the work presented by Wyman; we
present a controller architecture that will protect the safety logic and limit the consequences of an
attack.
CHAPTER 3.  THREAT MODEL

Understanding the physical consequences of a successful cyber attack on a particular ICS is vital to understanding how to protect that ICS. This chapter will show some potential physical consequences of a cyber attack and why it is important to limit them. Additionally, this chapter will explore various methodologies that could be employed by an attacker to force an ICS into an undesired state. Understanding these attacks and their consequences is vital to understanding the importance of the work in Chapter 4 which limits the consequences of a successful attack.

3.1 Consequences

It is important to realize the scope and breadth of a potential successful attack. The consequences can extend much further than just the immediate equipment failure or breach. We will divide these consequences into two main categories; health, safety, and environment (HSE) impacts, and operation impacts as seen in Figure 3.1. These impacts can extend far beyond just the initial consequences of the attack as demonstrated by the cascading consequences model developed by Wyman [42].

3.1.1 HSE Impacts

HSE impacts are any undesired physical consequences that impact the health and safety of people or the environment. These are the worst kinds of impacts and it is absolutely vital that they be contained. Incidents such as a loss of electrical or chemical containment, dams breaking, and explosions are all included under HSE impacts.
Health Impacts

When an incident is large enough and involves a loss of containment there are negative consequences not only for the company but also for the community that lives near it. This is especially true if the loss of containment can cause illness.

In 1986, one of the nuclear reactors at the Chernobyl nuclear power facility exploded during a routine test. The explosion blanketed the nearby area with deadly radioactive particles which led to the evacuation of the nearby city of Pripyat the next day. Residents were not allowed to take their personal belongings since they were told that they would be able to return soon. But the area was never reopened for them again and their belongings were too radiated to be recovered. The direct consequences of this industrial disaster included 56 direct deaths, 4000 additional cancer related deaths, and the permanent displacement of around 120,000 residents. The total cost of the incident is estimated to be around 1.2 billion dollars. [26] [43].

Safety Incidents

The worst outcome of any industrial incident would be a death or serious injury. ICSs often control dangerous processes that if not correctly controlled could lead to serious injury or death. Factories have large robots that can collide with a human, chemical plants often deal with deadly chemicals, electrical plants deal with thousands of volts and amps of electricity, transportation systems and amusement parks’ vehicles carry hundreds of people and can crash or malfunction.
in a variety of ways. Obviously one end goal of any system should be to provide a safe working environment for everyone that works with the system or is within its vicinity.

Environmental Impacts

In the case of systems that control hazardous chemicals such as oil distribution systems, chemical plants, and factories there is a risk to the environment if these chemicals are released improperly. The risk is demonstrated by the massive impact of the Deep Horizon Drill oil spill. While not the result of an attack, similar results could occur from a successful ICS attack. Over 4.9 million barrels of oil were released after an explosion at the drill site and this spill has ended up costing BP approximately 65 billion dollars [44] [45]. In addition to the costs involved, 11 workers died in the explosion and thousands of fish, birds, dolphins and other marine creatures were affected. As this example illustrates, there is a potential for huge environmental impacts that can last for years after the initial incident.

3.1.2 Operation Impacts

In addition to the HSE impacts, there are operation impacts associated with cyber attacks on ICSs. Operation impacts include equipment failure, additional wear and tear and reduced product quantity and quality.

Equipment Failure

This is one the first consequences of a successful attack. Most, if not all, attacks depend on the equipment operating abnormally. Equipment operating outside of normal bounds will lead to increased wear and tear and premature failure. The best known attack, as discussed in Section 1.1.3, Stuxnet, was designed specifically to slowly destroy the centrifuges [24]. Because the attack was slow, it was hard for the engineers to diagnose the actual problem with the centrifuges, thus delaying the response. Other attacks can be quicker and lead to a more noticeable loss of equipment as shown by the attack on the generator by Idaho National Laboratory scientists that will be discussed in Section 3.2.2. This attack led to an immediate and full loss of the generator. Fix-
ing or replacing equipment can be costly and can lead to other losses such as reduced production efficiency.

**Loss of Yield**

Another operation impact is reduced product yield and/or quality. If an industrial process is not operating exactly as designed, it is likely that the product being produced will suffer as well. These industrial processes are complex and rely on every piece of equipment operating as expected. Any deviation from this can affect the whole process. The result can be a reduction in throughput or a reduction in quality of the finished product.

**Loss of Reputation**

Even after the incident is stopped and the physical damage is repaired, the company can suffer cascading impacts such as long term business consequences. If customers believe that a company’s products or services are unsafe they may choose to spend their money elsewhere. This impact is more significant in industries that use control systems that interact with the general population such as the transportation and amusement park industries.

In 2015, two cars of the Smiler roller coaster at Alton Towers, located in Staffordshire, England, crashed due to human error. This caused serious injuries to passengers including the need to partially amputate the legs of two passengers. The owners were fined 5 million pounds, but the total cost of the incident was certainly higher than that. Two years later Alton Towers reported lower attendance at the park due to the incident [46]. A single safety incident can lead to negative impacts that last for years.

### 3.2 Attacks

The previous section discusses what can happen if an ICS is compromised; this section will explore what an attacker can do to cause one of these scenarios.

An attacker can cause damage by forcing the controller in a SCADA system to enter into a “Loss of Availability” state or into a “Loss of Integrity” state. Loss of Availability means that the controller cannot be reached or sent valid commands. A Loss of Integrity means that sent
Table 3.1: Consequences of Loss of Availability and Loss of Integrity Attacks in Level 2 and Level 1 Controllers

<table>
<thead>
<tr>
<th></th>
<th>Loss of Availability</th>
<th>Loss of Integrity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level 2 Devices</td>
<td>Operators cannot send commands to Level 1 devices, safety logic should keep system stable</td>
<td>Invalid commands sent to Level 1 devices, undesired results could occur, safety logic should keep system from catastrophic failure</td>
</tr>
<tr>
<td>Level 1 Devices</td>
<td>No output to Level 0 devices, system should shutdown, catastrophic failure possible</td>
<td>Invalid commands sent to Level 0 devices, safety logic bypassed, catastrophic failure likely</td>
</tr>
</tbody>
</table>

commands cannot be trusted. The commands can be unwanted, changed, or sent at the wrong time. Attackers can attempt to cause this in either the Level 2 controllers or the Level 1 controllers. While any loss of availability or integrity is not desired, there are worsening cases depending on what the attackers can cause, as shown in Table 3.1.

3.2.1 Level 2 Loss of Availability Attacks

In this attack scenario, the Level 2 devices would not respond to any control commands and would not send any outputs to the connected Level 1 devices. This would mean that an operator could not change anything in the process. Fortunately, the Level 2 devices will not send invalid data to the Level 1 devices, but they also will not send any valid commands or updated information. In this scenario, the Level 1 device’s safety logic should prevent any catastrophic problems.

One example of a Level 2 Loss of Availability attack is by launching a denial of service attack that floods the network with data. The network may not be able to handle this and legitimate traffic will not be delivered nor received. If the SCADA controllers do not receive accurate or timely sensor data from the RTUs and PLCs, they cannot effectively manage the system. Likewise, if the RTUs and PLCs do not receive timely control data from the SCADA controllers they will not function as expected. Additionally, technicians will not be able to trust the data reported by the human-machine interfaces and could make uninformed decisions which could have a wide variety of negative consequences.
Figure 3.2: Generator being destroyed as seen in video screenshot from the Idaho National Laboratory Operation Aurora test attack [49].

3.2.2 Level 2 Loss of Integrity Attacks

In this attack scenario, an attacker will attempt to force the Level 2 devices to send wrong commands to their attached Level 1 devices. This can include sending out invalid set-points (desired target levels in the industrial process) or sending valid commands in undesired sequences or with bad timing. Regardless of what commands are sent, if the safety logic in the Level 1 device is well formed, the EUC should remain in a safe, albeit potentially undesired, state.

Even valid set-points and data can lead to breakdowns and accidents if sent in the wrong order or with the wrong timing. In 2007, researchers at Idaho National Laboratory conducted a simulated attack on an electrical utility that resulted in the destruction of a generator. This test gained wide exposure after a report by CNN [47]. The simulated attack was performed by opening and closing breakers out of phase with the power which caused a massive torque on the motor which eventually destroyed it as seen in Figure 3.2 [48]. This was one of the first tests that showed actual physical damage could occur from a cyber attack.
3.2.3 Level 1 Loss of Availability Attacks

In a Level 1 Loss of Availability attack, the Level 1 controllers will not be able to send commands to the connected Level 0 devices. While this can lead to catastrophic results, it is more likely that the EUC will simply stop operating. For example, if a pump does not receive any commands, it simply will not pump. This could be very bad if the pump is responsible for keeping a dam from overflowing but not as bad if it is filling up a tank.

3.2.4 Level 1 Loss of Integrity Attacks

This is the worst attack scenario. In this case an attacker is able to send invalid commands directly to the Level 0 devices. These commands will have bypassed any safety logic in the Level 1 controller and will be able to cause any of the consequences described in Section 3.1. The controller architecture proposed in this thesis is designed to prevent this type of attack.

If an attacker can access the PLCs or other remote Level 1 devices, they can change the programming logic that determines their operation. This includes modifying the code to bypass implemented safety measures or causing the PLC to control the equipment in a way that is destructive. Many PLCs have little to no security measures. There are even occasionally hard coded passwords left inadvertently in the code [50]. Beresford found that in at least one instance a hard coded password had root access and allowed an attacker complete control over the PLC. This same PLC had an “Easter Egg” that consisted of an HTML file with dancing monkeys and some German text saying that “all work and no play makes Jack a dull boy” [51]. This shows the lack of strict code reviews to verify that all test code was removed from the PLC.

The few security measures that are sometimes present are usually ineffective and are easily exploitable. The communications are rarely encrypted which allows for easy replay attacks, where an attacker takes previously issued valid commands and resends them at a later time with malicious intent.

In addition to an attack, a well-meaning employee can accidentally send the wrong commands. One engineer is quoted as saying that “he doubted an attacker could do any more damage to the system than he or his colleagues have done” [42].

21
For example, in the year 2000, an engineer who had just recently returned from a plant site accidentally loaded trial code onto a live PLC at the plant while trying to connect to a PLC in the office. This caused the oil refinery to shut down for half a day and lose 10,000 barrels of production [52]. As seen by this accident, there is little to protect a system once an attacker has access to the network. A successful security solution should protect against both intentional attacks and unintentional commands.

It is imperative that the Level 1 controller’s outputs never suffer a Loss of Integrity so that the equipment under control will remain in a safe state. This can be accomplished by protecting the safety logic of the device from remote and unauthorized modification. Without protecting the safety logic, there can be no guarantee of safety in the ICS.
CHAPTER 4. PROPOSED CONTROLLER ARCHITECTURE AND DESIGN METHODOLOGY

This section proposes a controller architecture, the SAP Link architecture, for Level 1 SCADA devices, such as PLCs and RTUs, that will protect a control system from undesired physical impacts from a cyber attack. This architecture specifically prevents Loss of Integrity attacks in the Level 1 controllers by preventing remote modification to the safety logic. By preventing modification of the safety logic, the integrity of the Level 1 device’s outputs is preserved. Additionally, we present a design methodology to guide ICS designers implementing the SAP link architecture into their devices.

4.1 SAP Link Controller Architecture Overview

Safety measures implemented in industrial control systems ensure that the industrial process remains within the design specifications. Traditional methods have relied on physical limiters such as float switches and pressure valves. While effective, physical limiters can be difficult to modify if set-points need to change, and they are subject to mechanical failures. Since they are physically separated from the digital control system, they are very secure. The safety limits cannot be changed or overridden remotely.

Another methodology is to implement the safety logic within the Level 1 device. The controller uses the sensor inputs and the connected actuators to regulate the industrial system. This has the benefit of being easy to modify and can leverage complex computerized controllers. The downside is that safety protections can be overridden remotely if the device is not properly secured. Our controller architecture is inspired by the physical isolation of the physical limiters but combined with the benefits of the digital safety logic. This is accomplished by physically isolating the digital safety logic in a Level 1 controller from the rest of the ICS so that it cannot be modified remotely.
Figure 4.1 shows a typical Level 1 controller with its inputs and outputs. As can be seen, the controller receives inputs from a connected Level 2 device, its programming port, and sensor inputs from the Level 0 devices. It also outputs data to the connected Level 2 device, through its programming port, and as outputs to the connected Level 0 devices. There is a correlation between the inputs to the device and what it outputs. Since we want to secure the outputs to the Level 0 devices we need to analyze each input to the Level 1 device and assess its security risk.

We will assume that the power to the device is secured by a backup and an attacker does not have access to it. Additionally, we will assume that the programming port is disconnected and accessible only by those with physical access to the device. While physical security is important and often overlooked, techniques for providing it are beyond the scope of this paper [29]. Additionally, we assume that the physical inputs from the Level 0 devices are secure as they are usually direct connections to the Level 1 controller, and that any modification would require physical access. This leaves the inputs from the Level 2 device. We cannot guarantee that these inputs will be valid as it is possible that the network has been infiltrated and that all Level 2 and above devices
have been compromised. To secure the output to the Level 0 devices we will need some way to permit only valid commands from the Level 2 device to affect the outputs to the Level 0 devices.

This is accomplished by dividing the Level 1 device into three components as illustrated in Figure 4.2: the logic controller, the interface controller and the SAP link. The logic controller is a secure zone that is connected to the system’s Level 0 devices such as sensors and actuators. We call this the logic controller as it contains the control and safety logic for the device. The interface controller is a network accessible zone that contains the functions used to connect the device to the rest of the SCADA network and that gathers information from the logic controller and sends control information to it. We call this the interface controller as it provides the control interface to the rest of the ICS. The SAP (Simple as Possible) link is the vital connection between the logic controller and the interface controller. This secure and simple link protects the safety logic by limiting commands that can be sent from the interface controller to the logic controller.

Figure 4.2: SAP Link Controller Diagram
4.2 SAP Link Controller Design Methodology

By verifying the integrity of a Level 1 device’s outputs, we mitigate the physical consequences of a bad control command. In order to guarantee system integrity, we need to understand what physical consequences we are trying to mitigate. These vary from system to system depending on the industrial process and the specific implementation of the industrial control system. To successfully understand these potential consequences, a control system designer must understand every step of the industrial process that is controlled by the ICS. For example, they need to understand the chemicals that are used, machinery being controlled, and/or the product being created and delivered.

Therefore, the first step in this design methodology is to understand the physical process, the consequences of a control system failure, the outputs to the Level 0 devices that would lead to failure, and what cyber commands would need to be issued to cause these actions. There will be commonalities found between different systems, especially if they are similar industrial processes, but even in similar systems, care must be taken to not overlook small differences that could lead to significantly different consequences.

4.2.1 Physical Consequences of an ICS Failure

Knowing the extent of the potential physical consequences of an incorrectly operating control system is the first step to determine how the system should be developed. This requires an in depth study of all the equipment being controlled by the control system and the goods that are being processed by them. All potential incidents (both HSE and operation) need to be considered.

For example, a sewage system could overflow spilling untreated sewage into residents’ yards or into surface waters. A malfunctioning robotic arm in a car factory could reach beyond its limits and destroy equipment or injure personnel. A tank in a chemical plant could overflow and spill hazardous chemicals and injure plant workers. The potential consequences from a control system failure are as different as the industrial processes that they control.
4.2.2 Critical Level 1 Device Outputs

Once the physical consequences are understood, the next step is to evaluate what outputs from the Level 1 device would lead to those consequences. An ICS designer should let the physical consequences identified during the design process described in Section 4.2.1 guide this process.

For example, the control signals from the Level 1 device to the pump controls would be the critical outputs that must be kept within a valid state to keep a tank from overflowing. Every ICS will have a different set of critical physical operations and a system administrator must be sure to account for all of them. Sometimes a singular physical consequence could have different control signals that could lead to that state. The safety logic, if not compromised, will keep the system in a stable state by limiting these outputs from the Level 1 device.

4.2.3 Cyber Commands

Once a designer has identified the potential consequences of a malfunctioning ICS and the critical control outputs from the Level 1 device that could lead to them, a system designer needs to consider what cyber commands would need to be issued for those physical operations to cause undesired physical consequences. These critical cyber commands will help guide the system administrator as they design the safety logic and the SAP link.

For example, a cyber command would need to be issued to fill a tank to a higher set-point to cause the physical filling of the tank. If the Level 1 device has properly formed safety logic, it would check the tank level sensor before issuing any physical command. If the safety logic has been changed or is missing, then the command could be sent regardless of the tank level.

These invalid cyber commands could be issued by a mistaken operator or by an attacker. Regardless of the source, or how unlikely a particular command might be, they must all be considered. Every potential cyber command that could translate into an invalid physical command must be considered in the implementation of the safety logic. Careful evaluation of the system at this step in the design process will make the design of the safety logic more effective and more secure. Additionally, this will help guide the administrator to know what commands need to be accessible to a remote user.
4.3 Implement SAP Link Architecture

After the ICS risk is evaluated by following the steps in Section 4.2, the system administrator will decide how best to implement the control logic functions into the Level 1 controller. They will be implemented either in the logic controller or the interface controller. Functions implemented in the logic controller include any safety logic as well as any routines that interface directly with the Level 0 devices. Interface controller functions are used to communicate with the rest of the ICS network and facilitate sending Level 2 commands to the logic controller via the SAP link.

4.3.1 Interface Accessible Logic Controller Functions

In order to correctly control Level 0 actuators and sensors, commands will need to be sent from the Level 2 devices to the Level 0 devices via the Level 1 controllers. To facilitate this control there will be functions that are implemented by the logic controller but accessible via the interface controller. While these functions can be controlled from the interface controller, the control signals will be passed through the SAP link and must conform to any limits set by the safety logic. This interface, to be discussed in greater detail in Section 4.4, will isolate these commands from the interface controller to logically separate them.

4.3.2 Logic Controller Functions

These functions include functions such as safety logic, PID controllers and any functions that react directly to a sensor input such as an emergency stop that is hard-wired to the logic controller. These commands are vital to the control and operation of any equipment controlled by the Level 1 device. If these functions are modified, prevented from running, or accessed incorrectly, physical malfunctions will likely occur. These functions are often related to the functions described in the previous section. For example, an interface accessible logic controller function could be a request to fill a tank while the function that turns on the pump would be implemented in the logic controller with logic to check and verify the tank isn’t already full.
Safety Logic

Some of the most important functions implemented in the logic controller are the safety logic functions. This set of commands is implemented to keep the equipment under control within predetermined bounds and to prevent any incidents. It is the logic that will keep traffic lights from turning green in multiple conflicting directions at an intersection, the logic that will keep the mechanical arm from hitting people and the logic that will keep the water tank from overflowing. Great care must be taken to ensure that the safety logic covers all potential consequences as described in Section 4.2.1. Any gap in the safety logic can be exploited by an attacker to compromise the physical process.

4.3.3 Interface Controller Functions

The Level 1 devices need to be able to communicate with Level 2 devices to provide feedback and to receive control commands. Since the interface controller interfaces with the Level 2 devices, it will contain any processing functions required to communicate with them. This includes a Modbus, DNP3 or another SCADA communication controller. Additionally, the interface controller will parse data received, decide what is important to pass on to the logic controller and could contain some non-essential checks and bounds.

It is important to consider anything on this controller to be non-essential to the control of the equipment controlled by the Level 1 device. Since it will have a direct connection to the rest of the ICS network, any successful attack on the higher level devices or the network could compromise it.

4.4 SAP Link

The Simple As Possible (SAP) link is the critical connection between the interface controller and the logic controller as shown in Figure 4.3. This link provides a simple, explicitly defined link that physically limits the commands that can be sent across it. Although the link is simple, it still needs to be able to communicate all the necessary control functions from the interface controller to the logic controller. The SAP link ensures the secure operation of the logic controller since
an attacker cannot modify the safety logic in the logic controller (unless physically present) and because that logic obeys external directives only if they result in the system staying in a safe state.

### 4.4.1 Security Through Data Type Limits

The SAP link will allow only certain data types to be sent through it. For example, if a device requires only start and stop commands, then it will only allow start and stop commands and will block anything else. The logic controller should be designed to ignore anything that does not pass its filter. This requires a good initial design to explicitly define what kind of data is valid and desired. Since the link will not support the sending of arbitrary commands, it will be very difficult, if not impossible, for an attacker to change how the logic controller operates.

### 4.4.2 Security Through Data Rate Limits

As an additional security measure, a data rate limit should be enforced by the logic controller. This prevents an attacker from overwhelming the logic controller with too much data and slows down attackers attempting to launch brute force attacks. Additionally, a data rate limit makes the SAP link easier to debug and verify that it is operating as designed.
The link’s speed will depend on the industrial process and how much data is needed to be communicated. For a simple process, a slow link can be utilized while a faster link might be needed if more complicated control set-points are being sent. Optimal speed should be no faster than necessary to assure proper control.

4.4.3 Communications Protocols

The link’s communication protocol can vary from system to system. Some systems will need faster and more complicated links between them, while others will only need a slow and simple link. Thought should be placed into the design for this based on the ICS risk and where the control functions were implemented as identified in Section 4.3. This link is the key to protecting the safety logic described in Section 4.3.2. Any problem with this link could allow an attacker to bypass all the safety protections offered by the safety logic. Whichever communication protocol is utilized, it should be unidirectional and be able to ensure the data type and rate limits as discussed.

Possible implementations include a simple serial link that is designed to accept only certain commands or a shared memory that the interface controller writes to and the logic controller reads from.

4.4.4 Reporting Link

In addition to the SAP link, a reporting link from the logic controller to the interface controller needs to be implemented. As long as this link is truly unidirectional, which can be guaranteed by using opto-isolators, this link can be as robust as needed. Opto-isolators are ICs that use a light source, usually an LED, and a photoresistor to transfer a signal to another circuit with no physical connection.

It does not need to be simple like the SAP link since it is coming from a secure controller to an unsecured one. This link will provide sensor feedback and diagnostic information from the logic controller to the interface controller which will then send that data to higher levels of the ICS.
4.5 Test

After designing the safety logic and implementing it on to the controllers, it is important to test the system and verify that every legal state can be reached while illegal states are unreachable. While ideally this test would be an exhaustive examination of every possible input and configuration, this is likely not possible. However, the more that can be tested, the higher the assurance of a secure system. Tests should include sending invalid set-points, illegal commands, and attempts to modify or bypass the safety logic functions.
CHAPTER 5. DESIGN IMPLEMENTATION

This section describes an example implementation of a SAP Link controller and an industrial system to illustrate the design methodology proposed in this thesis. While the specific implementation details are unique for this ICS, the concepts are general and can be applied to other designs.

5.1 Ride System Overview

The example industrial system is based on a simple model of a motion simulator found at many amusement parks. The physical model of the simulator was designed in Autodesk Fusion 360 and then 3D printed [53]. The simulator ride vehicle has three servos; two to drive pan and tilt of the ride vehicle and a third that lifts the ride vehicle off the ground so that it can move freely. An example of an actual motion simulator ride vehicle can be seen in Figure 5.1. The Autodesk Fusion 360 3D model is shown in Figure 5.2, and the working model can be seen in Figure 5.3.

As shown in Figure 5.4, the example Level 1 device is implemented on an Arduino and a Raspberry Pi. The logic controller is implemented on the Arduino, while the interface controller is implemented on the Raspberry Pi. The SAP link that connects the two devices is a custom serial link utilizing an opto-isolator to provide complete electrical separation between the two systems. The Arduino is connected to the Level 0 devices which include actuators (the three servos and two sets of indicator LEDs) and sensors (three control buttons). A more detailed schematic of the controller can be found in Figure 5.5.

5.2 SAP Link Controller Design Methodology

This section will show how the example system follows the design steps and principles as discussed in Section 4.2. Even though the equipment under control is just a scale model of an imaginary ride system, we will evaluate it as if it was an actual ride.
5.2.1 Physical Consequences of an ICS Failure

A theme park ride, like this simulator, can cause many problems if the ICS fails to operate the machinery as designed. Potential consequences of the ride operating incorrectly include many of those described in Section 3.1. Both HSE and operation incidents are possible.

Possible HSE incidents include injury to or death of riders, maintenance personnel, ride attendants or onlookers. These could occur from the machine moving out of its normal range, parts falling off, unexpected movement or restraints opening prematurely or failing to lock. For example, if the ride is being loaded with passengers and the ride starts to move without authorization, it could strike passengers that are still loading, employees that are helping load the guests and any passengers that are on the ride but not yet properly restrained.

Operation incidents include premature failure of parts such as the motors or other mechanical devices. This could occur if the ride moves beyond predetermined safe bounds or moves quicker than designed. Additionally, if enough stress is placed on the system, there could be catastrophic failure of parts which could cause them to break immediately.
Furthermore, if the ride breaks down frequently or causes an injury, the ride, and therefore the park, would lose reputation among potential guests. This could scare away potential visitors and increase their concern about other attractions at the park which would reduce income from admissions. It could also scare away sponsors and reduce other forms of income.

5.2.2 Critical Level 1 Device Outputs

The critical outputs from the Level 1 device to the Level 0 devices are the movement commands to the pan, tilt and lift motors. Whenever any one of these is in operation, there is potential for an HSE or operation incident. This is especially true when there are people present. The safety logic should prevent the ICS from issuing any move commands to the motors if there are any sensor errors or it detects people in harm’s way. Additionally, movement commands beyond the specified limits should be prevented.

Another critical control output relates to the restraint locks. The system should never issue a physical command to unlock the restraints while the ride is in motion or when it is not in a safe physical state such as in the home position.
5.2.3 Cyber Commands

Risky cyber commands related to the movement would be any command that attempts to make the ride move beyond its structural limits or in some other undesired way. For example, if the ride vehicle should be limited to a tilt angle of 40 degrees, a command that would tilt it to 45 degrees would be undesirable. If the safety logic does not prevent this command from being issued from the logic controller to the tilt motors, additional stress would be placed on the ride vehicle.

Additionally, invalid start commands when the ride is not ready to move could cause movement when the ride is still being loaded or when personnel are within the ride space. The logic controller should not send any movement commands to the servos until all start conditions are met, which includes that all restraints are lowered and locked, the emergency stop is disabled, and ride personnel have requested the ride to start.

Cyber commands to open the restraints while the ride is either moving or the ride vehicle is not in the station should be denied. Since the ride vehicle lifts up and tilts and pans in odd angles, opening the restraints could cause riders to fall out of their seats and be seriously injured or killed. The restraints should be unlocked only if the ride is stopped and in the safe “home” position.
An emergency stop condition should also be in place to stop the ride if an operator notices a problem or the ride system detects a problem. This state should not be disabled unless done so manually by an approved operator who has verified that it is safe to do so. Due to the severity of an emergency stop, the ICS should not be able to disable the emergency stop remotely and should ensure that all safety conditions are met before allowing a manual reset to any normal operating state.

5.3 Implement SAP Link Architecture

The next step is to implement the necessary control functions that are needed for a functioning control system and prevent possible incidents. These operations will allow a ride operator to start and stop the ride, open and close the restraints, enable an emergency stop and reset the ride after an emergency stop. Additionally, these functions will keep the ride within safe bounds and prevent any unauthorized movement.
5.3.1 Interface Accessible Logic Controller Functions

The functions listed here are accessible via the interface controller and are the main ones utilized by the ride operator. They are implemented on the Arduino but can be actuated from the interface controller through the SAP link.

- Soft ESTOP (Emergency Stop) - This will stop the ride as quickly as can safely be done at anytime during the ride cycle. While a ride is in an emergency stop condition, the ride can not move or have its restraints opened until the system is reset directly from the logic controller. It is always safe to emergency stop. So even though an attacker can send undesired ESTOP commands, they cannot cause any damage by sending them. The ESTOP’s time and source will be logged which can be useful to detect an attacker. As an extra protection the ride cannot be removed from the ESTOP state by the interface controller but only from a manual command issued at the logic controller.

- Start the ride - This is essential for the operator to be able to operate the ride. Internal safety logic will prevent this command from having any effect if the ride is in an emergency stop condition or ride restraints are open and unlocked. When activated, the arm will move to an
upright position and the ride vehicle will pan and tilt as programmed in the logic controller. After finishing all programmed moves, the arm will return to the home position.

- Stop the ride - This will stop the ride and move it slowly back to its home position. This would be used to stop the ride for non-emergency situations such as a sick rider or a rider asking to get off.

- Lock Restraints - This will send the command to lock the restraints. Once restraints are locked, the riders can lower them and secure themselves. The lowering is simulated by a button connected to the logic controller.

- Unlock Restraints - Will open only if ride is not moving and in the home position. In a real system, there would be a mechanical way to unlock the seats directly at the ride vehicle.

- Move to a manual position - These commands allow a ride operator to set the angle of pan and tilt and how long it will take to get there directly from the GUI. This is accomplished by sending a manual move command code and then the next three data values received over the SAP link are interpreted as pan, tilt and time to move parameters. The logic controller has self-imposed pan and tilt limits that if exceeded will cause the ride to ESTOP. These commands can be sent multiple times to create a custom ride. Additionally, if the ride restraints are open or the ride is in an ESTOP condition, it will not move.

5.3.2 Logic Controller Functions

These functions are implemented in the logic controller but cannot be accessed remotely. They are activated from sensor inputs or from the internal state of the logic controller. This includes the safety logic.

- Disable ESTOP - This function is activated from a button connected to the logic controller. It resets the system from an emergency stop state to a normal run state.

- Reset Vehicle while ESTOPPED - This function is activated from a button connected to the logic controller and is required before an ESTOP can be deactivated. This will slowly move the ride back to its home position and resets the system so that the ESTOP can be deactivated.
• Restraints Lowered Sensor - This function sets a variable based on whether the restraints are lowered or not. When the restraints are lowered the variable is set to true and when they are still open, the variable is false. This is modeled by a button push to simulate the actual lowering of the restraints by guests.

• Feedback Functions - These functions report the current state of the logic controller to the interface controller. Every time the ride vehicle changes state, the logic controller sends a feedback string with data about the current pan, tilt, and lift motor angles and the current run and ESTOP states.

• Enable Motors - This activates the servos. This function was just for testing purposes.

• Disable Motors - This deactivates the servos. This function was just for testing purposes.
Safety Logic

The role of the internal safety logic is to prevent the simulator from operating outside of the design specifications or in any unsafe manner. Specifically it is designed to protect the ride vehicle from causing any of the incidents previously enumerated and listed in Section 5.2.1.

Before sending movement commands to the servos, the safety logic will always check the current operating conditions and verify that any desired set-points are within the limits specified in the safety logic. If it determines something is wrong or that a movement command will exceed the limits, the safety logic will immediately trigger an emergency stop condition which will need to be manually reset at the logic controller before it can continue operating. The SAP link prevents any unauthorized modification of these limits. If a limit needed to be changed, it would require the firmware of the Arduino to be rewritten which requires physical access.

Specific Limits:

- Tilt Limits - The simulator’s tilt must be between to -30 and 50 degrees.

- Pan Limits - The simulator’s pan must be between to -40 and 40 degrees.

- Start - The simulator will start only if the ESTOP is disabled, the simulator is in the home position, the restraints are locked and lowered and the simulator is not already running.

- Disable ESTOP - Will disable the ESTOP condition only if the ride is at home, not running and requested from the button press connected to the logic controller. The interface controller cannot request to disable the ESTOP state.

- Restraints Unlock - Will unlock only if the ride is at home and stopped.

- Raise Restraints - Will raise only if the ride is at home, stopped and restraints are unlocked.

- Restraints Lowered - Will set the restraints lowered variable only if the restraints are locked.

- Automatic Emergency Stops:
  - Restraints are opened while the ride is not at home.
  - A movement command exceeds the pan or tilt limits.
5.3.3 Interface Controller Functions

These functions are implemented on the interface controller and facilitate the GUI so a ride operator can operate the ride vehicle. The GUI can be seen in Figure 5.6. These functions are implemented in Python on the Raspberry Pi.

The interface controller sends the appropriate commands based on the button selected to the logic controller via the SAP link. Additionally, it receives diagnostic data from the logic controller via the reporting link and displays it in GUI. The red buttons are illegal commands that if sent to the Arduino will cause the Arduino to send a return signal back to the Raspberry Pi stating that those commands are unsupported. The commands can be enabled within the Arduino code for ease in testing but must be disabled in production code.

5.4 SAP Link

The SAP link that connects the Raspberry Pi to the Arduino is a low baud, software defined serial link. This link, as seen in Figure 5.7, is implemented using the GPIO pins of the Raspberry Pi and the Arduino. Between the Pi and the Arduino is an opto-isolator to help account for voltage differences between the systems and to further isolate the two controllers.
The interface controller portion of the SAP link is implemented on the Raspberry Pi by toggling the GPIO pin as shown in Listing 5.1. Before sending out the control signal, it sets the line high to trigger the Arduino so it knows that a command is about to be sent. Afterwards, the desired instruction code is sent bit by bit for a total of 8 bits. Between every bit, it waits for a set amount of time so that the signal is sent at a constant frequency. After the data packet is sent, it sets the line low to signal that it is done transmitting.

The logic controller portion of the SAP link is implemented on the Arduino as shown in Listing 5.2. After detecting a change on the connected GPIO pin, it samples the value of that pin at a fixed frequency that aligns with the frequency that the Raspberry Pi is sending the data. It stores the values of the pin in an array and after the 8 bits are received, it converts the array back to an integer. It will use this number to determine what command was sent.

As shown in Figures 5.8 and 5.9, regardless of the rate at which the Raspberry Pi sends the data, the Arduino will only record the current value of the pin at each sample time. This is valuable because it limits what an attacker can send and how quickly they can send it. It is not possible to send meaningful data more quickly than the logic controller is sampling. Additionally, because the SAP link is so simple, there is no way for an attacker to access or modify anything on the Arduino. Since the firmware is updated through the on-board serial link, the only way the firmware could be changed is to access the physical serial RX pin of the Arduino which is disconnected and left floating as shown in Figure 5.7. An attacker would have to gain physical access to the system which then removes much of the motivation to launch a cyber attack.
BAUDRATE = 64
FREQ = 1.0/BAUDRATE
NUMBITS = 8

```python
def send_thread(instr_code):
    GPIO.output(txPin, GPIO.HIGH)  #START CODE:
    time.sleep(FREQ)
    for i in range(NUMBITS):
        GPIO.output(txPin, int((instr_code/(2**i)) % 2))
        time.sleep(FREQ)
    GPIO.output(txPin, GPIO.LOW)  # reset line to idle
    time.sleep(2*FREQ)
```

Listing 5.1: Interface Controller SAP Code

In addition to the security protections afforded by the physical design of the link, additional security measures are implemented within the logic controller. The logic controller will only respond to the permitted functions as defined in Section 5.3.1. After receiving a command from the interface controller through the SAP link, the logic controller will check that command against a whitelist of valid commands as shown in Listing 5.3. Any valid command will call the function that corresponds to its command code. Any command that is not implemented will be ignored and an error message will be sent to the interface controller through the reporting link as shown in lines 29-30.

Sensor feedback from the logic controller to the interface controller is provided by the on-board serial communications chip-set on the Arduino to send data back to the interface controller. This link only utilizes the TX pin of the Arduino and runs at 9,600 baud. Additionally, an opto-isolator is placed between the TX pin on the Arduino and the RX pin on the Raspberry Pi to ensure that an attacker cannot launch an attack back through this link. The opto-isolator also helps with the different voltages since the Arduino runs at 5v and the Raspberry Pi runs at 3.3v.
```c
void ExternalCom::read_sap_link() {
    int cur_pi_value = !digitalRead(RASP_PIN);
    if (pi_active && !byte_received) {
        if (millis() - lastBitReadTime > FREQ && millis() > lastBitReadTime)
            // Process the next bit
            lastBitReadTime = millis();
            instr_code_array[cur_bit] = cur_pi_value;
            cur_bit++;
        if (cur_bit >= NUMBITS)
            // All bits are received. Process command
            int instr_code_int;
            for (unsigned int i = 0; i < NUMBITS - 1; i++) {
                instr_code_int += instr_code_array[i] * (1 << i);
            }
            if (instr_code_array[NUMBITS - 1] == 1) { // Negative number
                instr_code_int -= 128;
            }
            byte_received = true;
            process_command(instr_code_int);
    }
    // Pi has finished sending, waits for line to reset
    if (millis() > piResetTime && pi_active) {
        // Reset receiving line
        pi_active = false;
        byte_received = false;
    }
    if (cur_pi_value == 1 && !pi_active) {
        // Detected Pi sending a command
        cur_bit = 0;
        piResetTime = millis() + NUMBITS * FREQ + 2 * FREQ;
        lastBitReadTime = millis() + FREQ / 2;
        pi_active = true;
    }
}
```

Listing 5.2: Logic Controller SAP Code
void ExternalCom::process_command(int instr_code) {
  // instr_code is command integer received from interface controller
  if (man_ang_mode == 1) {
    man_tilt_ang = instr_code;
    man_ang_mode = 2;
  } else if (man_ang_mode == 2) {
    man_pan_ang = instr_code;
    man_ang_mode = 3;
  } else if (man_ang_mode == 3) {
    man_ang_time = 100 * instr_code;
    sim_ctrl->move(man_tilt_ang, man_pan_ang, man_ang_time);
    man_ang_mode = 0;
  } else if (instr_code == 0) {
    sim_ctrl->enable_e_stop("PI ENABLE ESTOP");
  } else if (instr_code == 1) { // * Restricted command
    Serial.println("Cannot Remotely Disable ESTOP.");
  } else if (instr_code == 2) {
    sim_ctrl->start_ride();
  } else if (instr_code == 3) {
    sim_ctrl->stop_ride();
  } else if (instr_code == 4) { // * Restricted command
    Serial.println("Cannot Remotely Return Home from ESTOP");
  } else if (instr_code == 5) {
    sim_ctrl->lock_restraints();
  } else if (instr_code == 6) {
    sim_ctrl->unlock_restraints();
  } else if (instr_code == 7) {
    man_ang_mode = 1;
  } else {
    Serial.print("INVALID CODE: ");
    Serial.println(instr_code);
  }
}

Listing 5.3: Legal SAP Functions
5.5 Test

To simplify the setup of the test bed, the Raspberry Pi acts as both the interface controller and as a stand in for the rest of the SCADA network. Implementing a complete SCADA system with all levels is beyond the scope of this work. This simplification does not prevent thorough testing of the design since the interface controller would be directly connected to the potentially compromised SCADA network. If the SCADA network is compromised, everything down to and including the interface controller can be compromised as well. This research focuses on the security protection that comes from the SAP link, assuming a compromised interface controller, and so can be modeled without the full system in place.

Testing for this system will verify that the ride simulator will always fail-safe regardless of the commands sent from the interface controller. This specifically means verifying the following conditions:

- If the ride is in an emergency stop condition, it will not move until manually reset
- The ride simulator will not malfunction if flooded with valid and/or invalid data over the SAP link
- The ride will not move beyond predesignated tilt and pan limits
- If the ride is not at home, then its restraints will not open

In addition to these safety and security tests, performance benchmarks can be measured such as SAP link latency and Arduino cycle time measurements. While the tests cannot prove conclusively that the ride is perfectly secure, these tests will show with a high level of confidence that the system is more secure than it would be without the SAP link and without well designed safety logic.
CHAPTER 6. RESULTS AND ANALYSIS

6.1 Results

This section presents results from the experiments conducted using the model system described in Chapter 5. These results show that the SAP Link Architecture can be implemented on low end hardware. All test results are based on data recorded from the logic controller and by visual observation. The logic controller sent diagnostic data through its serial port to the Raspberry Pi and a computer connected to the USB port of the Arduino which could be saved and then evaluated after the test was finished.

6.1.1 Security Tests

Test 1: Invalid Commands Test while Emergency Stopped

For this test, the ride was started and the vehicle moved into an upright position and allowed to start its normal run cycle before an emergency stop was triggered. While stopped, the logic controller was flooded with invalid commands. These invalid commands were created by randomly flipping the GPIO pin on the Raspberry Pi as to create a completely random bit pattern. Additionally, a random pause between each bit flip was inserted so that the bits would not conform to the expected timing. This was repeated 10,000 times. Listing 6.1 shows the Python code used to run this test.

```python
def random_send_invalid():
    for i in range(10000):
        GPIO.output(txPin, random.randint(0,1))
        time.sleep(random.random() / 8)
```

Listing 6.1: Invalid Commands Test while Emergency Stopped Test Code
While the test was running, the ride remained stationary, never exited the emergency stop state, and did not unlock the restraints. As the logic controller parsed the incoming data, it would attempt to interpret it as a command. If the command received was invalid, it would send a response to the interface controller stating this. Occasionally, the random data would form a valid command but since the ride was in an emergency stop condition, it would ignore the command and send a response to the interface controller stating that it could not run the operation. After the test was finished, the ride resumed normal operations once the emergency stop was disabled at the logic controller.

**Test 2: Valid Commands Test while Emergency Stopped**

For this test, the ride was started and then emergency stopped as in the previous test. Unlike the previous test, the logic controller was then flooded with random valid commands. These commands would normally actuate a command in the ride such as starting or stopping the ride. Since the ride was in an emergency stop state, the logic controller denied these commands and informed the interface controller via the reporting link that it could not actuate the command. Listing 6.2 shows the Python code for this test.

Once again the ride performed as expected. It remained in the emergency stop state and did not move nor unlock the restraints regardless of the sequence of valid commands received. The ride resumed normal operations once the emergency stop was disabled at the logic controller.

```python
def all_valid():
    commands = [0, 2, 3, 4, 5, 6]  # Valid Commands
    for i in range(100):
        send_thread(random.choice(commands))
```

Listing 6.2: Valid Commands Test while Emergency Stopped Test Code

**Test 3: Stress Test**

This test was designed to verify that the logic controller would continue to operate as normal even while receiving lots of data from the interface controller via the SAP link. This test
started the ride and let it run through its normal movements. While the ride was running, the interface controller sent random invalid commands that would be rejected by the logic controller. As the logic controller received these commands, it reported to the interface controller via the reporting link that it was ignoring the command. After the ride finished its run, the interface controller stopped sending invalid data and then sent a valid command to start the ride again. Once the ride was running again, the interface controller resumed sending invalid data. This was repeated 50 times. The test code can be found in Listing 6.3.

The ride performed as expected with no noticeable decrease in performance from the invalid commands. The logic controller started the ride, ran through ride movements as specified in the logic controller and ended the run all as designed. Additionally, the logic controller continued to report back to the interface controller that the commands it received were invalid.

In one instance the logic controller misinterpreted one of the invalid commands as a manual move command. The next commands received were then interpreted as the pan, tilt and time to move parameters. The pan parameter was beyond the valid limit of 40 degrees, so a pan limit error was triggered when the ride attempted to move beyond 40 degrees which triggered an emergency stop condition. While emergency stopped, the logic controller continued to deny invalid commands. After resetting the ride manually on the logic controller, the test continued as normal with no further problems. This further shows that this controller architecture is effective even if the logic controller misinterprets the commands being sent from the interface controller.

```
def run_ride_w_random_data():
    for i in range(50):
        ride_start = time.time()
        send_thread(2)  # start ride
        while (time.time() < ride_start + 21):
            send_thread(random.randint(34, 128))
```

Listing 6.3: Stress Test Code
Test 4: Manual Commands

This test attempted to exploit the manual move function by sending manual move commands to the logic controller with pan and tilt parameters beyond the allowed limits. The test code can be found in Listing 6.4.

The ride vehicle never reached or exceeded any of the out-of-bound limits. In every case, when the logic controller received a request from the interface controller to move the vehicle to an out-of-bounds limit, the logic controller would start to move the vehicle towards the out-of-bounds set-point but when the next move command to be issued to the servos would move the ride vehicle beyond the limit, the logic controller would trigger an emergency stop. After a manual reset at the logic controller, the ride system continued to respond to valid commands as normal.

```python
def manual_operation():
    for i in range(25):
        send_thread(7)  # move command
        send_thread(random.randint(-50,50))  # tilt parameter -50 to 50
        send_thread(random.randint(-70,70))  # pan parameter -70 to 70
        send_thread(random.randint(0,100))  # time parameter 0 to 100
        # Sleep to let it finish (at least partially) the move
        time.sleep(15)
```

Listing 6.4: Manual Commands Test Code

Test 5: Open Restraints While Ride is Running

This test sent repeated commands to open the restraints while the ride was running. A failure for this test would occur if the logic controller ever permitted the restraints to be opened while the ride was running or not in the home position. The Python test code can be found in Listing 6.5.

At no point in the tests did the ride vehicle open the restraints when the ride system was running or the vehicle was not in the home position. However, many of the open restraints commands from the interface controller were interpreted incorrectly by the logic controller. The command
to open restraints is the number 6 but the logic controller would often receive the number 3. The number 3 (0011) is shifted one bit compared to 6 (0110) in binary which shows that the logic controller was missing one of the bits in the control sequences. This occurred when the logic controller was performing lots of computations. An examination of this issue led to the conclusion that this particular implementation of the SAP link is not ideal, as will be discussed in depth in Section 6.2.

```python
def run_ride_trying_to_open_restraints():
    for i in range(50):
        time.sleep(8)
        ride_start = time.time()
        send_thread(2) #Start ride
        while (time.time() < ride_start + 12):
            send_thread(6) #Open Restraints Command
            time.sleep(2 * FREQ)
```

Listing 6.5: Open Restraints While Ride is Running Test Code

### 6.1.2 Performance Tests

#### Test 6: Latency of SAP Link

This test measures the latency of the SAP link by timing how long it takes for the logic controller to reply to the interface controller after a “ping” command is sent. When activated from the GUI, the interface controller sends the ping command to the Arduino and records what time it was sent. Upon receiving the command, the logic controller sends the string “PING” to the interface controller via the reporting link. When the interface controller receives the “PING” string, it calculates the difference between the time the command was sent and when the reply was received. This time measurement measures how long it took to send the command over the SAP link, processing time on the Arduino, and the time for the reply to be sent from the logic controller to the interface controller via the reporting link.

The results are shown in Table 6.1 and show that the SAP link is quite slow as it takes a minimum of .146 seconds for a byte to be sent. This is expected since the baud-rate implemented
Table 6.1: Latency of SAP Link Results

<table>
<thead>
<tr>
<th>Delay (s)</th>
<th>Ride System Idling</th>
<th>Ride is Running</th>
<th>Ride Running w/o Updates</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.146</td>
<td>0.192</td>
<td>0.147</td>
<td></td>
</tr>
</tbody>
</table>

in the controllers is 64 baud. Since each bit takes 1/64 of a second to send, 9 bits (8 data bits and 1 start bit) would take .140 seconds \(9 \times \frac{1}{64} = .140\). The difference of .006 seconds between calculated and measured times is due to processing time required to parse the ping command and the delay from the return link. In addition to the calculated .140 seconds it takes to send a data packet, there are two additional stop bits that must be sent before the Arduino will look for data. This brings up the total time per data packet to .172 seconds. These stop bits were not measured in the ping latency test.

We can also see that it takes longer to send a reply to the ping command when the ride is running. This is due to the Arduino sending an update to the interface controller every time the pan, tilt, or arm angle changes or the emergency stop or running state changes. As can be seen by the third column, when those updates are disabled the delay is virtually the same as when the ride is idle. This implies that the Arduino is able to continue processing commands via the SAP link and continue to process its control loop with little impact. The latency was longer because the reply to the ping command was delayed as the Arduino’s serial link could not keep up with all the data being sent.

**Test 7: Arduino Loop Time**

This test explores the impact on normal processing in the Arduino while receiving and processing data through the SAP link. An Arduino operates by utilizing a `while` loop that runs forever. Within this loop is all the code that is needed to run the process. Figure 6.1 shows two plots of the time it takes to iterate through these loops as the ride runs normally and when the ride is running while receiving a steady stream of invalid commands through the SAP link. Every 1000 loops through the Arduino code, the Arduino reports to the interface controller how long it took to iterate through these loops.

As can be seen in the chart, the Arduino completes 1000 loops in about 100ms when idle. While the arm is raising, it takes about 300ms to iterate through 1000 loops. Once the ride is
moving, there are some jumps where it takes over 1000ms between 1000 loops. This occurs when the ride is moving around quickly and requires additional processing time to determine the necessary outputs for the rapid changes in movement. After the ride finishes and the arm begins to lower, the loop time returns to 300ms per 1000 loops. Once the arm is lowered, the loop time returns to 100ms per 1000 loops. The change in loop time is expected since, when the ride is idling, each loop has to do little besides check if any variables have changed. Once the ride is running, it has to compute a lot more between each loop.

It is interesting that there is little noticeable difference between the ride running without any data from the SAP link and the ride running with a stream of invalid data from the SAP link. There is some lateral shifting but it seems that both the max and min amount of loop time is about the same. This shows that the SAP link is not overburdening the logic controller implemented on the Arduino.

![Arduino Loop Time Graph](image)

Figure 6.1: Arduino Loop Time Results
Table 6.2: Raspberry Pi Load Results

<table>
<thead>
<tr>
<th></th>
<th>CPU Usage</th>
<th>Memory Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>GUI Idle</td>
<td>6%</td>
<td>2.2%</td>
</tr>
<tr>
<td>GUI receiving serial data</td>
<td>23%</td>
<td>2.2%</td>
</tr>
</tbody>
</table>

**Test 8: Raspberry Pi Load**

Measurements show that the interface controller implemented on the Raspberry Pi still has plenty of processing power for other processes. This was measured by using the Linux command `top`. Figure 6.2 shows how it is used in the command line and the results from it. The average CPU and memory use when idle and while under load was computed using this command.

As shown in Table 6.2, when the GUI is running but not sending commands over the SAP link or receiving serial data from the logic controller, it uses approximately an average of 5 to 6% of the CPU. When running the ride and receiving lots of serial data from the logic controller, it jumps up to a maximum of 23% of the CPU. Memory usage stays the same at 2.2%.

![Figure 6.2: Raspberry Pi Usage Statistics](image)

**6.2 Analysis/Discussion**

**Limitations of this implementation of the SAP Link**

There were some unexpected problems with this implementation of the SAP link. The logic controller would often interpret the commands sent by the interface controller differently than what was intended. Most of that was due to the fact that the logic controller checked the SAP link pin once a loop and if the loop took too long due to many calculations then it might miss a change in the value from the interface controller. While this was a problem, it never forced the ride vehicle into a dangerous state which was the focus on this research.
In a real system, errors like this would not be acceptable. Care should be taken to design the SAP link in a way that would not cause these types of errors. This shows that this specific implementation of the SAP link is flawed but the idea of a SAP link is still valid.

6.2.1 SAP Link Architecture Limitations

While there are significant advantages in implementing the SAP link architecture there are always trade-offs in any design. This section explores some limitations and the effects they would have on a system.

Less Flexibility

One of the limitations of this design is that programmers need to be more deliberate in the initial programming of a system. Since the safety logic cannot be modified remotely, any changes in desired set-points will require physical access to the logic controller. This may not be a problem in a small, localized system, but in large SCADA systems, such as an electrical distribution grid, manually accessing hundreds or thousands of controllers hundreds of miles away from each other could be difficult, costly, or nearly impossible.

Reduction of Speed and Bandwidth

The SAP link is designed to be slow and low bandwidth. While it needs to be fast enough for the specific system, it will generally take longer for a set-point from a Level 2 device to reach the Level 0 devices through a SAP Link controller than a non SAP Link controller. Additionally, the number of commands that can be sent within a given time period will generally be lower than in an alternate design. This could present a problem if the system is designed incorrectly or if the amount of data that needs to be sent is underestimated in the design process.

Cost

Following this design methodology and implementing the SAP Link architecture can lead to somewhat higher initial development, production, and maintenance costs. Updating the safety
logic could be more costly as no updates can be done remotely. Additionally, if the logic controller was poorly designed, production from the ICS could be hindered, which would raise costs until it could be updated. Since it will take longer to update all the Level 1 devices in a system, costs could go up. That being said, the cost of the consequences of a cyber attack can be very high, which could easily be more expensive than the additional costs from implementing this design methodology.

Annoyances

Even if an attacker cannot force a system into an unsafe state, the effects of an attack can still be annoying. For example, in the system we modeled, an attacker would not be stopped from enabling the ESTOP state repeatedly. The problem with this sort of attack is that there is no way to stop it with safety logic since it should always be possible to activate an ESTOP since it is always safer to stop a process than to let it run uncontrollably. To stop this kind of attack, the attacker would need to be removed from the system.

6.2.2 Attack Scenarios

This section discusses how ICSs with SAP Link controllers are effective in the following attack scenarios.

DOS Attacks

If the ICS is attacked by a denial of service (DOS) attack, the PLCs and other Level 1 devices may not receive legitimate commands from the Level 2 devices. Additionally, they may not be able to provide accurate, timely information about sensor data to the Level 2 devices. While there might be a disruption in service, the safety logic should prevent the system from entering a catastrophic state and should be able to maintain a minimum of service. Additionally, the SAP link minimizes the processing required on the logic controllers for ill-formed and illegal commands by limiting the amount of data that can be sent across it and immediately rejecting any data that does not conform to what it expects.
Malicious Commands Sent

Since it is entirely possible that an attacker could infiltrate the system and send illegal commands from the interface controller to the logic controller, it must be decided how the system will respond to commands that are not permitted through the SAP link. Will it reject them? Will it alert the operator? And if so, how will it alert the operator? Will it attempt to follow the command until it reaches an illegal state? For example if a water tank is asked to overfill will it fill it to the top and then stop or will it just ignore the command? These are all questions that the system designer must address. The desired response could vary based on the system, the command, and even based on the current system state.

No matter how the system designer decides to handle invalid commands, the system must always reject any command that will cause it to enter into a dangerous state. Due to the protections offered by the SAP link, an attacker will not be able to modify the safety logic to allow any of these malicious commands. By design, attackers will always be limited in what they can do.

Insider Attacks

Insider attacks are especially hard to defend against as insiders likely have knowledge of how the system works and may still have access to passwords, accounts, and the network. They may be current or former employees and may have been involved in the design of the system’s defenses against attacks. Additionally, if they are still working there, they could be part of the team investigating the attack and could further hide their tracks. Insiders can launch many different attacks much more easily since they can bypass the traditional IT security measures and render them ineffective. It does not matter how good your lock is, if the attacker has a key.

Despite the wide range of possible attacks from an insider, the SAP link provides a critical protection against them since an insider would need to have physical access to change the safety logic. While this is still possible, it is much more difficult if the attacker does not work for the company anymore. Additionally, if proper physical protections are in place, and an attacker does attempt to modify the safety logic, it should be easier to detect who did it with camera feeds and door access logs.
6.2.3 Soft ESTOP vs Hard ESTOP

There could be many reasons why the emergency stop might need to be triggered on a given machine. The machine could detect that there is a malfunctioning sensor or a ride operator could notice that the machine is making an odd noise. Normally it is desired that the machine stops quickly in a way that is nondestructive. This is known as a soft ESTOP. However, there may be times that a quicker yet potentially harmful stop would be needed which is known as a hard ESTOP. For example, if an operator notices that a collision is imminent with a human, loss of equipment is preferred to loss of life so a hard ESTOP would be required. The destructive hard ESTOP should not be accessible to an attacker through the interface controller but the soft ESTOP should be. To accommodate this, a hardwired hard ESTOP connected directly to the logic controller could be utilized while a soft ESTOP can be enabled via the interface controller.

6.2.4 Implementation

No design, however well designed, will help if never implemented. ICSs are rarely updated due to the cost and size of the systems. In a legacy system, SAP link controllers can be used to replace existing controllers since the interface controller can support normal SCADA protocols such as Modbus and DNP3. This will allow system administrators to gradually replace equipment in a way that is cost effective. Level 1 devices that control especially dangerous processes should be the first to be retrofitted.

6.2.5 Other Applications

While this research has focused on industrial applications, this same design approach can be used for any cyber physical system. For example, it could be used in the consumer IoT market to protect consumers from attacks. One potential use is for a smart thermostat. If an attacker could remotely access the thermostat, they could raise the temperature as high as possible which would waste energy and could potentially create unsafe conditions within the home. This could be launched when the owners are not home and might not be detected for days or weeks. If the system is designed with a SAP Link controller, users could use switches directly connected to the
logic controller to set allowable ranges for temperature and an attacker would not be able to exceed those ranges without being physically present.

While this might be a trivial example, it could also be used in security systems. By forcing users to register a digital key fob at the logic controller instead of being able to remotely register them, a system could be protected from attackers trying to register their own key fob.

The SAP Link controller architecture is a secure method to protect both industrial and consumer products. Implementing it in conjunction with other security measures can do a lot to limit what an attacker can do. While there are some limitations, the benefits of the SAP Link controller architecture make it a worthwhile addition to any Level 1 controller.
CHAPTER 7. CONCLUSION

7.1 Contributions

The main contributions of this thesis are:

- Proposing the SAP Link controller architecture
- Proposing a design methodology to implement the SAP link architecture
- Demonstrating a proof-of-concept implementation of the proposed controller architecture
- A survey of current ICS security research

7.2 Future Work

Since this test did not integrate an actual SCADA system, additional experiments integrating a SAP link enabled Level 1 device into a functioning SCADA system would further show how this new controller architecture would fit into existing architecture. Additionally, having more complicated equipment to control would also help prove the concept. These tests would guide further research into the data requirements for the SAP link and what kind of link would be best.

The proof-of-concept tests of the controller architecture relied on internal data of the logic controller. Further tests could be completed by using an external device with independent sensors on each of the servos and indicator LEDs. This external device could verify that none of the constraints were violated. This would show that the logic controller really did operate as expected, and not just that it reported that it did.

Additionally, the safety logic would be more effective if it implements a system model of the industrial process and checks the actual state against the modeled state. This would not only verify that the industrial process is not exceeding safety limits, but would also ensure that it is operating as desired. This model should also compute the first derivative of the system variables to
ensure that they are not changing at a faster rate than desired. Since we have simplified the inputs to the logic controller by using the SAP link, future research in formal verification of the state model would demonstrate and prove the logic controller’s security.

There is still work to be done to determine the best communication protocol for the SAP link. More research could be done to determine how to select a link for a particular system or to design a flexible protocol that can be adapted on a case by case basis. This will be very helpful as the link that was implemented in Chapter 5 was slow and somewhat error prone. While the errors never resulted in an out-of-bounds operation, the logic controller did not always receive the correct information. Additional features of the link could include a quick way to enable an emergency stop and built-in error correction.

More research will be useful to determine the best hardware to implement the logic controller and interface controller. The Arduino and Raspberry Pi worked well as a proof-of-concept, but would likely be inadequate for anything but the most basic of control systems. Ideally, the logic controller, interface controller and SAP link would all be implemented on a PCB or even on the same chip. The physical separation would remain essential but research could explore how this is best done in a small package. This could provide additional benefits of lower power consumption and the smaller size would allow for more applications.
REFERENCES


