1-1-2012

Vulnerability Analysis Case Studies of Control Systems Human Machine Interfaces

Robert Wesley McGrew

Vulnerability analysis case studies of control systems human machine interfaces

By

Robert Wesley McGrew

A Dissertation
Submitted to the Faculty of
Mississippi State University
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy
in Computer Science
in the Department of Computer Science and Engineering

Mississippi State, Mississippi

May 2012

Vulnerability analysis case studies of control systems

human machine interfaces

By

Robert Wesley McGrew

Approved:

Name: Robert Wesley McGrew

Date of Degree: May Day, 2012

Institution: Mississippi State University

Major Field: Computer Science

Major Professor: Dr. Rayford B. Vaughn

Title of Study: Vulnerability analysis case studies of control systems human machine interfaces

Pages of Study: 122

Candidate for Degree of Doctor of Philosophy

This dissertation describes vulnerability research in the area of critical infrastructure security. The intent of this research is to develop a set of recommendations and guidelines for improving the security of Industrial Control System (ICS) and Supervisory Control and Data Acquisition systems software. Specifically, this research focuses on the Human-Machine Interface (HMI) software that is used on control panel workstations.

This document covers a brief introduction to control systems security terminology in order to define the research area, a hypothesis for the research, and a discussion of the contribution that this research will provide to the field. Previous work in the area by other researchers is summarized, followed by a description of the vulnerability research, analysis, and creation of deliverables. Technical information on the details of a number of vulnerabilities is presented for a number of HMI vulnerabilities, for which either the author has performed the analysis, or from public vulnerability disclosures where sufficient information about the vulnerabilities is available.

Following the body of technical vulnerability information, the common features and characteristics of known vulnerabilities in HMI software are discussed, and that information is used to propose a taxonomy of HMI vulnerabilities. Such a taxonomy can be used to classify HMI vulnerabilities and organize future work on identifying and mitigating such vulnerabilities in the future.

Finally, the contributions of this work are presented, along with a summary of areas that have been identified as interesting future work.

Key words: SCADA, HMI, control systems, vulnerabilities, exploits, security

DEDICATION

I would like to dedicate this work to my wife, Crystal, who has supported me throughout my pursuit and completion of a doctoral degree.

ACKNOWLEDGEMENTS

I would like to express his thanks to my family and friends, especially my wife Crystal, for supporting him through the process of obtaining my Ph.D. I would like to also thank the members of my committee: Dr. Rayford B. Vaughn, Dr. David A. Dampier, Dr. Mahalingham Ramkumar, and Dr. Thomas H. Morris, for encouraging this line of research and assisting in the preparation of the materials for presenting and publishing it. I would also like to thank the undergraduates, graduate students, and research associates that work in the National Forensics Training Center and Critical Infrastructure Protection Center for being available to discuss ideas, concepts, and vulnerability information.

Finally, I would like to thank the researchers who originally discovered, documented, and disclosed the vulnerabilities that I have described and used in the creation of a taxonomy in this dissertation, alongside the vulnerabilities that I have published work on. This especially applies to Luigi Auriemma, who has published a truly prolific collection of well-documented vulnerabilities in SCADA systems. I understand that the amount of work that is put into each vulnerability discovery far exceeds what anyone outside of the field of vulnerability analysis probably realizes, so I have a lot of respect for the other researchers in this field.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS, ABBREVIATIONS, AND NOMENCLATURE

ICS	Industrial Control System
SCADA	Supervisory Control and Data Acquisition
HMI	Human Machine Interface
PDD	Presidential Decision Directive
PLC	Programmable Logic Controller
RTU	Remote Terminal Unit
IDE	Interactive Development Environment
IT	Information Technology
HVAC	Heating, Ventilation, and Air Conditioning
DIDS	Distributed Intrusion Detection System

CHAPTER 1

INTRODUCTION

Critical infrastructure is a term used to define what services and processes are most important to the functioning of an organization. On a larger scale, national critical infrastructure is made up of the services that are required to guarantee the continued operation of a nation's services to its citizens and defense. In 1998, United States Presidential Decision Directive 63 (PDD-63) used the following as examples of services that are "essential to the minimum operations of the economy and government" in the United States [40]:

- Telecommunications
- Energy
- Banking and finance
- Transportation
- Water systems
- Emergency services

PDD-63 posits a scenario where enemies of the United States (including "nations, groups, or individuals") would attack in non-traditional ways that target critical infrastructure. Large portions of the critical infrastructure are created and maintained by the private sector, leaving the security of that infrastructure outside of the control of the national government. Assets owned by the private sector, only affected by the policies and compliance guidelines set by the government, include telecommunications, emergency health care and the distribution of critical goods such as food and fuel [40].

A unique problem with protecting national critical infrastructure is that the security of portions of it may be out of the control of that nation's government. Assets owned by the private sector, only affected by the policies and compliance guidelines set by the government, include health care and the distribution of critical goods such as food and fuel.

Critical infrastructure frequently relies on automation, and that automation takes the form of what are commonly referred to as industrial control systems (ICS) or supervisory control and data acquisition (SCADA) systems. SCADA is the acronym most commonly used to describe critical infrastructure automation in popular literature and media, however ICS is the more general term and encompasses systems that are not normally considered to be "supervisory". SCADA systems typically consist of end-point units, programmable logic controllers (PLCs) or remote terminal units (RTUs), that are responsible for monitoring and controlling systems in real-time, while parameters of that control are monitored and modified at centralized locations. In a SCADA system implemented with true supervisory control, the temporary lack of a communications channel would result in the end-point units continuing to operate the system normally within the parameters that were last sent to them. ICS systems are more generally defined to include systems where aspects of the control system are directly monitored and manipulated remotely. The real-time nature of such a system has the potential to make it more sensitive to the loss of a communications channel.

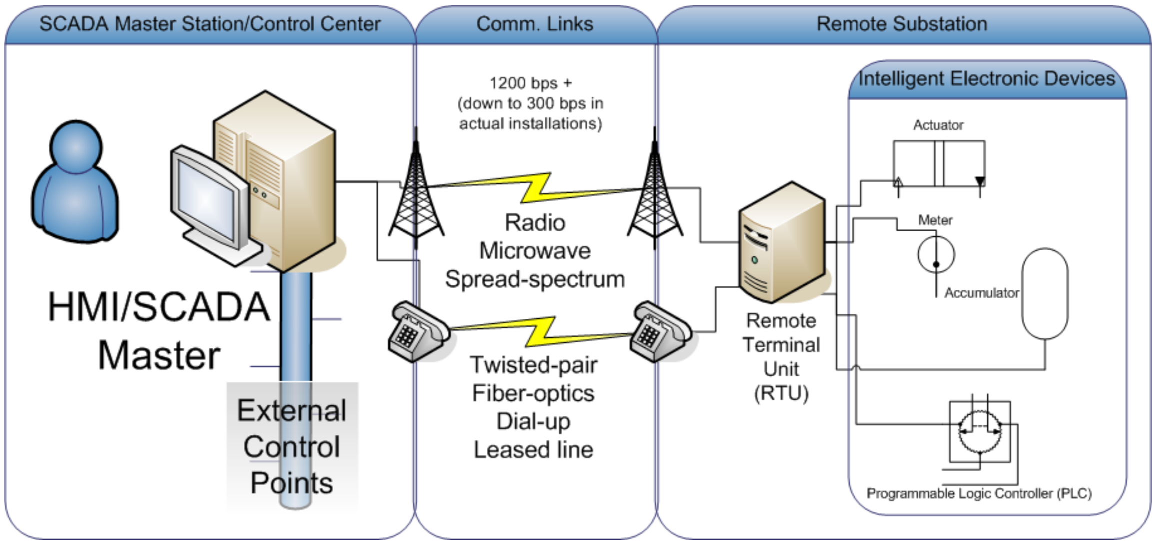

SCADA Master Station/Control Center
Comm. Links
Remote Substation
1200 bps + (down to 300 bps in actual installations)
Intelligent Electronic Devices
Actuator
Meter
Accumulator
Radio
Microwave
Spread-spectrum
HMI/SCADA Master
External Control Points
Twisted-pair
Fiber-optics
Dial-up
Leased line
Remote Terminal Unit (RTU)
Programmable Logic Controller (PLC)

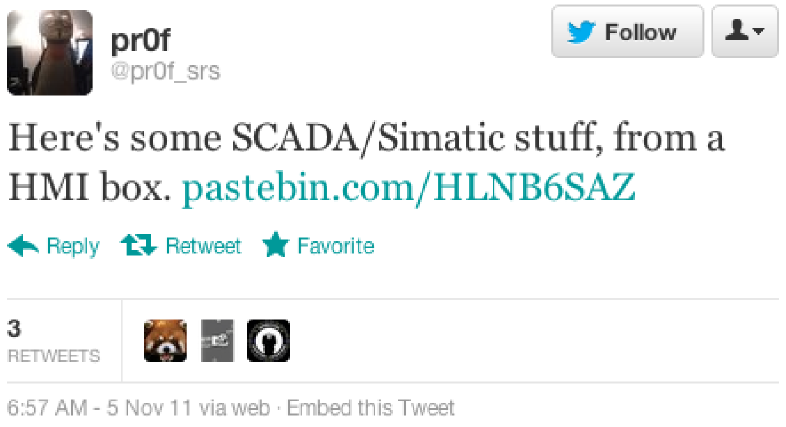
Figure 1.1

HMI and RTU/PLC elements of a SCADA system.[45]

Human-machine interface (HMI) software is used to monitor, manipulate, and log data on ICS/SCADA systems in all areas of critical infrastructure. A simplified view of the relationship between the HMI and RTU/PLC elements of a SCADA system can be seen in Figure 1.1.This software is typically found in control panels connected to hardware, as well as centralized touch-screens and general purpose computers located in central monitoring and management facilities. HMI software is typically distributed as development environments that can be used by the end-user or a solutions provider to develop a user interface for a specific ICS/SCADA system in a similar manner as interactive development environments (IDEs) are used to develop mainstream information technology (IT) software. This is necessary, since there is wide variation in HMI software features needed by different elements of infrastructure, or even between two separate implementations of the same infrastructure. The wide variety of features that HMI software implements increases the attack surface that might potentially contain vulnerabilities. It is not unusual to see an HMI system that implements everything from local user authentication and access control, to network communications with remote PLCs, for example. Many of the file formats and network protocols used by HMI systems are not common in mainstream IT software, so simple parsing bugs that result in exploitable vulnerabilities are likely to be common.

Recently, there has been a documented increase in interest in control systems by attackers, including those that are involved in the national critical infrastructure. In a recent National Cybersecurity and Communication Integration Center bulletin, the United States Department of Homeland Security discusses the rising interest that groups such as “Anonymous” have in locating and attacking critical infrastructure control systems. A reference is made in the bulletin to a posting by a member of Anonymous that has, on multiple occasions, posted to the Twitter social network boasting about compromising SCADA systems. In most cases, as in the screen shot in Figure 1.2, the attacker primarily interacts with the HMI portion of the system.

The first control systems hacker in the United States to be convicted and serve jail time for his SCADA hacking activity was Jesse William McGraw, a security guard at a Dallas, Texas area hospital. McGraw abused his privileged access to the building during the night-time hours to compromise the computer system that controlled the heating, ventilation, and air conditioning (HVAC) system of the facility. In addition to this, the system served other critical control roles for the hospital, such as maintaining the temperature of medicines that require specialized storage. McGraw posted screen shots of the compromised system online, boasting of his accomplishments to his hacker peers, and was in the



pr0f
@pr0f_srs

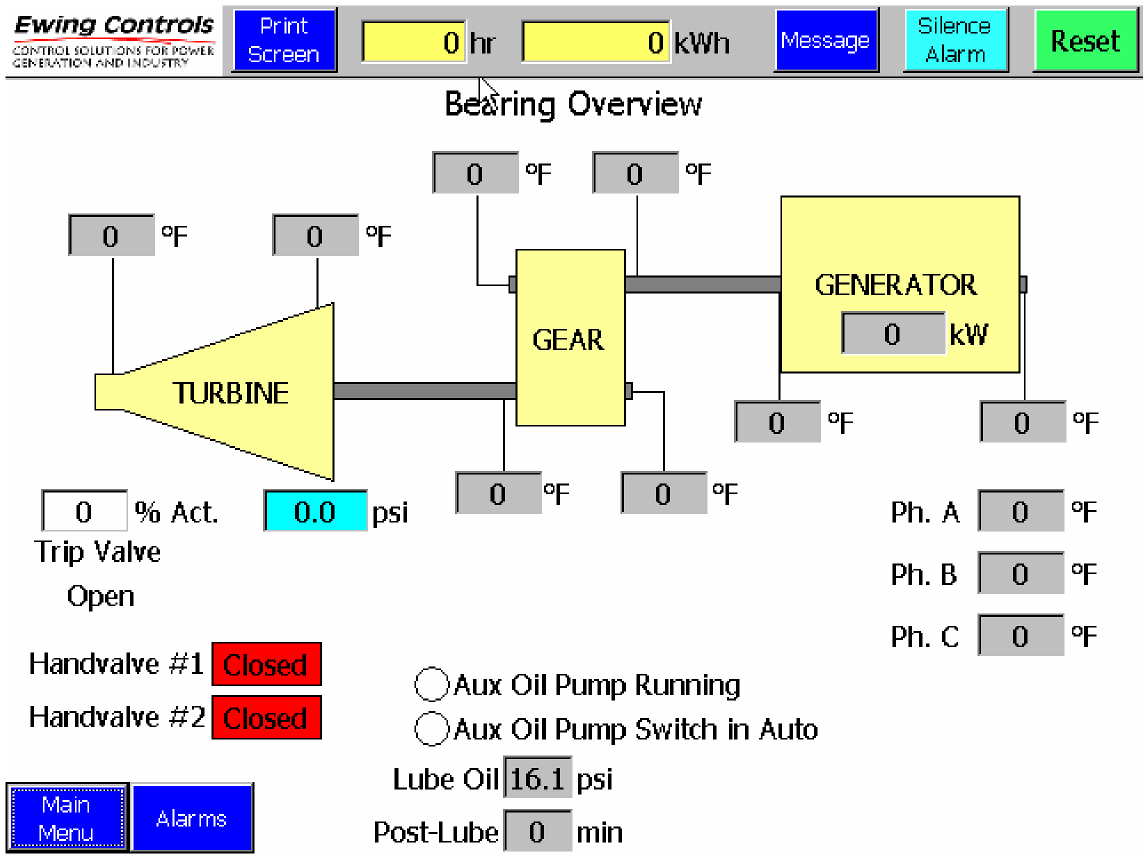
Follow

Here's some SCADA/Simatic stuff, from a HMI box. pastebin.com/HLNB6SAZ

Reply Retweet Favorite

3 RETWEETS

6:57 AM - 5 Nov 11 via web · Embed this Tweet



Ewing Controls
CONTROL SOLUTIONS FOR POWER GENERATION AND INDUSTRY

Print Screen | 0 hr | 0 kWh | Message | Silence Alarm | Reset

Bearing Overview

0 °F 0 °F 0 °F 0 °F

GENERATOR 0 kW

GEAR

TURBINE

0 °F 0 °F 0 °F 0 °F

0 % Act. 0.0 psi

Trip Valve
Open

Ph. A 0 °F
Ph. B 0 °F
Ph. C 0 °F

Handvalve #1 Closed
Handvalve #2 Closed

Aux Oil Pump Running
Aux Oil Pump Switch in Auto

Lube Oil 16.1 psi
Post-Lube 0 min

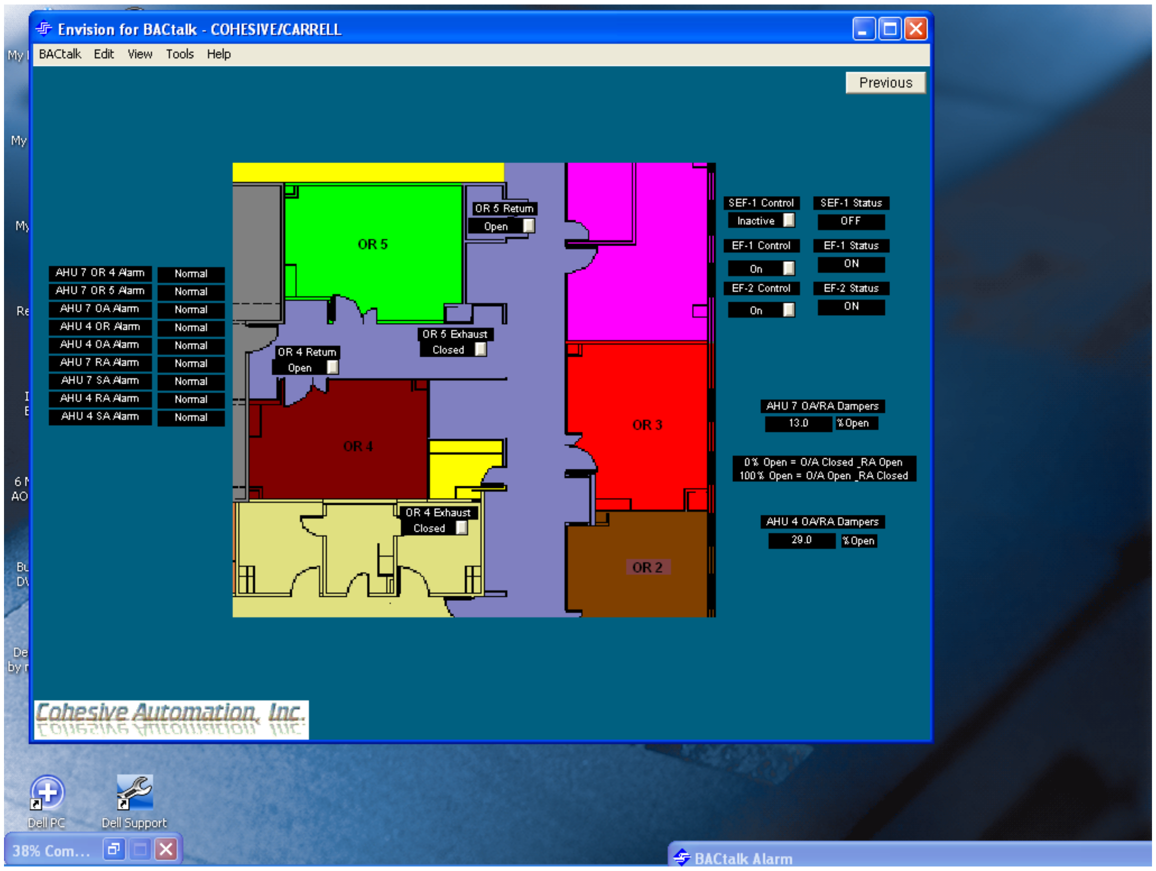
Figure 1.2

Anonymous-affiliated hacker boasts about SCADA HMI hacks process of compromising more systems for use in a distributed denial of service (DDOS) attack against United States government websites when he was arrested. The author of this dissertation discovered the on-line evidence of the crime being committed, investigated, and turned over the evidence and information gathered on McGraw to the FBI. A screen shot of the HMI system compromised by McGraw is shown in Figure 1.3, which displays a set of operating rooms, and controls related to ventilation in the displayed rooms. McGraw is currently serving a sentence of over nine years in federal prison for compromising the systems of the hospital. The law under which he was convicted has been changed since the PDD-63 document was published, in order to impose harsher punishment on those convicted of attacking systems that are related to public health and safety, such as many parts of our national critical infrastructure.

The widely reported-on and analyzed Stuxnet worm targeted the PLC portion of a control system. In order to infect the PLCs with malicious code, it looked for and used the presence of HMI software in order to bridge the gap between infecting PCs and PLCs [39]. This illustrates that the security of HMI software is critical to the security of a control system, and that the HMI is of high value to an attacker for gathering intelligence on the operation of a control system, and as a vector of attack for other portions of the system.

Vulnerability analysis, in the context of software application security, is a process that contains the following activities:

- The discovery of errors in programming or design that may constitute a security vulnerability
- Determine exploitability
- Development of proof-of-concept exploit code


Envision for BACtalk - COHESIVE/CARRELL
BACtalk Edit View Tools Help
Previous
OR 5
OR 4
OR 3
OR 2
Cohesive Automation, Inc.
Dell PC
Dell Support
38% Com...
BACtalk Alarm

AHU 7 OR 4 Alarm	Normal
AHU 7 OR 5 Alarm	Normal
AHU 7 OA Alarm	Normal
AHU 4 OR Alarm	Normal
AHU 4 OA Alarm	Normal
AHU 7 RA Alarm	Normal
AHU 7 SA Alarm	Normal
AHU 4 RA Alarm	Normal
AHU 4 SA Alarm	Normal

OR 5 Return: Open
OR 5 Exhaust: Closed
OR 4 Return: Open
OR 4 Exhaust: Closed

SEF-1 Control: Inactive
SEF-1 Status: OFF
EF-1 Control: On
EF-1 Status: ON
EF-2 Control: On
EF-2 Status: ON

AHU 7 OA/RA Dampers: 13.0 % Open
0% Open = O/A Closed ,RA Open
100% Open = O/A Open ,RA Closed
AHU 4 OA/RA Dampers: 29.0 % Open

Figure 1.3

Screen shot of HMI compromised in attack on hospital control system.

- Identification of potential mitigation and patch techniques

Vulnerability analysis, performed by malicious attackers, might give an attacker with the intent of compromising a system to damage or profit from it criminally a distinct advantage over other attackers and those responsible for defending an ICS/SCADA system. Vulnerability analysis undertaken by security researchers, as is described in this dissertation, has the potential to dramatically improve the security of ICS/SCADA systems by identifying weaknesses so that they may be patched or mitigated before malicious actors can exploit them. It is unknown how many vulnerabilities in ICS/SCADA HMI systems are known by attackers that are not known by the public, HMI vendors, or national critical infrastructure stakeholders. It is the goal of this research to take the benefits of vulnerability analysis a step further, and use the results to inform the development of more secure systems in the near future.

1.1 Hypothesis

It is hypothesized that building a taxonomy of vulnerabilities through case studies of discovering, analyzing, and exploiting previously unpublished HMI vulnerabilities, alongside existing vulnerability information available in the public domain, is possible and can serve as a set of guidelines for implementing sound basic security principles in control systems software in the future. Due to the unique set of tasks that HMI systems must handle, and the relative lack of consideration given to security in the design and architecture of HMI systems, many vulnerabilities will be the result of similar mistakes being made during the development process. A taxonomy of HMI vulnerabilities, with well-documented case studies, would allow for a set of practical and actionable recommendations to be made to HMI developers to prevent the same vulnerabilities from showing up in future HMI products. The same documentation would also provide vulnerability analysts a starting point for identifying potential vulnerabilities in existing HMI software.

If the vulnerabilities discovered through the process of compiling the case study have commonalities, such as specific security principles that they routinely violate, or design and implementation errors that are common across multiple products, then this would indicate that there is a lack of appropriate documentation and training on security available to the developers of many, if not most, HMI products. If the new vulnerabilities documented in this case study show common features with the body of existing and published HMI vulnerabilities, then this would strongly indicate that a common set of design and implementation flaws are responsible for weaknesses in national critical infrastructure.

1.2 Contribution

This dissertation, as a collection of recommendations and guidelines backed by examples and case-studies of actual HMI vulnerabilities, will serve as a resource for ICS/SCADA software developers, maintainers, stakeholders, and solutions providers. This resource will allow them to more easily take advantage of mainstream IT security research in improving the security of their own products and systems by identifying what is unique and important to the security of HMI software. This will be a significant step forward for practitioners in the area of HMI software security, who serve as one the most important lines of defense for the national critical infrastructure.

In addition to this primary goal, this research will serve as a useful resource for vulnerability researchers and "red team" penetration testers that are tasked with identifying new vulnerabilities in the development and deployment of HMI systems. While the developers mentioned in the earlier this dissertation have experience in the ICS/SCADA field and would benefit from the security testing aspects of the document, vulnerability researchers and penetration testers are familiar with the security aspects, and would benefit from the ICS/SCADA-specific material. By providing information on common vulnerabilities in HMI software, the time a "red team" would need to familiarize themselves with a SCADA target would be reduced, allowing them to be more efficient and effective in their testing. Overall, this would result in better security recommendations for improving the security of individual elements of national critical infrastructure.

This research may also form the basis for future research in generating recommendations for specialized areas of software development that have not traditionally seen much attention from vulnerability analysts. By following a similar pattern of vulnerability analysis and comparing the results for a set of software products in the same specialized area, researchers may be able to develop similar recommendations that are suited for those specific areas of software development. This dissertation may have some positive impact on the development and testing of secure software systems in other specialized areas.

An immediate benefit of the research will be the responsible disclosure of a set of vulnerabilities in HMI products. Disclosures will allow the vendors to patch existing HMI products, or at the very least publish mitigation strategies. End-users of the HMI products, including organizations responsible for national critical infrastructure, can implement these patches and mitigation strategies to improve the security of their ICS/SCADA systems. Publicly disclosed vulnerability information removes the power that criminal or malicious attackers may have in the same vulnerabilities that they have discovered, but not disclosed, for their own benefit.

By disclosing not only the details of the vulnerabilities and exploits, but also the vulnerability discovery processes and actions that led to discovery of the vulnerabilities, vulnerability analysts may benefit directly from this research. Vulnerability analysts that are tasked with finding vulnerabilities in other HMI products would benefit from being able to start their work from techniques presented in case-studies. Information security students would also benefit from having a start-to-finish example of how real-world security problems are identified, analyzed, and exploited.

CHAPTER 2

PREVIOUS WORK

Advances in vulnerability analysis have largely been driven by the private sector information security industry, due in parts to the monetary value of vulnerability information to businesses such as intrusion detection system signature providers and penetration testing firms. While the origins of some vulnerability analysis techniques such as “fuzzing” (a form of fault-injection that involves automated generation of inputs that may highlight security weaknesses) have their simplest roots in academia [30], it is typically the private sector that develops vulnerability analysis techniques into mature tools such as the ones that are used in this research for vulnerability analysis [38]. In the case of exploit development, many techniques have their roots in more “underground” publications, as is the case with the most common form of memory corruption exploitation: stack overflows [1]. One goal for the deliverable guidelines and recommendations in this research is to describe the vulnerability discovery process, vulnerability information, and exploitation in enough detail to provide useful insight to ICS/SCADA developers, and to provide better documentation to “red team” vulnerability analysts tasked with finding vulnerabilities in similar systems.

One of the threats that SCADA systems are most vulnerable to is the threat of malicious insiders. Insider threats are defined as being users that have elevated their privileges above and beyond what they are allowed by an organization's security policy, and outside of the scope of what they need to fulfill their duty in the organization. Insider attackers have unique opportunities when compromising systems [4]. For SCADA systems, one huge advantage that an insider has is physical access to the hardware and the software that runs on it. Given this level of access, it far more difficult to prevent escalation of privilege.

2.1 Vulnerability Taxonomy

Hansen et al [16] describes a taxonomy of vulnerabilities in implantable medical devices that categorizes vulnerabilities in terms of the proximity needed to exploit the vulnerability, the activity that is required for exploitation, the required state of the patient, components affected, and how permanent the damage is. SCADA systems share many traits of medical systems (and on a larger scale, medical systems may be implemented as SCADA) such as having software control physical processes, and the critical nature of its operation. The traits described in Hansen's taxonomy may be adaptable to a SCADA HMI classification, especially considering that proximity (local, remote, level of hardware access) is a factor in many SCADA vulnerabilities.

Yan and Randall [47] presented a vulnerability taxonomy to describe cheating in online gaming that involves classification of features along a two-dimensional grid, with one axis describing the method by which a compromise was made and the other axis describing traits of each vulnerability including these that may have application to SCADA HMI vulnerabilities:

- Design issues of the game system and underlying system
- Human error on the part of the game operator or player
- Integrity violation
- Denial of service
- Theft of information or possession
- Masquerading

From current work in HMI vulnerability analysis, it is obvious that a large portion of the vulnerabilities involve the amount of trust placed in client software to authenticate and authorize the end user and his or her actions. This is very similar to the challenges that on-line games face in loading bounds-checking and security code into the client, versus the server. In gaming, this is a trade-off of performance, latency, and security. In SCADA HMI systems, the architecture and design of the system may dictate where security features are implemented.

Briesemeister et al, in their description of a project for detecting, correlating and visualizing threats against critical infrastructure [5], describe the use of "event classes" in their system to help classify control system incidents across multiple sensors. Incidents are given a set of classes, where the most serious incidents involve denial of service or distress to the assets of a control system, and less severe incidents involve probes of the system and suspicious usage. Assets also contribute to the measure of criticality of an incident, where RTU/PLC incidents are given a much higher criticality than HMI incidents. While this is a categorization of incident types, and the relative criticality of one portion of the system or another may not be accurate in all cases, the terminology and divisions used may be useful in describing a taxonomy of HMI vulnerabilities.

2.2 Information Security Metrics

A vulnerability taxonomy for HMI vulnerabilities is one step in improving the ability to measure the security of a critical infrastructure control system as a whole. Information security metrics can be defined as a set of criteria for measuring the effectiveness of information security implementations. NIST, in their "Performance Measurement Guide for Information Security" [8], which supersedes a document entitled "Security Metrics Guide for Information Technology Systems", defines a metric or measurement as being the "results of data collection, analysis, and reporting". Among other measures, NIST describes measures of effectiveness and efficiency, which are used to determine if security controls are correctly implemented. In the area of HMI security, a taxonomy of vulnerabilities could be used to evaluate an HMI product in order to determine the effectiveness and efficiency of its controls.

Vaughn, Henning, and Siraj [43] propose a taxonomy of metrics that include types that are described by a set of categories, each of which can be classified in two states. These categories include objective/subjective, quantitative/qualitative, static/dynamic, absolute/relative, and direct/indirect. When using vulnerability information to measure HMI security, the metric may become more objective than subjective, due to there being a base set of common flaws that can be tested against. Also, the security of an HMI product may be more directly observed, given a taxonomy of common flaws. HMI security would be considered relative, as measuring the security of an HMI product does not take into account the mitigating controls surrounding it, and dynamic, as the product configuration, and known vulnerabilities, both may change over time. Metrics for technical objects are also presented, where a set of common HMI vulnerabilities may be useful in measuring the adversarial work factor needed to compromise the system, and the strength of the security of the products and implementation in general. A set of known vulnerabilities in a specific product can be used to measure the weakness that product has in an organization's environment, which would inform the decision of what security and mitigation techniques need to be layered around the product, such as firewalls, VPNs, and intrusion detection.

2.3 Vulnerability Analysis Tools

The following types of existing vulnerability analysis tools will be utilized heavily in the vulnerability discovery process. For each, a specific tool is named, as well as the purpose for each in the process of this research:

- System tracing/logging utilities - Used to locate important files and registry entries that products might be using to implement security features. The Sysinternals Suite contains a wide variety of these tools [29].
- Network monitoring software - Used to identify network traffic being generated and accepted by the products being tested. Necessary for reverse engineering network protocols. Wireshark is the primary monitoring tool that will be used [23].
- Debuggers - Used to trace through program execution to determine how security features are implemented, and to assist in the exploit development process. Immunity Debugger will be used for most of this testing [36].
- Disassemblers - Used to reverse engineer and understand the implementation of compiled binary software. IDA Pro provides the best graphing capabilities of any current disassembler [12].

- Fuzzing Tools - Used to generate random and mutated inputs for a program automatically, in order to expose network protocol and parsing flaws. SPIKE is a common fuzzer, however in this research it is likely that a variety of others will be used [38].
- Scripting tools - Used to develop exploits and ad-hoc tools for situations that existing vulnerability analysis tools do not adequately cover. Python and Ruby (as part of the Metasploit Framework) will be used for this purpose [36].

Fuzz testing, compared with the other tools and techniques in the above list, require much more involved techniques and processes for the vulnerability analyst. The earliest example of fuzz testing in academic literature was inspired by a program crashing after line noise over a modem connection introduced by a lightning strike caused a Unix program to crash attempting to parse the spurious input. This led to the development of a suite of utilities meant to test the error handling capabilities of command-line programs. These programs provided random input to the target software, monitored the software for crashes, and then logged the string combinations that cause the software to fail. Afterwards, the strings that caused crashes were analyzed alongside the code for the software in order to determine what had caused the crash [30]. Beyond the normal desire for reliability in software, crashes of this nature often indicate the presence of bugs that represent a security risk. By examining the cause of the crash, a vulnerability analyst may be able to control the execution path of the software, and create an exploit that allows for remote access, or escalation of privilege. Similar techniques are used today by many vulnerability analysts to discover the location of vulnerabilities in software where source code is not openly available. For closed-source software, this technique has the benefit of revealing the locations of potential vulnerabilities without prior knowledge of the internals of the software [38].

2.4 Applying Vulnerability Analysis to SCADA

Fernandez and Larrondo-Petrie have presented work on applying commonly accepted security patterns, or best practices, to the development of SCADA systems [13]. The focus of the work has been on vulnerabilities associated with communications between various aspects of a SCADA network, with much less focus on vulnerabilities and security features in the HMI components. Rather than focusing on the entire SCADA system, this work focuses on the secure development of the HMI, which has its own unique functionality within a SCADA network, and its own set of common vulnerabilities. For example, client-side file format attacks are much more feasible for attackers to perform successfully against HMI systems than other elements of a SCADA system.

Cagalaban and Kim have presented work on improving SCADA security through vulnerability analysis [7]. In this case, however, the term "vulnerability analysis" is used to describe a scoring system for rating the risk of a vulnerability, rather than the vulnerability discovery, examination, and exploitation process. This research focused on fault-injection, comparable to the fuzzing techniques used to discover vulnerabilities in mainstream IT software. This work focuses on the protocols involved in SCADA communications, rather than on the HMI.

2.5 SCADA Intrusion Detection

Many current efforts in SCADA intrusion detection focus on detecting attempt to exploit portions of a control system that involve protocols unique to SCADA, such as ModBus and DNP3. Shosha et al [37] describe a distributed intrusion detection system (DIDS) approach to detect attacks on SCADA networks. A DIDS would be able to use sensors in multiple locations in the control system network to detect anomalous behavior such as man-in-the-middle attacks on the DNP3 protocol. Other approaches, such as Barbosa and Pras' [2] involve taking advantage of features of control system networks that limit the range of various activity that take place when compared to traditional IT networks. By assuming that SCADA networks have a relatively fixed number of devices, small number of protocols in use per network, and regular patterns of communication defined by the programming of the PLCs and HMIs, anomalies that step outside of those boundaries become easier to detect.

One benefit of HMI vulnerability analysis, and building a taxonomy of HMI vulnerabilities, is that a large portion of the vulnerabilities involved are exploitable over conventional IT networks and protocols, such as web or remote desktop traffic implemented on top of a TCP/IP network, or locally as a form of privilege escalation on the host machine running the HMI. By increasing the body of knowledge surrounding these HMI vulnerabilities, signatures can be written that more generally detect the kinds of attacks that are commonly successful against these systems. This would give conventional IT intrusion detection and host protection products the ability to detect and react to many intrusions of HMI systems.

2.6 HMI Vulnerability Analysis

While HMI products have not seen the same vulnerability analysis attention as mainstream IT software, some vulnerabilities have been discovered and disclosed. One of the most popular HMI products, Wonderware, has had memory corruption vulnerabilities discovered and disclosed, which would open up the possibilities for attackers to run arbitrary, malicious code. In 2008, Core Security disclosed a vulnerability in Wonderware's SuiteLink service, which allows for communications between elements of a Wonderware system. This vulnerability would allow remote, un-authorized attackers to crash and possibly assume control over Wonderware systems. A mishandling of a large length field in the SuiteLink protocol is responsible for the vulnerability [10]. While the Core Security advisory does not elaborate on how the vulnerability was found, a protocol fuzzing process would likely be able to find the same vulnerability. It is common for vulnerability advisories to not include the processes used to discover the vulnerabilities in question, and it is not always immediately obvious from the vulnerability itself. Case-study documentation of these processes may be a useful contribution of this dissertation research.

Siemans' WinCC has also had a memory corruption vulnerability discovered and disclosed, in June/July 2011. The advisory for this vulnerability indicates that it is a file parsing bug that is responsible. With this kind of vulnerability, attackers would be able to, with minimal user interaction, assume control of HMI systems [19].

Luigi Auriemma has publicly disclosed many advisories on SCADA and HMI systems over the past few years [24]. Some of these vulnerabilities are trivial to exploit, including a recent one disclosed in the SpecView HMI software, where a directory traversal bug in a web server allows an attacker to access any file on the server that is hosting the software. In the Pro-face Pro-Server EX software, Luigi discloses memory leak and corruption vulnerabilities, in addition to the trivial encoding of the password (exclusive-or against a static key). These vulnerabilities echo characteristics of vulnerabilities found by the author in other HMI packages.

GLEG Ltd. has a collection of what it advertises as a pack of all publicly available SCADA exploits as a pack of exploits for the Immunity Canvas framework. While the exploit pack is a paid product, the list of updates provides insight into the types of vulnerabilities included, many of which revolve around password disclosure, information disclosure, and improper access control [15]. Many of the included exploits are based off of vulnerability information that Luigi has released, alongside several “zero day” vulnerabilities (despite the “public” advertised nature of the package). For this research the vulnerability information, coupled with the product names, can serve as a good source of products to include in case studies.

CHAPTER 3

RESEARCH AND EXPERIMENTATION PLAN

This chapter summarizes the plan for this research and experimentation, including discussion of the case study format and research questions.

3.1 Overview

As a research project directly involving case studies, Zainal recommends a set of steps that are involved in the process [48] and are adapted here for the purpose of a software vulnerability case study:

- Defining research questions
- Case selection
- Data collection
- Data analysis
- Reporting

3.2 Research Questions

In this dissertation research, analysis of the case studies should provide answer to the following questions:

- Do many HMI products have flaws in the implementation of security features?
- Are these flaws exploitable by attackers?

- Are there similarities to the flaws in HMI products from different vendors?
- Based on these similarities, can HMI vulnerabilities be classified into a taxonomy?
- From this taxonomy, can a set of recommendations be made of secure development practices that would prevent future products from having the same types of vulnerabilities?

3.3 Case Selection

Several ICS/SCADA Human-Machine Interface products were be selected for this research. Criteria for selection included popularity of the product, as determined by search engine ranking, level of discussion on public forums, and the presence of critical infrastructure clients in advertising. Other criteria included the level of discussion or interest in the HMI product on "hacking" forums that have been identified to discuss ICS/SCADA security, the functionality of the HMI product's demonstration or development copy, and the immediately obvious attack surface that is exposed. Using this process to select HMI products resulted in case studies and examples that practitioners in the field will find familiar, and will also have a higher immediate impact on the security of HMI products that are in wide use. Products that have multiple interfaces, such as a captive local GUI and an external web interface, were among those chosen.

3.4 Data collection

The processes described in this section were used to examine the HMI software in each case of the study, in order to locate vulnerabilities in the software, analyze the vulnerabilities, and gather the information needed to answer the research questions.

3.4.1 Vulnerability Discovery and Analysis

For each product, vulnerability analysis was undertaken to determine a set of possible security flaws. For each of these flaws, the other selected HMI products were analyzed for similar vulnerabilities. By using the results of vulnerability analysis for one HMI product to inform the analysis of the other selected products, vulnerabilities that are inherent to the design or development techniques being used by HMI developers were identified.

3.4.2 Attack Surface

The first step in vulnerability analysis of each product was to determine the attack surface available. This determination was made for both the external attack surface made available to remote attackers in the form of network services (such as web-based interfaces and ICS/SCADA protocols), as well as local interfaces that may give attackers the opportunity to escalate privileges. By determining the available attack surface of the product upfront, the rest of the vulnerability analysis process was tailored for that particular product, while maintaining some consistency across products.

3.4.3 Security Feature Targeting

For each product, an element of the vulnerability analysis involved identifying all of the "advertised" security features that are presented in the documentation and sales materials for the product. These are features that an end user of the product can reasonably assume to either protect them from compromise or maintain appropriate privileges for each class of user on the system. For each of these features, the implementation was examined to determine if it has been implemented with regard to current best practices in secure development, and to determine if there are any flaws in the implementation that can be exploited to circumvent the security control or otherwise take control of the product.

3.4.4 File Format Vulnerabilities

Potential file-format parsing vulnerabilities were explored in each product as well. A potentially serious attack vector for HMI products may be to compromise solutions provider machines by setting out “bait” on the public Internet in the form of graphical or code resource files that developers at solutions providers can use in HMI products to more efficiently create solutions for their clients. A file-format vulnerability in an HMI product can create a back door into the system once a specially crafted file is loaded up into the development or deployed interfaces in the HMI product. These vulnerabilities may be identified either by identifying publicly available libraries being used in HMI products that are known to contain vulnerabilities, or by using “fuzzing” techniques to craft files that might expose potential vulnerabilities on an automated basis.

3.4.5 Network Protocol Fuzzing

Potential remote exploitation vulnerabilities were explored by manipulating the protocols that are exposed by the HMI products. Common attacks were tested, such as directory traversal and cross-site request forgery attacks against web servers, to see if they can be used to remotely assume control or in some other way disrupt or influence the operation of the HMI. Protocol fuzzing techniques were to attempt to identify problems that were more specific to the individual products.

3.4.6 Exploit Development

For each potential vulnerability found, attempts were made at developing exploits that, at the very least, prove the concept of the vulnerability. Vulnerabilities that are known to be exploitable are typically considered by vendors and end-users to be more serious, which accelerates the development of patching and mitigation. Exploit information for vulnerabilities is also extremely useful to other vulnerability analysts and “red team” penetration testers that are tasked with testing ICS/SCADA systems. Some exploits were developed into more mature attacks that demonstrate the capabilities an attacker may have when facing an ICS/SCADA system, to illustrate the need for more secure development practices for HMI.

3.4.7 Pilot Study

A pilot study was conducted with the GE Fanuc iFix 4.5 HMI product [27]. In this study, it was observed that a feasible attack surface was identifiable and included both local and network authentication. By targeting the advertised functionality for authentication, access control, and secure storage, a suite of vulnerabilities were found that were the result of a software architecture that had evolved over time without sound security principles being given due attention. Network protocol issues were found and exploits were successfully developed for all of the identified vulnerabilities. The vulnerabilities that were identified were design and architecture issues, rather than programming errors, which may indicate that other similar products of similar functionality may have the same kinds of vulnerabilities.

3.5 Data Analysis

Each vulnerability in each product was documented, including characteristics that define the type of the vulnerability, protocols or file formats involved, and how it can be exploited. Across the set of HMI products, these vulnerabilities were compared to determine what similarities there are in the types of vulnerabilities that are found in HMI products. Where possible, the architecture and design elements of the products that were responsible for the vulnerabilities were examined, to identify places where many products make the same mistakes. The information gathered from examining and comparing vulnerability information were then be developed into a set of recommendations and best practices that can be used by HMI vendors to develop more secure HMI products in the future. Recommendations were also made on mitigation strategies that end-user stakeholders can use to protect their existing HMI product deployments.

3.6 Reporting

The recommendations and best practices were organized into taxonomies of vulnerability types or classes, with overview material that is generic to the type or class, and technical walk-throughs of each vulnerability. This technical description contains aspects of how the vulnerability was discovered and how it can be exploited (if exploitation is feasible). Mitigation strategies and alternative designs are presented that might have prevented the vulnerability, or reduce the impact of a malicious attack.

CHAPTER 4

VULNERABILITY CASE STUDIES

This chapter documents the case studies of vulnerabilities used in this research to identify common characteristics and develop a taxonomy of control systems human machine interface software vulnerabilities.

4.1 GE Fanuc iFIX

iFIX is a popular HMI package from General Electric's Intelligent Platforms division that provides both a development and runtime environment for graphically manipulating SCADA systems. The software attempts to provide security with a system for authenticating and separating privileges for a number of users[14]. The current version of iFIX, at the time of writing, is version 5.5. The vulnerabilities described here and in prior publications by the author were discovered in version 4.5 [27].

4.1.1 Password Disclosure

It is generally accepted by practitioners in the field of information security that passwords should only be stored persistently as the result of hashing the plain-text password and a per-user unique "salt" value with a cryptographic hashing algorithm. Hashing algorithms, such as SHA-1, are one-way functions that produce a unique and fixed-length output for any input of any size. The input of a cryptographically secure hashing algorithm should not be reversible from its output in a reasonable amount of time, nor should it be possible to find a “collision” (two inputs that result in the same output) in a reasonable amount of time. These properties of cryptographic hashing algorithms can be used to securely store passwords in a way that allows users to be authenticated easily, and, at the same time, prevent attackers from being able to recover plain-text passwords easily when the password storage is compromised. The result of hashing the password and salt value is stored alongside each user’s identification, and each time the user logs in, the provided password is hashed and compared with the hash stored with the user’s identification.

A common mistake made by developers implementing their own authentication schemes is to store passwords in plain-text, or using an encryption scheme that does not involve cryptographic hashing. Such schemes function as authentication normally, however in the event that the password storage is compromised (either via a compromised “admin” account, or through some other failure to protect that area of the system), these implementations fail in a much less safe fashion than a system using cryptographic hashing. If a system only stores securely hashed representations of the password, the password file or database is of little use to an attacker unless considerable resources are put towards attempting to “crack” the hashes by comparing them to the hashes an attacker generates off of commonly used passwords and derivations. A password traditionally accepted to be strong (10+ mixed-case, alphanumeric and symbol characters) hashed, along with a salt value, with a cryptographically strong algorithm will resist this kind of attack for an amount of time and computing resource that an attacker would find prohibitive.

In the case that passwords are stored in plain-text, the attacker instantly has every plain-text password for every user once he or she compromises the password storage. This is valuable for an attacker, even if authentication has been compromised in some other way, as password re-use by users may give the attacker a foothold as a user in a more secure system for which that they previously had no access. In cases where the password file, or individual passwords are encrypted, the attacker simply has to derive the key or mechanism by which the authentication system decrypts the passwords in order to check for authorization. Unlike cryptographic hashing, which has no requirement for stored plain-text or key, any form of encryption must involve stored secrets that the authentication system must use to perform authentication. If the attacker has access to this information as well, then passwords can be trivially recovered using the same process as the legitimate authentication.

This is classified as a design flaw, and can be easily corrected by utilizing standard hashing algorithms which are frequently available in libraries commonly found installed by default on modern operating systems. At some point before programming such a system, the design decision of password storage needs to be addressed to the extent of determining how passwords are to be salted, hashed, stored, and retrieved for authentication. Failing to do this can result in a system being implemented that works under normal circumstances, but provides much less security than would be desirable.

iFIX, in vulnerable versions, did not follow best practices in storing passwords. By monitoring file-system activity of the iFIX installation during the process of adding, removing, and changing the password of users to known values, it was determined that a file in the "/LOCAL/" subdirectory of the iFIX system contained a file named "XTCOM-

PAT.UTL" that was being used to store user credentials. The same file was accessed when users were authenticated and allowed to log into the iFIX system.

Under initial examination, it was obvious that "XTCOMPAT.UTL" did not contain ASCII or Unicode plain-text of the supplied passwords, indicating that the information was potentially encrypted or hashed. Upon further examination, however, it was observed that a change to a single character of the supplied password would result in only a single-byte change in the password file. This was immediately seen as a sign that the credentials were not stored securely. If a cryptographic hash was being used, each bit of the output hash would have a equal and random chance of matching the hash for a different password with even a single bit difference from the original. In other words, a single bit or byte change in the password should result in a completely different hash.

With each character position of the password being tied to a byte of the password storage, it was hypothesized that the user's password was being bit-wise exclusive-or'd with a key value before being written to password storage. While a correctly-implemented exclusive-or encryption scheme can provide perfect communications security (any possible plain-text being equally likely to correspond to a given cipher-text), it becomes trivial to break in cases where the key is shorter than the length of the plain-text (and thus repeats), where keys are re-used, or when the key cannot be stored or distributed securely. In situations where passwords need to be authenticated automatically, the key for an exclusive-or-based system must necessarily be stored somewhere, providing for less-than-ideal security.

To test this hypothesis, the location in the "XTCOMPAT.UTL" file for a user was determined by setting a variety of maximum-length passwords and observing the changed bytes. Then, a known password of maximum length (twenty “a”s in a row) was chosen for a user, and the resulting “encrypted” password in the file was observed. Finally, by taking advantage of the fact that exclusive-or is both commutative and associative, the key was derived by performing a bitwise exclusive-or of the known password and the on-disk representation. The resulting key was subsequently tested by using it to successfully decode the stored password, even after changes were made, and for other users and installations of iFIX.

Similar testing was used to extract the keys being used for other fields in the user data structures stored in “XTCOMPAT.UTL”. User records in this file can be decoded using the following keys:

- Offset: 0x04 bytes
- Length: 20 bytes
- XOR Key: 143A5B2BC39CF4B9019B40DE088B8BE8BAB4ED67

- Offset: 0x24 bytes
- Length: 20 bytes
- XOR Key: F84C300234F87780A890A22DCCD02C30621CF857

- Offset: 0x3A bytes
- Length: 6 bytes
- XOR Key: 9FE70758B8FD

When exploited with with a small program written to parse “XTCOMPAT.UTL”, this flaw allows an attacker to have immediate access to all of the user names and passwords for an iFIX installation if the contents of the “XTCOMPAT.UTL” are available to the attacker. Architectural and design problems with iFIX discussed in subsequent sections contribute to the ready disclosure of this file to attackers. An attacker who has access to an iFIX system in-person, over a compromised remote desktop, or authenticating over a local network is likely able to exploit this vulnerability. The impact of this vulnerability goes beyond access to the iFIX interface if passwords are re-used in other elements of the organization's network.

This vulnerability directly violates the principle of "open design" as described by Saltzer & Schroeder [33]. This principle states that the security of a mechanism should not depend on the ignorance of the process by attackers. As applied to a password authentication system, a system that follows this principle would not rely on a "secret" key that cannot be easily be protected.

HMI developers can avoid similar flaws in the design phase by using best-practices for hashing passwords with well-known and accepted cryptographic hashing algorithms designed for the purpose, or by taking advantage of modern operating systems built-in capabilities for identifying, authenticating, and providing separation between users. If at any time the development team finds themselves hard-coding a "secret" key in the application that is used to protect data from being disclosed, there should be a review of the design of that particular subsystem in order to determine a more accepted and secure way to implement the same functionality.

SCADA stakeholders can mitigate this vulnerability by more carefully controlling access to systems that use the iFIX HMI. A key aspect of this vulnerability is that it effectively "flattens" the different levels of access that iFIX allows for different classes of users. If an iFIX installation prevents, for example, a control panel operator from making changes to the interface, programming, or configuration of an iFIX project that an engineering role would be allowed to make, then this vulnerability can be used to escalate privileges from the lower level to the higher with relative ease. Unmonitored access should not be granted to those not cleared for operating the system at the highest level of privilege. Passwords should not be re-used for accounts on iFIX and non-iFIX systems, to limit the impact of a compromise. The iFIX system can be configured to use Windows API functionality to authenticate users via the operating system. While this configuration can be used to make password storage more secure, other vulnerabilities discovered and described in subsequent sections render this mechanism insufficient, by itself, for restricting access.

4.1.2 Intercepting Network Authentication

Communications security is important when authenticating over a network, to prevent attackers from intercepting users' credentials or replaying traffic in order to authenticate. It is generally recognized that some form of encryption is needed to ensure that, even in the case where an attacker is able to intercept and manipulate data in the authentication process, that the attacker would be unable to derive usable credentials or hijack the process in order to authenticate him or herself. Authentication should fail closed, in that modifications made to traffic should, at best for the attacker, only cause the authentication to fail for that attempt, rather than give access to anyone other than authorized users.

A classic example is the improvement in security provided by using the secure shell (SSH) protocol for shell access to remote systems over the unencrypted alternative, Telnet. An attacker with the ability to sniff network traffic between a client and server, either passively or through active means (such as ARP-spoofing), can easily intercept users authenticating via Telnet. SSH utilizes public-key cryptography in order to establish the identity of the host to the client, allowing the two to exchange a session key that can be used to securely encrypt authentication and subsequent access. A passively monitoring attacker would not be able to decrypt the traffic without private keys that are not transmitted, and would not be able to inject anything that would be interpreted as valid authentication.

iFIX, in vulnerable versions, provided functionality for authenticating over a network in cases where many computers are required to share the same set of users. There is, however a flaw in the way that iFIX performed this authentication. As a result, attackers can retrieve the entirety of the "XTCOMPAT.DLL" password storage file either during the process of attempted authentication, or by partially mimicking the process.

While exploring some of the authentication features available in iFIX, in order to find potential mitigations for the password disclosure vulnerability described in the prior section, it was observed that network authentication was available to be activated and used. This is implemented by placing the "security" files (included the "XTCOMPAT.UTL" described in the prior section) on a CIFS/SMB share, and changing the client security settings to point to the security file on the network share. This type of file sharing is built into the Windows operating system and is often used on home and small business networks for sharing files and other resources between computers. It is, by default, unencrypted. This functionality comes at the cost of very little added code to iFIX, as network shares of this type can be accessed by programs using the same Windows API calls as those used to access files on local drives.

By monitoring the network connection of a system authenticating using this mechanism, it was observed that the client iFIX process reads the entirety of the “XTCOMPAT.UTL” file over the network during authentication. If an attacker can observe this traffic, he or she can extract and recreate the password file from network traffic and use the techniques described in the previous vulnerability to decode passwords for all of the users of the system.

This problem is compounded by the fact that the entire file is transmitted, regardless of whether or not the attempt to authenticate is successful. An attacker, given knowledge of how the iFIX installation is configured, or through routine scanning common to attackers, could determine the location of the shared security files on the network shared folders and download the file independent of any iFIX process. This also results in the compromise of all users of the iFIX system.

This can be exploited by attackers on a local wired or wireless network segment through network monitoring. Remote exploitation may be possible if the network share is not suitably protected from incoming connections from external networks. This flaw results in the same level of compromise as the previous password disclosure vulnerability, though it may be easier to exploit in some configurations. It does not provide a more secure alternative to local storage of passwords, as one would hope or expect upon discovering the feature exists.

Like the prior password disclosure vulnerability, this is a flaw introduced in the design phase of development. Before development of a network authentication feature, requirements should be set forth on how it is to be implemented, and evaluated to ensure that it is being done safely. By relying on the ability of the file manipulation functions in the Windows API to read network-addressable files, essentially no extra code was needed to implement network authentication. The problem with this approach is that it also does not allow the the application developer to interject in the process any form of access control or authentication protocol.

This vulnerability violates the design principle of “fail-safe defaults” as described by Saltzer & Schroeder [33]. This principle argues that, by default, access to objects should be forbidden or restricted, and that access to only those objects needed for a specific task should be granted at any given time. This results in a conservative design where, in this case, only the information needed to authenticate one user should be accessed and transferred for a given attempt at authentication.

To mitigate this vulnerability, the HMI developers would need to either avoid this functionality entirely, or utilize a protocol, perhaps similar to Kerberos, that would allow for users to authenticate with a central authority that would then hand out the capabilities to the users to perform actions on the various systems in a network of iFIX installations. This would be considerably more complex than the current network authentication mechanism, and would likely require changes to the architecture of iFIX to support checking the access tokens at every access to resources.

The only effective mitigation for this vulnerability that SCADA stakeholders can implement is to disable this functionality, which may or may not be an option. The risk of compromise can be limited only if access to each workstation can be effectively controlled, as well as access to all of the network infrastructure between the authenticating system and the system hosting the shared password file. If a Windows domain controller system can be set up to centrally authenticate and identify users, this may serve as an alternative to iFIX's built-in network authentication, although there are vulnerabilities with this implementation as well, which will be discussed in the next section.

4.1.3 Authentication Bypass

For an authentication process to be secure, it must be implemented in a way that ensures that unauthorized users can't manipulate the process. From the moment a user provides their credentials, the process should be able to look up the user name, hash the provided password, check that password against a stored hash, and make a decision about allowing or denying access without outside interference. If this cannot be guaranteed, a malicious process may be able to change hashes to match, or short-circuit the final allow-or-deny decision.

In vulnerable version of iFIX, the entirety of the iFIX system, including the authentication process, operates in the context of the currently logged-in Windows user. While modern versions of Windows are multi-user and multi-tasking, and prevent processes from allocating and modifying the memory of other processes directly between different privileged users, there are programmatic ways that a process can access the memory space and context of another process running as the same user. This is used by debuggers and other development tools to examine and instrument running programs.

By "attaching" the Immunity Debugger to the authentication process, it was possible to trace the authentication process after a user-name and password was provided by the user. While directing the execution through the debugger, the authentication process was observed. The provided user-name and password were encrypted using the exclusive-or key described in a prior vulnerability, and compared with the contents of the "XTCOMPAT.UTL" file. Based on the results of this comparison, a branching instruction is used to determine if "allow" or "deny" code is executed.

The code that implements this process is located in a dynamic library called "secmgr.dll" that is loaded into the memory space of the authentication program "login.exe". The relevant function in "secmgr.dll" is called "sec_login". By tracing through "sec_login", the exact point at which a decision is made based on the authentication procedure was identified. That instruction took the form of "JNZ" (Jump if the Zero Flag is not set). A single bit of the opcode was changed to reverse the direction of this branching logic, turning it into a "JZ", or "JE", instruction (Jump if the Zero Flag is set). The modified code was written out to a copy of "secmgr.dll" in another location on the file-system, and a direct copy of "login.exe" was placed in the same directory. This modified copy of the login process could then be executed from this alternative location (or even an external USB drive), which allows an attacker to log into the system with the incorrect password, instead of the correct one. The same copy could be written over the original, in order to effect a denial-of-service that would allow incorrect passwords, while denying access to legitimate users. If measures were implemented to prevent all but the original process from authenticating, a program could be written to patch the relevant bytes of memory directly in the correct process.

At first glance, this authentication bypass would seem to be redundant for an attacker to utilize, when the password file itself is so easy to recover and decode. This is a more serious vulnerability, though, in that it is the result of an decision made by the developers about the architecture of the entire iFIX system, rather than something that can be fixed by dropping in a hashing algorithm as a replacement for the exclusive-or encoding that makes the password disclosure vulnerability possible. Also, while the password disclosure vulnerability can be mitigated partially by enabling the ability to authenticate via local or domain Windows accounts, that mitigation does not prevent this vulnerability, which allows an attacker to reverse the decision logic after the “LogonUserA” call is made by the authentication process.

This vulnerability can be exploited by any user who has local or remote access to the interface for logging into the iFIX system, that also has some mechanism for running additional code on that system. As will be demonstrated in the next vulnerability description, iFIX lacks the capability, even when operating in a restrictive user interface, to prevent third-party code from running if the host operating system is configured to “auto-run” code from USB or CD drives. While this functionality has been changed in recent versions of Windows, many iFIX installations operating on older versions of Windows XP will still allow for this bypass. In conjunction with other remote code execution vulnerabilities, this vulnerability would easily allow an unauthorized outside party to escalate their privileges within the iFIX system.

This flaw illustrates a serious violation of the “least privilege” principle. This principle states that each process should only operate using the least set of privileges necessary to accomplish the task at hand [33]. By operating the entirety of the iFIX system at the level of privilege required to perform the most secure actions it is allowed to undertake, it becomes impossible to prevent the end-user from modifying and manipulating processes that need to be secured, such as authentication. This architectural flaw has far-reaching implications that this specific vulnerability only begins to illustrate. There would also be nothing in this architecture to prevent third-party code from calling upon and manipulating other secure processes of the system independently, including code that directly communicates to the PLCs the HMI is monitoring and controlling.

To mitigate this vulnerability, the vendor would need to redesign the architecture of iFIX in order to isolate the components of the system that need to operate at a higher privilege from those that interact with the end-users at a lower level of privilege. Modern operating systems have the capability to prevent users and services running at different privilege levels from monitoring and manipulating each others' processes. The iFIX system could be redesigned as a set of services and client programs that only communicate to each other over well-defined and limited protocols. All accesses to all secure elements would need to be checked for authorization. For this specific vulnerability, a separate authentication or security service would be necessary to prevent malicious modification and replication of the process.

Much like the other iFIX vulnerabilities, the mitigation solution for SCADA stakeholders running the system is to control access to the systems running iFIX, and understand that the protections afforded by iFIX's documented ability to separate users of varying privilege level is ineffective. If a "flattened" user model, where all users of the iFIX system are trusted with any action allowed on the iFIX system, is acceptable, then access control of the system and network may be effective to maintain some degree of security.

4.1.4 Bypassing Run-Time Restrictions

With HMI software, it is often desirable to have a higher degree of control and limitation on local user actions than on mainstream IT systems. While most software takes advantage of multiprocessing to co-exist with other programs running in the same interactive session, allowing users to split their attention and time between multiple programs, HMI software is often meant to be the only software running with a user interface on a workstation, commanding the entirety of the user's time and attention. In this way, HMI software frequently seeks to enforce a "kiosk" operating mode, keeping the user captive in a highly controlled, full-screen interface. The name of this mode, utilized in some mainstream software as well, is in reference it its use on kiosk-style computers such as automated teller machines and vending machines.

In HMI software, this is meant to keep operator attention on the task at hand, preventing them from performing non-work related tasks on the HMI workstations. In addition, this acts as a security measure, preventing client-side attacks and malicious code from executing from external devices and network sources. This helps maintain the integrity of a SCADA system, which might be lacking the latest operating system patches due to isolation, driver or software incompatibility, or support issues.

If run-time restrictions such as these are not enforced completely, operators who find workarounds, either intentionally or by accident, may utilize them to maliciously circumvent controls and escalate privilege. Operators may also, with innocent intent, circumvent controls in order to perform non-work related computing tasks, such as web browsing, email, chat, and gaming on the HMI workstations. This activity often exposes HMI workstations to client-side security risks that they were not expected to mitigate by their design or deployment. Run-time restrictions such as these “kiosk” modes may also serve as a last-line of defense against surreptitious compromise by attackers that are able to establish physical or remote access to the HMI interface.

In iFIX, there are restrictions that can be placed on lower-privileged users that prevent them from switching the interface to other interactive programs or exiting the current full-screen HMI interface. In vulnerable versions of iFIX, key combinations such as “alt-tab” and “ctrl-alt-del” commonly used to exit and switch programs are intercepted by the iFIX software and denied to users that do not have the appropriate privilege level assigned to them in the iFIX user lists. Unfortunately, the software does not provide the ability to restrict code from automatically running from external USB drives and CD/DVD discs as they are inserted, on versions of Windows that allow for this behavior.

An attacker, or legitimate operator, may knowingly or unknowingly insert a USB drive that is configured with malicious software designed to switch user interfaces, exit the HMI program, or execute other malicious code outside of the scope of what the currently-logged-in user is allowed. This may take the form of malicious software that spreads unknowingly from drive to drive, or a malicious payload that targets the HMI directly to attack the other vulnerabilities listed here for iFIX. The attacker may not need to be physically present, if a network connection is present and suitable for exfiltrating the desired sensitive data or command and control. An example of such an attack would be to bribe or blackmail an employee with physical access (such as janitorial staff) to simply insert and eject a crafted USB device or CD. Another example would be to take advantage of an employee that utilizes open USB ports for charging complex devices such as portable media players and smart-phones that can be compromised. This kind of attack can even span across so-called "air gapped" networks that are otherwise isolated from the public Internet.

This is a violation of the Saltzer & Schroeder principle of "complete mediation", which states that every access to a resource must be checked to ensure the user holds the appropriate permission to access that resource[33]. By capturing some mechanisms of subverting the captive interface, and letting others occur, all accesses are not being mediated.

This is a programming flaw by the developers of iFIX, in that the design already exhibits an understanding that users may need to operate within a captive interface. This vulnerability is simply a matter of not taking into account every possible way in which malicious code might interactively execute. As with many vulnerabilities, the difficulty is inherent in the fact that while developers must think of every possible way of bypassing such controls, an attacker only has to find one mechanism that hasn't been blocked.

To mitigate this, developers of iFIX and HMI software seeking to offer a similar captive interface must add in the capability to either disable or otherwise hook the functionality of automatically running code from external devices and media. This is in addition to the current mechanisms for intercepting undesirable keystroke combinations. This specific vector, automatically running code from external media, can be blocked through registry modification.

SCADA stakeholders can mitigate this specific vulnerability by following Microsoft's instructions for disabling auto-run functionality in the registry. Physical access to interface ports, such as USB, should be restricted by either disabling the ports electronically, physically disconnecting, removing, or damaging the ports (a common solution is to fill the ports with epoxy), or by isolating the main "tower" of the computer away from the video, keyboard, and mouse interface required by the end-user.

4.2 Iconics Genesis 32

The following are vulnerabilities discovered as a part of this research in Genesis 32, an HMI package from Iconics. These vulnerabilities exist in version 9.22 of Genesis 32, and all prior versions.

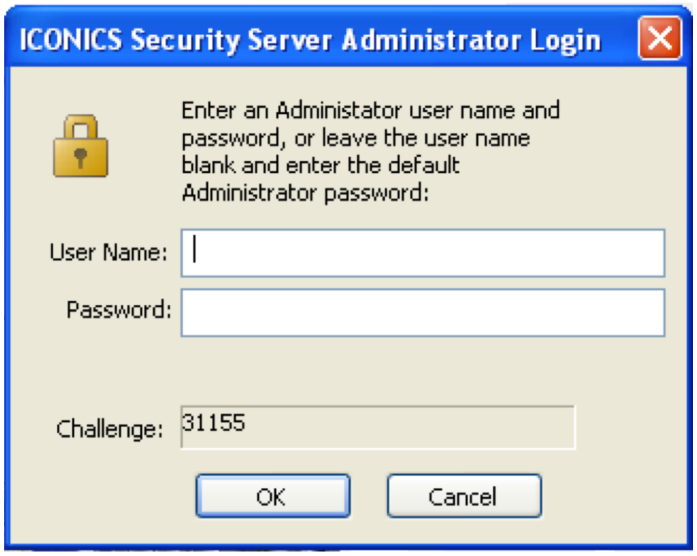
4.2.1 Authentication Bypass/Backdoor

For an authentication system to be secure, there must not be any "hidden" backdoors or similar functionality that would allow a user to bypass the authentication measure. It is, however, attractive for vendors and third-party solutions providers to insert mechanisms that make it convenient to remotely administer systems installed at remote client locations. End-users, not considering the security implications, may also see value in maintaining backdoor access for use in emergency situations and to maintain access when disgruntled employees leave. The latter fear may be well-founded, as in 2010, a former systems administrator for the city of San Francisco was sentenced to four years in prison for holding the login credentials of the city's computer networks "hostage" after receiving a poor performance review[46].

In mainstream IT software, the widespread adoption and connectivity of software by many users across the Internet has discouraged vendors from inserting backdoors (for fear of attackers gaining access). There have been recent examples. In January of 2013, SEC Consult Vulnerability Lab disclosed a vulnerability across Barracuda Networks' firewall and virtual private network (VPN) product line that involves an SSH service allowing remote administrative access to a range of "white-listed" public and private IP addresses. This backdoor access was intentional, in order to allow Barracuda to remotely support customers' devices, though it could result in compromises by attackers[35].

In Iconics Genesis 32, there is a program that allows administrative users the ability to add and remove users to the system, manage groups of users, and change the permissions and security levels of users. This interface can be accessed by any user that has an "Security System Administrator" flag set on their account with that user's user name and password. In vulnerable versions of Genesis 32, the security login form not only contains the usual fields for user name and password, but also provides a third, un-editable field marked "Challenge", shown in Figure 4.1.

According to the Genesis 32 documentation included with the program, the purpose of this feature is to provide SCADA stakeholders the ability to log into the system in "emergencies" by providing the code next to "Challenge" to Iconics' technical support hot-line. Presumably after Iconics verifies that the provider is a customer, an emergency password is provided to the caller that can be placed in the "password" field, while leav-



ICONICS Security Server Administrator Login

Enter an Administator user name and password, or leave the user name blank and enter the default Administrator password:

User Name:

Password:

Challenge: 31155

Figure 4.1

Iconics Genesis 32 security login, with “Challenge” feature.

ing the “username” field blank. If the correct emergency password is provided, then the security administration program will log the user in as the default security administrator user.

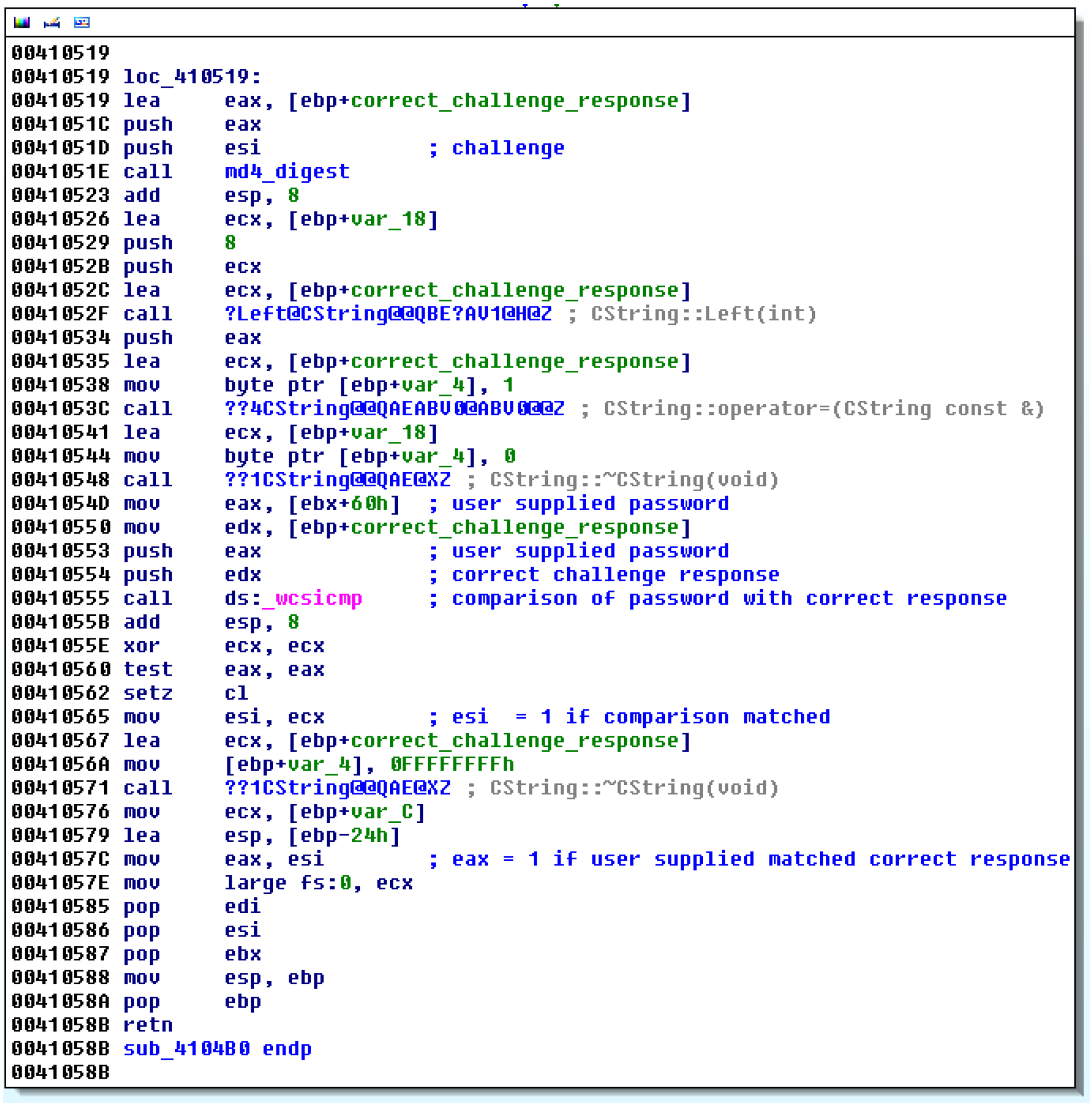
The challenge value is not randomly generated. It cycles through the positive values of a signed 16 bit integer at an approximate rate of 10 times per second, apparently “locks” into a value (or the value is determined as a function of current time) when the prompt is displayed. In other words, throughout the day, the challenge could be any integer from 0 through 32,767, repeating at different times of day. This portion of the program does not make a network connection back to Iconics, therefore any response that their technical support line provides must be checked locally within the security administration program.

It was also observed that this “challenge” value is not unique to each installation or Iconics customer. This, combined with the local nature of the response verification, leads to an immediate vulnerability that requires no reverse engineering or technical background. There is nothing to prevent one Iconics customer (or someone who can convincingly pose as as a customer on the phone) from acquiring a response code for another customer’s system for which they are not authorized. Also, reminiscent of the saying “even a broken clock is right twice a day”, anyone who has called in and received a response code has, given good timing, the ability to log into any Genesis 32 security configuration system at a few specific moments in the day.

Correct timing is not needed by attackers, however, as there is also a vulnerability in how this response is calculated. With all of the response calculations performed locally, it is possible to locate the code responsible for this calculation, analyze it, and create a tool that generates the correct response for a given challenge. By tracing the code through the password verification process with Immunity Debugger, and simultaneously examining and commenting the static disassembly in IDA Pro, the code that verified the user-supplied password against the correct response was located and analyzed.

In Figure 4.2 the relevant disassembly is presented. It was observed that the function call at 0x0041051E accepted the challenge value as an ASCII representation of the integer value, and placed what appeared to be a hash digest value into memory. To test this theory, a number of hashing algorithms were separately run, providing each the same challenge string until a matching algorithm, Message Digest 4 (MD4) was found. The subsequent string manipulation truncates the hash digest to the first 8 characters, and places the shorter digest into the local variable labeled in the disassembly as “correct_challenge_response” (note that this string was added for clarity by hand in the disassembly process and does not exist in the disk or memory image of the program). A pointer to this response and the value provided by the user in the password field is passed to a string comparison function at 0x00410555. If the strings match, then a value of 1 propagates through a set of registers, arriving back at EAX at the end of the function, causing the function to return the integer 1. If the strings do not match, the function returns a zero. The calling function uses this value to determine whether the user is authorized.

With the knowledge that the correct response to the challenge is simply the first 8 characters of the MD4 hash digest of the challenge, it is possible to implement this calculation in a way that does not involve running code on the target system. A smart-phone can be used with an appropriate calculator either as a native application or any of a number of common web applications that perform hash calculations. An attacker could even arrive at the system with a hard-copy printout of a subset of pre-calculated responses for a window of time that he or she has predetermined for the attack. Source code for a proof-of-concept python script that demonstrates how to calculate this value is included in Appendix A.1. This vulnerability can be exploited locally, or through a remote desktop connection to the vulnerable version of Genesis 32. As a result of this vulnerability, lower privileged users as well as attackers without any user account can add users, remove legitimate users, and escalate the privilege of existing users. No audit trail that would document this activity could be found, however if one exists, it could easily be modified by the same attacker or user.



```
00410519
00410519 loc_410519:
00410519 lea     eax, [ebp+correct_challenge_response]
0041051C push    eax
0041051D push    esi             ; challenge
0041051E call    md4_digest
00410523 add     esp, 8
00410526 lea     ecx, [ebp+var_18]
00410529 push    8
0041052B push    ecx
0041052C lea     ecx, [ebp+correct_challenge_response]
0041052F call    ?Left@CString@@QBE?AV1@H@Z ; CString::Left(int)
00410534 push    eax
00410535 lea     ecx, [ebp+correct_challenge_response]
00410538 mov     byte ptr [ebp+var_4], 1
0041053C call    ??4CString@@QAEABV0@ABV0@@Z ; CString::operator=(CString const &)
00410541 lea     ecx, [ebp+var_18]
00410544 mov     byte ptr [ebp+var_4], 0
00410548 call    ??1CString@@QAE@XZ ; CString::~CString(void)
0041054D mov     eax, [ebx+60h]  ; user supplied password
00410550 mov     edx, [ebp+correct_challenge_response]
00410553 push    eax             ; user supplied password
00410554 push    edx             ; correct challenge response
00410555 call    ds:_wcsicmp     ; comparison of password with correct response
0041055B add     esp, 8
0041055E xor     ecx, ecx
00410560 test    eax, eax
00410562 setz    cl
00410565 mov     esi, ecx        ; esi  = 1 if comparison matched
00410567 lea     ecx, [ebp+correct_challenge_response]
0041056A mov     [ebp+var_4], 0FFFFFFFFh
00410571 call    ??1CString@@QAE@XZ ; CString::~CString(void)
00410576 mov     ecx, [ebp+var_C]
00410579 lea     esp, [ebp-24h]
0041057C mov     eax, esi        ; eax = 1 if user supplied matched correct response
0041057E mov     large fs:0, ecx
00410585 pop     edi
00410586 pop     esi
00410587 pop     ebx
00410588 mov     esp, ebp
0041058A pop     ebp
0041058B retn
0041058B sub_4104B0 endp
0041058B
```

Figure 4.2

Iconics Genesis 32 challenge and response verification

This vulnerability violates the principle of "open design" as described by Saltzer & Schroeder [33]. This principle states that the security of a mechanism should not depend on the ignorance of the process by attackers. In this case, the secrecy of the response-generation algorithm is relied upon to prevent attackers from using it to bypass security mechanisms. Upon close examination by an attacker (which is possible even in the demonstration or trial version of the Genesis 32 product), the security of this mechanism goes away along with the secrecy of its algorithm. Using a more obscure hashing algorithm, or one of the developers' invention, would not make this more secure, as it would still be possible to analyze and replicate (or copy) the functionality into another program.

This represents a flaw in the design of the authentication system of this software. While the requirements for the software may have included functionality to allow for emergency authentication, the choice of procedure has resulted in this vulnerability. A more secure solution would likely need to integrate public-key cryptography, using a response that is digitally signed by the vendor (or other responsible party) that cannot be forged or replayed to the authentication process. The size and complexity of this response may dictate that it is to take place over the public Internet, which would be a security risk for systems that are intentionally air-gapped from public networks. The creation of a secure means to allow emergency authentication, or at least one that creates an immutable audit trail to an individual, is likely an interesting problem for future work in this area.

The vendor, Iconics, has stated that they are removing this functionality until a more secure solution can be found, so for SCADA stakeholders, the simplest mitigation is to upgrade to the latest version of Genesis 32. For stakeholders who are unable to upgrade their software, or require an emergency authentication backdoor, the only suitable mitigation is to prevent non-trusted users from directly accessing the Genesis 32 software. Despite any measures taken to separate users based on privilege and access, this vulnerability allows any user to escalate privileges and access to the highest levels.

4.3 KEP Infilink HMI

The following vulnerability was discovered as a part of this research in Infilink, an HMI package from KEP. This vulnerability exists in version 5.00.23 of Infilink, and all prior versions.

4.3.1 Password Disclosure

The vulnerability described in this section is similar to the password disclosure vulnerability in vulnerable versions of GE Fanuc iFIX described in section 4.1.1. While it is generally accepted that, to be stored securely, passwords for users must have a “salt” value applied and processed using a cryptographic hashing algorithm, some systems instead opt for encrypting or encoding passwords in a way that is inherently reversible. While the nature of a properly hashed password is that there is no procedure available to reverse the hash to a password in a reasonable amount of time (attackers are limited to brute-force and dictionary attacks), encoding the password in a way that the authentication mechanism reverses in order to perform its duty allows malicious programs the option of performing the same calculations to decrypt or decode passwords instantly. This is a design flaw, and is described in more detail in section 4.1.1.

In vulnerable versions of KEP Infilink HMI, much like iFIX, best practices are not followed in storing passwords. While setting up and examining the security functionality of an Infilink project, as described in its documentation, it was observed that when a password was set for a specific user, the number of asterisks in the masked version of the password that is displayed in the interface remained constant, and equal to the number of characters in the password, after exiting and re-entering the security interface, and even after exiting and reloading the HMI project itself. This led to the discovery of the insecure password storage.

When passwords are salted and hashed properly, the resulting hash value is of a fixed length, regardless of the length of the original password or hash. If this hash is stored, rather than storing the password itself, then knowledge of the length of the password is “lost” along with the details of each character. For this reason, in most programs that mask passwords with asterisk characters, the values stored in memory “behind” the asterisks in the edit box only represent the actual password when it is being interactively typed in by the user. When the form or window closes and is brought back up, the presence of a password is indicated by a fixed number of asterisks in the same field, but the value beings stored is a simple placeholder with no relation to the actual password. If a password field is observed that retains the length, in asterisks, of the original password, then either the original password is being stored in a way that is reversible (because the target software itself is able to do so), or the length of the original password is being stored independently of the hash. While it may not immediately seem to be as serious of a vulnerability to disclose the length of a password, it does greatly reduce the amount of time it would take an attacker to brute-force or dictionary-attack a password hash, by reducing the possible key space.

To confirm the visual indication that passwords were not being securely stored, the files contained within an Infilink HMI project were examined before and after password changes in order to determine where user information was being stored. Through this process, it was identified that user information must be contained within a file with the name “project.hmi”. This file was examined more closely during password changes, using a hex editor to observe the specific bytes that change after passwords change. This revealed that bytes relatively close to Unicode strings representing the user name changed each time the password was modified, and that the encoded password itself was the same length as the original password (identified in the binary data as a single-byte length field, followed by an encoded password of the length field’s size in bytes). This revealed that a secure cryptographic hashing algorithm was not being used to store passwords.

Suspecting that this encoded password may be using a similar exclusive-or scheme as iFIX, a similar process as described in the iFIX analysis was used to extract the key from Infilink HMI. A known password of maximum length was set for a user in the project, then ASCII representation of that password was exclusive-or’d bitwise with the encoded password in the project file. The output of this exclusive-or was an ASCII string beginning with the character “0”, and incrementing by one for each successive character. This key was verified as being the key used for the password for all users and all project files by using it to successfully testing it as an exclusive-or key with other users’ and projects’ encoded passwords. As a proof-of-concept, a small script was developed that locates user records within an Infilink “project.hmi” file, and quickly extracts and decodes the user names and passwords. The source code for this script is presented as Appendix A.2.

Like the iFIX vulnerability, this vulnerability is a violation of the Saltzer & Schroeder design principle of “open design”[33]. By relying on a “secret” key that is contained within a mechanism that can be easily examined by an attacker, the security of the system is compromised. While it is difficult to sufficiently protect passwords when the key to a reversible encoding without generating a new key for each installation and user and protecting that key, it is much easier and more secure to use a secure cryptographic hashing algorithm to store the password in an irreversible format.

In this case, the vendor responded by issuing a patch that obscures the entirety of the project.hmi file using a more complex encoding scheme than a simple exclusive-or. Upon testing this patch, it was observed that once the project file is loaded into memory by Infilink, the user names and passwords are still stored and encoded in the same easily-reversible format as before. The code responsible for decoding the encoded project files was located in the patched Infilink software, and it was observed that no user-supplied key is necessary to load and decode an encoded project file. With all of the requisite information to decode the project file located in the project file itself, it would be possible for an attacker to develop a new script to replicate this process, though this has not yet been demonstrated. It was verified, however, that an attacker need only use the code already existing in the patched Infilink executable to load and decode the project file, after which the decoded project file could be dumped to disk and decoded using the original script presented in Appendix A.2. As of the time of writing, Infilink HMI remains vulnerable to this attack.

SCADA stakeholders can mitigate this vulnerability, much like the iFIX vulnerability, by controlling access to systems that run the Infilink HMI. In the presence of a vulnerability that compromises authentication, the HMI's ability to separate privilege and access levels is ineffective. Therefore all users with local or remote access to the HMI must be trusted to the same high level of access. Passwords used on Infilink HMI systems are likely to be compromised, and should not be re-used for accounts on other systems.

4.4 WellinTech KingView

The vulnerabilities described in this section are in KingView, an HMI package from Wellintech. These vulnerabilities exist in version 6.5.3 of KingView, and all prior versions.

4.4.1 Password Disclosure

The vulnerability described in this section is similar to the password disclosure vulnerability in Infilink HMI described in the previous section, as well as the iFIX password disclosure vulnerability described in section 4.1.1. In vulnerable versions of WellinTech KingView, the file containing user information (including user names and passwords) for a project is encoded with a very simple transform on each byte that is easily reversed. As described in previous vulnerabilities in this chapter, this is also a violation of the Saltzer & Schroeder principle of "open design"[33].

Files associated with a KingView HMI project were observed while changes were made to the passwords of users within that project. Through this observation, it was confirmed that the “users.dat” file in the project directory contained user name and password information. A simple examination of the file in a hex editor did not reveal the plain-text presence of user information. By setting and changing the characters of known maximum-length passwords, it was determined that single-byte changes in the plain-text passwords had corresponding single-byte changes in the “users.dat” file, confirming that the passwords are not securely hashed. By comparing the encoded password with the known plain-text password, it was observed that every byte in the encoded data is the equivalent of the plain-text byte bitwise inverted.

While the full format of the “user.dat” file was not completely analyzed, a tool was written to invert all of the bytes in the file and display or save the resulting contents. By doing this it was observed that the entirety of user records, including user names, descriptions, and passwords are stored using the same scheme. The tool used to perform this transformation is presented as Appendix A.3.

The vendor released a patch that uses “a new encryption algorithm which is more effective and more practical”[44]. While the new algorithm has not been completely analyzed at the time of writing, it was observed that the new algorithm is completely reversible, though not as straightforward as a simple inversion. As before, the password fields retain the length of the passwords behind asterisks, indicating that plain-text passwords are visible in memory. This was confirmed by using BulletsPassView, a software package from NirSoft that will reveal the passwords stored in memory for a text box masked by asterisks. If BulletsPassView can recover a password from the application after it has opened a user data file, it proves that the mechanism for storing the password is reversible. Even in the absence of an easy-to-use script for decoding the user file, all an attacker needs to recover the password from a user data file is the free demonstration copy of KingView, and a copy of the free BulletsPassView utility.

To mitigate this vulnerability, SCADA stakeholders must evaluate the trust they place in any users that could gain access to the user.dat file. This would be any user with remote or local access to the system, including control panel operators that may not otherwise have permission to access development, project configuration, or higher-level engineering interfaces. Passwords for KingView HMI systems should not be re-used on other elements of the control system network.

4.4.2 History Server Heap-Based Buffer Overflow

It is important for the data structures of a running program to remain in a state that is consistent with what that program, and its associated libraries, expects and is able to process correctly. Specifically, when dealing with data structures that can be of variable length, such as strings and other arrays of binary data, it is important that the appropriate amount of space is allocated in RAM to store that data, and that the boundaries of that space are strictly enforced. Lower-level programming languages, such as C and C++, typically make memory management the responsibility of the programmer, compared to higher-level languages that dynamically resize data structures at runtime when boundaries are exceeded. In lower-level languages, if a programmer omits boundary-checking, or implements it incorrectly, then there is the potential for maliciously-formed data to write past the boundaries of the original data structure. In the case of structures allocated dynamically at runtime from heap memory, the implication of such an overflow is that, at a minimum, the program could crash, and potentially give the attacker the ability to run arbitrary code. Malicious code may run as a result of overwriting important data, located in memory just after unchecked boundaries, such as function pointers, important application-specific variables (such as privilege level), or data structures that are used to manage heap memory allocations. The overwrite typically attempts to take control over the processor's instruction pointer and direct it towards code the attacker injects as part of user input.

In vulnerable versions of KingView HMI, a flaw was discovered by researcher Luigi Auriemma in the way the History Server processes incoming network traffic[49]. The purpose of the History Server is to log data about control systems processes, to act as an audit trail or for process improvement, for example. When the History Server module receives network traffic on port 777, it processes each packet by checking an opcode field. For opcode 3, space is allocated in heap memory for incoming data based off of one field in the packet that describes the number of incoming data elements. Data from the packet is then copied into memory, with the amount based off another field that describes the packet size. If the size of the packet is set to be larger than the number of elements, then data from the packet will be written past the boundaries set for the memory allocated for incoming elements.

The implication of this flaw is that an attacker can easily identify the fields in question by examining legitimate traffic to a KingView History Server, and then use that information to craft data that will exploit the vulnerable code. At the most basic level, an exploit could corrupt the heap in a way that would cause the History Server to crash. This denial-of-service attack would, at least temporarily, prevent the History Server from logging the process information that it normally acquires, allowing the attacker to perform actions in other parts of the system that would, as a result, not be recorded. A more complex exploit would have the capability to execute arbitrary code provided by the attacker at system-level privileges on the system hosting the History Server, resulting in the complete compromise of the KingView HMI and any other software or service running on the same system. A complete remote-code execution exploit has not been published for this vulnerability and one has not yet been developed as part of this research, however the reporting organization, Zero Day Initiative, included in their advisory specifically that attackers "can exploit this to remotely execute arbitrary code in the context of the service (Local System)". Their calculated Common Vulnerability Scoring System (CVSS) score also indicates that the complexity of attacking this vulnerability is "low"[49].

This vulnerability can be classified as a programming error. Examining the disassembly reveals extensive use of the Microsoft Foundation Class Library (MFC) in the KingView software. MFC acts as a C++ wrapper around the Windows API, which indicates that it is likely that the KingView software was written in C++. In lower-level programming languages such as C++, it is the programmer's responsibility to enforce boundaries on arrays and strings. Fields in incoming structures (such as packet data) that contain information on the size of variable-length data are often used as cues to how much memory should be allocated to store the incoming data, however it is easy for a programmer to mistakenly allocate too little data. In this case, the confusion revolves around the use of two different size fields that, under normal circumstances are proportionate, but that could be set differently by an attacker. The vendor responded to this vulnerability with a patch that prevents this overflow from occurring.

To mitigate this vulnerability, SCADA stakeholders may install the patch provided by the vendor. This would likely require the KingView system to be restarted (though not the entire computer), which may have to wait for scheduled maintenance times for some stakeholders. Another mitigation stakeholders may choose to implement until patching is to adjust firewall rules to disallow access to port 777 on the History Server to potentially malicious hosts. Well-designed firewall rules should deny by default and only allow connections that are necessary, so this may already in place for some stakeholders. Patches should be applied as soon as possible, however, as firewall rules may not prevent attacks from normally-allowed hosts that have been compromised.

4.4.3 KingView ActiveX Control Heap-Based Overflow

ActiveX is a technology provided by Microsoft that is primarily used by developers to provide the ability to have native X86 code execute as a "control", or part of a website being displayed in a web browser on a Windows system. A memory corruption vulnerability similar to one that may be present in traditional server and client programs can, in an ActiveX control, open up a client system for an attack that arrives via malicious, attacker-controlled web pages.

In vulnerable versions of KingView HMI, a flaw was discovered by researcher Carlos Mario Penagos Hollmann in an ActiveX control used to provide an HMI interface to client systems as a web page in a browser[50]. The vulnerability is a heap-based overflow similar to the prior vulnerability, but in a different component of the software. The primary difference is that in this vulnerability, the vulnerable code executes on client systems that connect to KingView as a server. After a client system has the KingView ActiveX control installed, it is then vulnerable to having any potentially any website load that control and supply a malicious payload. This is in contrast to the prior vulnerability, which allowed an remote attacker to attack the server itself.

The researcher has published an exploit for this vulnerability that, as a proof-of-concept, launches calc.exe on the client system[50]. The code that launches calc.exe could be replaced by malicious code of an attacker's choice. ICS-CERT, in its advisory on this vulnerability, assessed that successful exploit code could be created, but would likely experience inconsistent results[17]. This indicates that while there might not be guaranteed success in exploitation, that repeated attempts would likely eventually succeed, as is sometimes the case with unreliable exploit code.

The implication of this for SCADA stakeholders is that a remote attacker could take control of a workstation normally used by legitimate users to perform HMI tasks on a KingView system. This would allow an attacker to then escalate to operator-level privileges on the KingView system itself by surreptitiously logging and using legitimate user credentials on the client system. If the client system is used to access other elements of the control system network, this ActiveX vulnerability could also lead to compromises in those areas as well.

This vulnerability, much like the previous one, can be classified as a programming error. This is, again, the result of not adequately checking the boundaries on an array, allowing attackers the ability to overwrite data structures that are critical to the security of the running process. While this can be rectified by inserting the bounds-check and replacing the vulnerable ActiveX control with a patched version (as the vendor has done), the design decision to have native compiled code execute on the client, executable by any website is questionable. The functionality of the control could likely be implemented in web technologies such as HTML5 and JavaScript, which would reduce the likelihood of remote code execution by taking advantage of rendering engines built into web browsers that are constantly being tested in the mainstream IT software vulnerability analysis community.

SCADA stakeholders may mitigate this vulnerability by applying the patch, which, since it does not require restarting the KingView system, should incur minimal downtime. If a stakeholder determines the risk to be too great to use a technology such as ActiveX that has a history of being a target of exploitation, then the web server component of KingView could be disabled, and remote access to the HMI could be accomplished through the use of remote desktop technologies. At this point, however, concerns about the local authentication measures and password storage, as described earlier in this work, become more important. If a stakeholder can implement filtering in such a way that web browsers on the client systems can only access the KingView web server, then this vulnerability becomes much more difficult for an attacker to successfully exploit.

4.5 Wonderware SuiteLink

The following vulnerability is in Wonderware, an HMI package from Ivensys. This vulnerability exists in version 54 of the SuiteLink service, and all prior versions.

4.5.1 Denial-of-Service Vulnerability

Denial-of-service vulnerabilities have a wide range of possible root causes, but in each case, the end result is that services, normally provided for local or remote programs and users, are disrupted in a way that can be damaging to an organization. In mainstream IT software, denial-of-service attacks are frequently used by attackers to disrupt the operations of businesses that social and political activists deem to be the targets of their protests. In January of 2011, the Federal Bureau of Investigation executed search warrants on members of hacker/activist group Anonymous, for denial of service attacks on payment processing services provided by Paypal, Visa, and Mastercard [32]. More recently, the same activist group switched from using the bandwidth of members and sympathizers to distributing malware that enlists computing resources owned by individuals and organizations that are not aware that their bandwidth is being used to attack targets that are even higher profile than before, such as the U.S. Justice Department[31]. The hospital HMI compromise documented in the introduction of this dissertation was performed, in part, in an attempt to build a "botnet" to be used in denial-of-service attacks against government systems on July 4, 2009[42]. Denial of service targets are frequently chosen for their strategic importance to target organizations, so it is feasible to consider that denial of service attacks on national critical infrastructure could in the future be launched by activist organizations such as Anonymous, which has recently declared "war" on the U.S. government[9]. Nation-state attacker may also target SCADA/HMI systems with denial-of-service attacks during, or as a precursor to, conventional warfare.

While denial-of-service attacks in mainstream IT software can have a serious impact on an organization's operations, public face, and revenue stream, the effects are typically limited to the duration of the attack. After a denial-of-service attack has finished launching, has been halted by the originator, or shut down by filtering the offending traffic, the target system either immediately comes back "on line" and operational, or does so shortly after a restart of the vulnerable software or operating system. Mainstream IT software frequently deals with outages of commodity Internet services, and will often restore operation after a denial-of-service attack in a similar way to how it would respond to any other outage. Software and hardware involved with control systems networks may have much more complex procedures that need to be followed to take them on and off-line, and may require human intervention. Systems that control physical processes may also have to cope with real-world physical damage to components due to loss-of-control or other effects of denial-of-service attacks. Among vulnerability analysts involved in testing mainstream IT software, denial-of-service vulnerabilities are not valued as highly as other types of vulnerabilities. In the control systems community, however, the inherent dangers posed by denial-of-service vulnerabilities should be considered.

In vulnerable versions of WonderWare SuiteLink, an application used to assist in connecting data sources to HMI software, Core Security discovered and published a denial-of-service vulnerability that can be used by attackers to crash the SuiteLink software[10]. A packet can be crafted by a malicious program to connect to a SuiteLink installation remotely over TCP port 5413. The malicious packet contains a field that controls the amount of memory allocated for incoming data, which can maliciously be set to a very large 32-bit integer value that is impossible for the memory allocation to fulfill within a 4 gigabyte process space. The memory allocation fails in such a case, and the SuiteLink process fails to check for this error, causing an crash to occur once data is attempted to be written to an invalid address. A published Metasploit framework module is publicly available that exploits this vulnerability[3].

While the original Core Security Advisory presents the vulnerability as a denial-of-service vulnerability, it states that exploitation as remote code execution “has not been proven, but it has not been eliminated as a potential scenario”[10]. While the vulnerability does not appear to be in the analysis performed as part of this research exploitable to directly execute remote code, it could serve a purpose in making the exploitation of other potential vulnerabilities in SuiteLink easier by filling memory with known patterns, or reducing the complexity of bypassing Address Space Layout Randomization using techniques presented by “Kingcope” recently[22].

For SCADA stakeholders, the immediate implication is that remote attackers can halt, disrupt, or degrade the quality of service provided by WonderWare HMI products. This may prevent operators from taking corrective action in case of an emergency, and may result in difficulties bringing the system on-line. Without an audit trail of what caused the outage, a SCADA system may go back on-line, only to be taken down again by the same attack several times before the root cause is determined and patched.

This vulnerability is a programming error in which the developers did not consider the possibility of a system call, such as one made for a memory allocation, to fail. Under normal circumstances and operation, it is unlikely that the allocation would ever fail.

Attackers with control over the parameters to system functions, even as simple as allocation functions that only take a length argument, can provide parameters that intentionally cause these functions to fail. Without proper error handling code after function calls, the error code is treated as an address (which would have been a valid address under normal circumstances) which results in an unhandled exception when data is written to the unallocated page in memory. Commonly accepted good programming practice is to check the return value of functions, however rarely is it explained that there are sometimes security implications associated with not doing so.

The vendor responded to this vulnerability by issuing a patch that corrects the programming error, and refers customers to a document available to customers on deploying secure industrial control systems. The Core Security advisory details the time-line of communicating the vulnerability to the vendor, which illustrates some of the difficulties that researchers face when reporting control systems security vulnerabilities to vendors that are unaccustomed to receiving and reacting to researchers' vulnerability reports. Difficulties were faced by Core in finding a security contact within the organization, proving the existence of the vulnerability (at one point the vendor asked Core for the free and widely available Python interpreter needed to run the proof-of-concept code), and in convincing the vendor that the vulnerability was exploitable in common installations of the software.

Mitigations taken by SCADA stakeholders to prevent vulnerabilities of this nature include keeping up-to-date with patches for software installed in the control systems network, and filtering network traffic in order to only allow trusted systems to connect. By limiting the systems that can connect, the potential for malicious attackers to connect and manipulate data being sent to services is greatly reduced, though not eliminated.

4.6 SpecView

The following vulnerability is in the SpecView HMI package. This vulnerability exists in version 2.5 (Build 853) of SpecView, and all prior versions.

4.6.1 Web Server Directory Traversal

Directory-traversal vulnerabilities take advantage of modern operating systems capability to refer to data on file-systems by “relative” paths. Each file on a file-system has a “fully qualified” path that uniquely identifies the file among the other files on the file-system. This path begins at the “root” of the file-system, and includes each hierarchical directory that contains the file. Currently running programs have a “current working directory”, that is a path to a folder in which file operations by the program takes place by default. For a web server, the current working directory might be the directory in which the main index files are kept. Relative paths may be used by users and programs to refer to files by their location relative to the current working directory. In relative paths, there are directory names with special meaning: “.” representing the current directory, and “..” representing the parent directory. The parent directory name can be concatenated one or more times to represent higher and higher level directories in the file system (“../../../” to represent “three directories up”, for example). Most programming language and operating system API functions accept either fully qualified or relative paths as part of the same parameter.

This gives rise to a class of vulnerability known as “directory traversal”. While a developer may desire to restrict access to files on a web server to only those in the web server’s document folder meant for public consumption, if web server paths used to access pages and files are directly passed into file system API functions, there is the potential for disclosing sensitive data to attacker who maliciously insert “..” directory names into web requests. Such vulnerabilities can reveal private source code, files containing user credentials (such as those described in this work as password disclosure vulnerabilities), and allow an attacker to build a profile of other vulnerable software on the system.

In vulnerable versions of SpecView, a flaw was discovered by researcher Luigi Auriemma that exists in the optional web server component of the software. Some implementations of HMI software make use of web servers to provide a remote interface to client systems running web browsers. In SpecView, a web server can be activated that hosts a continuously updated image of the main display of the program. This web server is vulnerable to a path traversal vulnerability that allows an attacker to traverse up the directory hierarchy past the limited location that is considered secure and available. As a result, attackers can request and download arbitrary files from the system hosting SpecView. Exploitation of this vulnerability requires no special tool or script, apart from a standard web browser[26].

Directory traversal vulnerabilities are typically the result of programmer error, rather than design decisions. By not filtering user input, or not using API functions to normalize paths before using them in file operations, file operations will read from unintended sources. Once such pre-access checks are inserted, it can be assured that file operations will only take place within directories that are allowed. The vendor responded to this vulnerability by releasing a new version that successfully addressed the issue[20].

SCADA stakeholders can mitigate this vulnerability by upgrading to the latest version of SpecView. If this is not an option, or if an upgrade must be delayed, then disabling the web server will prevent exploitation. If the web server must be in operation until an upgrade can be performed, then access to the server should be restricted to IP addresses that can be trusted not to be compromised and used to exploit the vulnerability.

4.7 Advantech Studio

The following vulnerability is in Advantech Studio, an HMI package from Advantech. This vulnerability exists in version 61.6.0.0 of the ISSymbol ActiveX Control, and all prior versions.

4.7.1 ISSymbol Multiple Heap and Stack-Based Overflow

The vulnerabilities described in this section are similar to the heap-based buffer overflow described in section 4.4.3. Advantech is a suite of tools for developing and deploying SCADA HMI systems. The ISSymbol ActiveX control for Advantech is used as part of a web-browser based interface to an Advantech HMI. Four buffer overflow flaws were discovered in the ISSymbol ActiveX control by Dmitriy Pletnev of Secunia Research[11].

In vulnerable versions of Advantech, there are four different overflow flaws that have been disclosed. The ISSymbol ActiveX control maintains an "InternationalOrder" property that can have string data copied to it past its normal boundaries. Another property, "InternationalSeparator", is also vulnerable in a similar way.

Stack-based buffer overflow vulnerabilities are similar to heap-based overflows, although they are frequently easier to exploit. Data allocated as local variables to a function, which are deallocated by the end of the function, are stored on the “stack”, a common data structure in computer architecture that allows for data to be “pushed” and “popped” from the top of the structure. When a function is called in programs compiled from C, C++, and many other high-level languages, parameters to the function are pushed to the stack, along with the address of the next instruction in memory after the function call. The memory address for the function is then moved into the instruction pointer, where execution of the function begins. At the beginning of the function, the pointer that denotes the top of the stack is moved to allocate space for local variables. Often, a base pointer is maintained to keep up with the “bottom” of the stack for each function at runtime, so this is frequently pushed to the stack at the beginning of functions, then restored with a pop instruction at the end to restore the stack state for the calling function. On Intel-based processors, and many others, the stack “grows” down in memory, meaning that memory allocation on the stack is accomplished by subtracting from the stack pointer. This means that character strings and other arrays that begin at an address in memory and end at a higher address, will, relative to the stack, begin “high” on the stack, and end “lower” on the stack. In cases where program code allows a malicious user to overflow data in arrays on the stack, it is possible to write past the boundaries of that variable, over the values of other variables on the stack, and then finally over the stored base pointers and return address for the current function. If an attacker can control the value on the stack that gets placed back into the instruction pointer after the function is complete, then the attacker can use that to redirect execution to code of the attacker's choice. This type of vulnerability saw its first widely distributed examples and explanation published by "Aleph One" in the hacker magazine "Phrack"[1]. Like heap-based buffer overflow vulnerabilities, stack-based buffer overflow vulnerabilities are typically the result of not checking the boundaries of a destination data structure before copying data into it, which is a programming error.

In vulnerable versions of Advantech's ISSymbol, there are stack-based buffer overflow vulnerabilities in a windowing procedure, as well as a log file generation procedure. By passing an over-sized amount of data to the "OpenScreen" method of the control, an exploitable overflow can occur. By setting a large string to the "LogFileName" property, an exploitable overflow can also occur in the log file generation process. These vulnerabilities allow attackers to run arbitrary code on the systems that have the ISSymbol control.

ICS-CERT has reported that the flaw has been fixed by the vendor in a newer version of the Advantech Studio product[21]. Since this is an attack that takes place against client systems that normally connect to Advantech systems, mitigations (other than patching) for SCADA stakeholders should either involve discontinuing use of remote access or controlling the web hosts that client systems are allowed to contact.

4.8 Sielco Sistemi Winlog

Vulnerabilities of four different types were analyzed, based on an advisory discussing 7 flaws discovered in the Sielco Sistemi Winlog SCADA HMI software by researcher Luigi Auriemma[25]. Two of the vulnerability types, stack-based buffer overflow and directory traversal, have been discussed in relation to flaws in other products earlier in this chapter, while the other two types, arbitrary writes and function pointer control, are introduced in this section. In vulnerable versions of Winlog, a TCP service can be activated to provide remote services to clients. It is this service that these vulnerabilities exist.

It is interesting to note, that for these vulnerabilities, an attempt was made by the vendor to repair the vulnerabilities, but the code inserted to check for parameters that indicate overflow conditions was not adequate initially. A version of the software was released that attempted to detect an overly-large amount of data being copied into the target buffer, however the check was performed using a signed integer, meaning that negative values that evaluated in an unsigned context as being very large, continued to allow for exploitation.

The vendor, Sielco Sistemi, after working with ICS-CERT and researchers including Auriemma that had reported the vulnerabilities independently, released an upgrade that repairs the programming flaws associated with these vulnerabilities. One mitigation option for SCADA stakeholders other than upgrading is to disable the TCP server functionality in the software. Unfortunately, it's unlikely that this was enabled without there being an operational need. While preventing connections to the TCP server from unknown hosts may prevent direct exploitation, it does not prevent attacks launched from other compromised machines on the control system network.

4.8.1 Function Pointer Control

While in Luigi Auriemma's advisory this type of vulnerability is classified as "code execution"[25], in this work, the term "function pointer control" is used. The purpose of this terminology change is to separate this type of vulnerability from other types (such as stack or heap-based buffer overflows) that are functionally different, yet also result in the execution of malicious code. Any code that calls functions by an address relative to a value either directly taken from or derived from user input runs the risk of having malicious code hijack the call to redirect execution in unexpected ways.

In vulnerable versions of Winlog, three vulnerabilities of this nature were discovered. The functions in which the vulnerable code exists are the "DblGetRecordCount", "@Db@TdataSet@Close@qqrv", "DblSetToRecordNo"functions. In each of these vulnerabilities, code exists that loads a pointer from user-controlled input data into a register, followed by a call instruction that sends the execution path to an address defined within the user-controlled data. The impact of this is that remote attackers can provide malicious code to execute, then easily direct the path of execution to that supplied code. Much like buffer overflow vulnerabilities, this type of vulnerability is the result of programmer error, failing to properly sanitize input data in such a way as to prevent users from being able to provide data that gets loaded into the target of a function call.

4.8.2 Arbitrary Write

In this section, as in the last, this work has taken the liberty of referring to the "write1" and "write4" with another term, "Arbitrary Write" that states the nature of the vulnerability in a more verbose form[25]. In the advisory's terminology, the numbers "1" and "4" are in reference to the number of consecutive bytes an attacker would have control over upon successful exploitation. This information is useful for vulnerability analysts and these are generally-accepted terms in that community, though for the purposes of this work it's useful to bring these terms under one broader category.

An arbitrary write vulnerability is a flaw in which an attacker is given the capability of potentially writing a sequence of bytes of the attacker's choice to the attacker's choice of addresses in memory. In general, this is the result of code being written that, in some way, allows an attacker to simultaneously control both a pointer to a location where data is copied, and the data to be copied to that location. This may also be the end-result of attempting to exploit a difficult overflow vulnerability as well. Often, in some way the value being written to memory is limited in some way (disallowed null bytes, for example), and/or the range of memory that can be written to is sometimes limited as well. The end-goal of an arbitrary write vulnerability is typically to overwrite a function pointer or important application-specific data structures (such as privilege levels for users).

One of the Winlog vulnerabilities is a four-byte arbitrary write involving a user supplied index used to access a file pointer, and is described as being "very theoretical" to exploit[25]. The more immediately exploitable vulnerability allows for a single null byte to be written outside of buffer in question. While a single null byte arbitrary write is difficult for an attacker to leverage into remote code execution, it could potentially be used to change data structures that are used in access control, and easily used to cause a crash in a denial-of-service attack.

4.8.3 Stack-Based Buffer Overflow

In vulnerable versions of Winlog, a stack-based buffer overflow exploit exists where a call is made to the standard C “sprintf” function[25]. The purpose of “sprintf” is to craft a string from a “printf”-style formatting string and a set of arguments (that fill in marked-up elements of the format string). The crafted string is placed in memory at a pointer passed to “sprintf” as the first argument to the function. The “sprintf” function relies on the programmer to ensure that the crafted string is not larger than the amount of memory allocated for it and passed as a pointer in the first argument. User-controlled format string arguments can often be provided that will result in the destination string being completely overwritten, along with whatever data is present on the stack or in-memory at higher addresses past the destination string. Stack and heap-based overflow vulnerabilities due to insecure use of “sprintf” are so widespread, that it’s frequently used as a classroom or training example, and US-CERT has included a discussion of the problem in a set of best practices for developing secure code[34].

4.8.4 Directory Traversal

In vulnerable versions of Winlog, a directory traversal vulnerability exists that is similar in concept to the one disclosed earlier in this chapter in the SpecView HMI package[25]. This vulnerability is in a TCP server that implements an application-specific protocol, which makes it a more difficult vulnerability to find and identify than a similar path traversal vulnerability in a web server, as was seen in SpecView. The implication, once discovered, however is the same: an attacker that can easily and reliably connect to this TCP service would be able to download any file on the file-system of the system running Win-Log.

4.9 Control Microsystems ClearSCADA

The following vulnerability is in ClearSCADA, an HMI package from Control Microsystems. This vulnerability exists in all 2005, 2007, and 2009 versions of the ClearSCADA software.

4.9.1 Authentication Bypass

A key principle of secure design described by Saltzer & Schroeder is that of "fail-safe defaults". This principle states, in part, that it is important that, in every failure mode of a software system, that the default access control decision revert to disallowing access, rather than granting access. In essence, if software "fails" it should do so into a "safe" state. When considering control-systems software, it may seem desirable for the software to fail into a state that allows for those on-hand during an emergency to take the steps necessary to intervene. Keeping this case in mind, however, it must be recognized that if malicious attackers realize that security is disabled or reduced in such failure states, then attacks that intentionally induce these states can be leveraged into much more serious complete compromises than simple denial-of-service. In any case, failing in a way that allows remote users to gain unauthenticated access to a system has far more dangerous implications than simply allowing local users to regain control of a system, and should be avoided when possible.

In vulnerable versions of Control Microsystems ClearSCADA HMI, a flaw was discovered by researcher Jeremy Brown that can be used to potentially bypass authentication measures in ClearSCADA by taking advantage of the lack of a fail-safe condition when the database server crashes[18]. Specifically, if an exception occurs in the “dbserver.exe” process, “Safe Mode” is activated, and remote users, including attackers, would then be able to log in without authentication. This behavior can essentially turn what would have been a temporary denial-of-service attack into a full remote compromise.

While a flaw that would allow an exception to be induced in “dbserver.exe” (which isn’t detailed in any advisory on this authentication bypass vulnerability) would be considered to be a programming flaw, the failure to prevent unauthorized logins when shut down into a safety state is a design flaw. While some applications of the software may require allowing on-hand personnel the capability to manipulate the system in such states, the option to fail-safe should be present, and arguably as the default.

The vendor, Control Microsystems, released a new version of the ClearSCADA software package that corrects this vulnerability by disallowing remote web logins when authentication is not possible due to a database server issue. The vendor also recommended other mitigations that SCADA stakeholders could employ to mitigate this and similar vulnerabilities, including limiting network access to the HMI web server, and implementing more secure communication for web communications, including installing security certificates to aid in validating the security of connections. It is interesting to note that in the ICS-CERT advisory dated August 2011, Control Microsystems states that they recommend an upgrade to ClearSCADA 2010 or newer, stating that versions from 2009 and earlier would not be patched. Considering that many SCADA systems and networks do not see frequent updates, and may run for many years on the same software with very little change (indeed, changes are often difficult to implement for technical and process safety reasons), a support window for software of roughly two years may leave many networks running ClearSCADA vulnerable to this issue despite the patch.

CHAPTER 5

VULNERABILITY TAXONOMY FOR SCADA HMI SOFTWARE

Based on the analysis of the vulnerabilities presented in the previous chapter, a taxonomy has been developed that utilizes common and significant characteristics of the vulnerabilities to place these vulnerabilities into descriptive categories. In this chapter, the characteristics and categories are discussed, as well as a presentation of the taxonomy itself. Finally, the implications and application of this taxonomy are discussed, including its impact on vulnerability discovery, stakeholder mitigation, education, and decisions that relate to mitigation strategies.

5.1 Defining Characteristics

This section describes some of the defining characteristics used in this work to categorize vulnerabilities.

5.1.1 Design Flaw vs. Programming Error

The clearest and most well-defined characteristic to arise from research into HMI vulnerabilities is the distinction between design flaws and errors introduced by mistakes made in programming. This usefulness of this distinction is uniquely high for SCADA HMI software as well, as it has been observed in the original vulnerability analysis conducted in this research that design flaws are relatively easy to find in established HMI products that are in wide use by stakeholders, compared to the difficulty of finding such vulnerabilities in software that is in wide use among mainstream IT software consumers such as businesses and individuals' personal computers on the public Internet.

The vulnerabilities discussed in this work that reflect design flaws are as follows:

- GE Fanuc iFIX - Password Disclosure
- GE Fanuc iFIX - Intercepting Network Authentication
- GE Fanuc iFIX - Authentication Bypass
- GE Fanuc iFIX - Bypassing Run-Time Restrictions
- Iconics Genesis 32 - Authentication Bypass/Backdoor
- KEP Infilink HMI - Password Disclosure
- WellinTech KingView - Password Disclosure
- Control Microsystems ClearSCADA - Authentication Bypass

The vulnerabilities discussed in this work that reflect programming error are as follows:

- WellinTech KingView - History Server Heap-Based Buffer Overflow
- WellinTech KingView - KingView ActiveX Control Heap-Based Buffer Overflow
- Wonderware SuiteLink - Denial-of-Service Vulnerability
- SpecView - Web Server Directory Traversal
- Advantech Studio - ISSymbol Multiple Heap and Stack-Based Buffer Overflow (4 vulnerabilities)
- Sielco Sistemi Winlog - Function Pointer Control (3 vulnerabilities)
- Sielco Sistemi Winlog - Arbitrary Write (2 vulnerabilities)
- Sielco Sistemi Winlog - Stack-Based Buffer Overflow
- Sielco Sistemi Winlog - Directory Traversal

5.1.2 Internal vs External Attack Vector

The language most frequently used in vulnerability advisories when describing the network and host locations from which a vulnerability can be exploited uses the terms “local” and “remote”. In this context, “local” vulnerabilities are those that require a presence on the system, and are typically used by authenticated users to escalate privileges past what they are allowed to have, and “remote” refers to vulnerabilities that can be exploited over a network connection to the target system. Vulnerabilities described as “remote” are typically further described in terms of whether or not a user must be authenticated to exploit the vulnerability in question. Pre-authentication remote vulnerabilities are obviously the most dangerous, to the extent that sometimes less attention is paid to other vulnerability types.

In the world of control systems software, specifically HMI software, with the targets being of high value and strategic importance, and a variety of software packages working together on a single network, it’s easy to imagine that attackers would be interested in leveraging vulnerabilities above and beyond “remote pre-authentication”. Once one system on a network has been compromised through such a vulnerability (or, indeed something as mundane as a client-side exploit or social engineering), an attacker then has a much broader selection of “local” exploits with which to move around in a system from a “remote” location.

It’s this nature of critical infrastructure systems that contain SCADA HMI software, that is the motivation to describe this characteristic in terms of “internal” and “external”. The meaning is nearly the same, but the implication is that this is a description of where the attack is launched, rather than the attacker's position. "Internal" can include malicious insiders and other physically "local" threats, but also includes remote attackers that have obtained a foothold in the system through some other vulnerability, and are motivated to continue their infiltration. "External", as the companion term, retains much of the original meaning and implication as the more common term, "remote".

The vulnerabilities discussed in this work that represent internal vulnerabilities include:

- GE Fanuc iFIX - Password Disclosure
- GE Fanuc iFIX - Authentication Bypass
- GE Fanuc iFIX - Bypassing Run-Time Restrictions
- Iconics Genesis 32 - Authentication Bypass/Backdoor
- KEP Infilink HMI - Password Disclosure
- WellinTech KingView - Password Disclosure

The vulnerabilities discussed in this work that reflect programming error are as follows:

- GE Fanuc iFIX - Intercepting Network Authentication
- WellinTech KingView - History Server Heap-Based Buffer Overflow
- WellinTech KingView - KingView ActiveX Control Heap-Based Buffer Overflow
- Wonderware SuiteLink - Denial-of-Service Vulnerability
- SpecView - Web Server Directory Traversal
- Advantech Studio - ISSymbol Multiple Heap and Stack-Based Buffer Overflow (4 vulnerabilities)
- Sielco Sistemi Winlog - Function Pointer Control (3 vulnerabilities)
- Sielco Sistemi Winlog - Arbitrary Write (2 vulnerabilities)
- Sielco Sistemi Winlog - Stack-Based Buffer Overflow
- Sielco Sistemi Winlog - Directory Traversal
- Control Microsystems ClearSCADA - Authentication Bypass

Note that in assigning this characteristic, the nature of the vulnerability itself and the exploitability of it externally was determined independent of any mitigating controls that a control system network could (and should) have in place to restrict access to such applications to the outside word, reducing attack surface.

5.1.3 Difficulty of Exploitation

This characteristic is a description of the difficulty of exploiting the vulnerability against a target, given currently available public information. While this characteristic can change over time (for example, a highly skilled attacker could publicly publish an exploit for a vulnerability that would be difficult for the average attacker to create "from scratch"), considering "exploitability" in this context is more useful than looking at the complexity of the vulnerability or exploit, since it is a measure of how likely it is, currently, for an attacker to make use of a certain vulnerability. It provides no service to analysts to measure the technical complexity of a vulnerability as being very high, when a publicly available exploit makes it accessible to relatively unskilled attackers.

The terminology used to describe the "difficulty of exploitation" characteristic of this taxonomy is limited to "trivial" and "non-trivial". "Trivial"-difficulty vulnerabilities are those which have working public exploits that are easily launched against target systems, or have enough publicly available information that an attacker with experience in a scripting language and a basic knowledge of exploitation techniques could quickly develop an exploit. "Non-trivial"-difficulty vulnerabilities are those for which the associated advisories have stated that exploitation is difficult or unproven, or where no exploit code has been given for more complex vulnerability types such as heap-based buffer overflow vulnerabilities.

It is worth noting that a vulnerability can change from "non-trivial" to "trivial" rapidly, given the attention of an experienced vulnerability analyst, the discovery of alternate methods of exploitation for a particular vulnerability, or a miscalculation of the original exploitability. It is simply meant as a guideline to the likelihood of seeing the vulnerability being used "in the wild", compared to seeing vulnerabilities that are known and documented with publicly available exploit code. The determination of exploitability is considered in relation to the vulnerable versions of the product, regardless of patches put in place by vendors since. Other characteristics describe vulnerabilities in terms of mitigation effectiveness.

The vulnerabilities discussed in this work that represent trivial to exploit vulnerabilities include:

- GE Fanuc iFIX - Password Disclosure
- GE Fanuc iFIX - Authentication Bypass
- GE Fanuc iFIX - Bypassing Run-Time Restrictions
- GE Fanuc iFIX - Intercepting Network Authentication
- Iconics Genesis 32 - Authentication Bypass/Backdoor
- KEP Infilink HMI - Password Disclosure
- WellinTech KingView - Password Disclosure
- Sielco Sistemi Winlog - Directory Traversal
- Sielco Sistemi Winlog - Stack-Based Buffer Overflow
- Wonderware SuiteLink - Denial-of-Service Vulnerability
- SpecView - Web Server Directory Traversal

The vulnerabilities discussed in this work that represent non-trivial to exploit vulnerabilities are as follows:

- WellinTech KingView - History Server Heap-Based Buffer Overflow
- WellinTech KingView - KingView ActiveX Control Heap-Based Buffer Overflow
- Advantech Studio - ISSymbol Multiple Heap and Stack-Based Buffer Overflow (4 vulnerabilities)
- Sielco Sistemi Winlog - Function Pointer Control (3 vulnerabilities)
- Sielco Sistemi Winlog - Arbitrary Write (2 vulnerabilities)
- Control Microsystems ClearSCADA - Authentication Bypass

5.1.4 Vendor Patch Status

While all of the vulnerabilities that are discussed in this dissertation have been addressed by the vendors, it is important to note that there is often a delay between public knowledge of a flaw and an adequate fix. In some cases, a vendor patch does not completely address the vulnerability, leaving systems open to continued exploitation. The characteristic terms used in this taxonomy to identify vendor patch status are "unpatched", for vulnerabilities that have not been addressed by the vendor; "addressed", for vulnerabilities that have been addressed by a patch or new version by the vendor, but the vulnerability remains exploitable in some way; and "patched", which indicates that a complete patch for the vulnerability has been distributed by the vendor.

All of the vulnerabilities in this work fall can be described as "patched", with the exception of the the following "addressed" vulnerabilities which have not been completely addressed:

- GE Fanuc iFIX - Authentication Bypass

- KEP Infilink HMI - Password Disclosure
- WellinTech KingView - Password Disclosure

No vulnerabilities in this work are described as “unpatched”. Even given the lack of case studies in this work that have this characteristic, it’s important to have it defined as many vulnerabilities go through an unpatched phase for a period of days, weeks, or months before a vendor patch is available. During this time, vulnerabilities that are unpatched are given careful analysis from vulnerability experts, and must be mitigated with other measures by SCADA stakeholders.

5.1.5 Non-Patch Mitigation Effectiveness

Another important characteristic of vulnerabilities in HMI software is the effectiveness of mitigations that stakeholders can put in place until patches are available and applied. Control systems networks often have requirements for verifying and testing patches and new versions of software, and limited maintenance windows of time in which changes can take place, so there is likely to be a larger amount of time between a patch being made available and it actually being applied to vulnerable systems in some control systems. In the time between the moment a stakeholder becomes aware of a vulnerability, and the availability and application of a patch, mitigation techniques may have to be applied (if available) to secure systems.

This characteristic is highly dependent on the details of each specific control system network. Individual networks’ security personnel may have to decide on how to apply this characteristic to vulnerabilities relative to their own organizations. For the purposes of this work, and as a recommendation for future work that assigns values to this characteristic outside of the context of a specific network, the term "effective" is used to describe vulnerabilities that have measures present that make exploitation more difficult for an attacker. For example, in this work, it is assumed that most web clients to HMI systems can be restricted to only viewing web pages related to the HMI system they are connected to, and such a mitigation could be described as "effective". "Ineffective" is used to describe vulnerabilities for which there are no mitigation techniques, the mitigation strategy provided is incapable of preventing exploitation, or the mitigation strategy is so restrictive that it is impractical to implement. An example of the latter would be a mitigation that requires a network service, that is presumably being used for some purpose, to be disabled.

The vulnerabilities discussed in this work that have mitigations judged, in the context of this work, to be effective include:

- GE Fanuc iFIX - Password Disclosure
- GE Fanuc iFIX - Authentication Bypass
- GE Fanuc iFIX - Bypassing Run-Time Restrictions
- WellinTech KingView - KingView ActiveX Control Heap-Based Buffer Overflow
- Iconics Genesis 32 - Authentication Bypass/Backdoor
- KEP Infilink HMI - Password Disclosure
- WellinTech KingView - Password Disclosure
- Advantech Studio - ISSymbol Multiple Heap and Stack-Based Buffer Overflow (4 vulnerabilities)

Note that vulnerabilities that can be mitigated by "wrapping" access to the vulnerable code in authenticated remote access, or by only allowing fully trusted users local access, are considered “effective” in this work. This may not be realistic in some real-world installations.

The vulnerabilities discussed in this work that either have no mitigation strategy, or have mitigations judged, in the context of this work, to be ineffective include:

- GE Fanuc iFIX - Intercepting Network Authentication
- WellinTech KingView - History Server Heap-Based Buffer Overflow
- Wonderware SuiteLink - Denial-of-Service Vulnerability
- SpecView - Web Server Directory Traversal
- Sielco Sistemi Winlog - Function Pointer Control (3 vulnerabilities)
- Sielco Sistemi Winlog - Arbitrary Write (2 vulnerabilities)
- Sielco Sistemi Winlog - Stack-Based Buffer Overflow
- Sielco Sistemi Winlog - Directory Traversal
- Control Microsystems ClearSCADA - Authentication Bypass

5.2 Categories

Through the comparison of characteristics that are common between a number of vulnerabilities, it is observed that across characteristics, categories of vulnerabilities can be defined that contain and summarize the characteristics of the vulnerabilities within them. While there will always be vulnerabilities that are exceptions to these categories, categorization of the most critical, easiest to find, or easiest to address vulnerabilities has implications for stakeholders and vulnerability analysts that will be presented in the last section of this chapter. In the next section, “Presentation”, a selection of the categories will be shown in tabular form.

In the analysis of the characterizations, and subsequent attempt to classify vulnerabilities into one (and only one) category each for the purpose of generating a taxonomy, it was observed that categories seen as useful for SCADA stakeholders have characteristics that overlap in categories that are useful for SCADA HMI vendors and vulnerability analysts. For this reason, a higher-level division is made, representing the categories in the form of two taxonomies, one primarily of interest to SCADA stakeholders seeking to protect and monitor SCADA HMI deployments, and a second taxonomy (built from the same set of characteristics) that is of interest to SCADA HMI vendors seeking to improve development practices and vulnerability analysts that are either seeking new HMI vulnerabilities or learning about vulnerability analysis in an educational environment.

5.2.1 Stakeholder Taxonomy

The following categories describe vulnerabilities in a way that is intended to be useful to assist SCADA stakeholders in prioritizing work on patching, mitigation, layered defenses, and monitoring. By categorizing vulnerabilities that require modifications (patches or upgrades) to software packages as critical, stakeholders can, as rapidly as possible, begin the process of testing and verification that may be needed to implement such modifications. For vulnerabilities that can be addressed by operational or configuration changes, independent of vendor software modification, categories define the type of monitoring stakeholders should implement in order to detect intrusion attempts.

5.2.1.1 Critical External

Vulnerabilities categorized as “Critical External” are the most dangerous to control system networks, and should be of the highest concern to those tasked with securing them. Vulnerabilities in this category have a Non-Patch Mitigation Effectiveness of “ineffective”, indicating that stakeholders must take steps to address the issue with vendor patches once they are made available. Finally, to narrow this category to vulnerabilities useful for remotely gaining a foothold in a control system network, vulnerabilities in this category have an Attack Vector characterization of “external”.

5.2.1.2 Critical Internal

Vulnerabilities categorized as “Critical Internal” should be of the high concern to those tasked with securing control system networks, as internally exploitable vulnerabilities are useful to attackers seeking to further damage and compromise control systems networks after a compromise. Vulnerabilities in this category have a Non-Patch Mitigation Effectiveness of “ineffective”, indicating that stakeholders must take steps to address the issue with vendor patches once they are made available. Finally, to narrow this category to vulnerabilities useful for inside threats and attackers who have established a foothold in a control system network, vulnerabilities in this category have an Attack Vector characterization of “internal”. This category is in place as a companion to Critical External, even in the absence of current vulnerabilities in the case studies that precisely match this criteria. Both categories must exist however, in order to categorize new vulnerabilities that match these characteristics before vendor patches and mitigation strategies are available.

5.2.1.3 Stakeholder Addressable - Logging

Vulnerabilities categorized as “Stakeholder Addressable - Logging” are those that control systems network operators or security administrators have the capability to secure their systems against with immediately available resources, and are trivial to exploit by attackers. While vulnerabilities in this category are easily exploited by attackers with relatively low skill and resources, they are also easily mitigated by stakeholder action. “Logging” in the context of this category indicates that stakeholders should keep separate logs of attempts against vulnerabilities in this category, at least until vendor patches are made available, in order to detect opportunistic attacks by attackers that hope to find a target that hasn’t implemented mitigations. These logs could help identify threats and trends in attack traffic. Vulnerabilities in this category have a Non-Patch Mitigation Effectiveness of “effective”, and a Difficulty of Exploitation characteristic of “trivial”.

5.2.1.4 Stakeholder Addressable - Alert

Vulnerabilities in the category of “Stakeholder Addressable - Alert” are those that should be monitored for exploitation attempts with alerts generated for immediate attention, even in cases where the vulnerabilities are mitigated, due to the difficulty of exploitation. Attempted exploits against difficult-to-exploit vulnerabilities are indicative of advanced threats or the new availability of public exploits (and the need to re-characterize and reclassify associated vulnerabilities). Vulnerabilities in this category have a Non-Patch Mitigation Effectiveness of “effective”, and a Difficulty of Exploitation characteristic of “non-trivial”.

5.2.2 Vendor and Analysis Taxonomy

Though the immediate benefit of this work is to improve the security of critical infrastructure by giving stakeholders the tools needed to categorize and prioritize the way they address vulnerabilities, assisting HMI software vendors and vulnerability analysts also can have an impact on the security of critical infrastructure. Educating vendors and analysts on the design principles and programming practices of secure software development, in the context of existing vulnerabilities that meet certain characteristics, can help identify other similar vulnerabilities that are currently undiscovered, and prevent the same problems from arising in future software. For vulnerability analysts, the distinction between "internal" and "external" has an impact on how the vulnerabilities are exploited in proof-of-concept code. For vendors, the same distinction can serve as a means for prioritizing efforts in patching and secure development.

5.2.2.1 Design Flaw Internal

Vulnerabilities in this category have the characteristic of being the result of "design flaws", and an attack vector of "Internal". Privilege escalation vulnerabilities that take advantage of secure design principle violations (such as a lack of complete mediation) can frequently be categorized as "Design Internal". Vulnerabilities in this category are often the easiest to analyze, and could serve as educational examples in training analysts to find future critical issues in SCADA HMI software.

5.2.2.2 Design Flaw External

Vulnerabilities in this category have the characteristic of being the result of “design flaws”, and an attack vector of “external”. An example of this category of vulnerability can be seen in the network authentication of iFIX, where authentication is performed over the network in a way that allows attackers to intercept all of the users’ credentials at once. Vulnerabilities in this category could be prioritized for patching by vendors.

5.2.2.3 Programming Error Internal

Vulnerabilities in this category have the characteristic of being the result of “programming error”, and an attack vector of “internal”. Locally-exploitable programming errors, can result in privilege escalation. Internal programming errors are often easier to exploit and demonstrate, and can serve as introductory training material for both analysts and developers.

5.2.2.4 Programming Error External

Vulnerabilities in this category have the characteristic of being the result of “programming error”, and an attack vector of “external”. Vulnerabilities in this category represent very dangerous vulnerabilities, in that they will not show up in a review of the design, and are exploitable by remote attackers. For this reason, they are uniquely attractive for vendors to prevent, and for vulnerability analysts to locate.

5.3 Presentation

The following tables present the characteristics of each case study vulnerability in a selection of the taxonomy categories (that have representation in the case study vulnerabilities) described in the previous section. Following the taxonomy categories is a combined table of all of the case study vulnerabilities with their associated characteristics, so that they can be easily compared directly.

Table 5.1

Stakeholder Taxonomy - Critical External

Vulnerability	Design/Coding	Internal/External	Exploitation	Patch	Mitigation
iFIX - Auth. Intercept	Design	External	Trivial	Patched	Ineffective
SuiteLink - DoS	Coding	External	Trivial	Patched	Ineffective
SpecView - Directory Traversal	Coding	External	Trivial	Patched	Ineffective
Winlog - Stack Overflow	Coding	External	Trivial	Patched	Ineffective
Winlog - Directory Traversal	Coding	External	Trivial	Patched	Ineffective
ClearSCADA - Auth. Bypass	Design	External	Non-Trivial	Patched	Ineffective
KingView - Server Heap Overflow	Coding	External	Non-Trivial	Patched	Ineffective
Winlog - Function Ptr. Control	Coding	External	Non-Trivial	Patched	Ineffective
Winlog - Arbitrary Write	Coding	External	Non-Trivial	Patched	Ineffective

Table 5.2

Stakeholder Taxonomy - Stakeholder Addressable - Logging

Vulnerability	Design/Coding	Internal/External	Exploitation	Patch	Mitigation
iFIX - Password Disclosure	Design	Internal	Trivial	Patched	Effective
iFIX - Auth. Bypass	Design	Internal	Trivial	Addressed	Effective
iFIX - Run-Time Bypass	Design	Internal	Trivial	Patched	Effective
Genesis 32 - Auth. Bypass	Design	Internal	Trivial	Patched	Effective
Infilink - Password Disclosure	Design	Internal	Trivial	Addressed	Effective
KingView - Password Disclosure	Design	Internal	Trivial	Addressed	Effective

Table 5.3

Stakeholder Taxonomy - Stakeholder Addressable - Alert

Vulnerability	Design/Coding	Internal/External	Exploitation	Patch	Mitigation
KingView - ActiveX Heap Overflow	Coding	External	Non-Trivial	Patched	Effective
Advantech - Multiple Overflows	Coding	External	Non-Trivial	Patched	Effective

Table 5.4

Vendor and Analysis Taxonomy - Design Flaw Internal

Vulnerability	Design/Coding	Internal/External	Exploitation	Patch	Mitigation
iFIX - Password Disclosure	Design	Internal	Trivial	Patched	Effective
iFIX - Auth. Bypass	Design	Internal	Trivial	Addressed	Effective
iFIX - Run-Time Bypass	Design	Internal	Trivial	Patched	Effective
Genesis 32 - Auth. Bypass	Design	Internal	Trivial	Patched	Effective
Infilink - Password Disclosure	Design	Internal	Trivial	Addressed	Effective
KingView - Password Disclosure	Design	Internal	Trivial	Addressed	Effective

Table 5.5

Vendor and Analysis Taxonomy - Design Flaw External

Vulnerability	Design/Coding	Internal/External	Exploitation	Patch	Mitigation
iFIX - Auth. Intercept	Design	External	Trivial	Patched	Ineffective
ClearSCADA - Auth. Bypass	Design	External	Non-Trivial	Patched	Ineffective

Table 5.6

Vendor and Analysis Taxonomy - Programming Error External

Vulnerability	Design/Coding	Internal/External	Exploitation	Patch	Mitigation
KingView - Server Heap Overflow	Coding	External	Non-Trivial	Patched	Ineffective
KingView - ActiveX Heap Overflow	Coding	External	Non-Trivial	Patched	Effective
SuiteLink - DoS	Coding	External	Trivial	Patched	Ineffective
SpecView - Directory Traversal	Coding	External	Trivial	Patched	Ineffective
Advantech - Multiple Overflows	Coding	External	Non-Trivial	Patched	Effective
Winlog - Function Ptr. Control	Coding	External	Non-Trivial	Patched	Ineffective
Winlog - Arbitrary Write	Coding	External	Non-Trivial	Patched	Ineffective
Winlog - Stack Overflow	Coding	External	Trivial	Patched	Ineffective
Winlog - Directory Traversal	Coding	External	Trivial	Patched	Ineffective

Table 5.7

Combined Case Study Vulnerability Characteristics

Vulnerability	Design/Coding	Internal/External	Exploitation	Patch	Mitigation
iFIX - Password Disclosure	Design	Internal	Trivial	Patched	Effective
iFIX - Auth. Intercept	Design	External	Trivial	Patched	Ineffective
iFIX - Auth. Bypass	Design	Internal	Trivial	Addressed	Effective
iFIX - Run-Time Bypass	Design	Internal	Trivial	Patched	Effective
Genesis 32 - Auth. Bypass	Design	Internal	Trivial	Patched	Effective
Infilink - Password Disclosure	Design	Internal	Trivial	Addressed	Effective
KingView - Password Disclosure	Design	Internal	Trivial	Addressed	Effective
ClearSCADA - Auth. Bypass	Design	External	Non-Trivial	Patched	Ineffective
KingView - Server Heap Overflow	Coding	External	Non-Trivial	Patched	Ineffective
KingView - ActiveX Heap Overflow	Coding	External	Non-Trivial	Patched	Effective
SuiteLink - DoS	Coding	External	Trivial	Patched	Ineffective
SpecView - Directory Traversal	Coding	External	Trivial	Patched	Ineffective
Advantech - Multiple Overflows	Coding	External	Non-Trivial	Patched	Effective
Winlog - Function Ptr. Control	Coding	External	Non-Trivial	Patched	Ineffective
Winlog - Arbitrary Write	Coding	External	Non-Trivial	Patched	Ineffective
Winlog - Stack Overflow	Coding	External	Trivial	Patched	Ineffective
Winlog - Directory Traversal	Coding	External	Trivial	Patched	Ineffective

5.4 Implications

This section describes implications of this work's taxonomy for SCADA stakeholders, HMI developers, and vulnerability analysis.

5.4.1 Mitigation Strategy

For SCADA stakeholders, this taxonomy, with adjustment of mitigation effectiveness for the specifics of an organization, and maintenance of vendor patch status over time, could serve to help prioritize efforts in securing and monitoring HMI software. "Stakeholder Addressable" vulnerabilities should be immediately addressed in order to bring a system to a baseline of attainable security. "Critical External" vulnerabilities should cause stakeholders to evaluate temporary changes in operational procedure or connection to the outside world can be made to move Mitigation Effectiveness to "effective". If not, externally-facing systems must be closely monitored for compromise. "Critical Internal" vulnerabilities should also be the target of monitoring. Finally, "Stakeholder Addressable - Alert" vulnerabilities should have monitoring in place, locally at stakeholder organizations, and perhaps in a distributed and collaborative form as part of industry groups, in order to determine when advanced attackers or new public exploit code arises.

5.4.2 Development

For SCADA HMI developers tasked with maintaining existing HMI software, "Design Flaw External" and "Programming Error External" vulnerabilities should be prioritized for patching. For those developing new HMI software, or attempting to re-architect existing

HMI software to provide more security, “Design Flaw Internal” vulnerabilities should be used in education and training material to illustrate concepts that are fundamental to designing secure software.

5.4.3 Vulnerability Discovery Strategy

Experienced vulnerability analysts tasked with finding vulnerabilities in HMI software may find the memory corruption vulnerabilities that they expect to find in mainstream IT software, and miss the even lower-hanging-fruit of design flaws that are common to HMI software, but infrequent in modern mainstream IT software. Analysts should examine the vulnerabilities in the “Design Flaw Internal” category for examples of design flaws that may be found in other HMI software. This category of vulnerability has already been used by the author of this work in the classroom for the purpose of educating undergraduate and graduate students of information security on the basics of vulnerability analysis, and to demonstrate the impact of design flaws to SCADA stakeholders.

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

In this final chapter, conclusions are formed based on the analysis of vulnerability information and discussed in the form of answers to the original research questions posed by this dissertation. Contributions to the fields of information assurance, information operations, critical infrastructure security, and education as a result of the taxonomy and associated research are also discussed. Finally, potentially useful topics of future research are presented, based on the questions that arose during the research of this taxonomy of vulnerabilities.

6.1 Research questions

The research presented in this dissertation has suggested answers to the original research questions:

It is clear, from the vulnerabilities presented in this work, that the vulnerabilities in HMI software are common. Control systems software packages have traditionally seen less attention from vulnerability analysts than mainstream IT software, resulting in vulnerabilities that have existed in legacy code far longer than it would have, given closer scrutiny. This was observed in HMI packages in this research, specifically iFIX, containing vulnerabilities that have been in place in legacy code for decades. While analysis and exploitation of vulnerabilities in any software package takes considerable time and effort, the initial identification of design flaws that lead to vulnerabilities is relatively straightforward.

It is also clear that the numerous flaws found in HMI software are, for the most part, exploitable by a variety of attackers. Vulnerabilities that are likely to be exploitable from outside of a control systems network appear to primarily be memory corruption vulnerabilities that are in place due to programmer error. Design flaws appear to primarily be exploitable by local attackers attempting to escalate privilege, or by remote attackers that have gained a foothold into the target network via a vulnerability in another component (control system specific or software packages common with mainstream IT software).

It was observed that, across multiple HMI vendors, similar flaws with common characteristics exist that can be categorized together into a taxonomy and addressed in those categories to make mitigation, discovery, and prevention easier. SCADA stakeholders, developers, and vulnerability analysts all can benefit from this taxonomy. Through the development of the taxonomy, and the analysis of case studies that lead up to it, recommendations were identified that can be made to ensure he future development of more secure HMI products. Primarily, it was identified that the body of design principles by Saltzer & Schroeder should be integrated into secure coding education for SCADA HMI developers in order to prevent the common design flaws found in case studies presented in this work. Finally, future work has been identified that will help in promoting secure development practices for these products and assist in repairing vulnerabilities that currently exist and remain to be discovered.

6.1.1 Contributions

The primary contribution of this work is the development of a taxonomy for SCADA HMI vulnerabilities, based on characteristics observed in case studies performed in this dissertation research. The taxonomy illustrates a set of categories which assists SCADA stakeholders in focusing efforts on addressing vulnerabilities and monitoring, ultimately improving the security of national critical infrastructure. For vulnerability analysts (experienced and new), and SCADA HMI developers, a set vulnerabilities were shown to exist that could serve as excellent teaching tools to find and avoid similar vulnerabilities in the future. This will lead to less hard-to-mitigate vulnerabilities for stakeholders in the future.

In the research for this dissertation, new vulnerabilities were discovered in a number of different HMI packages. Each vulnerability was identified and analyzed, and proof of concept code developed to illustrate exploitability. Vulnerabilities were reported to the vendors via US-CERT and ICS-CERT, published, and presented at industry conferences in order to present control system vulnerability information to a wider audience, including stakeholders that run vulnerable HMI software in their own organizations. Through this work, awareness has been raised in academia and industry about vulnerabilities in this type of system, which may, over time, result in more secure code and deployments of SCADA HMI systems.

This research has also resulted in the documentation of processes that can be used to identify and exploit vulnerabilities in HMI software. This information is useful for vulnerability analysts that are tasked with identifying new vulnerabilities to fix. In this way, this research branches out to find and remedy far more vulnerabilities than were possible in the scope of this research alone. Vulnerability information can be used by penetration testing and "red team" groups to test the deployment of specific SCADA systems in order to determine if patches and mitigation methods have been applied, and that secure configurations are in use. The vulnerability data in this research has been used successfully in the classroom with undergraduate and graduate computer science students in an information security course in modules about SCADA software and introductory vulnerability analysis. The vulnerabilities found in HMI software served as excellent examples, due to the ease of exploitation and application to real-world critical infrastructure security.

HMI developers can use the information in this dissertation to begin the process of developing more secure code, in the form of patches for current HMI systems, and the future secure development of new HMI software and added features on existing packages. The technical information found in the case studies provided in this dissertation are adequate to identify areas that should be focused on in finding vulnerabilities in source code of existing HMI products that developers are currently maintaining. While more work should be done on educating HMI vendors on secure programming and design practices, the existing published work should begin to assist developers that seek it out.

6.1.2 Publications

Research proposed this dissertation has already been published by the author in the Journal of Systems and Software, Volume 82, Issue 4, under the title of Discovering Vulnerabilities in Control System Human-Machine Interface Software. This publication describes a representative case-study of vulnerabilities found in an HMI system that illustrate architectural flaws that could prove to be common across many HMI products [27]. An invited article to The CIP Report (Volume 7, Number 8) has been published on the same topic, under the title “Vulnerability Analysis of SCADA HMI Systems” [28]. The vulnerabilities discussed in these two publications were responsibly disclosed to the vendor and the US Computer Emergency Readiness Team (US-CERT), where it was assigned the vulnerability identifier VU#310355 [41]. Talks on this subject have been given at conferences, such as the Systems and Software Technology Conference and the SANS SCADA Summit, as well.

Information discovered by the author on vulnerabilities in Genesis 32, KingView HMI, and Infilink HMI, was published in the proceedings of DEFCON 20 and presenting in July of 2012. DEFCON 20 is the largest (with attendance estimated between 10,0000 and 15,000) and highest profile security conference each year. The talk was presented in the “Track One” of the conference, in the largest lecture hall, to a large audience including vulnerability analysts, penetration testers, SCADA stakeholders, and members of the media (hundreds more watched the talk later on the Internet). A more personal question-and-answer session immediately followed, with a number of SCADA stakeholders and media in attendance. A workshop on discovering and analyzing HMI vulnerabilities was also presented at the local regional security conference, BSides Jackson, in Jackson Mississippi, in November of 2012. The results of taxonomy development are currently being put into an article to be submitted by the end of April for journal publication.

6.2 Future Work

Future work will be performed to make improvements upon the taxonomy created in this work. Guidelines for categorizing vulnerabilities should be further developed, and ongoing work is needed in categorizing new SCADA HMI vulnerabilities as they are discovered. By applying the characterizations through the life-cycle of new vulnerabilities as they are discovered, the changes in category over time for vulnerabilities can be observed and documented. User studies should be conducted to determine the impact of the vulnerability on the operations of elements of national critical infrastructure.

Further investigation into finding previously undiscovered HMI vulnerabilities should take place. This investigation should continue to look for design flaws as well as programming error vulnerabilities (such as memory corruption flaws). Vulnerabilities that are the result of programmer error are frequently reported in SCADA software by researchers who have a background in discovering similar vulnerabilities in mainstream IT software. It has been demonstrated here that, alongside the same vulnerabilities found in mainstream IT software, there are design flaws that are common across multiple SCADA HMI software products, such as similar password disclosure vulnerabilities found in three different products. Further work in discovering more design flaws that violate the same principles, and other basic principles of secure software design, would further validate and extend the taxonomy presented in this work.

The vulnerability discovered in Iconics Genesis 32 was the result of a feature designed to allow access in emergency situations. More so than in mainstream IT software, SCADA and control systems software frequently automates physical processes that are either responsible for public health and safety or could endanger the safety of humans, either nearby (nuclear reactors, for example) or the beneficiaries of the service provided (water treatment, for example). In such safety-critical situations, there is a need to always have the ability to shut down the process or change parameters in the event of a catastrophic failure. This requires a user with appropriate credentials to always be on hand (along with as many backup users as necessary to ensure the appropriate level of safety), or, more realistically, a way for the users that are on-hand to intervene in the event of an emergency, regardless of their usual access level.

There is a need to develop a means for allowing access to critical software in emergency situations for those who would not normally need such access. This may involve a secure means of a remote authority (a vendor, a solutions provider, or a set of trusted users on-call) digitally signing orders to raise privileges. In cases where this is determined to be too much of a burden, there should be a mechanism for logging emergency access in a way that is impossible to modify or delete for the user that initiated the access. Such a mechanism has the potential to fully compromise the security of a system, if an outside attacker gained access to a user-level account with the capability of increasing privilege at-will, and this would likely be the biggest challenge associated with researching and developing such a mechanism.

While this document is meant to serve as a resource for SCADA HMI developers and vulnerability analysts examining SCADA HMI systems, there is a need for further education and training. Course-ware that focuses on the application of the principles of secure software development as it specifically relates to control systems security would be useful to developers of HMI software. Specialized classes that take a technical look at the details of control systems software vulnerabilities and the steps taken to identify and exploit them would be useful to security researchers and penetration testers that are tasked with testing the security of such systems, and would give them an opportunity to devote time and effort to match that of persistent attackers.

Specifically, there should be further investigation into the tendency of vendors to respond to password disclosure vulnerabilities appropriately. In all three cases where it was determined that passwords were not hashed securely, and were reversible from their encoded state, the vendors of the software were not able to repair the issue. One vendor chose to not change the mechanism, and the two vendors that issued patches simply changed the mechanism to a more complex, yet still reversible function. In both cases the new functionality seemed to act more towards increasing the obfuscation beyond a simple exclusive-or or inversion, rather than solving the essence of the problem (and following the proper security principles). It is ironic in both cases that it likely took far longer to develop a more complex encoding scheme than it would have taken to simply generate or derive a salt value, concatenate it to the given password, and pass the result through an industry-standard hashing function that the developers would not even have to design (such as MD5 or SHA1). This is certainly an issue of education, and could be focused on in a classroom environment, or a more widespread awareness program targeting control systems developers.

Due to the higher stakes involved in securing control systems, an investigation should take place into the impact of denial-of-service attacks on the safety of control systems.

While there have been dramatic examples of attacks that result in physical destruction of equipment[6], and documentation exists on the results of the Stuxnet[39] malware, a more extensive study should be conducted on the effects of denial of service attacks that exist in SCADA software to measure their impact on short and long-term operations. A similar study could also examine the impact on control systems of denial-of-service vulnerabilities determined to be low-impact in mainstream IT software, where those vulnerabilities also exist in control systems deployments (in software such as operating systems and other support or server software).

While modern operating systems that provide multi-user, multi-tasking capability, such as Microsoft Windows, offer features of memory protection and authentication that SCADA HMI software can leverage to improve security, it could also be said that the status of Windows as a general purpose operating system allows for more functionality than is absolutely necessary for HMI software to operate. As an example, with the ability to launch a web browser, comes the temptation for users to switch out of the HMI interface to browse the public Internet. An operating system with all of the capabilities needed to support HMI software, as well as the ability to assist in locking down the interface to only the HMI, with no additional functionality adding to the attack surface, would be useful in some situations.

In some situations, the unacceptability of down-time, hardware compatibility, or other circumstances might delay or prohibit the installation of patches or new software systems that repair vulnerabilities such as those pointed out in this document. In cases where requirements or cost make the deployment of newer, more secure systems difficult, a wrapper that would allow security features to be layered on top of the insecure HMI packages would be useful. Such a wrapper would encapsulate or virtualize the software, while implementing access control, monitoring, and other security measures. Challenges would include testing the security of the wrapper, and determining how to implement varying levels of access to different users.

An effort should be made to determine forensic artifacts of attacks on control systems, including HMI. Vulnerabilities described in this work, as well as future vulnerabilities could be tested in a controlled environment to determine their impact on memory and disk images of the systems targeted. Using this data, indicators of compromise and evidence items could be determined that would reveal the presence, methods, tools, and potential identifying information about attackers.

REFERENCES

[1] "Smashing The Stack For Fun And Profit," *Phrack*, vol. 7, no. 49, Nov. 1996.

[2] R. R. R. Barbosa and A. Pras, "Intrusion detection in SCADA networks," *Proceedings of the Mechanisms for autonomous management of networks and services, and 4th international conference on Autonomous infrastructure, management and security*, Berlin, Heidelberg, 2010, AIMS'10, pp. 163–166, Springer-Verlag.

[3] Belay Tows, "WonderWare SuiteLink 2.0 Remote Denial of Service Exploit," http://www.exploit-db.com/exploits/6474/.

[4] M. Bishop, S. Engle, S. Peisert, S. Whalen, and C. Gates, "We have met the enemy and he is us," *Proceedings of the 2008 workshop on New security paradigms*, New York, NY, USA, 2008, NSPW '08, pp. 1–12, ACM.

[5] L. Briesemeister, S. Cheung, U. Lindqvist, and A. Valdes, "Detection, Correlation, and Visualization of Attacks against Critical Infrastructure Systems," *Eighth Annual Conference on Privacy, Security and Trust*, Ottawa, Ontario, Canada, August 2010.

[6] Bruce Schneier, "Staged Attack Causes Generator to Self-Destruct," http://www.schneier.com/blog/archives/2007/10/staged_attack_c.html.

[7] G. Cagalaban, T. Kim, and S. Kim, "Improving SCADA control systems security with software vulnerability analysis," *Proceedings of the 12th WSEAS international conference on Automatic control, modelling &#38; simulation*, Stevens Point, Wisconsin, USA, 2010, ACMOS'10, pp. 409–414, World Scientific and Engineering Academy and Society (WSEAS).

[8] E. Chew, M. Swanson, K. M. Stine, N. Bartol, A. Brown, and W. Robinson, *SP 800-55 Rev. 1. Performance Measurement Guide for Information Security*, Tech. Rep., Gaithersburg, MD, United States, 2008.

[9] Chris Taylor, Mashable, "Anonymous Hacks US Government Site, Threatens Supreme 'Warheads'," http://mashable.com/2013/01/26/anonymous-hack-government-website-declares-war/.

[10] Core Security, "Wonderware SuiteLink Denial of Service vulnerability," http://www.coresecurity.com/content/wonderware.

[11] Dmitriy Pletnev, Secunia Research, "Advantech Studio ISSymbol ActiveX Control Buffer Overflow Vulnerabilities," http://secunia.com/secunia_research/2011-37/.

[12] C. Eagle, *The IDA Pro Book: The Unofficial Guide to the World's Most Popular Disassembler*, No Starch Press, San Francisco, CA, USA, 2008.

[13] E. B. Fernandez and M. M. Larrondo-Petrie, "Designing Secure SCADA Systems Using Security Patterns," *Proceedings of the 2010 43rd Hawaii International Conference on System Sciences*, Washington, DC, USA, 2010, HICSS '10, pp. 1–8, IEEE Computer Society.

[14] GE Intelligent Platforms, "iFIX by GE Intelligent Platforms," http://www.ge-ip.com/products/proficy-hmi-scada-ifix-5-5/p3311.

[15] GLEG Inc., "SCADA+ Pack Latest Updates," http://gleg.net/agora_scada_upd.shtml.

[16] J. A. Hansen and N. M. Hansen, "A taxonomy of vulnerabilities in implantable medical devices," *Proceedings of the second annual workshop on Security and privacy in medical and home-care systems*, New York, NY, USA, 2010, SPIMACS '10, pp. 13–20, ACM.

[17] Industrial Control Systems Cyber Emergency Response Team, "ICSA-11-074-01WELLINTECH KINGVIEW ACTIVEX CONTROL," http://ics-cert.us-cert.gov/pdf/ICSA-12-137-02.pdf.

[18] Industrial Control Systems Cyber Emergency Response Team, "ICSA-11-173-01CLEARSCADA REMOTE AUTHENTICATION BYPASS," http://ics-cert.us-cert.gov/pdf/ICSA-11-173-01.pdf.

[19] Industrial Control Systems Cyber Emergency Response Team, "ICSA-11-175-02Siemens Simatic WinCC Exploitable Crashes," http://www.us-cert.gov/control_systems/pdf/ICSA-11-175-02.pdf.

[20] Industrial Control Systems Cyber Emergency Response Team, "ICSA-12-137-02ADVANTECH STUDIO ISSYMBOL ACTIVEX BUFFER OVERFLOWS," http://www.us-cert.gov/control_systems/pdf/ICSA-11-175-02.pdf.

[21] Industrial Control Systems Cyber Emergency Response Team, "ICSA-13-011-02 SPECVIEW DIRECTORY TRAVERSAL," http://www.us-cert.gov/control_systems/pdf/ICSA-11-175-02.pdf.

[22] KingCope, "Attacking the Windows 7/8 Address Space Randomization," http://kingcope.wordpress.com/2013/01/24/attacking-the-windows-78-address-space-randomization/.

[23] U. Lamping, R. Sharpe, and E. Warnicke, *The Wireshark User's Guide*, 2008.

[24] Luigi Auriemma, "Advisories," http://aluigi.altervista.org/adv.htm.

[25] Luigi Auriemma, "Sielco Sistemi Winlog," http://aluigi.altervista.org/adv/winlog_2-adv.txt.

[26] Luigi Auriemma, "SpecView Web Server Directory Traversal," http://aluigi.altervista.org/adv/specview_1-adv.txt.

[27] R. W. McGrew and R. B. Vaughn, "Discovering vulnerabilities in control system human-machine interface software," *J. Syst. Softw.*, vol. 82, April 2009, pp. 583–589.

[28] R. W. McGrew and R. B. Vaughn, "Vulnerability Analysis of SCADA HMI Systems," *The CIP Report*, vol. 7, February 2009, pp. 6–9.

[29] Microsoft, "Windows Sysinternals," Microsoft Technet, July 2007, Microsoft Corporation, http://www.microsoft.com/technet/sysinternals/default.mspx.

[30] B. P. Miller, L. Fredriksen, and B. So, "An empirical study of the reliability of UNIX utilities," *Commun. ACM*, vol. 33, December 1990, pp. 32–44.

[31] Quinn Norton, Wired, "Anonymous Tricks Bystanders Into Attacking Justice Department," http://www.wired.com/threatlevel/2012/01/anons-rickroll-botnet/.

[32] Ryan Singel, Wired, "FBI Knocks Down 40 Doors in Probe of Pro-WikiLeaks Attackers," http://www.wired.com/threatlevel/2011/01/fbi-anonymous/.

[33] J. H. Saltzer and M. D. Schroeder, "The Protection of Information in Computer Systems," *Proceedings of the IEEE 63-9*, 1975.

[34] Sean Barnum, Cigital, Inc., "US-CERT, Build Security In, SPRINTF," https://buildsecurityin.us-cert.gov/bsi/articles/knowledge/coding/840-BSI.html.

[35] SEC Consult, "SEC Consult Vulnerability Lab Security Advisory ¡ 20130124-0 ¿," https://www.sec-consult.com/fxdata/seccons/prod/temedia/advisories_txt/20130124-0_Barracuda_Appliances_Backdoor_wo_poc_v10.txt.

[36] J. Seitz, *Gray Hat Python: Python Programming for Hackers and Reverse Engineers*, No Starch Press, San Francisco, CA, USA, 2009.

[37] A. Shosha, P. Gladyshev, S.-S. Wu, and C.-C. Liu, "Detecting cyber intrusions in SCADA networks using multi-agent collaboration," *Intelligent System Application to Power Systems (ISAP), 2011 16th International Conference on*, sept. 2011, pp. 1 –7.

[38] M. Sutton, A. Greene, and P. Amini, "Fuzzing: Brute Force Vulnerability Discovery," 2007, pp. 351–417.

[39] Symantec Security Response, "W32.Stuxnet Dossier Version 1.4," February 2011.

[40] The Clinton Administration, "Presidential Decision Directive 63 (PDD-63): Policy on Critical Infrastructure Protection," May 1998.

[41] United States Computer Emergency Readiness Team, "GE Fanuc Proficy HMI/SCADA iFIX uses insecure authentication techniques," http://www.kb.cert.org/vuls/id/310355.

[42] U.S. Attorney's Office, Northern District of Texas, "Arlington Security Guard Arrested on Federal Charges for Hacking into Hospital's Computer System," http://www.fbi.gov/dallas/press-releases/2009/dl063009.htm.

[43] R. B. Vaughn, R. Henning, and A. Siraj, "Information Assurance Measures and Metrics: State of Practice and Proposed Taxonomy," *Hawaii International Conference on System Sciences*, vol. 9, 2003, p. 331c.

[44] Wellintech, "september.10,2012-Patch for KingView6.53," http://www.wellintech.com/index.php/news/35-september102012-patch-for-kingview653.

[45] Wikimedia Commons: Lemaymd, "DNP Overview," http://en.wikipedia.org/wiki/File:DNP-overview.png.

[46] Xeni Jardin, "Former SF City sysadmin gets 4 year sentence for refusing to hand over passwords," http://boingboing.net/2010/08/07/former-sf-city-sysad.html.

[47] J. Yan and B. Randell, "A systematic classification of cheating in online games," *Proceedings of 4th ACM SIGCOMM workshop on Network and system support for games*, New York, NY, USA, 2005, NetGames '05, pp. 1–9, ACM.

[48] Z. Zainal, "The Case Study as a Research Method," *Review of Educational Research*, vol. 15, no. 5, 1945, p. 352.

[49] Zero Day Initiative, "WellinTech KingView HistoryServer.exe Opcode 3 Parsing Remote Code Execution Vulnerability," http://www.zerodayinitiative.com/advisories/ZDI-11-351/.

[50] Zero Day Initiative, "WellinTech KingView 'KVWebSvr.dll' ActiveX Control Heap Buffer Overflow Vulnerability," http://www.securityfocus.com/bid/46757/info.

APPENDIX A

SOURCE CODE

A.1 Genesis 32 Response Calculator Source

```
#!/usr/bin/python

# gen32_challenge_response.py

# Uses this library: http://www.oocities.org/rozmanov/python/md4.html

import md4

import binascii

import sys

md4_context = md4.new()

md4_context.update(sys.argv[1])

print binascii.hexlify(md4_context.digest())[:8]
```

A.2 KEP Infilink HMI Userfile Decoder Source

```
#!/usr/bin/python

# decode Infilink passwords

import sys

def decode_password(enc):

    dec = ''

i=0

for c in enc:

        dec += chr(ord(c)^(0x30+i))
        i += 1
    return dec
fp = open(sys.argv[1],'rb')
data = fp.read()
fp.close()
while True:
    index = data.find('\x20\x03\x00\x00\x01\x00')+6
    if index == 5:
        break
    username_length = ord(data[index])
    index += 5
    username = data[index:index+username_length*2]
    print 'Username: %s' % (username)
    index += username_length*2
    description_length = ord(data[index])
    if description_length == 0:
        index+=4
    else:
        index+=5
        description = data[index:index+description_length*2]
        print 'Description: %s' % (description)
    index += description_length*2
    user_level = ord(data[index])
    print 'Userlevel: %i' % (user_level)
    index += 12
    password_length = ord(data[index])
    index += 4
    password = data[index:index+password_length]
    print 'Password: %s\n' % (decode_password(password))
    index += password_length
    data = data[index:]
```

A.3 WellinTech KingView User File Decoder Source

```
#!/usr/bin/python
import sys
fp = open(sys.argv[1],'rb')
data = fp.read()
fp.close()
decoded = ''
for c in data:
    decoded += '%c'%(chr(255-ord(c)))
print decoded
```