# A framework for Cybersecurity of Supervisory Control and Data Acquisition (SCADA) Systems and Industrial Control Systems (ICS)

Alaa Al Ghazo
*Iowa State University*

# A framework for Cybersecurity of Supervisory Control and Data Acquisition (SCADA) Systems and Industrial Control Systems (ICS)

by

**Alaa Al Ghazo**

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Co-majors: Electrical Engineering; Computer Engineering

Iowa State University

Ames, Iowa

2020

## DEDICATION

I would like to dedicate this thesis to my beloved wife, Arwa Al Anber, who has been an endless source of love, support, and encouragement. To my son, Waseem, who enlighten my life. I also would like to dedicate it to my father, Taha Al Ghazo, and My mother, Suad Al Shwayyat, who have been my source of inspiration and who continuously provide their moral, spiritual, and emotional support. Finally, I would like to dedicate it to my brother and sisters for their love and support.

# TABLE OF CONTENTS

LIST OF TABLES . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . v

LIST OF FIGURES . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . vi

ACKNOWLEDGEMENTS . . . . . . . . . . . . . . . . . . . . . . . . . . . . . viii

ABSTRACT . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . ix

CHAPTER 1. INTRODUCTION . . . . . . . . . . . . . . . . . . . . . . . . . . 1
1.1 Background . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1
1.1.1 The SCADA/ICS Security Problem . . . . . . . . . . . . . . . . . 2
1.1.2 Differences between SCADA/ICS and IT . . . . . . . . . . . . . . 3
1.2 Proposed Framework . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
1.3 Literature Review . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 9
1.4 Thesis Organization . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14

CHAPTER 2. ANDVI: AUTOMATED NETWORK DEVICE AND VULNERABILITY IDENTIFICATION IN SCADA/ICS BY PASSIVE Monitoring . . . . . . . 15
2.1 Proposed Hybrid Approach to Device Recognition . . . . . . . . . . . . . 20
2.1.1 Learning Stage: Building Reference Database . . . . . . . . . . . . 22
2.1.2 Recognition Stage . . . . . . . . . . . . . . . . . . . . . . . . . . 25
2.1.3 A CPS Case Study: iTrust Water Treatment SCADA . . . . . . . . 27
2.1.4 Discussion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 29
2.2 Minimizing Data Collection Points: Role of Routers . . . . . . . . . . . 30
2.3 Modbus-data Analysis for Fingerprinting . . . . . . . . . . . . . . . . . 31
2.3.1 Modbus-data based fingerprinting . . . . . . . . . . . . . . . . . . 32
2.3.2 A CPS Case Study: S7-300 PLC SCADA . . . . . . . . . . . . . . 35
2.3.3 Discussion . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 36
2.3.4 Automated Network Device and Vulnerability Identification (ANDVI) Tool . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 37

CHAPTER 3. A2G2V: AUTOMATIC ATTACK GRAPH GENERATION AND VISUALIZATION AND ITS APPLICATIONS TO COMPUTER AND SCADA NETWORKS . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 38
3.1 Computer Network Illustrations and their Model Formulation . . . . . . . 40
3.1.1 Model Formulations of Computer Networks . . . . . . . . . . . . . 42

3.2 Model-based Attack Graph Formulation, Generation, and Illustration . . . 45
3.2.1 Automated Attack Graph Generation and Visualization . . . . . . . 46
3.2.2 Computer Network based Illustration of A2G2V . . . . . . . . . . . 50
3.2.3 Discussions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 54
3.3 A CPS Case Study: Water Treatment CPS . . . . . . . . . . . . . . . . . 55
3.3.1 Discussions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 62

CHAPTER 4. CRITICAL-ATTACKS SET IDENTIFICATION IN ATTACK-GRAPHS FOR COMPUTER AND SCADA NETWORKS . . . . . . . . . . . . . . . . . . 63
4.1 Critical-Attacks Set Computation . . . . . . . . . . . . . . . . . . . . . . . 65
4.1.1 Strongly Connected Components-induced Min Label-Cut . . . . . . 67
4.1.2 Implementing an Exact MLC . . . . . . . . . . . . . . . . . . . . . 71
4.1.3 Implementing state-of-art Approximation Algorithm [1] . . . . . . . 72
4.2 Case-studies: Computer Networks and Water Treatment SCADA CPS . . . 74
4.2.1 Computer Network Case-studies . . . . . . . . . . . . . . . . . . . 74
4.2.2 Water Treatment SCADA CPS Case-study . . . . . . . . . . . . . . 78
4.3 Results and Discussion . . . . . . . . . . . . . . . . . . . . . . . . . . . . 85

CHAPTER 5. SUMMARY AND FUTURE WORK . . . . . . . . . . . . . . . . . . 88
5.0.1 Future Work . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 91

BIBLIOGRAPHY . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 93

## LIST OF TABLES

Table 2.1 SCADA/ICS distinguishing features summary . . . . . . . . . 23
Table 3.1 3-hosts Network Attack Graph Search Summary . . . . . . . 51
Table 3.2 5-hosts Network Attack Graph Search Summary . . . . . . . 53
Table 3.3 Water Treatment SCADA CPS Attack Graph Search Summary 61

## LIST OF FIGURES

Figure 1.1 SCADA/ICS System . . . . . . . . . . . . . . . . . . . . . . . 2
Figure 1.2 Proposed Framework . . . . . . . . . . . . . . . . . . . . . . 7
Figure 2.1 Hybrid approach for devices recognition . . . . . . . . . . . 20
Figure 2.2 Standard SCADA/ICS control hierarchy . . . . . . . . . . . 21
Figure 2.3 PLCs data analysis . . . . . . . . . . . . . . . . . . . . . . . 24
Figure 2.4 Logix 1765 PLC and the Allen Bradly HMI data analysis . 24
Figure 2.5 Control hierarchy identification example . . . . . . . . . . . 26
Figure 2.6 Architecture of iTrust Water Treatment SCADA . . . . . . 28
Figure 2.7 iTrust Water Treatment SCADA control hierarchy identification output . . . . . . . . . . . . . . . . . . . . . . . . . . . . 29
Figure 2.8 Water Treatment SCADA device recognition output . . . . 29
Figure 2.9 Ring topology SCADA system architecture . . . . . . . . . . 30
Figure 2.10 Routers based SCADA system architecture . . . . . . . . . . 31
Figure 2.11 Modbus RTU over TCP . . . . . . . . . . . . . . . . . . . . 32
Figure 2.12 Modbus TCP/IP packet example . . . . . . . . . . . . . . . 34
Figure 2.13 Modbus-data analysis based devices recognition . . . . . . . 34
Figure 2.14 A simulated SCADA water treatment system . . . . . . . . 35
Figure 2.15 Automated Network Device and Vulnerability Identification (ANDVI) framework . . . . . . . . . . . . . . . . . . . . . . . . . 37
Figure 3.1 Three Hosts Network Example. . . . . . . . . . . . . . . . . 40
Figure 3.2 Five Hosts Network Example. . . . . . . . . . . . . . . . . . 42
Figure 3.3 A2G2V Algorithm Architecture. . . . . . . . . . . . . . . . . 48
Figure 3.4 A2G2V generated attack-graph for 3-hosts network . . . . 52
Figure 3.5 A counterexample from JKind output automatically encoded, and disjuncted with security property in Lustre . . . . . . . . . . . 53
Figure 3.6 A2G2V generated Attack-graph for 5-hosts network . . . . . 54
Figure 3.7 SCADA Architecture of Water Treatment System. . . . . . 56
Figure 3.8 A2G2V generated Attack-graph for Water Treatment SCADA CPS . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 61

Figure 4.1 Attack-Graph Example . . . . . . . . . . . . . . . . . . . . . . 66
Figure 4.2 Architecture of the Critical-Attacks Identification (CAI) Algorithm. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 68
Figure 4.3 3-Hosts Networked-system example. . . . . . . . . . . . . . . 75
Figure 4.4 3-Hosts Networked-system generated attack-graph. . . . . . 75
Figure 4.5 5-Hosts Networked-system example. . . . . . . . . . . . . . . 76
Figure 4.6 5-Hosts Networked-system generated attack-graph. . . . . . 77
Figure 4.7 SCADA Architecture of Water Treatment System. . . . . . 78
Figure 4.8 Generated Attack-graph for Water Treatment SCADA CPS 84

## ACKNOWLEDGEMENTS

All thanks to Almighty Allah for giving me the strength and the ability to reach this milestone in my life.

I would like to take this opportunity to express my thanks to those who helped me with various aspects of conducting research and the writing of this dissertation. First, I am extremely grateful to my major adviser Professor Ratnesh Kumar for his support, patience, and guidance throughout my research. I am also thankful to my committee members and all the professors who gave me courses at Iowa State University for their efforts and contributions to this work. Furthermore, I would like to express my gratitude to the Hashemite University for the Fellowship support as well as Iowa State University for Teaching and Research assistantships, which was based on support from NSF, during my PhD study at Iowa State University. And finally, I thank my friends for all the wonderful memories.

## ABSTRACT

The motivation behind this thesis is to provide an efficient and comprehensive solution to secure Supervisory Control and Data Acquisition (SCADA) systems and Industrial Control Systems (ICS). SCADA/ICS systems used to be on isolated networks. However, due to the increase in popularity and advancements of wireless networking and cloud technologies, SCADA/ICS systems have begun to expand their connectivity to the cloud; the extent of such connectivity can vary from system to system. Benefits of connecting to the internet/cloud are substantial, but such connectivity also makes those system vulnerable, for no longer being isolated.

Device recognition is useful first step in vulnerability identification and defense augmentation, but due to the lack of full traceability in case of legacy SCADA/ICS systems, the typical device recognition based on document inspection is not applicable. leading to the possibility of unaccounted security vulnerabilities in such systems. We propose a hybrid approach involving the mix of communication patterns and passive fingerprinting to identify unknown device types, manufacturers, and models. In addition, our ANDVI implementation maps the identified devices to their known vulnerabilities

To identify how interdependence among existing atomic vulnerabilities may be exploited by an adversary to stitch together an attack that can compromise the system, we propose a model-checking based Automated Attack-Graph Generator and Visualizer (A2G2V). The proposed A2G2V algorithm uses existing model-checking tools, an architecture description tool, and our own code to generate an attack-graph that enumerates the set of all possible sequences in which atomic-level vulnerabilities can be exploited to compromise system security.

Attack-graphs analysis enables security administrators to establish appropriate security measurements to secure their system but practical considerations on time and cost can pose limit on their ability to address all system-level vulnerabilities at once. In this thesis, we propose an approach that identifies label-cuts within an attack-graph to automatically identify a set of critical-attacks that, when blocked, renders the system secure. The identification of a minimal label-cut is in general NP-complete, and in order to deal with this computational complexity, we propose a linear complexity approximation utilizing the Strongly-Connected-Components (SCCs) to identify a cut possessing a minimum number of labels and representing a critical-attacks set. Also, we compare our proposed algorithm to an exact minimum label-cut algorithm and to an approximation algorithm, both taken from the literature and report the improvements.

The proposed approaches were tested on real-world case studies, including two IT network systems and a SCADA network for a water treatment cyber-physical system.

# CHAPTER 1. INTRODUCTION

## 1.1 Background

SCADA/ICS are industrial systems that use control devices, network protocols, and graphical user interfaces for gathering and analyzing real-time data. SCADA/ICS systems are utilized to monitor and command a plant or other industrial equipment such as nuclear plants, telecommunications, water and waste control, oil and gas refining, and energy. Today, since cloud computing and the Internet of Things (IoT) revolutions offer increasingly rapid innovation, flexible resources, and help in lowering operating costs, SCADA/ICS are transitioning to cloud computing and IoT to improve supervisory and control processes by sharing real-time information among machines, manufacturing chains, suppliers, and customers [2]. Since SCADA/ICS systems feature unique cyber and physical interaction and were originally built as isolated systems, connecting them to the internet creates potential security problems. This Thesis proposes solutions to secure SCADA/ICS systems.

Figure 1.1 shows a typical SCADA/ICS system network. Such a network can be divided into three layers: the plant layer that consists of physical components such as machines, sensors, and actuators; the SCADA layer that contains on-site supervisory systems and Programmable Logic Controllers (PLC); the enterprise layer that represents remote access devices [3].

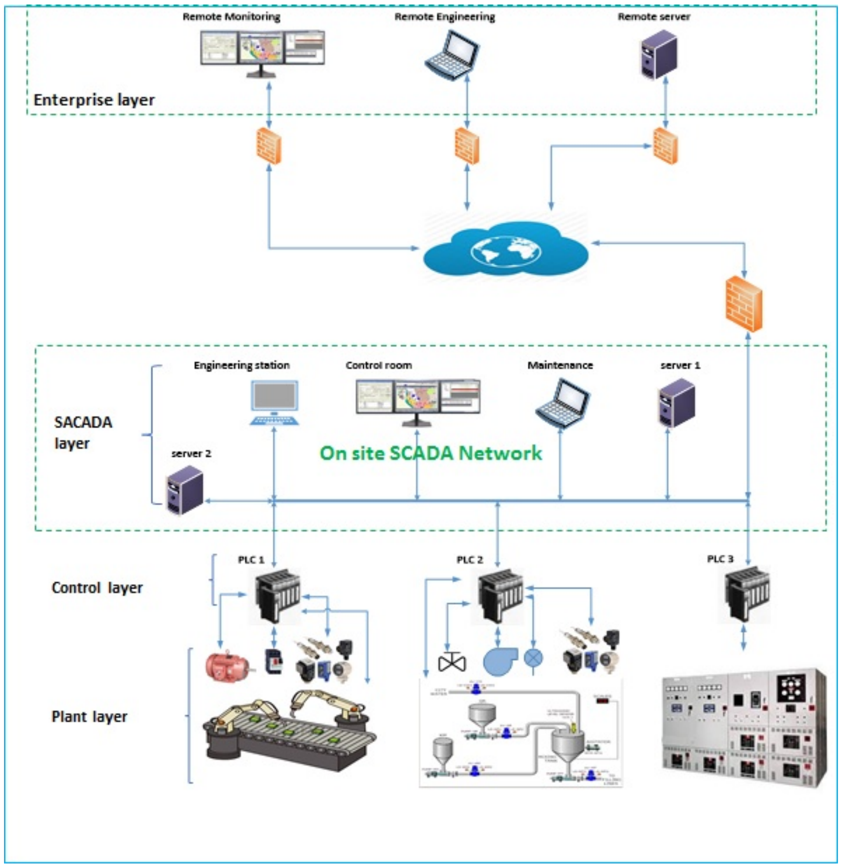


Figure 1.1: SCADA/ICS System

### 1.1.1 The SCADA/ICS Security Problem

Over the past few years, SCADA/ICS systems have been subjected to an increasing number of major cyber-attacks such as Stuxnet, Shamoon, Dragonfly, and Night Dragon [4]. Stuxnet was designed to compromise PLCs by modifying the behavior of centrifuges used in the enrichment of uranium. It would first auto-execute itself on a USB drive, then determine whether the target was running a Siemens PLC, and if so, it modified the PLC control files and played a recorded measurement in a loop to avoid detection. Shamoon, designed to erase computer hard drives, targeted energy companies in the Middle East. It first created a system file to execute itself remotely, then collected files and overwrote them either with a JPEG image or 192KB blocks of random data. Finally, it reported back to a command and control server. Dragonfly and Night Dragon were intended to target energy suppliers and industrial processes to collect sensitive information [3]. Such nefarious efforts have made it clear that SCADA/ICS systems need enhanced cybersecurity.

Enhancement of SCADA/ICS systems security can be achieved through improvement in three main areas: Access control and encryption to prevent unauthorized access to system resources, Vulnerability assessment and resolution for Identifying security vulnerabilities and mitigate them before an actual attack occurs, and Intrusion detection and reconfiguration that continually monitors system activities to detect suspicious behaviors and reorganize the system if an attack is detected. Both Access control and Vulnerability assessment must be performed during security system design of an SCADA/ICS system before an actual attack occurs, while Intrusion detection is an online monitoring activity. This thesis will focus on Vulnerability assessment and resolution.

### 1.1.2 Differences between SCADA/ICS and IT

Since security problems in SCADA/ICS systems originate with the connection to the internet and cloud computing, a reader may assume that an IT security solution is the only thing required to secure SCADA/ICS systems. The truth is, while IT security may resolve some aspects of a security problem, it usually will not be able to fully secure SCADA/ICS systems due to differences between SCADA/ICS systems and IT systems [2, 5, 6].

The security goals of IT systems are confidentiality, integrity, and availability, with confidentiality being the most important one. However, SCADA/ICS system goals could be prioritized differently in a situation where a major disaster could happen if a system component such as a controller becomes unavailable, possibly leading to serious injury or damage to the environment. SCADA/ICS security is thus more concerned about 24/7 availability of all system components [5]. Software patching and frequent updates like those used in IT are not suitable safeguards for SCADA/ICS systems. For example, upgrading an SCADA/ICS system will require a long planning time, and it is also difficult to suspend operation of an industrial computer on a regular basis to install new security patches that themselves may violate control logic [6]. For example, on March 7, 2008, A Georgia nuclear power plant was accidentally shut down because a business network computer, the computer used to monitor the plant's chemical and diagnostic data, rebooted after a software update. The data on the SCADA control system were reset when the computer rebooted, and the resulting lack of data made the safety systems believe there was a drop in level in the water reservoirs that cooled the plant's radioactive nuclear fuel rods [7].

[5] argue that different layers in SCADA/ICS system can experience different types of attacks and security solutions, since the communication protocols and the layers' functions are usually not the same those of IT. The Enterprise layer uses transmission control protocol/internet protocol (TCP/IP), so attacks occurring in this layer are the same as those in the IT domain, even though some security details might be different. At this layer, the industrial firewall is considered the prime security protection entity. Firewalls usually define zones using rules to restrict accessibility to ports. Industrial firewalls are able to block all inbound network traffic except that explicitly required by operations. The SCADA/ICS layer is different from that for IT where communication protocols such as PROFINET and POWERLINK are based on industrial Ethernet. The functions of this layer are configuring control system parameters and monitoring the overall health of control systems, so security at this layer will not be achieved by just applying IT security solutions; it should instead be implemented to be consistent with the application characteristics. For example, an active protection line of defense can be implemented to control what a user or a device can do and what programs can be executed. The third layer, the control layer, includes numerous types of communication protocols, like PROFIBUS, controller area network (CAN), RS485, and industrial Ethernet. Since the physical layer is directly connected to the controllers, the control layer directly affects the physical layer, and feedback from the physical layer affects controller decisions. Security Measures must thus have strong adaptability to ensure

a plant's ability to continue operating safely even when an attack occurs. Finally, we must consider the physical layer, where security issues can exist due to either direct or indirect attacks. A direct attack is one that happens by physical contact, while an indirect attack is one that can only be launched from the control layer because other layers have no direct access.

Generally, while IT has developed a mature security technology and design principles such as authentication, access control, message integrity, and separation of privilege that may help guard SCADA/ICS systems against cyber-attacks, IT researchers have sometimes not considered how attacks affect estimation and control algorithms and how they also can affect the physical world. We argue that Studying SCADA/ICS security by itself as a research problem can produce a better understanding of the consequences of an attack on SCADA/ICS. This implies designing novel attack-detection algorithms to fit into the SCADA/ICS domain and designing new attack-resilient algorithms and architectures more suited to SCADA/ICS security.

In summary, the major differences between traditional IT security and SCADA/ICS security are:

- Physical domain dynamics (such as sensor/ actuator/ plant behaviors) can be monitored/ manipulated to bring down the system, a problem unique to SCADA/ICS systems, since IT systems have no physical domain dynamics.
- Availability is crucial, because loss of availability has greater consequence in SCADA/ICS systems than in typical IT systems, so the burden of ensuring security and resiliency is higher, even though it is also a common concern with IT systems.
- Due to continuous dynamics, "patching" or "reconfiguration" can be a lot trickier and must be more precisely described, e.g., when one switches a control policy, replaces/adds sensors/actuators, etc., the switch might destabilize the underlying con-

tinuous dynamics, so switching from one equilibrium condition to a subsequent equilibrium condition must be navigated carefully while maintaining security.

- From the time of their inception, SCADA/ICS systems have been developed independently of IT systems, so, except for the Enterprise Layer, SCADA/ICS systems have evolved their own proprietary protocols such as PROFINET, PROFIBUS, CAN, industrial Ethernet, etc., with their own specialized set of vulnerabilities and fixes, making their security concerns different from the traditional IT systems.

## 1.2 Proposed Framework

We are proposing an SCADA/ICS security enhancement framework as a comprehensive solution that covers all aspects of vulnerability assessment. The framework should analyze SCADA/ICS systems, discover vulnerabilities, and introduce resolutions for them, hence enhancing the security of the system. The proposed solution consists of three building blocks: system component recognition, component vulnerability scan, global vulnerability assessment, and vulnerability resolution, and developing these building blocks will require solving some research problems associated with the intended functions. Figure 1.2 is a block diagram of the proposed framework.

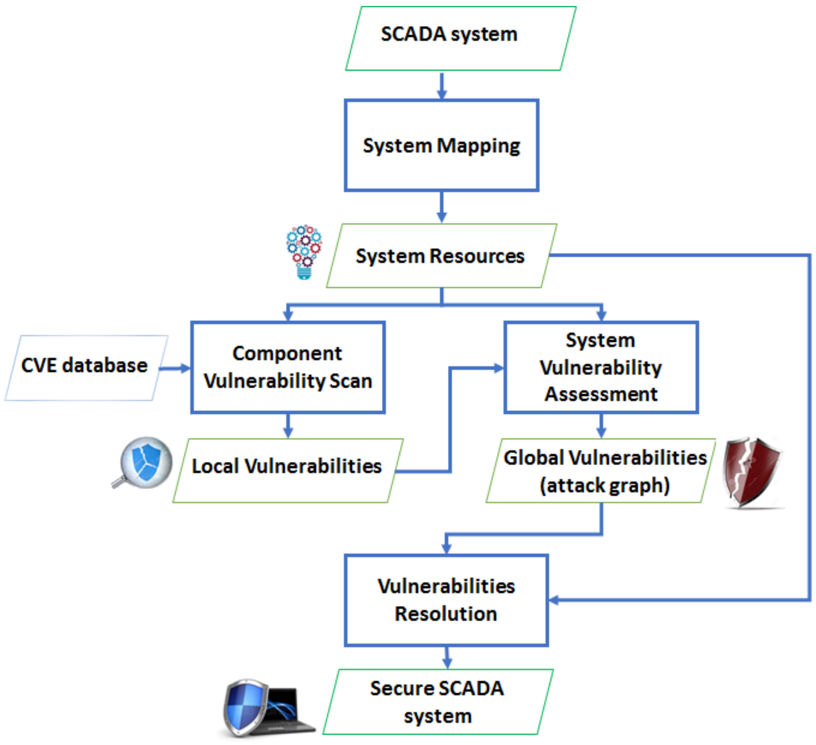

SCADA system
System Mapping
System Resources
CVE database
Component Vulnerability Scan
System Vulnerability Assessment
Local Vulnerabilities
Global Vulnerabilities (attack graph)
Vulnerabilities Resolution
Secure SCADA system

Figure 1.2: Proposed Framework

**System Components Recognition:** Because of the way SCADA/ICS systems were formerly designed, and because of the properties of some legacy systems, SCADA/ICS network resources such as devices, connectivity, protocols, data, access controls, trust relations, etc., have not always been fully known (For example, we might know a reserved IP address but not the physical component to which it is connected), so when there is an attempt to connect existing systems to the internet and to cloud computing, knowledge of SCADA/ICS network resources is very difficult to achieve practically, because that would require going through all documentation and connections. We propose an Automated Network Device and Vulnerability Identification (ANDVI) approach to identify the unknown devices' types, manufacturers, and models. The algorithm uses the SCADA/ICS devices's communication-patterns to recognize the control hierarchy levels of the devices. In conjunction, certain distinguishable features in the communication-packets are used to recognize the device manufacturer, and model. We have implemented this hybrid approach in Python, and tested on traffic data from a water treatment SCADA testbed in Singapore (iTrust). In addition, we propose a Modbus-data analysis based passive fingerprinting approach to identify SCADA/ICS unknown devices, allowing a smaller number of data collection points, which further enhances the applicability of the proposed solution to real-world systems. The Modbus-data analysis based approach was also implemented in Python and tested on data from a water treatment SCADA system. Finally, our ANDVI implementation also maps the identified devices to their known vulnerabilities.

**Global Vulnerabilities Assessment:** Securing Cyber-Physical systems (CPS), and Internet of things (IoT) systems requires the identification of how interdependence among existing atomic vulnerabilities may be exploited by an adversary to stitch together an attack that can compromise the system. Therefore, accurate attack-graphs play a significant role in systems security. A manual construction of the attack-graphs is tedious and error-prone, this work proposes a model-checking based Automated Attack-Graph Generator and Visualizer (A2G2V). The proposed A2G2V algorithm uses existing model-checking tools, an architecture description tool, and our own code to generate an attack-graph that enumerates the set of all possible sequences in which atomic-level vulnerabilities can be exploited to compromise system security. The architecture description tool captures a formal representation of the networked system, its atomic vulnerabilities, their pre- and post- conditions, and security property of interest. A model-checker is employed to automatically identify an attack sequence in form of a counterexample. Our own code integrated with the model-checker parses the counterexamples, encodes those for specification relaxation, and iterates until all attack sequences are revealed. Finally, a visualization tool has also been incorporated with A2G2V to generate a graphical representation of the generated attack-graph. The results are illustrated through application to computer as well as control (SCADA) networks.

**Vulnerability resolution:** Use of attack-graphs has been proposed to represent complex attacks scenarios, that exploit interdependence among the atomic component-/device-level vulnerabilities to stitch together the attack-paths that might compromise a security property at the system-level. While such analysis of attack scenarios enables security administrators to establish appropriate security measurements to secure their system, practical considerations on time and cost can pose limit on their ability to address all system-level vulnerabilities at once. In this work, we propose an approach that identifies label-cuts within an attack-graph to automatically identify a set of critical-attacks that, when blocked, renders the system secure. The identification of a minimal label-cut is in general NP-complete, and in order to deal with this computational complexity, we propose a linear complexity approximation utilizing the Strongly-Connected-Components (SCCs) of the given attack graph to generate an abstracted version of the attack-graph, namely, a tree over the SCCs, and next use an iterative backward search over this tree to identify set of backward reachable SCCs, along with their outgoing edges and their labels, to then identify a cut of the tree, possessing a minimum number of labels and representing a critical-attacks set. We also report the implementation and validation of the proposed algorithm on real-world case studies, including two IT network systems and a SCADA network for a water treatment cyber-physical system. We also compare our proposed algorithm to an exact minimum label-cut algorithm and to an approximation algorithm, both taken from the literature and report the improvements.

## 1.3 Literature Review

Many authors investigated the SCADA/ICS security problem. [8] introduce a risk assessment framework designed for GPS-based railway SCADA systems. This framework applies Hierarchical Holographic Modeling (HHM) where to identify risks it maps the Control Objectives for Information and Related Technology (CobiT) onto a holographic model. [9]

suggest a vulnerability assessment methodology for SCADA systems based on professional experience. [10] propose a methodology for quantitative estimation of cyber risk reduction for SCADA system. To estimate such risk reduction, a cyber-attack directed graph is first generated for both Improved and unimproved systems, followed by measuring and analyzing the variation of time needed to compromise each system.[11] studied SCADA system integrated risk assessment and probabilistic assessment in Energy Management Systems and Management Information Systems. A scenario-based approach to vulnerability assessment is used by the Control Systems Security Center (CSSC) for the National Cybersecurity Division of the Department of Homeland Security [12]. [13] introduced a method for SCADA system security level qualitative assessment that helps managers make decisions about implementing security countermeasures. [14] introduced a three Petri net model for quantifying the risk of cyber-attacks on SCADA systems, aiming to identify all high-consequence attack states.

[15] approach vulnerability assessment by developing an infrastructure hypergraph and an evolution graph (a more detailed version of an attack graph. [16] introduced a Network Security Risk Model, a directed graph representing an attack, whose goal is to aid in the selection of risk management controls by providing a measure of risk and by calculating the measure for a baseline and for a security improved system. [17] introduced the Adversary View Security Evaluation (ADVISE) method that adds the characteristics of an adversary to an attack graph. The purpose of that method is to identify the most likely attack path by using an executable state-based security model to calculate an attack's probability of success. [18] proposed a SCADA security framework RAIM with a four-part network, including real-time monitoring, anomaly detection, impact analysis, and mitigation strategies. [19] suggested a cybersecurity risk assessment methodology that can be applied during the process of instrumentation and control systems design in nuclear power plants. It outlines the steps that should be followed to perform vulnerability assessment during both system and component design and equipment supply. [20] proposed a Boolean logic Driven Markov

Processes (BDMP) modeling approach that combines fault trees with Markov processes to achieve both qualitative and quantitative vulnerability assessment. [21] suggested a Phasor Measurement Unit (PMU) based on a risk assessment framework for SCADA systems of power grids. It identifies and quantifies vulnerabilities within such systems using the Duality Element Relative Fuzzy Evaluation Method (DERFEM), then generates an attack graph that can be used to find intrusion scenarios. [22] discussed the application of game theory to cybersecurity analysis of a smart grid SCADA system where they modeled the attacker as a two-player, non-zero sum, noncooperative, sequential, and perfect information game for the SCADA system administrator. [23] suggested a methodology for quantitative assessment of vulnerability assessment in SCADA systems based on optimal power flow and power flow tracing.

A limited number of studies exist that proposed approaches to identify the SCADA/ICS components. [24] proposed identifying the devices based on specific SCADA protocol functions (e.g., in Modbus protocol there is a function that returns the device information). Such approaches are limited to the specific protocol used, and do not generalize. The SCADA search engine Shadon [25], that can discover the devices connected to the internet, has inspired works as [26] and [27] to use port scanning methods to identify SCADA devices. However, these approaches employ active fingerprinting, and if applied without proper study and preparation, can interrupt and even “freeze” some devices, as mentioned earlier.

[28] proposed the use of TCP/IP information to construct a device fingerprint. [29] generated a device fingerprint based on the response time. This approach is only device-specific, and not device-model-specific, thus offering limited resolution of devices. [30] identified the type of a device based on the SCADA/ICS communication-patterns, but is limited to two layer systems. The SCADA/ICS device fingerprinting problem was also considered in [31], where the author explored the existing approaches, without proposing a new one; The paper concluded that current tools are not fully compatible with SCADA/ICS

systems. [32] evaluated Shadon search engine and concluded that Shadon search engine can be a threat to SCADA/ICS systems as it can identify Internet-facing industrial control devices; The paper did propose a mitigation technique to protect from Shadon, and did not propose a new device recognition tool.

Some authors focused on attack-graph generation to evaluate the SCADA/ICS system vulnerability. [33] proposed a process calculus based model to generate a graph, by translating algebraic specifications into formulas, and next backward chaining those formulas into a formula corresponding to the success of the attacker goal. [34] used logic programming to construct attack-graphs. The logic reduction was applied starting from the primary facts until reaching the objective facts. [35] used generalized stochastic Petri nets to model intrusions into the control networks. [36, 37] constructed attack graphs using network connectivity information where an attacker can reach the next desired system location from the current system location if it is adjacent to it. This search strategy was applied recursively until the entire attack graph was constructed. [38] used a multilevel Bayesian network to construct an attack-graph by integrating knowledge of attacks, system functions, and hazardous incidents. [39] proposed a layered search algorithm. The bottom layer represents the attacker initial privileges. An upper layer represents new computed privileges after each iteration. [40] proposed a "worst-case scenario attack-graph". Each node represents a network host, and the edges represent the atomic attack that an attacker can use to gain the highest privilege on the next target node (the worst-case attack). [41] proposed reducing the search complexity by constructing a reachability matrix using grouped reachability conditions. To eliminate duplication, the algorithm uses pre-stored hashes. The author also proposed a new attack-graph structure called multiple-prerequisite graphs, where nodes represent the reachability conditions among network hosts. [42] proposed a bidirectional search method to construct attack-graphs. The forward search starts from initial states, while the backward search starts from the goal states. [43] introduces a parallel and distributed memory-based algorithm that builds vulnerability-based attack graphs on a distributed multi-agent platform. These approaches work at certain levels of abstraction and may miss finer details of how atomic attacks can be concatenated to compromise a system.

[44] considered a model-based approach for attack generation where it generates a single attack path from a single reported counterexample. [45] introduced the model-based approach to create an attack-graph for a networked system. The state space of the system was formally modeled, using configuration parameters and attacker privileges. The state-transitions represent the atomic attack actions. The approach in [45] is novel, yet limited to the $AG(safe)$ type security property, whose violation corresponds to the reachability unsafe locations in the model. More general requirements, such as $AG(p \rightarrow AXq)$ that are violated by the reachability of certain paths (rather locations) can exist.

Finally, few Papers focused on Vulnerabilities resolutions using attack-graph analysis. [46] showed the NP-completeness of finding the minimum critical-attacks set (equivalently, the min Label-Cut, MLC) by reducing the minimum cover problem to it. [47] established the NP-completeness by reducing the Hitting-Set problem to the MLC, and also presented a greedy algorithm to the Hitting-Set problem that picks the elements with the highest hits first. [48] showed that a relaxed version of MLC (one that seeks a certain approximation to MLC) itself is NP-Complete. They proposed an approximation algorithm for finding a solution to MLC within a factor of $O(\sqrt{n})$. [1] improved upon this work by proposing an approximating solution to MLC within a factor of $O(n^{2/3})$. Their approach computes the minimum polytope corresponding to an "approximate" hitting set.

[49, 50] studied "minimum-cost network hardening" that is essentially a similar problem of finding the minimum-costing hardening measures of network components, exploring their dependency relations (as in case of an attack graph). [51] used generalized stochastic Petri nets to quantitatively evaluate the control networks intrusions probability. [52, 53] proposed the use of Google's page-rank based approach for network security, in which the idea was to successively remove the highest page-ranked nodes to maximally decrease the overall connectivity. [54] proposed a genetic algorithm for finding a minimum cut set in AND/OR

dependency attack-graphs, with the cut set representing a set of security countermeasures that prevent attackers from exploiting the underlying vulnerabilities. [55] studied a related problem of finding a minimum label spanning tree and a minimum label path, but those do not necessarily induce a cut. [56] proposed an exact algorithm to compute a MLC for a special case, where the graph is disjoint undirected graph with label frequency at most 2, in polynomial complexity. The paper also proposed an approximation algorithm for a more general undirected graph. However, since an attack-graph is a directed graph, the proposed algorithms do not apply to them.

Examining the state of the art in vulnerability assessment reveals that further improvement and research is required in areas such as overcoming cyber-attacks and failures, reliability improvement, evaluation and validation, and tool support. Vulnerability assessment is also missing a comprehensive method that would cover all stages of the risk management process. In this thesis, we propose solutions and enhancements to fill this research gap in the vulnerability assessment.

## 1.4 Thesis Organization

The rest of this thesis is organized as follows:

Chapter 2 ANDVI: Automated Network Device and Vulnerability Identification in SCADA/ICS by Passive Monitoring.

Chapter 3 A2G2V: Automatic Attack Graph Generation and Visualization and its Applications to Computer and SCADA Networks.

Chapter 4 Critical-Attacks Set Identification in Attack-Graphs for Computer and SCADA Networks

Chapter 5 Summary and future work.

# CHAPTER 2. ANDVI: AUTOMATED NETWORK DEVICE AND VULNERABILITY IDENTIFICATION IN SCADA/ICS BY PASSIVE Monitoring

SCADA/ICS systems are installed by third party contractors, maintained by site engineers, and operate for a long time. This makes tracing the documentation of the systems' changes and updates challenging since some of their components' information (type, manufacturer, model, etc.) may not be up-to-date, leading to possibly unaccounted security vulnerabilities in the systems.

To catch any existing vulnerabilities in an SCADA/ICS system, a complete understanding of devices, practices, protocols, and applications must be established to be able to enumerate the previously identified vulnerabilities for that set of components, and develop the defenses against the potential attacks (such as providing patches to remove the existing vulnerabilities). To develop such desired knowhow of a legacy system, one important prerequisite is the identification of the devices (e.g., Programmable Logic Controller (PLC), Human Machine Interface (HMI)), protocols, and applications/services on the network. This is also referred to as device fingerprinting, and if proper documentations were available, a simple solution could be to refer to those. However, per the DHS report on common cybersecurity vulnerability in SCADA/ICS [57], it turns out that in many instances, the documentation and implementation differ owing to the inadequately documented changes and updates, rendering the referral to any such documentation erroneous/incomplete. This situation is also encountered in traditional IT infrastructures, and a typical solution is to resort to network fingerprinting techniques. This can be either active or passive fingerprinting.

Active fingerprinting attempts to identify the devices on the network by actively requesting information from the devices: For example it may send probing packets to an IP address and analyze any response [58]. One may attempt to implement such an active approach for an SCADA/ICS network, but it raises two concerning issues. Firstly, it may add to the traffic load of the network, whereas most SCADA/ICS systems have congestion constraints with strict latency requirements, and their violation may raise certain safety issues. Secondly, some legacy devices on an SCADA/ICS system are not designed to robustly support any unsupported packets, and their introduction may set the device into an unpredictable state. These concerns make the active fingerprenting for SCADA/ICS systems circumspect, and so not preferred by SCADA/ICS users, as applying it requires further analysis to assess the impacts on the systems.

Passive fingerprinting uses a network sniffer to capture traffic already generated by the system devices, and analyzes this traffic to identify the devices. For passive fingerprinting to function appropriately, it is pertinent to define the distinguishing features in the network traffic to facilitate device identity. IT infrastructure network traffics have known distinguishable features that can be used to identify the network devices [31]. In an SCADA/ICS, a network does not typically provide *explicit* signatures for a passive fingerprinting, as its equipment are often simple and unable to offer extra-functional services. The passive fingerprinting task can be approached by first inferring the device type, that can reduce the amount of features needed to identify the devices in the network and thereby improve its accuracy. We use the communications-pattern to first identify the device hierarchy level to reveal its type, and next use the passive fingerprinting for an enhanced device recognition.

SCADA/ICS systems have specific network topology requirements, two of which are very useful for our approach. (1) Each device is only allowed to communicate with pre-programmed devices. (2) The devices follow a chain-of-command based communication-pattern to prevent any command conflict. Such topological information about communication can be used to identify the control hierarchy level of a device within that chain-of-command, where the levels of the SCADA/ICS hierarchy help identify a device type (SCADA station, HMI, PLC, etc.).

Accordingly, we introduce a two-step hybrid approach that uses both the communication-patterns and device-fingerprints (constructed through passive fingerprinting techniques). Our two-step algorithm first uses the network traffic to explore the SCADA/ICS communication-patterns and identify the control hierarchy, and next a fingerprint for each device obtained from its communication packets and is compared with the learned ones in the reference database (explained in next sentence) to determine the exact model details of the device. To learn the reference database, the traffic of a network with known devices is gathered and analyzed to identify the device specific values in certain fields of the communication packets.

We implemented this hybrid device recognition approach in Python, and tested it on traffic data from a water treatment testbed, reported in [59]. Our reference fingerprint database was constructed from the traffic data reported in [59, 60, 61]. [59] described a water treatment system controlled by Allen Bradly PLCs, HMI, and SCADA station. [60] presented a sub system consisting of Direct Logic 205, Siemens S7 1200, and a SCADA station. [61] provided one-to-one communication between a SCADA station and Siemens S7 1500, S7 1200, S7 300, and S7 400 PLCs. In other words, our reference dataset represents a variety of SCADA/ICS devices from different vendors to be able to validate the generality of our approach.

One of the challenges that any passive device fingerprinting faces is the number of data collection points. Due to the hierarchical nature of the SCADA system, the devices packet only exist between the parents/children devices, and do not extend beyond that level. So, a data collection scheme needs to collect data for each device. In some SCADA systems, routers exist to connect different children devices to their common parent, so a data collection point between the router and the parent device can capture all the required data, reducing the number of data collection points. For this a more in-depth packet inspection is required to fingerprint a device. We describe this for SCADA/ICS utilizing Modbus for device communication, which is one of the most widely used application-layer protocol [62].

Nowadays, the Modbus protocol is integrated with the TCP/IP protocol, i.e., the Modbus packets are encapsulated into the TCP/IP packets. Since the routers are not allowed to modify the TCP/IP data, the Modbus messages are not affected when transmitted over the network even if routers exist in the path between the source and the destination. Using this observation, in this work, we propose passive fingerprinting approach to identify SCADA/ICS devices that utilize Modbus-protocol for communication. The fingerprints are created by analyzing the Modbus messages and identifying the read/write register ranges for the unknown devices, which is then compared to the fingerprints in a reference database. SCADA/ICS device datasheets include a register map that identifies the read/write registers for each SCADA/ICS device, and so the reference fingerprints database can be created directly from the manufacturers' datasheets. We implemented this device recognition approach in Python, and tested it on traffic data from a water treatment SCADA system.

A final step in our work on Automated Network Device and Vulnerability Identification (ANDVI) is the mapping of the discovered devices to their known vulnerabilities. To do this, our tool extracts that information availability in Common Vulnerabilities and Exposures (CVE) database and National Vulnerability Database (NVD). When combined with our work on A2G2V (Automated Attack Graph Generation and Visualization) [63], this can map device vulnerabilities to attack graphs of all ways a system-level security property of interest may be compromised. Further, by integrating with our work on SCCiMLC (Strongly Connected Component induced Min Label Cut) [64], a minimal set of vulnerabilities can be identified whose mitigation renders the entire system secure. The main contributions of the work presented here are:

1. To the best of our knowledge, for the first time, a hybrid communication-pattern and passive fingerprinting approach is proposed to identify the SCADA/ICS devices' type, manufacturer, and model.

2. To the best of our knowledge, for the first time, the SCADA/ICS communication-pattern is used to identify the device control hierarchy level, and next use it to refine the device type. The paper identified a set of features in the communication packets that can be used to distinguish among the devices based on their control hierarchy level.

3. We developed a software implementation for the proposed Hybrid approach, and report its validation using data from a real-world water treatment CPS system from iTrust [59].

4. To the best of our knowledge, for the first time, a passive fingerprinting approach is proposed to identify the SCADA/ICS devices' type, manufacturer, and model by passive analysis of Modbus data by mapping out the read/write registers of the devices and comparing those to a reference database.

5. We developed a software implementation of the Modbus-data analysis based approach, and its validation using a water treatment CPS example.

6. We integrated our devices recognition algorithms with known vulnerabilities databases to automatically identify the devices and their associated known vulnerabilities.

## 2.1 Proposed Hybrid Approach to Device Recognition

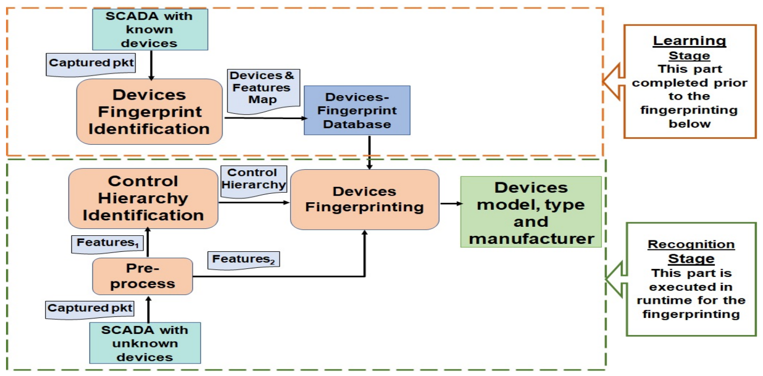

SCADA with known devices
Captured pkt
Devices Fingerprint Identification
Devices & Features Map
Devices-Fingerprint Database
Learning Stage
This part completed prior to the fingerprinting below
Control Hierarchy Identification
Control Hierarchy
Devices Fingerprinting
Devices model, type and manufacturer
Features1
Features2
Pre-process
Captured pkt
SCADA with unknown devices
Recognition Stage
This part is executed in runtime for the fingerprinting

Figure 2.1: Hybrid approach for devices recognition

Figure 2.1 depicts the architecture of the proposed hybrid approach for device recognition in an SCADA/ICS network. The approach is based on the TCP/IP passive fingerprinting where the communication packets are captured for analysis and device recognition. This is a two-stage process: the first stage is a learning stage where a reference fingerprints database is created from an SCADA/ICS system containing a known set of devices, using the identified values of certain features in the communication packets of the devices. Creating such a database requires identifying the reliable distinguishable features in the captured TCP/IP packets that can be used to generate a distinguishable fingerprint for each device. The second stage is a recognition stage where packets captured from a SCADA system with unknown devices are processed, and passed into a two-step algorithm: a control hierarchy identification step and a device fingerprinting step.

SCADA/ICS systems control hierarchy follows the standard architecture shown in Figure 2.2 [65]. The hierarchy is imposed to prevent conflicts in the control commands, requiring each node in the system to communicate only with its direct parent, its immediate children, or with nodes at the same level. This characteristic can be used to identify the SCADA/ICS devices' level using our control hierarchy identification algorithm (see Algorithm 1 in Section 2.1.2). Knowing the device control hierarchy level, allows us to infer the device type (e.g., local controller level is a PLC, plant supervisory controller level is either HMI or SCADA station, production coordinator controller level and control center level are SCADA stations).

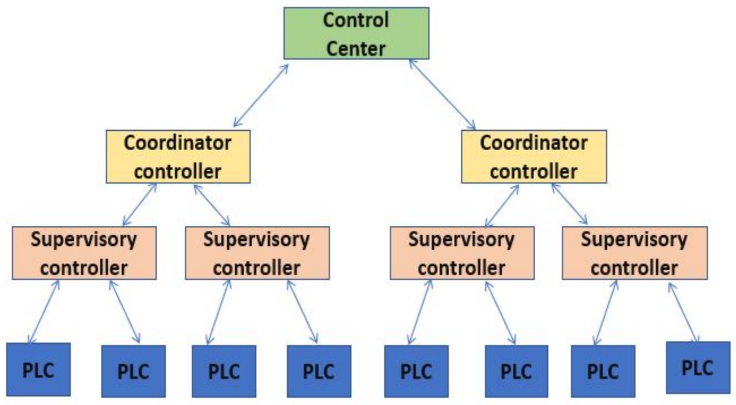


Figure 2.2: Standard SCADA/ICS control hierarchy

The above step of control hierarchy level identification is useful when SCADA/ICS devices in the same system are from the same vendors and use same programmer protocols. In such cases, different types of SCADA/ICS devices have similar TCP/IP communication packet features (Section 2.1.1 gives an example of such a case), making it challenging to distinguish devices based solely on communication packets. However, by first inferring their control hierarchy, we can identify a device type (PLC vs. HMI), and this together with the distinguishable features of the communications packets (the second step) then further pinpoints the device.

The packet processing in the recognition stage is simply to extract the features from the TCP/IP packets and store those into two matrices: $Features_1$ and $Features_2$. $Features_1$ has two columns for each packet and the total number of rows equals the number of packets captured. Column 1 records the source of the packet and column 2 records the destination. $Features_2$ has 3 columns for each packet captured, and again the total number of rows is the number of packets captured. Each column records a distinguishable packet feature, described below. To be able to fully recognize all the devices, the captured packets need to span all the devices in the SCADA/ICS network, and we assume that to be the case.

### 2.1.1 Learning Stage: Building Reference Database

The first stage in device recognition is to build a reference fingerprints database by analyzing the captured packet from an SCADA/ICS network with *known* devices, and identifying the values of certain features in the communication packets that are unique to the devices. To identify such distinguishable features in the captured network traffic, we examined data generated from several different SCADA/ICS devices Allen Bradley Logix 1765 PLC, Siemens S1200 PLC, Siemens S1500 PLC, Siemens S7 300 PLC, Siemens S7 400, and DirectLogic 205 PLC, reported in [59, 61, 60]. We examined a total of 20 TCP/IP packet features including: frame length, vendor MAC ID, TCP-segment length, IP identification numbers, Total IP packet length, time to live, and tcp-window size. These 7 features have been plotted below in Figure 3 to illustrate our fingerprinting approach. Among these 7 features 3 were found to be unique to a device. These 3 features are described in bit more detail next.

When a packet is generated, the source of the packet specifies the duration a packet remains valid, called time-to-live, and denoted ($TTL$). The $TTL$ values for the packets from the same source/model are always the same, and so serve as one distinguishing feature. Another feature that a packet source specifies is the packet ID, that it increases in the same fixed increments, and that increment amount is different for different source/model. Thirdly, each generated packet also specifies the source MAC address, in which first four bytes form the vendor ID. The vendor ID in the MAC address can be used to identify the device manufacturer using the databases that enlist the devices' vendors and their MAC vendor IDs (for instance the OUI Lookup Tool from Wireshark [66]). Generally, vendors use more than one MAC vendor ID, specific to a factory or manufacturing time period. This can be used to further differentiate among the devices from the same vendor. An example for this is Siemens S1200 (by Siemens Numerical Control Ltd) versus S1500 (manufactured by Siemens AG); S1200 MAC vendor ID is 001C06 while S1500 MAC vendor ID is 001B1B.

The analysis results of packet feature values for different packets, and of different devices, are plotted in Figure 2.3, where each diagram is for a single device, and in each diagram, the x-axis is discrete corresponding to the 7 features mentioned above, and the y-axis is also discrete showing the values of the packet-features (encoded as numeric values). Eg, packets from the device S7 1500 are plotted in the first diagram. It can be seen that several of the SCADA/ICS packet features from a same device vary from packet-to-packet, except for the 3 distinguishable features mentioned above that are constant, and turn out to be adequate to differentiate among the 6 different PLCs. These 3 features are: Time To Live ($TTL$), the difference in the IP.IDs of two consecutive packets (which we denote as, $IP.ID_{diff}$), and the vendor MAC ID ($Vendor_MAC$). Note given that $TTL$, $Vendor_{MAC}$, and $IP.ID_{Diff}$ can take 255 (1 byte), $4.294 \times 10^9$ (4 bytes), and $65.535 \times 10^3$ (2 bytes), values respectively. Thus using these 3 features, one can potentially distinguish a total of $255 \times (4.294 \times 10^9) \times (65.535 \times 10^3) = 7.176 \times 10^{16}$ number of devices.

Table 2.1 summarizes the results from our analysis of the communication packets and consolidating those into the 3 distinguishing features $TTL$, $IP.ID_{diff}$, and $Vendor_{MAC}$, that can be used for look up to identify the device manufacturer and model. This table reports the communications-pattern based reference database created during the learning stage of our approach for identifying different SCADA/ICS devices.

Table 2.1: SCADA/ICS distinguishing features summary

| SCADA/ICS device | $TTL$ | $IP.ID_{diff}$ | $Vendor_{MAC}$ |
|---|---|---|---|
| Logix 1765 PLC | 64 | 256 | 001d9c |
| Direct Logic 205 | 32 | 1 | 00E062 |
| S-1200 | 30 | 1 | 001C06 |
| S-1500 | 30 | 1 | 001B1B |
| S7-300 | 30 | 1 | 000e8c |
| S7-400 | 128 | 3 | 000e8c |

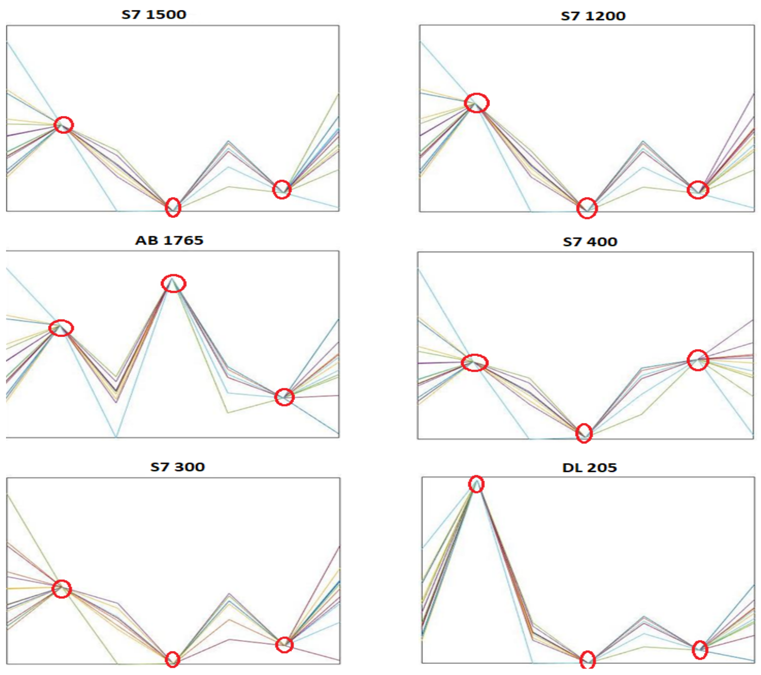


Figure 2.3: PLCs data analysis

Note while the above 3 packet features are constant across the packets from the same device, there can still exist more than one device with identical set of 3 features. An example pair is the Logix 1765 PLC versus the Allen Bradly HMI; the two devices are identical with respect to the aforementioned 3 TCP/IP features as shown in Figure 2.4. However, such devices are not placed at the same control hierarchy, and so we use the extra information about the control hierarchy to further pin-point a device.

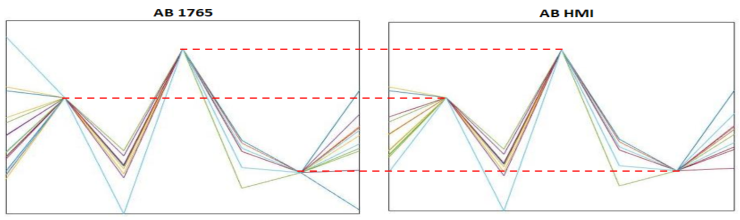


Figure 2.4: Logix 1765 PLC and the Allen Bradly HMI data analysis

### 2.1.2 Recognition Stage

This stage is used to recognize the unknown devices in a SCADA system by monitoring the network communications packets in the run-time, by first deciphering the control hierarchy, and next utilizing the packet features of Table 1 to identify the devices on the network.

To maintain a high level of reliability and safety, the SCADA/ICS systems architecture and communication follow the IEC 6113 [67] and the IEC 62264 [68] standards. Using such standardized control hierarchy architecture as a basis, we identify the SCADA/ICS devices' level as formulated in Algorithm 1 below.

The algorithm proceeds in bottom-up fashion, by identifying the bottom-most (also, called level-0) devices of the control hierarchy, and once the devices of a level have been identified, the devices of a next level can be identified as the destinations of the former. To identify the level-0 devices, first the devices with identical destinations are grouped into a same group. Then the level-0 devices correspond to the members of those groups whose destinations are not larger than those of any other groups. Once the level-0 devices are recognized, finding the higher levels is easier: Given level-$k$ devices for some $k \geq 0$, their set of destination devices, that do not belong to any lower levels, form the level-$(k+1)$ devices. The algorithm terminates when all the devices are already accounted for in one of the levels.

Here we use the example in Figure 2.5 as a running example to illustrate these ideas. Initially, using the $Features_1$ data, that contains the list of all source-destination pairs, we compute for each source node $s$, its set of destination devices $D_s$. In Figure 2.5 for example, $D_{s_1} = \{s_1, s_2, s_3, s_6, s_7\}$.

Next we identify the sources that are at the bottom of the control hierarchy. For this, the sources with *identical destinations* are grouped together into their own single groups. Then the sources in those groups are in the bottom of control hierarchy if their destinations are no larger than those of any other groups. For example, since $D_{s_1} = D_{s_3}$, we group

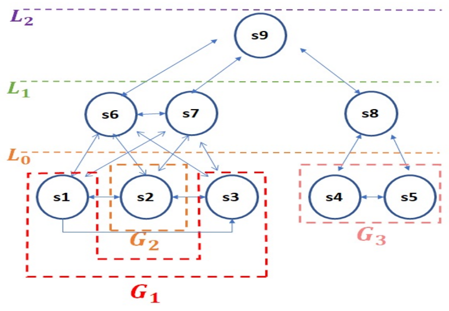


Figure 2.5: Control hierarchy identification example

those into the group, $G_1 = \{s_1, s_3\}$. Similarly, $G_2 = \{s_2\}$, $G_3 = \{s_4, s_5\}$, $G_4 = \{s_9\}$, etc. Next, the groups whose destinations are not larger than any of the other groups' destinations are identified. So for example, the destinations of $G_1$ and $G_2$ are smaller than those of group $G_5 = \{s_6, s_7\}$ whose destinations are $D_{s_6} = D_{s_7} = \{s_1, s_2, s_3, s_6, s_7, s_9\}$, whereas the destinations of $G_3$ includes $s_8$ that is not a destination for $G_1$ or $G_2$, etc. Similarly, the destination of $G_3$ namely, $\{s_4, s_5, s_8\}$, are smaller than the destination of $G_6$, namely $\{s_4, s_5, s_8, s_9\}$, etc. With such grouping and destination comparison, the sources in $G_1, G_2, G_3$, are identified to be the lowest level groups in the control hierarchy, and this is recorded in $L_0 = G_1 \cup G_2 \cup G_3$.

Once the level-$k$ sources in $L_k$ get recorded for any $k \geq 0$, the level-$(k+1)$ sources in $L_{k+1}$ are identified as the union of those groups which contain a destination of a level-$k$ source, and which do not belong to the level-$k$ or lower. So for example, $L_1 = G_5 \cup G_6$, and next $L_2 = G_4$. The above steps are formalized in the following algorithm.

Once the control hierarchy levels have been identified using the above algorithm, the TCP/IP packet features ($TTL$, $IP.ID_{diff}$, $Vendor_{MAC}$) stored in the $Features_2$ dataset are compared to the reference database to identify the device manufacturer and model.

It should be noted that, because of the hierarchical nature of the SCADA system discussed above, and since packets generated by the SCADA devices do not travel beyond their parent node level, the data collection needs to be done between the parent/child hierarchy level of each device, to ensure the inclusion of data from all system devices. This

**Algorithm 1** Control Hierarchy Identification Algorithm

```
1: Input: Set of packets
2: Output: sources S, control levels $L$, control level mapping $C : S \rightarrow L$ that
   maps each source $s \in S$ to its control level $C(s) \in L$.
3: main
4: for each source $s$, add it to a set $S$, and also construct its destination $D_s$
5: compare $D_s$ for all $s \in S$
6: group sources with same destinations into groups $G_i$'s
7: $\forall i : G_i \subseteq L_0$ if $\forall j : G_j \not\subset G_i$
8: $\forall i, \forall k \geq 0$: $G_i \subseteq L_k + 1$ if $(\exists s \in L_k : D_s \subseteq G_i) \wedge (\nexists l \leq k : G_i \subseteq L_l)$
9: Terminate when all $s \in S$ are mapped to some level $k \geq 0$.
```

also ensures that the data fingerprints we consider for device identification, namely, TTL, IP.ID$_{diff}$, Vendor$_{MAC}$, do not undergo alteration in the network.

### 2.1.3 A CPS Case Study: iTrust Water Treatment SCADA

To demonstrate the applicability of our algorithm to a real-world cyber-physical system (CPS), a water treatment system was chosen as a case-study (Figure 2.6). The system is based on a fully-functional water treatment CPS testbed in the iTrust laboratory [59]. The system consists of Programmable Logic Controllers (PLCs), Human Machine Interfaces (HMIs), Supervisory Control and Data Acquisition (SCADA) workstation, and a SCADA Server. The 6 PLCs control the 6 stages of water treatment, while the HMI and the SCADA station monitor and coordinate between the PLCs. The SCADA station reports to and coordinates with the SCADA server. These devices and their control hierarchy are shown in Figure 2.6.

The Secure Water Treatment testbed dataset comprises of data from 7 days of continuous normal operation. For the device recognition purposes, only one full control cycle of data is needed, and so for our purposes the network traffic for 4 hours of normal operation was sufficient, and was analyzed. The information in the pcap captured files were extracted and stored in CSV files using Tshark [69].

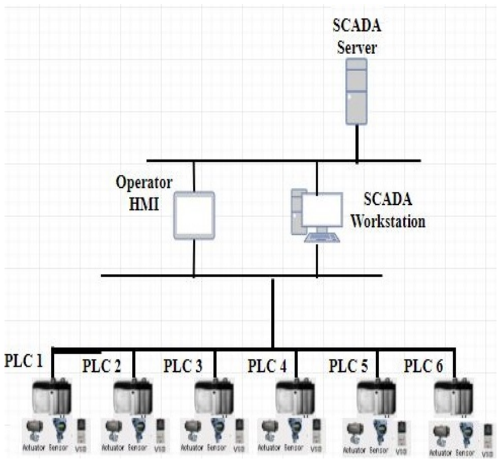
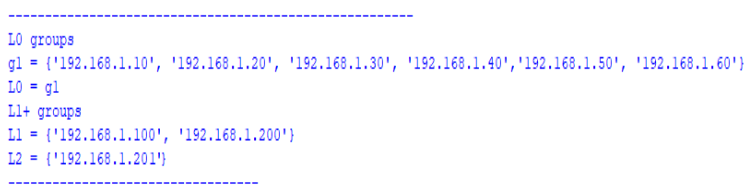


Figure 2.6: Architecture of iTrust Water Treatment SCADA

Initially, a preprocessing step was applied to the collected packets to obtain the $Features_1$ and $Features_2$ matrices. The algorithm started by enumerating the set of sources $S$ and the destinations $D_s$ for each source $s \in S$. 9 members were found as sources. Next, $s =$ 192.168.1.10 was randomly selected, and its destination set was compared to the destinations of other devices. As a result, 192.168.1.10, 192.168.1.20, 192.168.1.30, 192.168.1.40, 192.168.1.50, and 192.168.1.60 were grouped in $G_1$. Next, $s =$ 192.168.1.100 was selected, and comparing its destinations to those of the other remaining devices resulted in $G_2 = \{192.168.1.200, 192.168.100\}$. Finaly, $s =$ 192.168.1.201 was selected; it being the last source in $S$, it was finalized that $G_3 = \{192.168.1.201\}$. $G_1$ is the only group that does not contain the other groups as a subset, so $L_0 = G_1$. $G_2$ contains the destination for $G_1$ devices, so $L_1 = G_2$. Similarly, $G_3$ contains the destinations for $G_2$ devices, and so $L_2 = G_3$. At this point all sources in $S$ have been mapped to a control level, so the algorithm terminated. The result from our Python encoding of Algorithm 1 is shown in Figure 2.7.

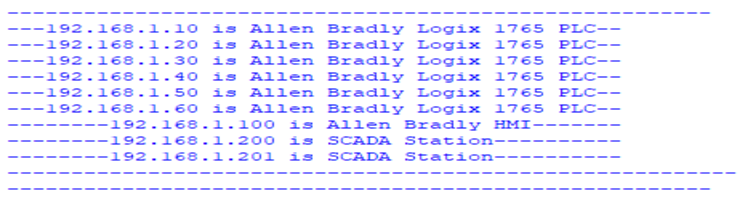
Based on the control hierarchy identification outcomes, the bottom level devices, 192.168.1.10, 192.168.1.20, 192.168.1.30, 192.168.1.40, 192.168.1.50, and 192.168.1.60 can be identified to be PLCs. The level-1 devices, 192.168.1.100 and 192.168.1.201 can either be a SCADA



```
-------------------------------------------------------
L0 groups
gl = {'192.168.1.10', '192.168.1.20', '192.168.1.30', '192.168.1.40','192.168.1.50', '192.168.1.60'}
L0 = gl
L1+ groups
L1 = {'192.168.1.100', '192.168.1.200'}
L2 = {'192.168.1.201'}
----------------------------------
```

Figure 2.7: iTrust Water Treatment SCADA control hierarchy identification output

station or a HMI device. Finally, the top level device, 192.168.1.200 is identified as a SCADA station. Subsequently, the vendor ID was used to distinguish between the HMI and the SCADA station: the vendor ID for both 192.168.1.201 and 192.168.1.200 were Microsoft, while the vendor ID for 192.168.1.100 was Rockwell Automation. Based on this, 192.168.1.201 and 192.168.1.200 were identified to be SCADA stations, while 192.168.1.100 was identified as an Allen Bradly HMI. The device fingerprinting step constructed a fingerprint for each PLC in the form of $Feature_2$ matrix, which was then compared against the lookup table (Table 2.1) to identify their manufacturer and model. This step then revealed that the PLCs are the Allen Bradly Logix 1765 PLC. A log of these results is shown in Figure 2.8.



```
-------------------------------------------------------
---192.168.1.10 is Allen Bradly Logix 1765 PLC--
---192.168.1.20 is Allen Bradly Logix 1765 PLC--
---192.168.1.30 is Allen Bradly Logix 1765 PLC--
---192.168.1.40 is Allen Bradly Logix 1765 PLC--
---192.168.1.50 is Allen Bradly Logix 1765 PLC--
---192.168.1.60 is Allen Bradly Logix 1765 PLC--
--------192.168.1.100 is Allen Bradly HMI-------
--------192.168.1.200 is SCADA Station----------
--------192.168.1.201 is SCADA Station----------
---------------------------------------------------------
-------------------------------------------------------
```

Figure 2.8: Water Treatment SCADA device recognition output

### 2.1.4 Discussion

The experimental results from the hybrid device recognition algorithm show that it can successfully identify the SCADA system devices type, manufacturer, and model. In addition, its ability to map SCADA devices to their levels in control hierarchy helps improve the passive fingerprinting accuracy, since this limits the search for an unknown device to only those devices that are eligible to that level of hierarchy.

The computational complexity of the algorithm is determined by the number of nodes in the network ($N$) and has a worst-case running time of $O(N^3)$. Which is the maximum time required to construct the system control hierarchy (list all nodes (N), identifying their destination list $D_s$, and comparing $D_s$ for all nodes). On a standard computer, Core i5/2.2GHz/4GB RAM running Win 10, our algorithm took 15 seconds to identify the devices for the above real-world case study.

## 2.2 Minimizing Data Collection Points: Role of Routers

As noted earlier, since the packets generated by the SCADA devices do not travel beyond their parent node level, the data collection needs to be done between the parent/child hierarchy level of each device. To understand the implication of this fact, consider the two SCADA systems of Figure 2.9 and Figure 2.10. The one shown in Figure 2.9 is a Siemens PLCs based system that controls a water treatment facility. Here the SCADA system network is based on ring topology that does not require any routers. Figure 2.10 is a Mitsubishi PLCs based system that controls an animal food manufacturing factory, and uses routers. Note both systems are existing working systems in use.

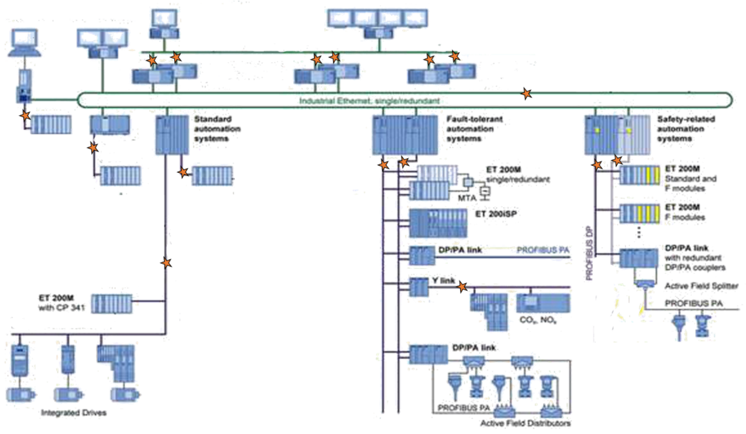

Industrial Ethernet, single/redundant
Standard automation systems
Fault-tolerant automation systems
Safety-related automation systems
ET 200M single/redundant
MTA
ET 200iSP
DP/PA link
PROFIBUS PA
Y link
CO$_2$, NO$_x$
DP/PA link
PROFIBUS PA
Active Field Distributors
ET 200M Standard and F modules
ET 200M F modules
PROFIBUS DP
DP/PA link with redundant DP/PA couplers
Active Field Splitter
PROFIBUS PA
ET 200M with CP 341
Integrated Drives

Figure 2.9: Ring topology SCADA system architecture

Examining the two systems reveals that the minimum number of data-collection points needed in the ring topology to identify the system components (shown as orange stars) is much less than in a router-based system ( total 15 vs. 31). Yet the latter type system (with routers) is more common being cheaper (ring topology requires extra connections). Motivated by this observation, we propose a passive fingerprinting approach that additionally exploits the features of Modbus protocol, extensively used in SCADA/ICS networks for communication among its devices, and thereby minimizes the number of data collection points, by collecting data at the routers (rather than at the devices). In Figure 2.10, the minimum number of data-collection points needed to identify the devices are shown as green stars ( total 31 as opposed to 11). The proposed Modbus-data analysis based approach ensures that only the part of the data that is not affected by router actions is examined for device fingerprinting (and so data can be collected at routers as opposed to directly at devices).

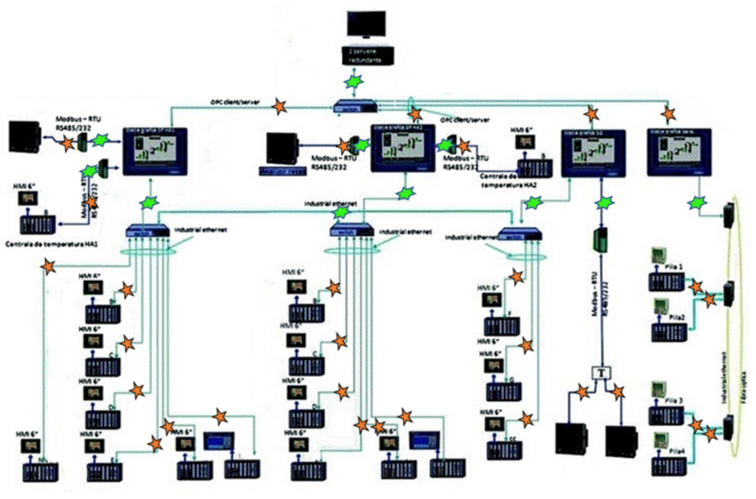


Figure 2.10: Routers based SCADA system architecture

## 2.3 Modbus-data Analysis for Fingerprinting

Modbus TCP / IP (also known as Modbus-TCP) is a Modbus RTU protocol operating on Ethernet with a TCP interface. i.e., the Modbus RTU message is transmitted with a

TCP/IP wrapper, and sent over a network (rather over serial lines). Figure 2.11 demonstrates how a Modbus message is embedded within a TCP/IP message.

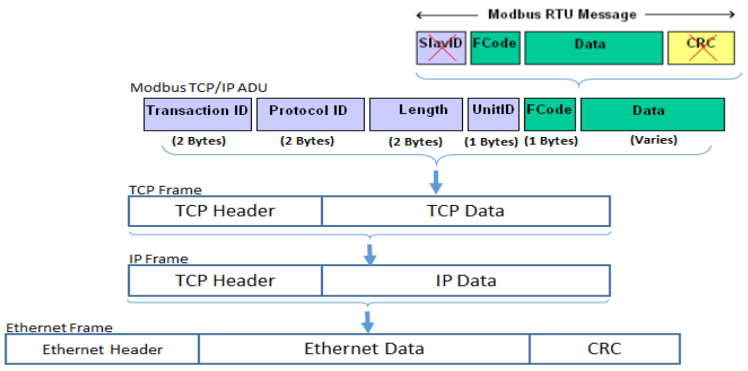


Figure 2.11: Modbus RTU over TCP

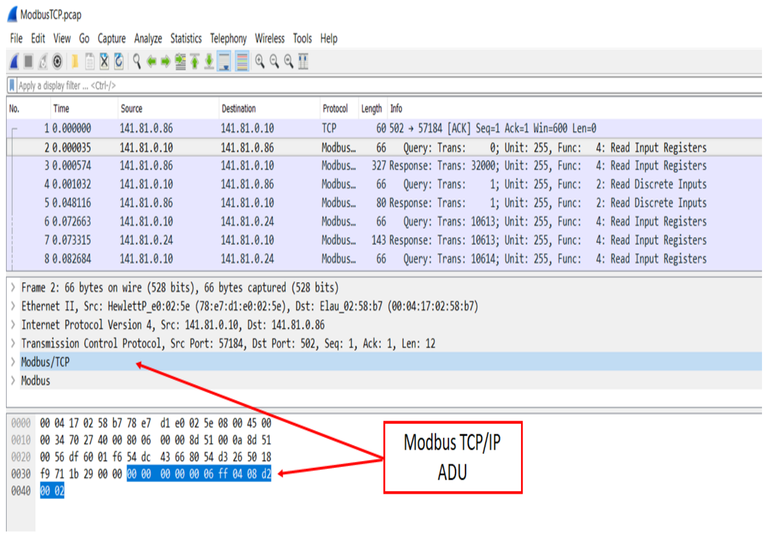
Knowledge of how the Modbus messages are organized helps in analyzing their content. For example, if the captured packet *Modbus TCP/IP ADU*= 0001 0000 0006 11 03 006$B$ 0003, then: *Transaction ID*= 0001, *Protocol ID*= 0000, *Message Length*= 0006 (6 bytes to follow), *Unit ID*= 11, *Function Code*= 03, *Unit ID*= 11, *Data Address of the first register requested*= 006$B$, and *total number of registers requested*= 0003. Such knowledge helps identify the aspects of the Modbus-data that are not altered by router actions, for the purposes of device fingerprinting.

### 2.3.1 Modbus-data based fingerprinting

A first step is to extract the *Modbus TCP/IP ADU* from a TCP/IP packet. Since we know how the Modbus packet is embedded inside a TCP/IP packet, once we have captured a packet, say using Wireshark, we can locate the Modbus packet and extract it. Figure 2.12 shows a captured Modbus TCP/IP packet example.

Following the Modbus-packet extraction, a next step is to identify the features withing it to be able to use towards device fingerprinting. Our study of the Modbus protocol and literature revealed that, while the range of registers that the protocol can read from or write to is predefined in the protocol standards, upon studying and analyzing the Modbus-

TCP datasheets from different SCADA/ICS vendors such as Siemens, Schneider, GE, ABB, Wago, MegaTec, and Eaton Corporation we found that the register addresses that are read/written by Modbus are different for different manufacturers, as was also noted in [70]. In practice, each I/O PLC module is mapped to a predefined specific range of registers (by the manufacturer), e.g., GE Genius analog output registers are mapped from 401501 to 401900 [71], and Schneider ION7500/ION7600 analog output registers are mapped from 40011 to 40277 [72]. In Siemens s7 PLCs the user can optionally select a start and an end address, and the programming portal configures the devices to fit those selected addresses to registers between 40001 and 49999 (note the user selected addresses will not be the same as the addresses of the actual registers; refer to [73] for more details). Since the programming portal automatically links the SCADA devices to the Modbus registers, it adds an offset between two consecutive devices, e.g., if the first device mapping begins at 40001 and ends at 40100, the second device mapping will start at 40150 and ends at 40250. Since each device behaves differently and uniquely based on its manufacturer's setting, a device fingerprint can be created for the unknown SCADA devices by monitoring the registers used by Modbus protocol, then comparing those to devices' reference database. We utilize this principle in fingerprinting of devices connected to SCADA/ICS networks employing Modbus-protocol.


ModbusTCP.pcap
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Modbus TCP/IP ADU

```
No.  Time       Source       Destination  Protocol  Length Info
   1 0.000000   141.81.0.86  141.81.0.10  TCP        60 502 → 57184 [ACK] Seq=1 Ack=1 Win=600 Len=0
   2 0.000035   141.81.0.10  141.81.0.86  Modbus…    66    Query: Trans:     0; Unit: 255, Func:   4: Read Input Registers
   3 0.000574   141.81.0.86  141.81.0.10  Modbus…   327 Response: Trans: 32000; Unit: 255, Func:   4: Read Input Registers
   4 0.001032   141.81.0.10  141.81.0.86  Modbus…    66    Query: Trans:     1; Unit: 255, Func:   2: Read Discrete Inputs
   5 0.048116   141.81.0.86  141.81.0.10  Modbus…    80 Response: Trans:     1; Unit: 255, Func:   2: Read Discrete Inputs
   6 0.072663   141.81.0.10  141.81.0.24  Modbus…    66    Query: Trans: 10613; Unit: 255, Func:   4: Read Input Registers
   7 0.073315   141.81.0.24  141.81.0.10  Modbus…   143 Response: Trans: 10613; Unit: 255, Func:   4: Read Input Registers
   8 0.082684   141.81.0.10  141.81.0.24  Modbus…    66    Query: Trans: 10614; Unit: 255, Func:   4: Read Input Registers

> Frame 2: 66 bytes on wire (528 bits), 66 bytes captured (528 bits)
> Ethernet II, Src: HewlettP_e0:02:5e (78:e7:d1:e0:02:5e), Dst: Elau_02:58:b7 (00:04:17:02:58:b7)
> Internet Protocol Version 4, Src: 141.81.0.10, Dst: 141.81.0.86
> Transmission Control Protocol, Src Port: 57184, Dst Port: 502, Seq: 1, Ack: 1, Len: 12
> Modbus/TCP
> Modbus

0000  00 04 17 02 58 b7 78 e7  d1 e0 02 5e 08 00 45 00
0010  00 34 70 27 40 00 80 06  00 00 8d 51 00 0a 8d 51
0020  00 56 df 60 01 f6 54 dc  43 66 80 54 d3 26 50 18
0030  f9 71 1b 29 00 00 00 00  00 00 00 06 ff 04 08 d2
0040  00 02
```

Figure 2.12: Modbus TCP/IP packet example

Figure 2.13 depicts our proposed Modbus-data analysis based device fingerprinting approach, consisting of two stages: The first is an offline learning stage in which information from SCADA devices manufacturer datasheets are used to create a reference fingerprint database. The second stage is the online recognition stage that captures communication packets from the network at each of the routers, extracts the relevant Modbus registers information to create the fingerprints for the unknown devices, and finally compares the created fingerprints to the fingerprints' reference database, obtained during the learning stage, to identify the devices.

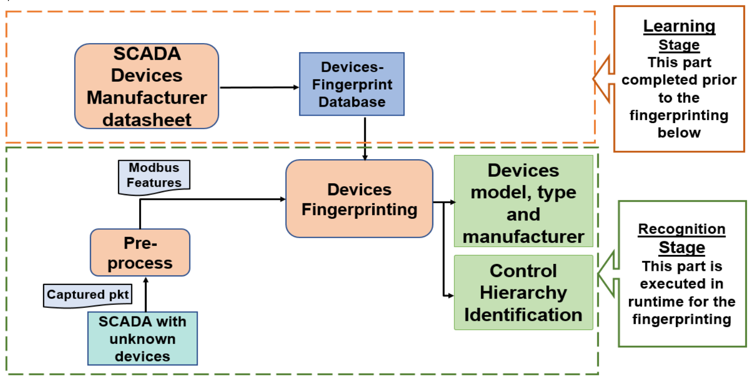

SCADA Devices Manufacturer datasheet
Devices-Fingerprint Database
Learning Stage
This part completed prior to the fingerprinting below
Modbus Features
Devices Fingerprinting
Devices model, type and manufacturer
Pre-process
Captured pkt
Control Hierarchy Identification
SCADA with unknown devices
Recognition Stage
This part is executed in runtime for the fingerprinting

Figure 2.13: Modbus-data analysis based devices recognition

The above analysis can be augmented with control hierarchy information for a further refinement/confirmation of fingerprinting. The Modbus protocol is a master-slave protocol in which the master device sends requests to slave devices. Modbus exchanged messages incorporate this information by specifying whether a device is a master or a slave. Thereby, monitoring such a feature allows for the identification of control hierarchy. For example, considering Figure 2.5 and monitoring the Modbus messages, indicates that $\{s_1, s_2, s_3\}$ are slave PLCs for both $s_6$ and $s_7$, $\{s_6, s_7, s_8\}$ are slave PLCs for $s_9$, and $\{s_4, s_5\}$ are slave PLCs for $s_8$. Using this information, the devices are mapped into their control hierarchy in a simple manner.

### 2.3.2 A CPS Case Study: S7-300 PLC SCADA

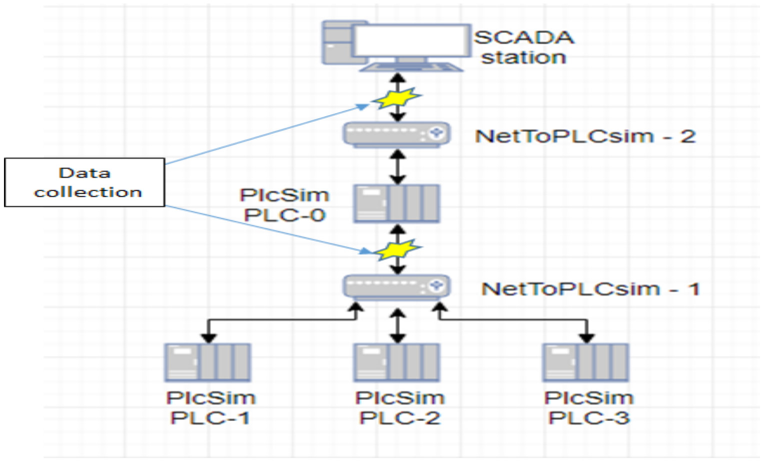


Figure 2.14: A simulated SCADA water treatment system

To validate the Modbus-data analysis based fingerprinting, we implemented a simulated Siemens s7-300 SCADA system. To guarantee that the simulated system matches a real-world one, we constructed it using “SIMATIC S7-PLCSIM” [74] that allows users to create virtual Siemens PLCs to perform a comprehensive simulation (that is used by developers to validate their design prior to commissioning). S7-PLCSIM simulates a single PLC at a time, and so we also used “NetToPLCsim” [75] that allows network communication with the S7-

PLCSIM based virtual PLC, using the network interface of the PC on which the S7-PLCSIM is running. Using these tools, allowed us to create a realistic simulated control system that generates real data. We created a simulated system with the control architecture shown in Figure 2.14, that consists of 4 s7-300 PLCs, one master PLC, three slave PLCs, and one control room station. "NetToPLCsim" mimic routers to simulate the network.

We implemented the proposed Modbus-data analysis based fingerprinting approach in a Python tool, and tested it on the collected data. The tool successfully identified the PLCs as Siemens s7-300 PLCs, and also successfully identified the system control hierarchy. The collection points were placed at the routers (as opposed to at devices), validating that the Modbus-data analysis based fingerprinting approach worked with a minimal number of data collection points (2 as opposed to 4).

### 2.3.3 Discussion

The experimental results of the proposed Modbus-data analysis based device recognition confirmed that it could successfully identify the SCADA system control hierarchy and system devices type, manufacturer, and model. On a standard computer, Core i5/2.2GHz/4GB RAM running Win 10, our algorithm took 12 seconds to identify the devices of the case study.

The fact that, the proposed Modbus-data analysis based fingerprinting approach is not affected by router actions on data, reduced the number of data collection points needed to identify unknown devices. For example, in the simulated water treatment SCADA system, only two collection points were needed, while 4 data collection points exists at the 4 devices. While the Modbus-protocol is one of the most commonly used protocols in SCADA systems, there are real-world examples, like the water treatment CPS from iTrust, that do not employ Modbus. In those cases, our earlier hybrid approach works, except additional effort is need in data collection at all the devices (as opposed to at all the routers).

### 2.3.4 Automated Network Device and Vulnerability Identification (ANDVI) Tool

Once the SCADA/ICS devices have been identified, their potential known vulnerabilities can also be automatically mapped out by a look-up to trusted and up-to-date databases. The two most trusted vulnerabilities databases are the Common Vulnerabilities and Exposures (CVE) database and the National Vulnerability Database (NVD). CVE is a database of known security threats, sponsored by the United States Department of Homeland Security. NVD is a U.S. government repository of standards-based vulnerabilities that include databases of security checklist references, security-related software flaws, misconfigurations, product names, and impact metrics. The information provided by these two databases can be integrated with our devices recognition algorithms to automatically identify the system devices and their associated known vulnerabilities. To do this end, our ANDVI tool (its architecture shown in Figure 2.15) takes the output from either of the two device identification approaches, and matches the discovered devices to their known vulnerabilities listed in the CVE database, and then uses the NVD database to rank those vulnerabilities according to their NVD threat index values.

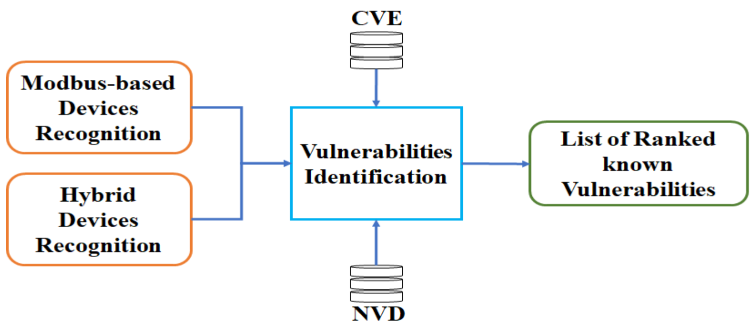

CVE
Modbus-based Devices Recognition
Hybrid Devices Recognition
Vulnerabilities Identification
List of Ranked known Vulnerabilities
NVD

Figure 2.15: Automated Network Device and Vulnerability Identification (ANDVI) framework

# CHAPTER 3. A2G2V: AUTOMATIC ATTACK GRAPH GENERATION AND VISUALIZATION AND ITS APPLICATIONS TO COMPUTER AND SCADA NETWORKS

Attack-graphs are graphical data structures representing all attack scenarios, in the form of sequences of atomic attacks, that an attacker can pursue to compromise a system [76]. Once an attack-graph has been generated for a system, it can be used in understanding the complex system-level vulnerabilities derivable from the atomic component level ones, and thereby in system-level security risk assessment. Note vulnerability assessment is an important research field of its own, and the interested readers may refer to [77, 78, 79] for details. Traditionally, attack-graphs are generated manually by a team of experts that analyze the system, but this is tedious and time-consuming, and subject to being error-prone and incomplete. Therefore, an approach that is based on a complete and accurate model and that is automated in generating the attack graph is highly desired.

The existing research on attack-graph generation is "non-model-based" and hence cannot be automated (see Section 1.3); the only exception being [45]. Similar to the work in [45], our attack graph generation algorithm is model-based, i.e., it utilizes *a formal description of the system model (describing architecture and connectivity, components and behaviors, assets, defenses, vulnerabilities, and atomic attacks), as well as of the system-level security properties of interest, along with defining the associated state-space of interest and specifying how that evolves when atomic-attack actions of interest occur, to enable the automatic computing the associated attack-graph.* The limitation of [45] is that it is restricted to those security properties whose violation corresponds to the reachability of unsafe locations in the model. Our approach enables the generation of an attack-graph for *any* security property of interest, including those that may be violated by executing an acyclic path, rather only by reaching a location.

We present an augmentation of a model-checker for state-space exploration to find *all* those acyclic counterexample-paths where the given specified security properties are violated. A model-checker in general only produces a *single* counterexample, and if executed more than once, it may reproduce the same counterexample. Our algorithm ensures that a distinct counterexample is generated each time a model-checker is called. To ensure that the same counterexample is not repeated, our A2G2V implementation iteratively *relaxes* the existing specification by disjuncting it with the encoding of the latest counterexample [80, 81]. This ensures that the latest counterexample ends up satisfying the relaxed specification and can no longer act as a counterexample.

Our C-program based A2G2V tool takes as input the system architecture and the atomic attack behaviors as captured in Architecture Analysis and Design Language (AADL) [82] and its AGREE Annex. The AADL description is used to formally represent the underlying networked-system, whereas its atomic vulnerabilities, their pre- and post- conditions, and security property of interest using the AGREE [83] Annex. The AADL+AGREE model is then translates into its Lustre equivalent, to enable further analysis using the model-checker JKind [84]. The A2G2V tool calls the JKind model-checker iteratively, each time with a newly *relaxed* specification, to generate a new attack sequence in each iteration in the form of a new counterexample. Further, the Graphviz tool has also been integrated into A2G2V for the visualization of the generated attack graph (see Figure 3.3 in Section 3.2).

The proposed algorithm was tested on two computer network examples and one CPS example: A three hosts network from [45, 47], an extended five hosts network, and a water treatment CPS. For the second example, we consider a more general security property which involves the reachability of a certain path rather than just a location. (Such a case for example cannot be handled in the prior works [45, 47].)

The main contributions of the work presented here are:

1. A model-based automated attack graph generator by way of automated generation of all acyclic counterexamples of a model; this requires relaxing the specification with the encoding of the earlier counterexamples so distinct counterexamples are found in each iteration.

2. The approach supports general security properties and not just those represented as reachability of certain unsafe states.

3. C-program based implementation A2G2V of our algorithm that interfaces with AGREE (for AADL and Annex based modeling and its Lustre translation), iteratively with JKind (for generation of all acyclic counterexamples), and further integration with Graphviz for visualization of the set of all attack sequences in a graph form.

4. A water treatment CPS and two IT network examples for illustrating the above steps.

## 3.1 Computer Network Illustrations and their Model Formulation

As a concrete example to illustrate the problem of attack-graph generation, we adapt the networked-system example from [47, 45] shown in Figure 3.1.

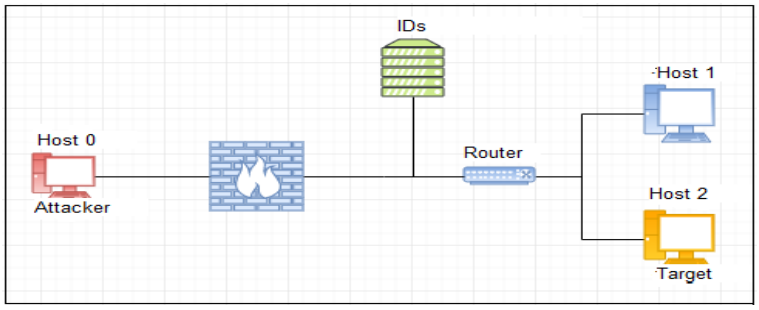


Figure 3.1: Three Hosts Network Example.

The network has three hosts: host-0, where the attacker is located, whereas host-1 and host-2 are the two target hosts. Host-1 runs ftp and sshd, while host-2 is running database and ftp. Also, there exists a firewall separating the targets from the rest of the internet. An IDS (intrusion detection system) supervises the network traffic between the network hosts and the outside sources. The firewall does not place any access control restrictions on the flow of network traffic, rather lets the IDS monitor the traffic flow between (host-0; host-1) and (host-0; host-2), but not between (host-1; host-2). For an atomic-attack (see their list below) that is detectable, the IDS triggers an alarm upon its detection, while a stealthy attack remains undetected. Owing to the aforementioned services running on the two hosts, there exists these three vulnerabilities:

1. *wdir*: a writable ftp home directory; this vulnerability is exploitable by an ftp rhosts attack on hosts 1 and 2, running the ftp service.

2. *fshell*: an executable command shell assigned to the ftp user name; this vulnerability is also exploitable by an ftp rhosts attack on hosts 1 and 2, running the ftp service.

3. *xterm*: the xterm is vulnerable to a buffer overflow attack on host 1 running the ftp and sshd services, and host 2 running ftp service.

The aforementioned vulnerabilities can be exploited resulting in the following atomic attacks:

1. sshd buffer overflow (*sbo*): This attack exploits the xterm vulnerability, giving the attacker root access to the victim host. This attack can be either stealthy or detectable by the IDS.

2. ftp rhosts (*ftpr*): This attack uses wdir vulnerability or the fshell vulnerability where the attacker establishes a remote login trust by creating a .rhosts file in the ftp home directory. This is an stealthy attack that the IDS is unable to detect.

3. remote login (*rlog*): This attack uses an existing trust relationship between two hosts gained by executing ftpr attack. The attacker logs into one host from another and gets user access even without a password. This attack is detectable by the IDS.

4. local buffer overflow (*lbo*): This attack also exploits the xterm vulnerability to gain access to the setuid root file to give the attacker root access to a local host. This attack is stealthy, and not detectable by the IDS.

To illustrate the capability of our approach in generating attack-graphs for complex and general security properties such as $AG(p \rightarrow AFq)$, we expanded the above example to a 5-host system of Figure 3.2. Here host-3 and host-4 are additional target hosts. These two hosts are running ftp and sshd services as in the case of host-1, and hence have the same vulnerabilities as host-1. As a result, this system then has eight possible atomic attacks. As before, the firewall does not place any access control restrictions on the flow of network traffic rather only on monitoring: the traffic flows between (host-0; host-1), (host-0; host-2), (host-0; host-3), and (host-0; host-4) are monitored by the IDS, while those among (host-1; host-2; host-3; host-4) are not monitored.

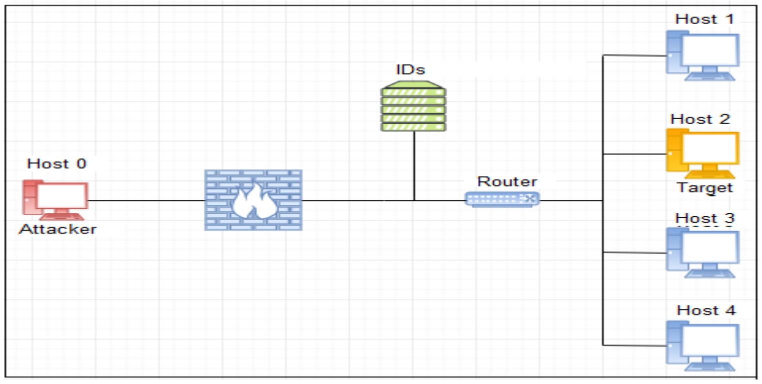


Figure 3.2: Five Hosts Network Example.

### 3.1.1 Model Formulations of Computer Networks

Following presents the formal description of the three host system of Figure 1.

1. Set of hosts $H = 0, 1, 2$; variable $i \in \{0, 1, 2\}$ (static parameters).

2. System connectivity, $C \subseteq H \times H$ ; Boolean $c_{ij} = 1$ iff host $i$ connected to host $j$ (static parameters).

3. System services $S$; Boolean $s_i = 1$ iff service $s \in \{ftp, sshd, data\}$ is running on host $i$ (dynamic variables).

4. System vulnerabilities $V$ ; Boolean $v_i = 1$ iff vulnerability $v \in \{dir, fshell, xtermg\}$ exists on host $i$ (static parameters).

5. Attack instances $A_I \subseteq A \times H \times H$; labeled $a_{ij} \equiv$ attack $a$ from source $i$ to target $j$, $a \in \{sbo, ftpr, rlog, lbog\}$, $sbo :=$ $sshd$ buffer overflow, $ftpr := ftp\_rhosts$, $rlog :=$ remote login, $lbo :=$ local buffer overflow (static parameters).

6. Trust relation $T \subseteq H \times H$ ; Boolean $t_{ij} = 1$ iff $i$ is trusted by $j$ (dynamic variables)

7. Attacker level of privilege $L$ on host $i$; variable $l_i \in \{none, user, root\}$ (dynamic variables).

8. Intrusion detection system $IDS : A \times H \times H \to \{0, 1\}$; Boolean $ids(a_{ij}) = 1$ iff attack $a$ from source $i$ to target $j$ is detectable (static parameters).

9. A global Boolean $d_g$ tracks whether an $IDS$ alarm has been triggered for any previously executed atomic attack (dynamic variable).

10. Attack pre-conditions:

    - $Pre_{(sbo_{ij})} \equiv c_{ij} = 1 \wedge (l_i \geqslant user) \wedge (l_j < root) \wedge (sshd_j = 1)$
    - $Pre_{(ftpr_{ij})} \equiv c_{ij} = 1 \wedge (l_i \geqslant user) \wedge (\exists k \in H : t_{kj} = 0) \wedge (ftp_j = 1) \wedge (wdir_j = 1 \wedge fshell_j = 1)$
    - $Pre_{(rlog_{ij})} \equiv c_{ij} = 1 \wedge (l_i \geqslant user) \wedge (l_j = none) \wedge (t_{ij} = 1)$
    - $Pre_{(lbo_{ij})} \equiv c_{ij} = 1 \wedge (l_j = user) \wedge (xterm_j = 1)$

11. Attack post-conditions:

    - $post_{(sbo_{ij})} \equiv (l_j = root) \wedge (sshd_j = 0) \wedge ((i = d_g = 0) \Rightarrow (d_g = 0)(d_g = 1))$
    - $post_{(ftpr_{ij})} \equiv (\forall k \in H : t_{kj} = 1)$

- $post_{(rlog_{ij})} \equiv (l_j = user) \wedge (i = 0 \Rightarrow d_g = 1)$
- $post_{(lbo_{ij})} \equiv (l_j = root)$

12. Intial states : $l_0 = root \wedge (l_1 = l_2 = none) \wedge (\forall ij \in H \times H : t_{ij} = 0) \wedge (ftp_1 = ftp_2 = sshd_1 = data_2 = 1) \wedge d_g = 0$
(Initially, the attacker has root privilege on host-0 and no privilege on other hosts, none of the hosts trust each other, ftp is running on host-1 and host-2, sshd on host-1, database on host-2, and IDS has not detected security violation.)

13. The security property $\phi$ is violated if the attacker has privilege level below root on host-2 and it remains undetected by the IDS. This can then be described by a Computational Tree Logic (CTL) formula:

$$\neg\phi \equiv AG(l_2 = root \wedge d_g = 0).$$

Here “AG” stands for “All paths Globally at all states”. The property reads as, “for all paths, globally at all states, either attacker’s privilege level on host-2 is below root, or this gets detected”. The 5-host system’s formal description is an expansion of the 3-host system’s formal description; host-1, host-3, and host-4 are equivalent to host-1 in the 3-host system, so they retain similar formal descriptions. Host-0, host-2, and the IDS maintain the same functionality as the 3-host system, so they retain the same formal description as in the 3-host system. Therefore, the above formal description, with the slight modification to include host-3 and host-4, applies to the 5-host system. For the 5-host system, the security property is violated if the attacker is able to get root access on host-2 only after compromising host-3, and while remaining undetected, i.e.,

$$\neg\phi \equiv AG((l_3 \geq 1) \rightarrow AF(l_2 = root \wedge d_g = 0)).$$

Here ”AG” stands for ”All paths Globally at all states” and ”AF” stands for ”All paths in a Future state”. The property reads as, ”for all paths, globally at all states, if attacker’s privilege level on host-3 is root, then for all paths (originating from that state) in a future state, attacker privilege level on host-2 is root, and this remains undetected”.

## 3.2 Model-based Attack Graph Formulation, Generation, and Illustration

The model-based approach to attack graph generation requires a formal security model of the overall system, and a system-level security property of interest. Using these, the associated attack-paths and the attack-graph is obtained in automated fashion using our algorithm. To begin, we formally define a system security model, an attack path, and an attack graph.

**Definition 1** *A system security model* ($M = (S, E, s_0)$) *is a state-transition diagram whose locations* $S$*, with* $s_0 \in S$ *denoting an initial location, capture the security status of the system (e.g., which host is running which service, what the attacker's access privilege level is at each host, what the trust level is among each pair of hosts, and what the detection status is of security violation), and whose transitions* $E$ *describe how the atomic attack actions cause an update in the system security status. In general, the transitions are guarded by pre-conditions on state-variables, and their execution entails certain post-conditions on the same state-variables.*

To construct the system security model one needs a complete system description; its components and connectivity; their assets, services, and vulnerabilities; the atomic attack actions that exploit those vulnerabilities; their pre- and post- conditions; and also their detectability by an IDS. The execution of a sequence of state transitions (namely, atomic attacks) in $M$ can result in the violation of a security property $\phi$, expressed in a suitable temporal logic.

**Definition 2** *Given a security model* $M$ *and a security property* $\phi$ *of interest, an Attack Path* ($AP$) *is a finite acyclic path of a sequence of states in* $M$*,* $AP = (s_0, s_1 \cdots s_f)$*, where* $s_0$ *is an initial state in* $M$*, while any two adjacent states in the path belong to the transition set* $E$*, such that the execution of* $AP$ *leads to the violation of* $\phi$*.*

**Definition 3** *An Attack-Graph* ($AG$) *is a data structure representing a union of all attack paths*

$$AG = \{AP_i | \ AP_i \text{ is an attack path in } M\}.$$

### 3.2.1 Automated Attack Graph Generation and Visualization

Our A2G2V (Automated Attack Graph Generation and Visualization) approach is based on extension of model-checking. Model-checking is a tool for verifying if a system model $M$ fulfills a certain temporal logic property $\phi$ of interest. If the property is not satisfied by some runs of $M$, then a model-checker reports an offending run as a counterexample.

**Algorithm 2** A2G2V: Automated Attack Graph Generator and Visualizer

```
Input: AADL+AGREE model ($M$)
Output: Attack Graph ($AG$)
procedure
    Translate AADL Model ($M$) to Lustre Model ($Ml$) % Using AGREE ; set $\phi_0 = \phi$
    Generate attack path $AP$ % Using JKind find the counterexample that violate $\phi$
    loop 1
    if $AP$ is Cyclic then
        $K = K - 1$
        generate $AP$
        goto loop 1
    if $AP$ is Acyclic then
        $K = |AP|$ % $|\cdot|$ computes the length of its argument
        loop 2
        generate $AP_i$
        if new attack-path then % if all APs of length $\leqslant K$ are not yet enumerated
            $\phi_{i+1} = \phi_i \vee AP_i$
            Add $AP_i$ to $AG$
            goto loop 2
        else % if all APs of length $\leqslant K$ have been enumerated
            $K = K + 1$
            generate $AP$
            if $AP$ is Acyclic then
                goto loop 2
            else
                return $AG$
    generate graphical $AG$ using Graphviz
```

In our work, the system description of components and their interfaces and connections is first captured using Architecture Analysis and Design Language (AADL) [82], within the open-source environment (Osate2) [85]. Osate2 is an upgraded Eclipse-based platform, an open-source Integrated Development Environment (IDE). Osate2 extends the Eclipse platform with textual and graphical editors for AADL, and supports Extensible Markup Language (XML) based interchange format for AADL, based on its metamodel specification. Within an AADL architecture level model, the system components can be added graphically, and also extensions can be introduced through Annexes, that can offer new categories of model elements such as behaviors and properties of the components. In our case, we use the AGREE Annex developed by Rockwell [83] to specify the component models, and system-level security properties.

AGREE also translates the AADL+Annex models and properties to Lustre, and interacts with JKind (a model-checker for Lustre models) for correctness verification. AGREE can only be run from the OSATE2 GUI and does not support command line instructions, and cannot be called iteratively for the generation of more than one counterexamples, without possibly redesigning the AGREE's source code. So we developed our own C-based implementation, A2G2V, that takes the Lustre output from AGREE and iteratively interacts with JKind to generate the attack-sequences one after another. Figure 3.3 shows the proposed algorithm architecture.

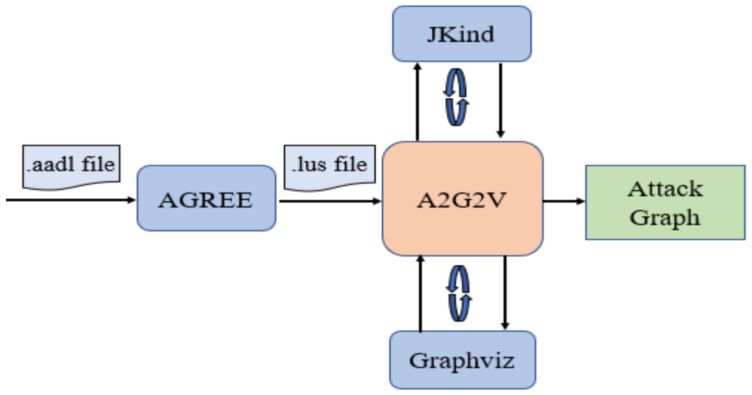


Figure 3.3: A2G2V Algorithm Architecture.

The approach behind our main algorithm A2G2V [Algorithm 2] is as follows. User inputs the system description in AADL+Annex using the AGREE front end. This is translated into Lustre by AGREE and fed to A2G2V in form of a security model $M$ and system-level security property $\phi$. If $M$ complies with the given security property $\phi$, JKind reports that $\phi$ is satisfied. If $M$ violates $\phi$, then JKind reports a counterexample representing a state transition path of concatenated atomic attacks (namely, the transitions in $E$) that the system follows to violate $\phi$. Note once an attacker has compromised the system through a sequence of atomic attacks, it need not launch additional atomic attacks. Thus, without loss of generality, all attack-paths are acyclic. Since we are only interested in acyclic counterexample paths, the length of an attack-path is upper-bounded by the ”depth” (i.e., the length of the maximum acyclic path) of system model $M$.

This upper-bound however is not known a priori, and so requires an iterative exploration. In our case, JKind uses Bounded Model Checking (BMC) for verification: It searches for counterexamples is of length bounded to a search-depth, with a default initial value of $K = 200$. Note this default value *does not* affect the end result: If a smaller length counterexample exists or the system is shown to be safe using the default induction depth of $K = 200$, then $K = 200$ is already a workable choice. If $K = 200$ induction-depth is inconclusive, then

JKind automatically increases $K$ incrementally until it finds a counterexample or concludes that the system is safe.

From JKind's first run, if no counterexample is found, then the attack-graph is simply an empty graph. On the other hand, if the first reported $AP$ is cyclic, then A2G2V decreases $K$ by 1 and re-runs the model-checker JKind. It repeats this step until finding an acyclic $AP$. (Note this task of finding an acyclic path by iteratively decrementing the search-depth can be skipped if an acyclic path is found in the first iteration.) At this point, A2G2V sets $k$ to be the length of that $AP$. Then it enumerates all the attack-paths of length $\leqslant K$ by iteratively disjuncting the reported $AP_i$ to the security property as follows: $\phi_0 := \phi$; $\phi_{i+1} := \phi_i \vee AP_i$, and then re-runs the model-checker JKind. After enumerating all attack-paths of length $\leqslant K$, JKind increments the search-depth $K$ by 1 and re-runs the algorithm. This process is repeated until reaching a $K$ that reports a cyclic $AP$, meaning that the model-depth has been exceeded. (Note this task of finding a cyclic path by iteratively incrementing the search-depth can be skipped if a cyclic path is found in the first iteration.) The next step is to pass the union of the reported acyclic attack-paths to Graphviz to generate a graphical visualization of the corresponding attack-graph.

Note the *soundness* of Algorithm 1 follows from the fact that the algorithm generates *only* the counterexamples, whereas its *completeness* follows from the fact that it generates *all* the acyclic counterexamples, which is a finite set since the model itself has a finite state-space, with finite number of dynamic state-variables, each taking finitely many values. Also as a result, the termination is guaranteed since the search is only over the acyclic paths, which is a finite set. Implementing the A2G2V tool required building three main functions: a counterexample parsing function, a cyclic testing function, and a Luster model editing function. The counterexample parsing function extracts the generated counterexamples from the JKind output. The cyclic testing function tests if the generated counterexample is cyclic or acyclic. The model editing function transfers the counterexample into its luster representation, by first encoding it in the Luster syntax, and next disjuncting the encoded counterexample with the property being model-checked. The main program coordinates among the three functions to execute the algorithm described above.

The number of pre-/post-conditions is linear in the number of atomic attacks and the dynamic state-variables. Also, in general, the computation complexity is determined by the size of the model and the length of the property. For the class of so called CTL (Computational Tree Logic) properties, the complexity is known to be polynomial in the size of the model and the length of the security property [86]. The model-checking has gained tremendous industrial adoption for safety critical applications, owing to its applicability to practical sized-systems.

### 3.2.2 Computer Network based Illustration of A2G2V

The computer network examples presented in Section 3.1.1 were first encoded in AADL+AGREE Annex. The AADL description formally captured the architectural description of the computer networked systems: their components and their connectivitiy. The information about their behaviors: their dynamic state variables, the pre- and post-conditoins of the atomic attacks, and the security properties of interest were encoded in the AGREE Annex. These were translated to Lustre using AGREE to obtain the system security model $M$, which was then fed to our A2G2V tool for the atack-graph generation and visualization.

For the 3-host system of Figure 3.3, the model $M$ consists of a total of $57 \times 10^6$ states and $3.2 \times 10^{15}$ transitions, whereas the 5-host system of Figure 3.2, there are $1.4 \times 10^{17}$ states and $1.8 \times 10^{34}$ transitions in its model $M$.

For the 3-host system, the goal of the attacker is to gain access to the database on host-2, for which the attacker needs to gain root access to host-2 without being detected by the IDS. Hence, the property $\phi$ that should not be violated is that either the attacker never gains root access to host-2 or it gets be detected by the IDS.

Table 3.1 summarizes the A2G2V algorithm search results for enumerating the attack-paths, and the search depth (or path-length), denoted $K$, in the 3-host network. In the first iteration, the A2G2V set $K$ to 200, and found an acyclic 4 steps attack-path. A2G2V used $K = 4$ and iterated until no new 4 steps attack-path could be found, resulting in 10 different attack-paths. Next, $K$ was incremented to 5 and all attack-paths of length 5 were generated (57 new attack-path were found). Finally, A2G2V incremented $K$ to 6, but that resulted in a cyclic attack-path. A2G2V then terminated the attack-path search.

Table 3.1: 3-hosts Network Attack Graph Search Summary

| $K$ | Generated paths information | Next $K$ |
|---|---|---|
| 200 (default) | A 4-step acyclic attack-path | $K$=4 |
| 4 | 10 acyclic attack-paths | $K$=5 |
| 5 | 57 acyclic attack-paths | $K$=6 |
| 6 | A cyclic attack-path generated | Terminate A2G2V |

Note due to a feature of the model-checker when searching for a counterxample of a certain length, the lower length counterexamples are also explored, and this causes the model-checker to duplicate the counterexamples (so for example when searching for $K = 5$, a $K = 4$ counterexample is generated for a second time). Table 3.1 reports all such results. However, our graph visualizer parses all attack-paths and automatically removes the duplicates. This can be seen from the generated attack graphs (Figures 3.4, 3.6 and 3.8).

Figure 3.4 depicts the attack-graph generated by the A2G2V algorithm for this property. Each node describes the attacker privileges and the trust relationships between the network hosts at that step. Each transition represents an atomic-attack.

Any path from the initial node to the final node represents a sequence of atomic-attacks that the attacker can use to attain its goal without being detected. For example, the path highlighted in the red has the following atomic-attacks: ftp-02 establishes trust between host-0 and host-2, sbo-01 gives the attacker root access to host-1, rlog-12 gives the attacker user access to host-2, lbo-02 gives the attacker root access to host-2, and finally ftp-01

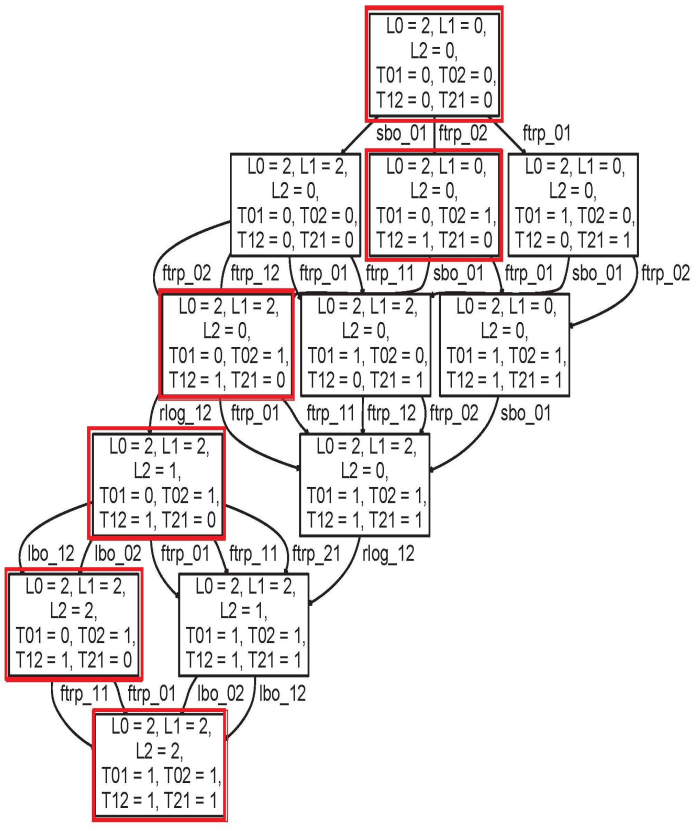

L0 = 2, L1 = 0, L2 = 0, T01 = 0, T02 = 0, T12 = 0, T21 = 0
sbo_01
ftrp_02
ftrp_01
L0 = 2, L1 = 2, L2 = 0, T01 = 0, T02 = 0, T12 = 0, T21 = 0
L0 = 2, L1 = 0, L2 = 0, T01 = 0, T02 = 1, T12 = 1, T21 = 0
L0 = 2, L1 = 0, L2 = 0, T01 = 1, T02 = 0, T12 = 0, T21 = 1
ftrp_02
ftrp_12
ftrp_01
ftrp_11
sbo_01
ftrp_01
sbo_01
ftrp_02
L0 = 2, L1 = 2, L2 = 0, T01 = 0, T02 = 1, T12 = 1, T21 = 0
L0 = 2, L1 = 2, L2 = 0, T01 = 1, T02 = 0, T12 = 0, T21 = 1
L0 = 2, L1 = 0, L2 = 0, T01 = 1, T02 = 1, T12 = 1, T21 = 1
rlog_12
ftrp_01
ftrp_11
ftrp_12
ftrp_02
sbo_01
L0 = 2, L1 = 2, L2 = 1, T01 = 0, T02 = 1, T12 = 1, T21 = 0
L0 = 2, L1 = 2, L2 = 0, T01 = 1, T02 = 1, T12 = 1, T21 = 1
lbo_12
lbo_02
ftrp_01
ftrp_11
ftrp_21
rlog_12
L0 = 2, L1 = 2, L2 = 2, T01 = 0, T02 = 1, T12 = 1, T21 = 0
L0 = 2, L1 = 2, L2 = 1, T01 = 1, T02 = 1, T12 = 1, T21 = 1
ftrp_11
ftrp_01
lbo_02
lbo_12
L0 = 2, L1 = 2, L2 = 2, T01 = 1, T02 = 1, T12 = 1, T21 = 1

Figure 3.4: A2G2V generated attack-graph for 3-hosts network

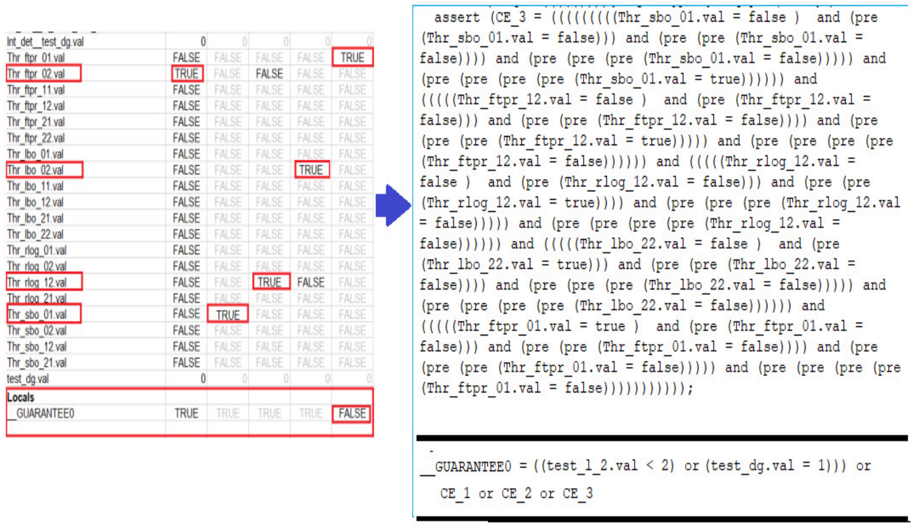
establishes trust between host-1 and host-2. Figure 3.5 shows the JKind reported counterexample for this path, and how the A2G2V algorithm encodes it and disjuncts it with the security property within the Lustre file automatically.

We also applied A2G2V to the five hosts system, with a more general security property that we introduce earlier, and is being repeated here:

$$\neg\phi = AG((l_3 \geq 1) \rightarrow AF(l_2 = root \wedge d_g = 0)).$$

This property represents all attack scenarios that an attacker may take to compromise host-3 first, and later compromise host-2, without being detected.

The A2G2V algorithm started at default $K$ value of 200. The first iteration generated an acyclic 4 steps attack-path. So $K$ was set to 4 and A2G2V generated all attack-paths



| | | | | | |
|---|---|---|---|---|---|
| Int_det__test_dg.val | 0 | 0 | 0 | 0 | 0 |
| Thr_ftpr_01.val | FALSE | FALSE | FALSE | FALSE | TRUE |
| Thr_ftpr_02.val | TRUE | FALSE | FALSE | FALSE | FALSE |
| Thr_ftpr_11.val | FALSE | FALSE | FALSE | FALSE | FALSE |
| Thr_ftpr_12.val | FALSE | FALSE | FALSE | FALSE | FALSE |
| Thr_ftpr_21.val | FALSE | FALSE | FALSE | FALSE | FALSE |
| Thr_ftpr_22.val | FALSE | FALSE | FALSE | FALSE | FALSE |
| Thr_lbo_01.val | FALSE | FALSE | FALSE | FALSE | FALSE |
| Thr_lbo_02.val | FALSE | FALSE | FALSE | TRUE | FALSE |
| Thr_lbo_11.val | FALSE | FALSE | FALSE | FALSE | FALSE |
| Thr_lbo_12.val | FALSE | FALSE | FALSE | FALSE | FALSE |
| Thr_lbo_21.val | FALSE | FALSE | FALSE | FALSE | FALSE |
| Thr_lbo_22.val | FALSE | FALSE | FALSE | FALSE | FALSE |
| Thr_rlog_01.val | FALSE | FALSE | FALSE | FALSE | FALSE |
| Thr_rlog_02.val | FALSE | FALSE | FALSE | FALSE | FALSE |
| Thr_rlog_12.val | FALSE | FALSE | TRUE | FALSE | FALSE |
| Thr_rlog_21.val | FALSE | FALSE | FALSE | FALSE | FALSE |
| Thr_sbo_01.val | FALSE | TRUE | FALSE | FALSE | FALSE |
| Thr_sbo_02.val | FALSE | FALSE | FALSE | FALSE | FALSE |
| Thr_sbo_12.val | FALSE | FALSE | FALSE | FALSE | FALSE |
| Thr_sbo_21.val | FALSE | FALSE | FALSE | FALSE | FALSE |
| test_dg.val | 0 | 0 | 0 | 0 | 0 |
| **Locals** | | | | | |
| __GUARANTEE0 | TRUE | TRUE | TRUE | TRUE | FALSE |

```
  assert (CE_3 = (((((((((Thr_sbo_01.val = false )  and (pre
(Thr_sbo_01.val = false))) and (pre (pre (Thr_sbo_01.val =
false)))) and (pre (pre (pre (Thr_sbo_01.val = false))))) and
(pre (pre (pre (pre (Thr_sbo_01.val = true)))))) and
(((((Thr_ftpr_12.val = false )  and (pre (Thr_ftpr_12.val =
false))) and (pre (pre (Thr_ftpr_12.val = false)))) and (pre
(pre (pre (Thr_ftpr_12.val = true))))) and (pre (pre (pre (pre
(Thr_ftpr_12.val = false)))))) and (((((Thr_rlog_12.val =
false )  and (pre (Thr_rlog_12.val = false))) and (pre (pre
(Thr_rlog_12.val = true)))) and (pre (pre (pre (Thr_rlog_12.val
= false))))) and (pre (pre (pre (pre (Thr_rlog_12.val =
false)))))) and (((((Thr_lbo_22.val = false )  and (pre
(Thr_lbo_22.val = true))) and (pre (pre (Thr_lbo_22.val =
false)))) and (pre (pre (pre (Thr_lbo_22.val = false))))) and
(pre (pre (pre (pre (Thr_lbo_22.val = false)))))) and
(((((Thr_ftpr_01.val = true )  and (pre (Thr_ftpr_01.val =
false))) and (pre (pre (Thr_ftpr_01.val = false)))) and (pre
(pre (pre (Thr_ftpr_01.val = false))))) and (pre (pre (pre (pre
(Thr_ftpr_01.val = false)))))))))));
```

```
__GUARANTEE0 = ((test_l_2.val < 2) or (test_dg.val = 1))) or
  CE_1 or CE_2 or CE_3
```

Figure 3.5: A counterexample from JKind output automatically encoded, and disjuncted with security property in Lustre

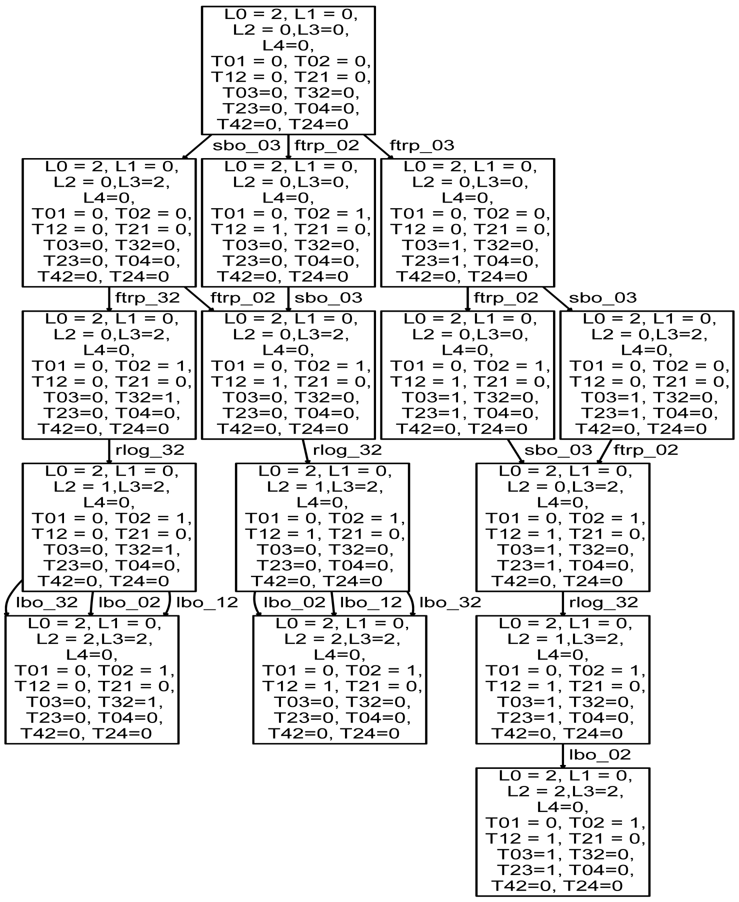
of length 4, a total of 59 different attack-paths. $K$ was then incremented to 5, generating 150 different attack-paths. A2G2V then incremented $K$ to 6, but that generated a cyclic attack-path, terminating the search for the attack-paths. Table 3.2 summarizes this process, while the set of all attack-paths is depicted in graphical form in Figure 3.6. This graphical depiction is also automatically obtained using A2G2V.

Table 3.2: 5-hosts Network Attack Graph Search Summary

| $K$ | Generated paths information | Next $K$ |
|---|---|---|
| 200 (default) | A 4-step acyclic attack-path | K=4 |
| 4 | 59 acyclic attack-paths | K=5 |
| 5 | 150 acyclic attack-paths | K=6 |
| 6 | A cyclic attack-path generated | Terminate A2G2V |

### 3.2.3 Discussions

The experimental results for the A2G2V algorithm show that it could successfully generated attack-graphs in an automated fashion for the computer networks. Such an attack-graph can be used to analyze the security of the system. For example, it can be seen from Figure 3.4 that rlog-12 attack is common to most of the attack paths, so if the resources can be used to mitigate this attack, then the security of the networked-system will hugely improve. Figure 3.6 shows how an attacker can compromise the system through host-3. Such analysis can be helpful in the design-phase where the systems administrators can introduce appropriate security measures to improve the system-level security.



L0 = 2, L1 = 0, L2 = 0,L3=0, L4=0, T01 = 0, T02 = 0, T12 = 0, T21 = 0, T03=0, T32=0, T23=0, T04=0, T42=0, T24=0
sbo_03
ftrp_02
ftrp_03
L0 = 2, L1 = 0, L2 = 0,L3=2, L4=0, T01 = 0, T02 = 0, T12 = 0, T21 = 0, T03=0, T32=0, T23=0, T04=0, T42=0, T24=0
L0 = 2, L1 = 0, L2 = 0,L3=0, L4=0, T01 = 0, T02 = 1, T12 = 1, T21 = 0, T03=0, T32=0, T23=0, T04=0, T42=0, T24=0
L0 = 2, L1 = 0, L2 = 0,L3=0, L4=0, T01 = 0, T02 = 0, T12 = 0, T21 = 0, T03=1, T32=0, T23=1, T04=0, T42=0, T24=0
ftrp_32
ftrp_02
sbo_03
ftrp_02
sbo_03
L0 = 2, L1 = 0, L2 = 0,L3=2, L4=0, T01 = 0, T02 = 1, T12 = 0, T21 = 0, T03=0, T32=1, T23=0, T04=0, T42=0, T24=0
L0 = 2, L1 = 0, L2 = 0,L3=2, L4=0, T01 = 0, T02 = 1, T12 = 1, T21 = 0, T03=0, T32=0, T23=0, T04=0, T42=0, T24=0
L0 = 2, L1 = 0, L2 = 0,L3=0, L4=0, T01 = 0, T02 = 1, T12 = 1, T21 = 0, T03=1, T32=0, T23=1, T04=0, T42=0, T24=0
L0 = 2, L1 = 0, L2 = 0,L3=2, L4=0, T01 = 0, T02 = 0, T12 = 0, T21 = 0, T03=1, T32=0, T23=1, T04=0, T42=0, T24=0
rlog_32
rlog_32
sbo_03
ftrp_02
L0 = 2, L1 = 0, L2 = 1,L3=2, L4=0, T01 = 0, T02 = 1, T12 = 0, T21 = 0, T03=0, T32=1, T23=0, T04=0, T42=0, T24=0
L0 = 2, L1 = 0, L2 = 1,L3=2, L4=0, T01 = 0, T02 = 1, T12 = 1, T21 = 0, T03=0, T32=0, T23=0, T04=0, T42=0, T24=0
L0 = 2, L1 = 0, L2 = 0,L3=2, L4=0, T01 = 0, T02 = 1, T12 = 1, T21 = 0, T03=1, T32=0, T23=1, T04=0, T42=0, T24=0
lbo_32
lbo_02
lbo_12
lbo_02
lbo_12
lbo_32
rlog_32
L0 = 2, L1 = 0, L2 = 2,L3=2, L4=0, T01 = 0, T02 = 1, T12 = 0, T21 = 0, T03=0, T32=1, T23=0, T04=0, T42=0, T24=0
L0 = 2, L1 = 0, L2 = 2,L3=2, L4=0, T01 = 0, T02 = 1, T12 = 1, T21 = 0, T03=0, T32=0, T23=0, T04=0, T42=0, T24=0
L0 = 2, L1 = 0, L2 = 1,L3=2, L4=0, T01 = 0, T02 = 1, T12 = 1, T21 = 0, T03=1, T32=0, T23=1, T04=0, T42=0, T24=0
lbo_02
L0 = 2, L1 = 0, L2 = 2,L3=2, L4=0, T01 = 0, T02 = 1, T12 = 1, T21 = 0, T03=1, T32=0, T23=1, T04=0, T42=0, T24=0

Figure 3.6: A2G2V generated Attack-graph for 5-hosts network

On a standard computer, Core i5/2.2GHz/4GB RAM running Win 10, our algorithm took 2 hours and 15 mins to construct an attack-graph for the 5 host network with safety type security property (that is violated by reachability of certain locations). For the more general “temporal” security property, that is violated by the reachability of certain paths (rather locations), our algorithm generated the attack-graph in 15 mins for the 5-host network. While [45] can also generate the attack graph for a safety property, it does not have the capability to generate the attack graph for the more general temporal security properties. Ours is the first algorithm to do so. Also, note that the algorithm in [44] does not generate an attack-graph, rather only a single attack-path.

## 3.3 A CPS Case Study: Water Treatment CPS

To demonstrate the applicability of our A2G2V algorithm to a real-world cyber-physical system, a water treatment system was chosen as a third case-study (Figure 3.7). The system is based on a fully functional water treatment CPS testbed in the iTrust laboratory [59]. The system consists of twelve hosts: Host-0 is the remote station; host-1 is the SCADA server; host-2 is the SCADA workstation; host-3 is the engineer’s laptop; host-4 is the operator Human Machine Interface (HMI); host-5 is the control room; host-6 to host-11 are the Programmable Logic Controllers (PLC) that control the physical process through sensors and actuators. We suppose that the attacker has secured access to the remote station (Host-0), and the PLCs (Hosts 6-11) are the attacker’s main target. There exists a firewall separating host-0, host-1, host-2, host-3 and the rest of the system. The firewall restricts the access from the remote station to host-1, host-2, and host-3. Also, it allows access from host-1, host-2, and host-3 to the rest of the system. The communication with the PLCs goes through host-5 and host-6. The PLCs communicate with host-5 and host-6 but not with each other.

The SCADA system components and services are all Siemens products. Online databases [87, 88] have enlist that those components may have the following vulnerabilities, which we

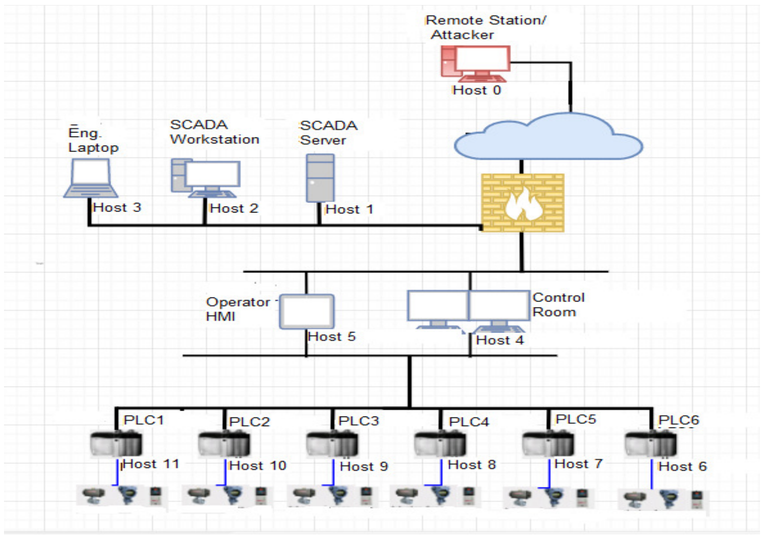


Figure 3.7: SCADA Architecture of Water Treatment System.

assume to be the case for an illustration of our approach:

1. The SCADA server is a SINEMA server, in which

   (a) *iws*: The integrated web servers at Port 4999/TCP and Port 80/TCP may allow unauthenticated remote code execution if an attacker has a network access to the server.

   (b) *lmw*: Local Microsoft Windows operating system users can escalate their privileges if the affected products are not installed under their default path ($\backslash C :$ $\backslash\backslash ProgramFiles \backslash\backslash *$" or the localized equivalent).

2. The SCADA workstation is a SIMATIC Wincc flexible runtime, in which

   (a) *rmm*: The remote management module of SIMATIC Wincc flexible panel transmits weakly protected credentials over the network. Attackers capturing network traffic of the remote management module can reconstruct the credentials.

   (b) *inws*: Siemens Wincc flexible allows remote attackers with user access to inject arbitrary web script or HTML via unspecified vectors.

3. The engineering Laptop runs a Wincc TIA portal, in which

   (a) *bac*: When a user logs in, the application sets predictable authentication token/cookie values. This can allow an attacker to bypass authentication checks.

   (b) *inws*: Wincc TIA portal allows remote attackers to inject arbitrary web script or HTML via unspecified vectors.

4. The HMI runs a Wincc runtime advanced, in which

   (a) *rmm*: The remote management module of Wincc runtime advanced transmits weakly protected credentials over the network. Attackers capturing network traffic of the remote management module can reconstruct the credentials.

   (b) *inws*: Siemens Wincc runtime advanced allows remote attackers to inject arbitrary web script or HTML via unspecified vectors.

5. The control center runs a Wincc runtime professional, in which

   (a) *rmm*: The remote management module of SIMATIC Wincc runtime professional transmits weakly protected credentials over the network. Attackers capturing network traffic of the remote management module can reconstruct the credentials.

   (b) *inws*: Siemens Wincc flexible allows remote attackers to inject arbitrary web script or HTML via unspecified vectors.

6. The PLCs are Siemens SIMATIC S7-1500 CPU PLC devices, in which

   (a) *rng*: The random-number generator on Siemens SIMATIC S7-1500 CPU PLC devices does not have sufficient entropy, which makes it easier for remote attackers to defeat cryptographic protection mechanisms and hijack sessions via unspecified vectors.

The above listed SCADA system atomic vulnerabilities can be exploited resulting in the following atomic attacks:

1. Remote code execution (*rce*): This attack exploits the vulnerability in SINEMA integrated web servers at Port 4999/TCP and Port 80/TCP if an attacker has network access to the server. This attack gives the attacker user access to the attacked host.

2. Unquoted service paths (*usp*): This attack exploits the vulnerability in SINEMA server windows operating system which allows a local user to escalate their privileges to gain root access to the attacked host.

3. User credentials construction (*ucc*): This attack exploits the weakly protected credentials in SIMATIC Wincc flexible, SIMATIC Wincc advanced, and SIMATIC Wincc professional. This attack gives the attacker user access to the attacked host.

4. Cross-site scripting (*xss*): This attack exploits the web script and HTML code injection vulnerability in Wincc TIA portal, SIMATIC Wincc flexible, SIMATIC Wincc advanced, and SIMATIC Wincc professional. This attack gives the attacker root access to the attacked host.

5. Authentication token/cookie (*atc*): This attack exploits the predictable authentication token/cookie values in a Wincc TIA portal. This attack gives the attacker root access to the attacked host.

6. Cryptographic protection mechanisms (*cpm*): This attack exploits the vulnerability in SIMATIC S7-1500 CPU random-number generator if the attacker has network access to the PLCs. This attack gives the attacker root access to the attacked host.

The water treatment system can be formally specified as follows:

1. Set of hosts $H = 0, 1, 2, \cdots, 11$; variable $i \in \{0, 1, 2, \cdots, 11\}$ (static parameters).

2. System connectivity, $C \subseteq H \times H$ ; Boolean; $c_{ij} = 1$ iff host $i$ connected to host $j$ (static parameters).

3. System services $S$; Boolean; $s_i = 1$ iff service $s \in$ { SCADA server is SINEMA, SIMATIC Wincc runtime flexible, Wincc runtime advanced, Wincc runtime professional, Wincc TIA portal, SIMATIC S7-1500 CPU } is running on host $i$ (dynamic variables).

4. System vulnerabilities $V$ ; Boolean; $v_i = 1$ iff vulnerability $v \in$ { $iws$, $lmw$, $rmm$, $inws$, $bac$, $rng$ } exists on host $i$ (static parameters).

5. Attack instances $A_I \subseteq A \times H \times H$; labeled $a_{ij} \equiv$ attack $a$ from source $i$ to target $j$, $a \in \{rce, usp, ucc, xss, atc\}$, $rce :=$ remote code execution, $usp :=$ unquoted service paths, $ucc :=$ user credentials construction, $xss :=$ cross-site scripting $atc :=$ authentication token/cookie (static parameters).

6. Trust relation $T \subseteq H \times H$ ; Boolean; $t_{ij} = 1$ iff $i$ is trusted by $j$ (dynamic variables).

7. Attacker level of privilege $L$ on host $i$; variable $l_i \in \{none, user, root\}$ (dynamic variables).

8. Attack pre-conditions:

   - $Pre_{(rce_{ij})} \equiv c_{ij} = 1 \wedge (l_i \geqslant user) \wedge (l_j = none) \wedge iws = 1$
   - $Pre_{(usp_{ij})} \equiv c_{ij} = 1 \wedge (l_i \geqslant user) \wedge (l_j < root) \wedge lmw = 1$
   - $Pre_{(ucc_{ij})} \equiv c_{ij} = 1 \wedge (l_i \geqslant user) \wedge (l_j = none) \wedge rmm = 1$
   - $Pre_{(xss_{ij})} \equiv c_{ij} = 1 \wedge (l_i \geqslant user) \wedge (l_j < root) \wedge inws = 1$
   - $Pre_{(atc_{ij})} \equiv c_{ij} = 1 \wedge (l_i \geqslant user) \wedge (l_j = none) \wedge bac = 1$
   - $Pre_{(cpm_{ij})} \equiv c_{ij} = 1 \wedge (l_i \geqslant user) \wedge (l_j = none) \wedge rng = 1$

9. Attack post-conditions:

   - $post_{(rce_{ij})} \equiv (l_j = user) \wedge (iws = 0)$
   - $post_{(usp_{ij})} \equiv (l_j = root) \wedge (lmw = 0)$

- $post_{(ucc_{ij})} \equiv (l_j = user) \wedge (rmm = 0)$
- $post_{(xss_{ij})} \equiv (l_j = root) \wedge (inws = 0)$
- $post_{(atc_{ij})} \equiv (l_j = user) \wedge (bac = 0)$
- $post_{(cpm_{ij})} \equiv (l_j = root) \wedge (rng = 0)$

10. Intial states : $l_0 = root \wedge (l_1 = l_2 = \cdots = l_{11} = none) \wedge (\forall ij \in H \times H : t_{ij} = 0) \wedge (iws = lmw = rmm = inws = bac = rng = 1)$
(Initially, the attacker has root privilege on host-0 and no privilege on other hosts, none of the hosts trust each other, and $iws$, $lmw$, $rmm$, $inws$, $bac$, and $rng$ are running on the SCADA system components.)

11. The security property $\phi$ of interest is violated if the attacker has the root privilege level on host-6. This can then be described by a Computational Tree Logic (CTL) formula:

$$\neg\phi \equiv AG(l_6 = root).$$

An AADL+AGREE description that captures the formal description of the above system, its components, connectivity, dynamic state variables, atomic attacks, and security property of interest was encoded. The constructed system model $M$ consists of a total of $1.9 \times 10^{14}$ states and $3.6 \times 10^{28}$ transitions, and was fed as input to our A2G2V tool.

We analyzed the security of the water treatment system, using A2G2V for $\phi = AG(l_6 \geq 1)$). This property represents all attack scenarios that an attacker may take to compromise host-6. Our A2G2V algorithm successfully generated the attack-graph for this property as demonstrated in Figure 3.8.

The A2G2V algorithm started by setting default search depth for attack-paths to $K = 200$. The first iteration generated an acyclic 5 steps attack-path, and so $K$ was set to 5. A2G2V generated all attack-paths of length 5 , generating 100 different attack-paths. Then $K$ was incremented to 6. This generated a cyclic attack-path, which then terminated the search for the attack-paths. Table 3.3 summarizes this process.

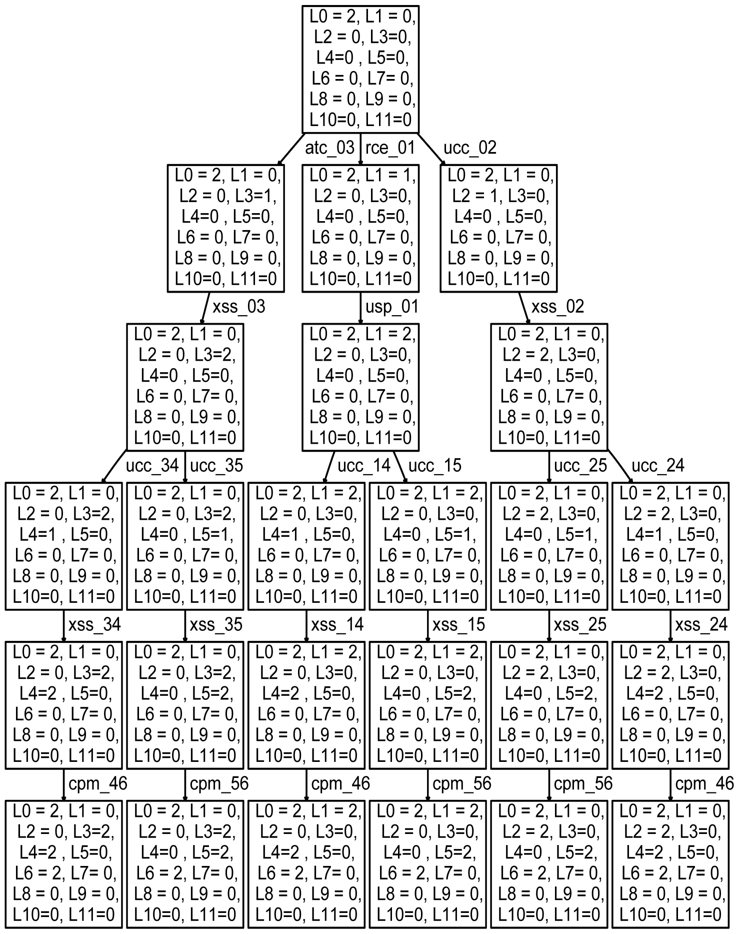

L0 = 2, L1 = 0, L2 = 0, L3=0, L4=0 , L5=0, L6 = 0, L7= 0, L8 = 0, L9 = 0, L10=0, L11=0
atc_03
rce_01
ucc_02
L0 = 2, L1 = 0, L2 = 0, L3=1, L4=0 , L5=0, L6 = 0, L7= 0, L8 = 0, L9 = 0, L10=0, L11=0
L0 = 2, L1 = 1, L2 = 0, L3=0, L4=0 , L5=0, L6 = 0, L7= 0, L8 = 0, L9 = 0, L10=0, L11=0
L0 = 2, L1 = 0, L2 = 1, L3=0, L4=0 , L5=0, L6 = 0, L7= 0, L8 = 0, L9 = 0, L10=0, L11=0
xss_03
usp_01
xss_02
L0 = 2, L1 = 0, L2 = 0, L3=2, L4=0 , L5=0, L6 = 0, L7= 0, L8 = 0, L9 = 0, L10=0, L11=0
L0 = 2, L1 = 2, L2 = 0, L3=0, L4=0 , L5=0, L6 = 0, L7= 0, L8 = 0, L9 = 0, L10=0, L11=0
L0 = 2, L1 = 0, L2 = 2, L3=0, L4=0 , L5=0, L6 = 0, L7= 0, L8 = 0, L9 = 0, L10=0, L11=0
ucc_34
ucc_35
ucc_14
ucc_15
ucc_25
ucc_24
L0 = 2, L1 = 0, L2 = 0, L3=2, L4=1 , L5=0, L6 = 0, L7= 0, L8 = 0, L9 = 0, L10=0, L11=0
L0 = 2, L1 = 0, L2 = 0, L3=2, L4=0 , L5=1, L6 = 0, L7= 0, L8 = 0, L9 = 0, L10=0, L11=0
L0 = 2, L1 = 2, L2 = 0, L3=0, L4=1 , L5=0, L6 = 0, L7= 0, L8 = 0, L9 = 0, L10=0, L11=0
L0 = 2, L1 = 2, L2 = 0, L3=0, L4=0 , L5=1, L6 = 0, L7= 0, L8 = 0, L9 = 0, L10=0, L11=0
L0 = 2, L1 = 0, L2 = 2, L3=0, L4=0 , L5=1, L6 = 0, L7= 0, L8 = 0, L9 = 0, L10=0, L11=0
L0 = 2, L1 = 0, L2 = 2, L3=0, L4=1 , L5=0, L6 = 0, L7= 0, L8 = 0, L9 = 0, L10=0, L11=0
xss_34
xss_35
xss_14
xss_15
xss_25
xss_24
L0 = 2, L1 = 0, L2 = 0, L3=2, L4=2 , L5=0, L6 = 0, L7= 0, L8 = 0, L9 = 0, L10=0, L11=0
L0 = 2, L1 = 0, L2 = 0, L3=2, L4=0 , L5=2, L6 = 0, L7= 0, L8 = 0, L9 = 0, L10=0, L11=0
L0 = 2, L1 = 2, L2 = 0, L3=0, L4=2 , L5=0, L6 = 0, L7= 0, L8 = 0, L9 = 0, L10=0, L11=0
L0 = 2, L1 = 2, L2 = 0, L3=0, L4=0 , L5=2, L6 = 0, L7= 0, L8 = 0, L9 = 0, L10=0, L11=0
L0 = 2, L1 = 0, L2 = 2, L3=0, L4=0 , L5=2, L6 = 0, L7= 0, L8 = 0, L9 = 0, L10=0, L11=0
L0 = 2, L1 = 0, L2 = 2, L3=0, L4=2 , L5=0, L6 = 0, L7= 0, L8 = 0, L9 = 0, L10=0, L11=0
cpm_46
cpm_56
cpm_46
cpm_56
cpm_56
cpm_46
L0 = 2, L1 = 0, L2 = 0, L3=2, L4=2 , L5=0, L6 = 2, L7= 0, L8 = 0, L9 = 0, L10=0, L11=0
L0 = 2, L1 = 0, L2 = 0, L3=2, L4=0 , L5=2, L6 = 2, L7= 0, L8 = 0, L9 = 0, L10=0, L11=0
L0 = 2, L1 = 2, L2 = 0, L3=0, L4=2 , L5=0, L6 = 2, L7= 0, L8 = 0, L9 = 0, L10=0, L11=0
L0 = 2, L1 = 2, L2 = 0, L3=0, L4=0 , L5=2, L6 = 2, L7= 0, L8 = 0, L9 = 0, L10=0, L11=0
L0 = 2, L1 = 0, L2 = 2, L3=0, L4=0 , L5=2, L6 = 2, L7= 0, L8 = 0, L9 = 0, L10=0, L11=0
L0 = 2, L1 = 0, L2 = 2, L3=0, L4=2 , L5=0, L6 = 2, L7= 0, L8 = 0, L9 = 0, L10=0, L11=0

Figure 3.8: A2G2V generated Attack-graph for Water Treatment SCADA CPS

Table 3.3: Water Treatment SCADA CPS Attack Graph Search Summary

| *K* | Generated paths information | Next K |
|---|---|---|
| 200 (default) | A 5-step acyclic attack-path | K=5 |
| 5 | 100 acyclic attack-paths | K=6 |
| 6 | A cyclic attack-path generated | Terminate A2G2V |

### 3.3.1 Discussions

On a standard computer, Core i5/2.2GHz/4GB RAM running Win 10, our algorithm took 40 mins. The experimental results demonstrated the usefulness of our A2G2V algorithm showing that it successfully generated attack-graphs in an automated fashion for realistic applications such as the SCADA case study. The generated attack-graph helps the system administrators to utilize the resources to maximize the security while minimize the cost of consequences. An inspection reveals that the *ucc* and the *xss* are the most common attacks. Also, the generated attack-graph demonstrates that an *xss* attack can never be applied successfully without successfully executing a *ucc* attack first. So, by way stopping the *ucc* attacks, the system administrators can also eliminate the *xss* attacks, which would greatly improve the system security.

# CHAPTER 4. CRITICAL-ATTACKS SET IDENTIFICATION IN ATTACK-GRAPHS FOR COMPUTER AND SCADA NETWORKS

In general, even small systems may have large and complex attack-graphs, and so for system administrators with limits on budgets and resources, securing all vulnerabilities included in an attack-graph might not be desirable. Further, the scale of the attack-graphs make those difficult to fully comprehend through manual inspection. Thus securing an SCADA/ICS system requires an automated tool to help system administrators analyze an attack-graph to identify the *minimum* defense enactments that can help secure their system in a viable fashion. We refer to such minimal set of attacks, a critical attacks-set, mitigating which would render the attack-graph disconnected, and thereby disabling all attack-paths that an adversary may exercise to compromise the given system.

Attack-graphs are directed labeled graphs, where each vertex corresponds to a certain system status including attacker privileges on various devices/assets, each edge represents an atomic attack that an attacker might execute to gain more privileges, and each path from the initial node to a final node represents an attack path an attacker may follow to compromise a system-level security property. The goal for a system administrator then is to find a minimal number of attacks that could be prevented such that no viable path remains from the initial node to a final node. In graph theory this is known as a cut. Since the attack-graph is a directed labeled graph, finding the minimal number of attacks, whose prevention would render the disconnection of the initial node from the final nodes, is an instance of a "min label-cut" (MLC) [46, 47] that requires finding the minimum number of attack-labels whose edges must be removed from a graph to cut all paths from the initial node to the final nodes. A direct application of the MLC to identify a set of critical attacks is computationally limiting, since MLC is in general an NP-complete problem [46, 47].

In this chapter, we propose a linear complexity, automated critical-attacks set identification approach, not necessarily minimal, based on the analysis of the attack-graph. The first step within our approach is to compute the Strongly Connected Components (SCCs) of a given attack-graph, and use those to generate an abstracted version of the attack-graph, namely, the tree over its SCCs. (Recall that a subgraph of a directed graph, also called a component, is strongly connected if there is a directed path from any vertex of the component to every other vertex of the component, and where all vertices along the path are within the component.) The next step is to employ a backward search over the abstracted attack-graph (a tree over the SCCs), one hop at a step, starting from its terminal node, to iteratively find the set of nodes that can reach the terminal node in a increasing number steps, and identify the labels of their outgoing edges (each of which form a cut). Finally, we identify the cut with a minimal number of labels, termed a SCC-induced Min Label-Cut (min SCCiMLC).

To validate the SCCiMLC algorithm we present three case-studies: two computer network systems and one water treatment SCADA CPS system from the iTrust Lab [59]. Further to compare the SCCiMLC algorithm efficiency, we compare it against an exact and an approximation algorithm. For the former, we first generalize the exact MLC algorithm proposed by [56], that works only for disjoint undirected graph with label frequency of at most 2, to the general labeled-graphs. This exact MLC-algorithm searches over the source-to-terminal paths ($s$-$t$ paths in short): It starts by selecting an $s$-$t$ path and picks the labels included in the path. New $s$-$t$ paths, whose all labels are not yet among the picked labels, are selected iteratively and those new labels of the path are added to the selected labels, until a MLC results. In addition to this exact algorithm, we also compare the SCCiMLC algorithm to the state-of-art approximation algorithm of [1], one that is shown to possess the best approximation factor reported in literature. [1] proposes to compute an approximation to a MLC by optimizing over the minimum 0/1-polytope corresponding to an "approximate" hitting set (details can be found in the cited reference).

The main contributions of this chapter are summarized as follows.

1. A linear, automated, and Strongly-Connected-Components (SCCs) based, critical-attacks set identification algorithm, termed SCCiMLC.
2. An exact MLC algorithm inspired by [56], by way of its generalization.
3. A comparison between the proposed SCCiMLC algorithm performance, the generalized exact algorithm, and a state-of-art approximation algorithm.
4. Implementation of the proposed and two comparison algorithms, along with their validation against two IT network examples and a water treatment CPS provided by the iTrust Lab. The comparison and validation results are very promising for the 3 case-studies examined: The proposed SCCiMLC has the same accuracy as that of the exact algorithm, while the same speed as that of the approximation algorithm, which is $> 65$ times faster than the exact algorithm.

## 4.1 Critical-Attacks Set Computation

Consider an attack-graph $G = (V, E, s, t, L, \ell)$, where $V$ is the set of vertexes, $E \subseteq V \times V$ is the set of edges that connect a vertex to another, $s$ is the source vertex, $t$ is the terminal vertex, $L$ is a set of labels where each label represents an atomic attack, and $\ell : E \to L$ is an edge-labeling function.

Figure 4.1 shows an attack-graph example in which each sequence of edges, starting from the source vertex $s$ and ending at the terminal vertex $t$, represents an attack sequence that can compromise a system under attack. Model-based approaches to computing an attack-graph, given the system description, its atomic attack actions, and its security property of interest, are reported in [47, 89, 63]

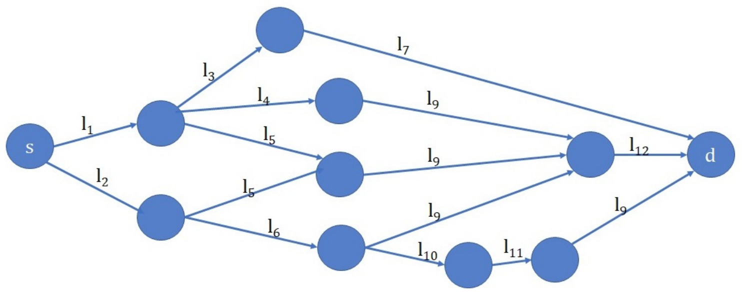


Figure 4.1: Attack-Graph Example

Atomic attack actions can be mitigated by eliminating their root-cause vulnerabilities (e.g., by way of providing available software patches for them), which is equivalent to removing corresponding labeled edges from the attack graph. To make the overall system secure so that a desired system-level security property of interest cannot be violated through a sequence of atomic attack actions, a set of atomic attacks can be mitigated, so that the removal of the corresponding labeled edges from the attack-graph renders it disconnected, making the terminal vertex unreachable from the source vertex.

Such a set of atomic attacks can then be viewed as inducing a *labeled cut* of the attack-graph, where a labeled cut is simply a set of labels $L' \subseteq L$ such that the removal of edges carrying those labels disconnects the source and terminal vertices. An attack set that can disconnect a given attack-graph is called a *critical-attacks set*. A desirable goal then is to find a *minimal critical-attacks* set, finding which can be viewed as an instance of a "min labeled-cut" (MLC) problem [1], whose goal is to identify the smallest set $L' \subseteq L$ of labels, such that $s$ and $t$ become disconnected when the edges carrying the labels in $L'$ are deleted from the attack-graph.

MLC is known to be an NP-complete problem [1, 48]. In order to find a "near-minimal critical-attacks set" $CA \subseteq L$ in a polynomial complexity manner, in this chapter we introduce the notion of a "Strongly Connected Components-induced Min Label-Cut (SC-CiMLC)" to approximate a minimal critical-attacks set. We also extend [56] to obtain an exact MLC algorithm and implement it for comparison. Also for comparison, we also implement the state-of-art MLC approximation algorithm proposed in [1]. The said three implemented algorithms are compared with respect to three case studies provided in the next section.

### 4.1.1 Strongly Connected Components-induced Min Label-Cut

In our earlier work [89, 63] we developed an algorithm, A2G2V (Automated Attack Graph Generation and Visualization), for model-based automated generation of attack-graphs, extending the initial work of [47] to address security properties that can be a predicate over state-trajectories, rather just a predicate of the states. A2G2V, in turn, relies on our method for Automated Network Device and Vulnerability Identification (ANDVI) tool [90] that passively observes network traffic to discover network connectivity and fingerprint network devices, and next refers to their vulnerability databases to enumerate the device vulnerabilities as well as the associated atomic attack actions. A2G2V uses the output of ANDVI (device vulnerabilites and their connectivity) to then enumerate the set of all nesting of atomic attack actions to yield an attack-graph, consisting of all sequences of atomic attack actions ("attack-paths") that can compromise a system-level security property.

The proposed SCCiMLC algorithm (Figure 4.2) approximates a solution to the MLC problem, identifying a critical-attacks set in linear time in the following manner. It first creates an abstracted tree version (i.e., an acyclic graph) of the given attack-graph, defined over the vertex set of its strongly connected components (SCCs), and next uses a backward search starting from the terminal node to iteratively identify a set of nodes that can reach the terminal node in $k$ or less steps ($k \geq 1$), and also identifies the associated cuts in form of the outgoing edges of the $k$-step backward reachable nodes. The cut withe smallest number of labels yields a SCCiMLC.

To visualize the proposed SCCiMLC, consider a simpler case in which all the edges in an attack-graph from source $s$ to terminal $t$ are forward edges, i.e., the given attack-graph has no cycles (is a tree), and so each node is its own SCC. Starting from $t$ and reaching backwards one hop at a time, one can iteratively identify the set of nodes that reach $t$ in 1 or less, 2 or less, 3 or less, etc. steps. For each such set of nodes that can reach $t$ in $k$ or less steps ($k \geq 1$), there is a set of outgoing edges, cutting which induces a cut. Among those cuts, the one with the minimum number of labels is the SCCiMLC. This same idea can be applied to the tree graph over the SCCs of the given attack-graph, backward searching over that tree to yield SCCs that are $k$ or less steps away from the terminal node, and identifying the labels of their outgoing edges as the label-cuts.

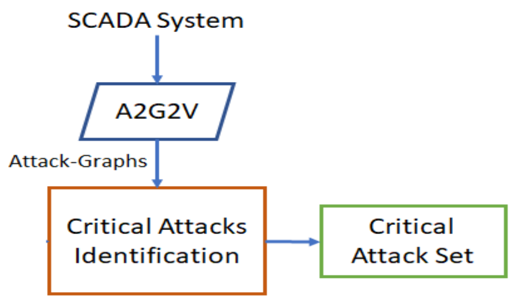


Figure 4.2: Architecture of the Critical-Attacks Identification (CAI) Algorithm.

**Definition 4** *For a graph $(V, E)$, $C \subseteq V$ is a* Strongly Connected Component (SCC) *if for all $u, v \in C$, exists a sequence of edges, $(u, v_1)(v_1, v_2)\ldots(v_i, v_{i+1})(v_{i+1}, v_{i+2})\ldots(v_n, v)$, connecting $u$ to $v$, such that $v_1 \ldots v_n \in C$. The edge set $E_C$ of the SCC $C$ is simply defined to be, $E_C := E \cap (C \times C)$.*

**Definition 5** *For the attack-graph $(V, E, s, t, L, \ell)$, we obtain an abstracted graph over its set of SCCs, $(\mathcal{V}, \mathcal{E}, C_s, C_t, L, \ell)$, where $\mathcal{V}$ is the set of SCCs of $(V, E)$; $\mathcal{E} := \{(C, C') \in \mathcal{V} \times \mathcal{V} | \exists u \in C, v \in C', (u, v) \in E\}$ is the set of edges connecting any two SSCs; $C_s, C_t \in \mathcal{V}$ are the SCCs containing $s$, $t$ respectively (i.e., $s \in C_s, t \in C_t$); and for any edge $(C, C') \in \mathcal{E}$, $\ell((C, C')) = \{l \in L | \exists u \in C, v \in C', l = \ell(u, v)\}$, i.e., it is the union of all labels of the edges connecting $C$ to $C'$.*

Once the abstracted graph (a tree over the SCCs) has been generated, we use a iterative backward reach over the tree, one hop at a iteration, starting from its terminal SCC $C_t$ to identify the set of SCCs $\mathcal{V}_k \subseteq \mathcal{V}$ that can reach the terminal one in $k \geq 0$ iterations, with initial value $\mathcal{V}_0 = C_t$. We also the set of outgoing edges $\mathcal{E}_k \subseteq \mathcal{E}$ of $\mathcal{V}_k$ and their labels $L_k \subseteq L$, with the initial values $\mathcal{E}_0 = L_0 = \emptyset$. The following iterative computation over $0 \leq k \leq |\mathcal{V}| - 1$ performs these steps.

**Initialize:**

$k = 0; \mathcal{V}_k = \{C_t\}; \mathcal{E}_k = L_k = \emptyset;$

**Iterate:**

- $\mathcal{V}_{k+1} := \{C \in \mathcal{V} | \exists C' \in \cup_{i \leq k} \mathcal{V}_i \text{ such that } (C, C') \in \mathcal{E}\};$
- $\mathcal{E}_{k+1} = \{(C, C') \in \mathcal{E} | C \in \mathcal{V}_{k+1}, C' \in \cup_{i \leq k} \mathcal{V}_i\};$
- $L_{k+1} := \ell(\mathcal{E}_{k+1});$

**Terminate:**

If $\cup_{0 \leq i \leq k+1} \mathcal{V}_i = \mathcal{V}$, then set $i^* := \arg\min_{1 \leq i \leq k+1} |L_i|, L^* := L_{i^*}$ and stop; else, set $k = k + 1$ and goto Iterate.

The algorithm picks $i^*$ to be the index with the smallest number of labels ($i^* = \arg\min_{1 \leq i} |L_i|$). Then $L_{i^*}$ is a desired SCCiMLC that approximates the MLC.

The proposed SCCiMLC steps are formalized in the following algorithm:

**Algorithm 3** SCCiMLC Algorithm

```
1: **Input:** $G = (V, E, s, t, L, \ell)$
2: **Output:** A critical-attacks set, $CA \subseteq L$
3: **main**
4: Compute SCCs of $G$ as in Definition 1
5: Generate the tree over the SCCs, $(\mathcal{V}, \mathcal{E}, C_s, C_t, L, \ell)$ as in Definition 2
6: $k = 0, \mathcal{V}_k = \{C_t\}, \mathcal{E}_k = \emptyset, L_k := \ell(\mathcal{E}_k)$;
7: For $k = 1$ to $k \leq |\mathcal{V}| - 1$:
8:   $\mathcal{V}_{k+1} := \{C \in \mathcal{V} | \exists C' \in \cup_{i \leq k} \mathcal{V}_i \text{ s.t. } (C, C') \in \mathcal{E}\}$;
9:   $\mathcal{E}_{k+1} = \{(C, C') \in \mathcal{E} | C \in \mathcal{V}_{k+1}, C' \in \cup_{i \leq k} \mathcal{V}_i\}$;
10:   $L_{k+1} := \ell(\mathcal{E}_{k+1})$;
11: $k^* := \arg\min_{k \geq 1} |L_k|$;
12: $CA := L_{k^*}$.
13: **end**
```

In Algorithm 3, the generation of the SCCs, the construction of the tree over the SCCs, and also the backward reachability over that tree to find a SCCiMLC, can all be performed in complexity that is linear in the size of the given attack-graph, meaning the proposed ALgorithm 3 of finding SCCiMLC is of linear complexity. We encoded the above algorithm for finding a SCCiMLC in the C language. It receives the output from our A2G2V algorithm (that generates an attack-graph), which in turn receives input from our ANDVI algorithm that processes network traffic to identify (i) the network devices and maps those against the existing databases to identify their vulnerabilities and the corresponding atomic attack actions, and (ii) the network connectivity.

To validate our min SCCiMLC implementation, we tested it on attack-graphs of two computer network examples and the SCADA network used for the control of a water treatment testbed at the iTrust lab in Singapore [59]. We also compared the results to an exact

MLC algorithm inspired by [56] and to Dutta et al. [1] approximation algorithm, and both of those algorithms are described below.

### 4.1.2 Implementing an Exact MLC

In an attempt to provide a polynomial complexity algorithm for computing an exact MLC for a special case, [56] first showed that MLC is NP-hard if the maximum length of any $s$-$t$ path is longer than two, or if the maximum label frequency (the maximum number of times that a label appears in a graph) $f_{max} > 2$. [56] also proved that, when restricted to disjoint-path undirected graphs, MLC can be solved in polynomial time if $f_{max} = 2$. For the sake of comparison with the proposed SCCiMLC, we extend the algorithm in [56] to a general directed attack-graph. Note that the time complexity of the extended algorithm is no longer polynomial.

Algorithm 3.4 in [56] is initialized by picking a path and all its edge-labels, say $L_0 \subseteq L$ (those edge-labels from a candidate set of labels to be removed as part of a label-cut). Let $L_k$ be the set of labels at $k$th iteration (from past $k$ selected paths). If cutting edges with labels $L_k$ does not induce a cut, then there must exist a $s$-$t$ path whose none of the labels are included in $L_k$. In the iterative step, such a path is found, and all its labels are added to $L_k$ to obtain $L_{k+1}$. The iteration terminates when a set of labels $L_k$ that induces a label-cut is found. The size of this label-cut can be further reduced by searching over all subsets of $L_k$ to find a minimal subset $CA \subseteq L_k$ that also induces a cut. The exact algorithmic steps are formalized below in Algorithm 4.

**Algorithm 4** MLC Exact Algorithm

```
1: **Input:** $G = (V, E, s, t, L, \ell)$
2: **Output:** A critical-attacks set, $CA \subseteq L$
3: **main**
4: Let $L_k = \phi$
5: while $L_k$ does not induce a cut
6:   Pick any path $p \in G$ such that $\ell(p) \cap L_k = \emptyset$
7:   $L_{k+1} = L_k \cup \ell(p)$
8: Identify all subsets $\mathcal{A} := \{A \subseteq L_k\}$
9:   Pick a minimal subset $A \in \mathcal{A}$ that induces a cut
10: $CA := A$
11: **end**
```

We encoded the above exact MLC algorithm in C, and used the implementation to compare the performance against our own proposed algorithm in Section 4.2.

### 4.1.3 Implementing state-of-art Approximation Algorithm [1]

As note above in Section 1.3, a few prior studies have proposed algorithms for approximating a solution to the MLC problem, and among those, a few do not directly apply to a general attack-graph and hence not suitable for comparison (for example the algorithm in [47] requires an atomic attack to appear only once on an $s$-$t$ path, which is generally not the case as can be seen from our examples in Section 4.2.) Among the ones that are directly applicable to a general attack-graph, we chose [1] for implementation and comparison, as it can be considered the state-of-art given that it provides the best known approximation factor.

It is known from [47] that finding a MLC is an instance of a hitting set problem: Given a set of labels, one for each $s$-$t$ path, namely, $\mathcal{L}\{L_p \subseteq L \mid p$ a $s$-$t$ path in $G\}$, a hitting-set is a minimal subset $CA \subseteq L$ that intersects with each $L_p$, i.e., $\forall p \in G : L_p \cap CA \neq \emptyset$. [1]

proposed finding an approximation to the MLC solution by optimizing over a 0/1-polytope which corresponds to a hitting set for the set for $\mathcal{L}$.

**Definition 6** *[91] A 0/1-polytope in d-dimensional space is the convex hull of a set of d-dimensional end points $Q \subseteq \{0,1\}^d$, whose vertices have 0/1-coordinates only (i.e., it is a convex subset of the hypercube $\{0,1\}^d$ with constraint on its coordinates being 0/1) .*

The above mentioned hitting-set problem can be solved via a 0/1-polytope optimization problem. We use $\mathcal{P}_G$ to denote the set of all $s$-$t$ paths in $G$. Define set of end points in the $|L|$-dimensional unit-cube:

$$Q := \{x \in \{0,1\}^{|L|} \mid \forall p \in \mathcal{P}_G, \exists l_k \in \ell(p) : x(k) = 1\},$$

where $l_k$ denotes the $k$th label in the set $L$. In other words, each $x \in Q$ "selects" at least one label from each $s$-$t$ path. Next, solve for:

$$\min_{x \in Q} \sum_{k \in \{1,\ldots,|L|\}} x(k).$$

The solution to the above optimization provides an exact solution to MLC. To obtain an approximate solution, a relaxed version of the hitting-set problem is formulated in [1], where the end-points are allowed to be unit-interval valued numbers:

$$\widehat{Q} := \{x \in [0,1]^{|L|} \mid \forall p \in \mathcal{P}_G : |p| \leq |V|^{\frac{2}{3}}, \exists l_k \in \ell(p) : x(k) \geq \frac{1}{|V|^{\frac{2}{3}}}\},$$

and the following optimization is performed:

$$\min_{x \in \widehat{Q}} \sum_{k \in \{1,\ldots,|L|\}} x(k).$$

Let $x^* \in \widehat{Q}$ be the minimizer. Then a label-cut is found as follows:

$$\begin{aligned}
L' &:= \{l_k \in L \mid x^*(k) \geq \frac{1}{|V|^{\frac{2}{3}}}\},\\
E' &:= \{e \in E \mid \ell(e) \notin L_1\},\\
L'' &:= \text{ a MLC for } G' = (V, E', s, t, L, \ell),\\
CA &:= L' \cup L''.
\end{aligned}$$

For implementing the above hitting-set solution based computation of an approximation to MLC, we used the MatLab code from [92], where we modified the "separation oracle" function to match the separation hyperplane algorithm in [1].

## 4.2 Case-studies: Computer Networks and Water Treatment SCADA CPS

Realistic applications of two computer network examples and a real-world water treatment testbed of the iTrust Lab [59], were chosen as case-studies to demonstrate the applicability of the proposed SCCiMLC algorithm, and to compare its performance to both the exact algorithm (our generalization of [56]) and the state-of-art approximation algorithm of [1].

### 4.2.1 Computer Network Case-studies

As a concrete example illustrating the problem of critical-attacks set identification, we adapt the networked-system example from [46, 47, 89, 63] shown in Figure 4.3.

There are three hosts in the network: host-0, where the attacker is located, while the two target hosts are host-1 and host-2. Host-1 runs sshd and ftp, while host-2 runs ftp and database. As a result, this system has four possible atomic attacks per host. There is also a firewall that separates the targets from the rest of the network. The firewall does not place any restrictions on access control over network traffic flow. The network traffic between network hosts and external sources is supervised by an intrusion detection system (IDS). The IDS can monitor the flow of traffic between (host-0 ; host-1) and (host-0 ; host-2), but not between (host-1 ; host-2) because of its network positioning. The full system description and its implementation within our A2G2V (Automate Attack Graph Generation and Visualization) tool can be found in [89, 63]. Figure 4.4 shows the automatically generated attack-graph of this system using our tool A2G2V.

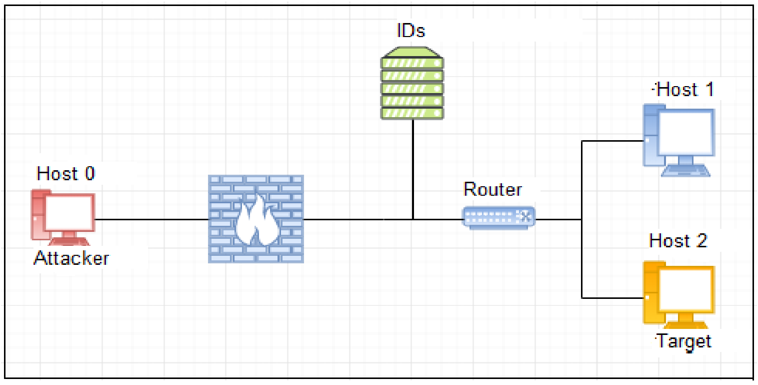


Figure 4.3: 3-Hosts Networked-system example.

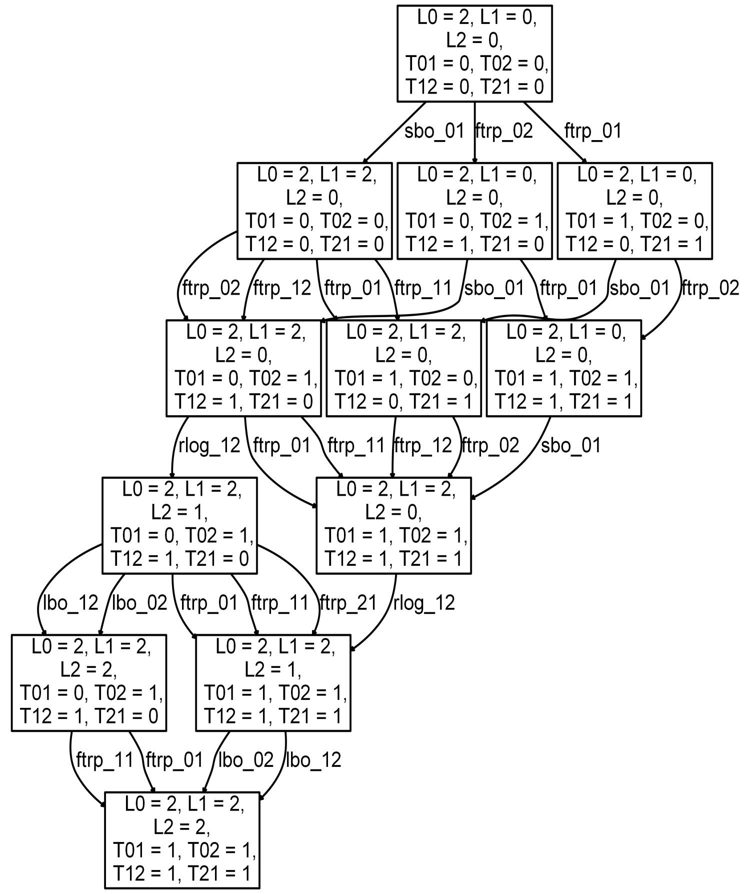

L0 = 2, L1 = 0, L2 = 0, T01 = 0, T02 = 0, T12 = 0, T21 = 0
sbo_01
ftrp_02
ftrp_01
L0 = 2, L1 = 2, L2 = 0, T01 = 0, T02 = 0, T12 = 0, T21 = 0
L0 = 2, L1 = 0, L2 = 0, T01 = 0, T02 = 1, T12 = 1, T21 = 0
L0 = 2, L1 = 0, L2 = 0, T01 = 1, T02 = 0, T12 = 0, T21 = 1
ftrp_02
ftrp_12
ftrp_01
ftrp_11
sbo_01
ftrp_01
sbo_01
ftrp_02
L0 = 2, L1 = 2, L2 = 0, T01 = 0, T02 = 1, T12 = 1, T21 = 0
L0 = 2, L1 = 2, L2 = 0, T01 = 1, T02 = 0, T12 = 0, T21 = 1
L0 = 2, L1 = 0, L2 = 0, T01 = 1, T02 = 1, T12 = 1, T21 = 1
rlog_12
ftrp_01
ftrp_11
ftrp_12
ftrp_02
sbo_01
L0 = 2, L1 = 2, L2 = 1, T01 = 0, T02 = 1, T12 = 1, T21 = 0
L0 = 2, L1 = 2, L2 = 0, T01 = 1, T02 = 1, T12 = 1, T21 = 1
lbo_12
lbo_02
ftrp_01
ftrp_11
ftrp_21
rlog_12
L0 = 2, L1 = 2, L2 = 2, T01 = 0, T02 = 1, T12 = 1, T21 = 0
L0 = 2, L1 = 2, L2 = 1, T01 = 1, T02 = 1, T12 = 1, T21 = 1
ftrp_11
ftrp_01
lbo_02
lbo_12
L0 = 2, L1 = 2, L2 = 2, T01 = 1, T02 = 1, T12 = 1, T21 = 1

Figure 4.4: 3-Hosts Networked-system generated attack-graph.

As a second computer network system example, we used a 5-hosts network Figure 4.5, which is an extended version of the 3-hosts network descried above. Here, host-3 and host-4 are additional target hosts, that run ftp and sshd services as in the case of host-1, and hence have the same vulnerabilities as host-1. As before, this system has four possible atomic attacks per host. Also, as before, the firewall does not place any access control restrictions on the flow of network traffic. Also due to the placement of IDS on the network, the traffic flows between (host-0; host-1), (host-0; host-2), (host-0; host-3), and (host-0; host-4) are monitored by the IDS, while those among (host-1; host-2; host-3; host-4) are not monitored. The full system description and its implementation within our A2G2V tool can be found in [89, 63]. Figure 4.6 shows the automatically generated attack-graph of this system using our tool A2G2V.

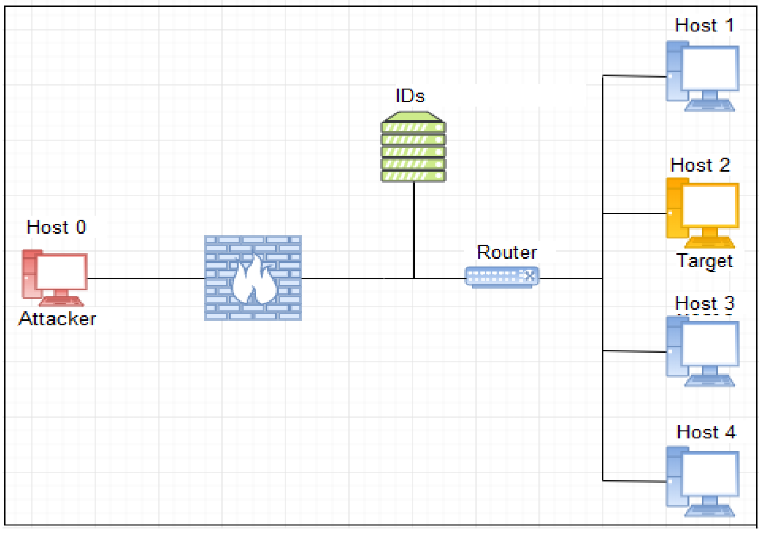


Figure 4.5: 5-Hosts Networked-system example.

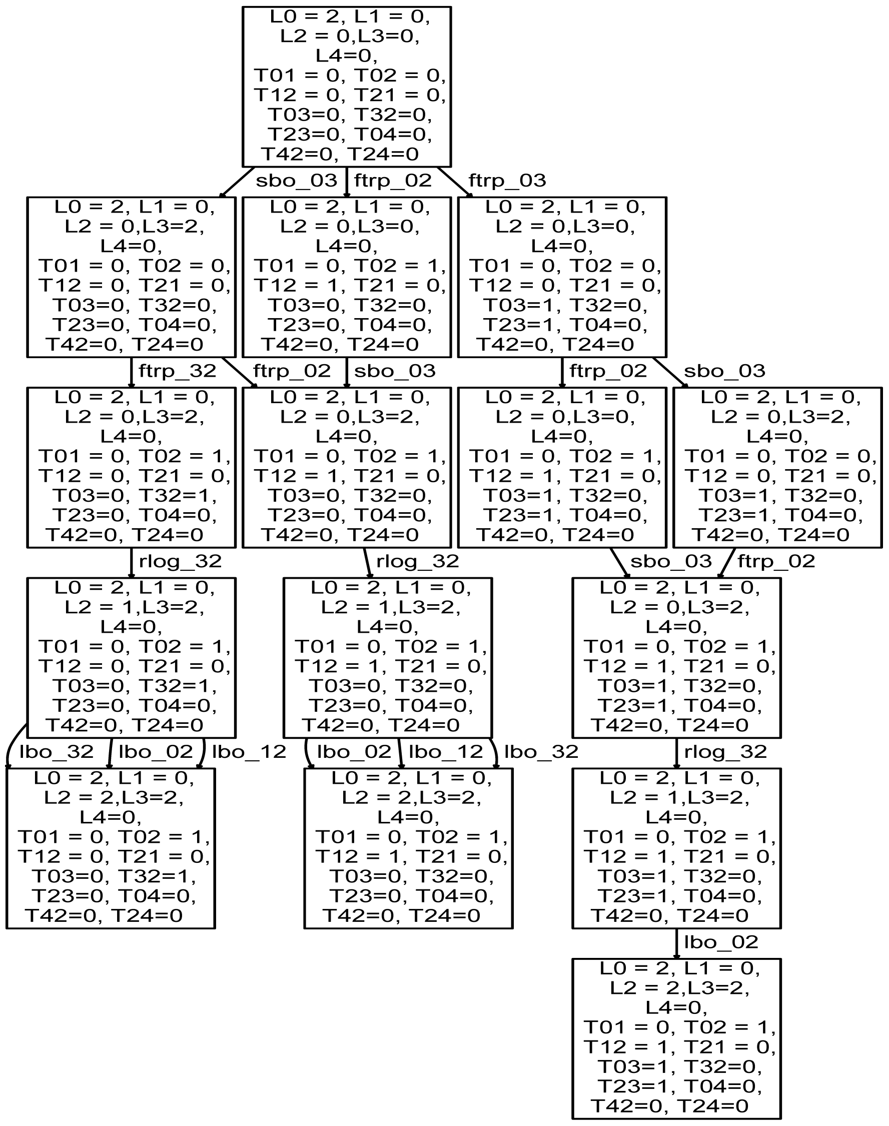

L0 = 2, L1 = 0, L2 = 0,L3=0, L4=0, T01 = 0, T02 = 0, T12 = 0, T21 = 0, T03=0, T32=0, T23=0, T04=0, T42=0, T24=0
sbo_03
ftrp_02
ftrp_03
L0 = 2, L1 = 0, L2 = 0,L3=2, L4=0, T01 = 0, T02 = 0, T12 = 0, T21 = 0, T03=0, T32=0, T23=0, T04=0, T42=0, T24=0
L0 = 2, L1 = 0, L2 = 0,L3=0, L4=0, T01 = 0, T02 = 1, T12 = 1, T21 = 0, T03=0, T32=0, T23=0, T04=0, T42=0, T24=0
L0 = 2, L1 = 0, L2 = 0,L3=0, L4=0, T01 = 0, T02 = 0, T12 = 0, T21 = 0, T03=1, T32=0, T23=1, T04=0, T42=0, T24=0
ftrp_32
ftrp_02
sbo_03
ftrp_02
sbo_03
L0 = 2, L1 = 0, L2 = 0,L3=2, L4=0, T01 = 0, T02 = 1, T12 = 0, T21 = 0, T03=0, T32=1, T23=0, T04=0, T42=0, T24=0
L0 = 2, L1 = 0, L2 = 0,L3=2, L4=0, T01 = 0, T02 = 1, T12 = 1, T21 = 0, T03=0, T32=0, T23=0, T04=0, T42=0, T24=0
L0 = 2, L1 = 0, L2 = 0,L3=0, L4=0, T01 = 0, T02 = 1, T12 = 1, T21 = 0, T03=1, T32=0, T23=1, T04=0, T42=0, T24=0
L0 = 2, L1 = 0, L2 = 0,L3=2, L4=0, T01 = 0, T02 = 0, T12 = 0, T21 = 0, T03=1, T32=0, T23=1, T04=0, T42=0, T24=0
rlog_32
rlog_32
sbo_03
ftrp_02
L0 = 2, L1 = 0, L2 = 1,L3=2, L4=0, T01 = 0, T02 = 1, T12 = 0, T21 = 0, T03=0, T32=1, T23=0, T04=0, T42=0, T24=0
L0 = 2, L1 = 0, L2 = 1,L3=2, L4=0, T01 = 0, T02 = 1, T12 = 1, T21 = 0, T03=0, T32=0, T23=0, T04=0, T42=0, T24=0
L0 = 2, L1 = 0, L2 = 0,L3=2, L4=0, T01 = 0, T02 = 1, T12 = 1, T21 = 0, T03=1, T32=0, T23=1, T04=0, T42=0, T24=0
lbo_32
lbo_02
lbo_12
lbo_02
lbo_12
lbo_32
rlog_32
L0 = 2, L1 = 0, L2 = 2,L3=2, L4=0, T01 = 0, T02 = 1, T12 = 0, T21 = 0, T03=0, T32=1, T23=0, T04=0, T42=0, T24=0
L0 = 2, L1 = 0, L2 = 2,L3=2, L4=0, T01 = 0, T02 = 1, T12 = 1, T21 = 0, T03=0, T32=0, T23=0, T04=0, T42=0, T24=0
L0 = 2, L1 = 0, L2 = 1,L3=2, L4=0, T01 = 0, T02 = 1, T12 = 1, T21 = 0, T03=1, T32=0, T23=1, T04=0, T42=0, T24=0
lbo_02
L0 = 2, L1 = 0, L2 = 2,L3=2, L4=0, T01 = 0, T02 = 1, T12 = 1, T21 = 0, T03=1, T32=0, T23=1, T04=0, T42=0, T24=0

Figure 4.6: 5-Hosts Networked-system generated attack-graph.

### 4.2.2 Water Treatment SCADA CPS Case-study

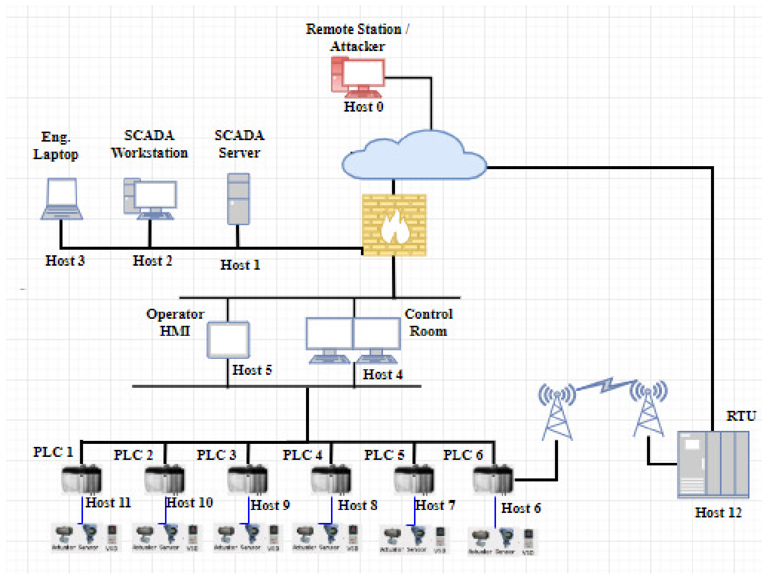


Figure 4.7: SCADA Architecture of Water Treatment System.

The system shown in Figure 4.7 is that of a fully functional water treatment CPS testbed in the iTrust laboratory [59]. The system consists of thirteen hosts: Host-0 is the remote station; host-1 is the SCADA server; host-2 is the SCADA workstation; host-3 is the engineer's laptop; host-4 is the operator's Human Machine Interface (HMI); host-5 is the control room; host-6 to host-11 are the Programmable Logic Controllers (PLCs) that control the physical process through sensors and actuators; host-12 is a Remote Terminal Unit (RTU). We suppose that the attacker has been able to secure access to the remote station (host-0).Host-0, host-1, host-2, host-3 and the remainder of the system are separated by a firewall. The firewall restricts access to host-1, host-2, and host-3 from the remote station. It also allows access to the remainder of the system from host-1, host-2, and host-3. The communications with and among the PLCs go through host-4 and host-5. The PLCs communicate with host-4 and host-5 but not with each other. The remote station (host-0) communicates with the RTU (host-12). the RTU (host-12) communicates with PLC-6.

The components and services of the SCADA system are all products from Siemens. Online databases [87, 88] list that these components may have the following vulnerabilities, which we assume to be the case to illustrate our approach:

1. The SCADA server is a SINEMA server, in which:
   (a) *iws*: The built-in web servers at Port 4999/TCP and Port 80/TCP may allow remote execution of unauthenticated code if the server is accessed by an attacker.
   (b) *lmw*: If the impacted products are not mounted under their default route ("$C:\backslash\backslash ProgramFiles\backslash\backslash *$" or the localized equivalent), local Microsoft Windows operating system users may increase their privileges.
2. The SCADA workstation is a SIMATIC Wincc flexible runtime, in which:
   (a) *rmm*: The SIMATIC Wincc flexible panel remote management module transmits weakly protected credentials across the network. The credentials can be reconstructed by attackers capturing remote management module network traffic.
   (b) *inws*: Siemens Wincc flexible allows remote attackers with user access to inject arbitrary web script or HTML via unspecified vectors.
3. The engineering Laptop runs a Wincc TIA portal, in which:
   (a) *bac*: The application establishes predictable token / cookie authentication values when a user logs in. This can enable an attacker to bypass checks on authentication.
   (b) *inws*: Wincc TIA portal enables remote attackers to use unknown vectors to inject arbitrary web script or HTML.

4. The HMI runs a Wincc runtime advanced, in which
   (a) *rmm*: The SIMATIC Wincc flexible panel remote management module transmits weakly protected credentials across the network. The credentials can be reconstructed by attackers capturing remote management module network traffic.
   (b) *inws*: Siemens Wincc runtime advanced allows remote attackers to inject arbitrary web script or HTML via unspecified vectors.
5. The control center runs a Wincc runtime professional, in which:
   (a) *rmm*: The SIMATIC Wincc flexible panel remote management module transmits weakly protected credentials across the network. The credentials can be reconstructed by attackers capturing remote management module network traffic.
   (b) *inws*: Siemens Wincc flexible allows remote attackers to inject arbitrary web script or HTML via unspecified vectors.
6. The PLCs are Siemens SIMATIC S7-1500 CPU PLC devices, in which
   (a) *rng*: There is insufficient entropy in the random number generator on Siemens SIMATIC S7-1500 CPU PLC systems, making it simpler for remote attackers to defeat cryptographic protection mechanisms and hijack sessions via unknown vectors.
7. The RTU (SICAM RTUs SM-2556 COM Modules)
   (a) *osdi*: The impacted devices' embedded internet server (port 80/tcp) could allow remote attackers to acquire sensitive device data over the network.
   (b) *uxac*: The integrated web server (port 80/tcp) of the affected devices could allow unauthenticated remote attackers to execute arbitrary code on the affected device.

The above listed SCADA system atomic vulnerabilities can be exploited resulting in the following atomic attacks:

1. Remote code execution (*rce*): This attack exploits the vulnerability in Port 4999/TCP and Port 80/TCP embedded SINEMA web servers if the server is accessed by an intruder. This attack provides user access to the attacked host.

2. Unquoted service paths (*usp*): This attack exploits the vulnerability in the operating system of SINEMA server windows that enables a local user to increase their privileges in order to obtain root access to the host being attacked.

3. User credentials construction (*ucc*): This attack exploits SIMATIC Wincc flexible, SIMATIC Wincc advanced, and SIMATIC Wincc professional's weakly protected credentials. This attack provides user access to the attacked host.

4. Cross-site scripting (*xss*): In Wincc TIA portal, SIMATIC Wincc flexible, SIMATIC Wincc advanced, and SIMATIC Wincc professional, this attack exploits the vulnerability of the web script and HTML code injection. This attack provides root access to the host attacked by the attacker.

5. Authentication token/cookie (*atc*): This attack takes advantage of a Wincc TIA portal's predictable authentication token / cookie values. This attack provides root access to the host attacked by the attacker.

6. Cryptographic protection mechanisms (*cpm*): This attack takes advantage of the weakness in the random number generator of the SIMATIC S7-1500 CPU if the attacker has access to the PLCs. This attack provides root access to the host attacked by the attacker.

7. Unauthorized remote snifing (*urs*): This attack exploits the vulnerability of embedded web servers of SICAM RTUs SM-2556 at Port 80/TCP if the server is accessed by an adversary. This attack provides user access to the attacked host.

8. Unauthorized remote execution ($urx$): This attack exploits the vulnerability of embedded web servers of SICAM RTUs SM-2556 at Port 80/TCP if the server is accessed by an adversary. This attack provides root access to the attacked host to the attacker.

The water treatment system can be formally specified as follows:

1. Set of hosts $H = 0, 1, 2, \cdots, 12$; variable $i \in \{0, 1, 2, \cdots, 12\}$ (static parameters).

2. System connectivity, $C \subseteq H \times H$ ; Boolean; $c_{ij} = 1$ iff host $i$ connected to host $j$ (static parameters).

3. System services $S$; Boolean; $s_i = 1$ iff service $s \in \{$ SCADA server is SINEMA, SIMATIC Wincc runtime flexible, Wincc runtime advanced, Wincc runtime professional, Wincc TIA portal, SIMATIC S7-1500 CPU, SICAM RTUs SM-2556 $\}$ is running on host $i$ (dynamic variables).

4. System vulnerabilities $V$ ; Boolean; $v_i = 1$ iff vulnerability $v \in \{$ $iws$, $lmw$, $rmm$, $inws$, $bac$, $rng$, $osdi$, $uxac\}$ exists on host $i$ (static parameters).

5. Attack instances $A_I \subseteq A \times H \times H$; labeled $a_{ij} \equiv$ attack $a$ from source $i$ to target $j$, $a \in \{rce, usp, ucc, xss, atc, cpm, urs, urx\}$ (static parameters).

6. Trust relation $T \subseteq H \times H$ ; Boolean; $t_{ij} = 1$ iff $i$ is trusted by $j$ (dynamic variables).

7. Attacker level of privilege $L$ on host $i$; variable $l_i \in \{none, user, root\}$ (dynamic variables).

8. Attack pre-conditions:

   - $Pre_{(rce_{ij})} \equiv c_{ij} = 1 \wedge (l_i \geqslant user) \wedge (l_j = none) \wedge iws = 1$
   - $Pre_{(usp_{ij})} \equiv c_{ij} = 1 \wedge (l_i \geqslant user) \wedge (l_j < root) \wedge lmw = 1$
   - $Pre_{(ucc_{ij})} \equiv c_{ij} = 1 \wedge (l_i \geqslant user) \wedge (l_j = none) \wedge rmm = 1$
   - $Pre_{(xss_{ij})} \equiv c_{ij} = 1 \wedge (l_i \geqslant user) \wedge (l_j < root) \wedge inws = 1$

- $Pre_{(atc_{ij})} \equiv c_{ij} = 1 \wedge (l_i \geqslant user) \wedge (l_j = none) \wedge bac = 1$
- $Pre_{(cpm_{ij})} \equiv c_{ij} = 1 \wedge (l_i \geqslant user) \wedge (l_j = none) \wedge rng = 1$
- $Pre_{(urs_{ij})} \equiv c_{ij} = 1 \wedge (l_i \geqslant user) \wedge (l_j = none) \wedge osdi = 1$
- $Pre_{(urx_{ij})} \equiv c_{ij} = 1 \wedge (l_i \geqslant user) \wedge (l_j = none) \wedge uxac = 1$

9. Attack post-conditions:

   - $post_{(rce_{ij})} \equiv (l_j = user) \wedge (iws = 0)$
   - $post_{(usp_{ij})} \equiv (l_j = root) \wedge (lmw = 0)$
   - $post_{(ucc_{ij})} \equiv (l_j = user) \wedge (rmm = 0)$
   - $post_{(xss_{ij})} \equiv (l_j = root) \wedge (inws = 0)$
   - $post_{(atc_{ij})} \equiv (l_j = user) \wedge (bac = 0)$
   - $post_{(cpm_{ij})} \equiv (l_j = root) \wedge (rng = 0)$
   - $post_{(urs_{ij})} \equiv (l_j = user) \wedge (osdi = 0)$
   - $post_{(urx_{ij})} \equiv (l_j = root) \wedge (uxac = 0)$

10. Intial states : $l_0 = root \wedge (l_1 = l_2 = \cdots = l_{11} = none) \wedge (\forall ij \in H \times H : t_{ij} = 0) \wedge (iws = lmw = rmm = inws = bac = rng = osdi = uxac = 1)$

    (Initially, the attacker has root privilege on host-0 and no privilege on other hosts, none of the hosts trust each other, and *iws*, *lmw*, *rmm*, *inws*, *bac*, *rng*, *osdi*, and *uxac* are running on the SCADA system components.)

11. The security property $\phi$ of interest is violated if the attacker has the root privilege level on host-4, host-5, or host-6. This can then be described by a Computational Tree Logic (CTL) formula:

$$\neg\phi \equiv AG((l_4 = root) \vee (l_5 = root) \vee (l_6 = root)).$$

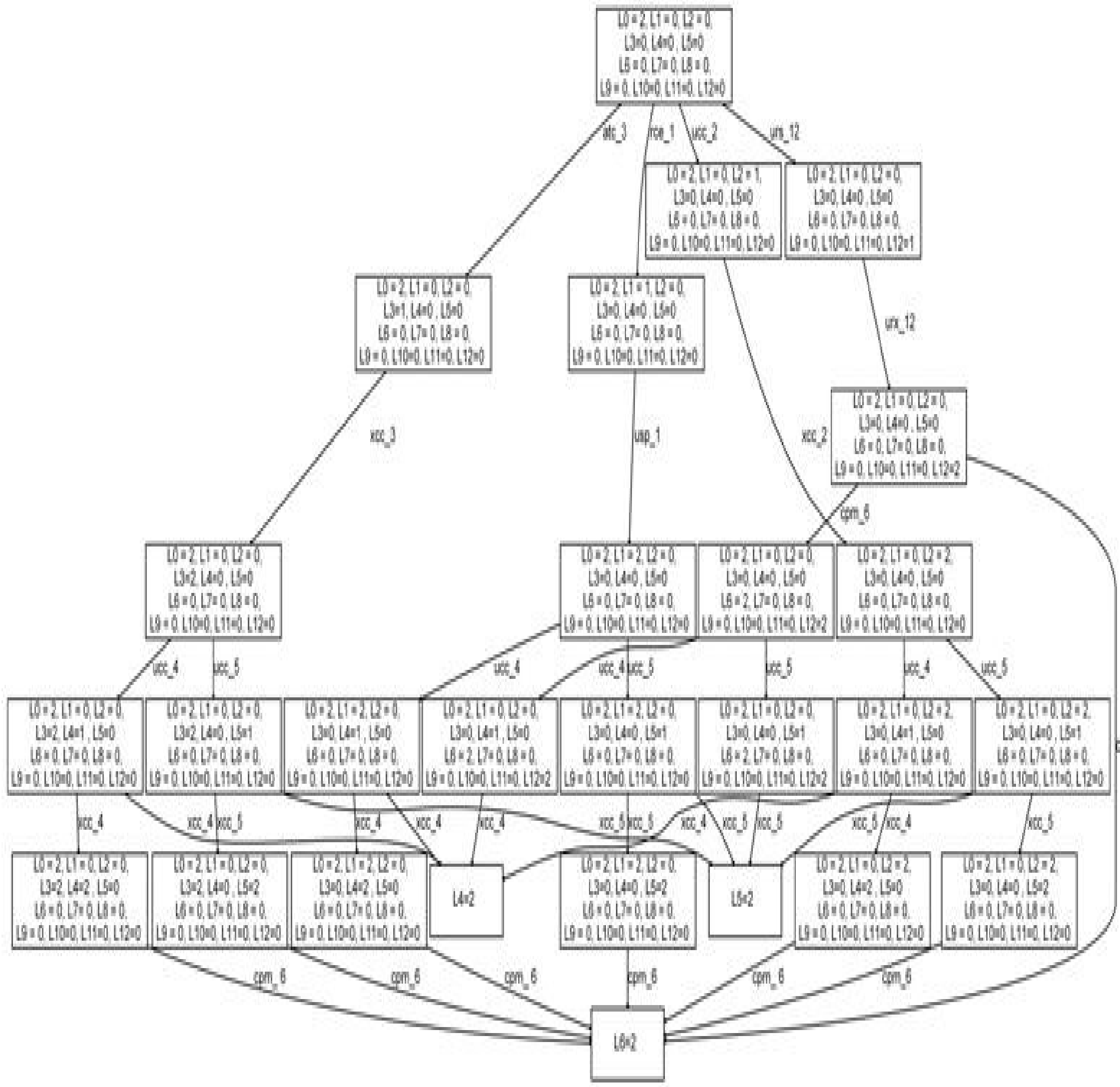


Figure 4.8: Generated Attack-graph for Water Treatment SCADA CPS

We used our previously developed A2G2V tool [89, 63] to generate the attack-graph for compromising this property, as shown in Figure 4.8. The input to A2G2V consists of the network topology, specified in AADL (Architecure Analysis & Development Language) [85], as well as the atomic attack actions associated with each of the network devices (and their pre- and post-conditions), specified in the AGREE Annex [83] of AADL. Note that, the water treatment SCADA system has 12 devices with 23 number of connections among them.

The devices are exposed to 7 different types of vulnerability and subject to 8 different types of atomic attacks per host (listed above). An atomic attack can be mounted on a device to escalate attacker's privilege level on a connected device, under certain pre-conditions. The attack-graph then tracks the dynamically evolving privilege level of the attacker on all the 12 devices, as the atomic attacks occur by exploiting the vulnerabilities of the connected devices. These atomic attacks can be nested to form 150 different attack-sequences (see Figure 4.8) that an attacker may execute to violate the system-level security property (namely, gain root access on host-4, host-5, or host-6), all of which were automatically computed using our A2G2V tool, and that is also graphically displayed automatically.

## 4.3 Results and Discussion

Our implementation of the proposed SCCiMLC computed the critical-attacks set as, $CA = \{rlog\_12\}$ for the 3-hosts system, $CA = \{rlog\_32\}$ for the 5-hosts system, and $CA = \{cpm\_6, xcc\_5, xcc\_4\}$ for the water treatment SCADA CPS, meaning that applying security measures to prevent an attacker from exploiting these attacks, can guarantee the desired system-level security property of interest. It turns out that, a quick analysis confirms that removing any smaller set of atomic attacks, i.e., a proper subset of $CA$ does not render the attack-graph disconnected. Thus the proposed SCCiMLC algorithm was able to find a *minimal* critical-attacks set for the above applications we analyzed, which is quite encouraging.

It can be seen that identifying the critical-attacks set hugely reduces the effort required to secure the system; in the 3-hosts system securing 1 atomic attack ($rlog\_12$) out of $4 \times 2 = 8$ atomic attack actions is required to secure the system; in the 5-hosts system securing 1 atomic attack ($rlog\_32$) out of $4 \times 4 = 16$ atomic attack actions is required to secure the system; in the water treatment SCADA system securing 3 atomic attack ($cpm\_6, xcc\_5, xcc\_4$) out of $8 \times 12 = 96$ atomic attack actions is required to secure the system.

To compare the performance of the proposed SCCiMLC we computed the critical-attacks set using an exact algorithm (our generalization of [56]) and a state-of-art approximation algorithm of [1] (both described in Section 4.1). The exact algorithm identifies the critical-attacks set as $CA = \{rlog\_12\}$ for the 3-hosts system, $CA = \{rlog\_32\}$ for the 5-hosts system, and $CA = \{cpm\_6, ucc\_5, ucc\_4\}$ for the water treatment SCADA CPS. The approximation algorithm identifies the critical-attacks set as $CA = \{sbo\_01, ftrp\_02, ftrp\_01\}$ for the 3-hosts system, $CA = \{sbo\_03, ftrp\_02, ftrp\_03\}$ for the 5-hosts system, and $CA = \{atc\_3, rce\_1, xcc\_2, urx\_12\}$ for the water treatment SCADA CPS. It can be seen that the our proposed algorithm matches the exact one in the size of the MLCs (although the solutions in the case of water treatment SCADA differ), whereas it supersedes the approximation algorithm, which computed label-cuts of larger size in all 3 cases, compared to our proposed SCCiMLC algorithm.

On a standard computer, with Core i5/2.2GHz/4GB RAM running Win 10, our SCCiMLC algorithm took 9 sec to compute the *minimal* critical-attacks set for the 3-hosts system attack-graph, 9 sec to compute the *minimal* critical-attacks set for the 5-hosts system attack-graph, and 15 sec to compute the *minimal* critical-attacks set from the water treatment SCADA system attack-graph. In contrast, the exact algorithm (our generalization of [56]) took 10 min to compute the *minimal* critical-attacks set for the 3-hosts system as well as 5-hosts attack-graph, and 18 min to compute the *minimal* critical-attacks set from the water treatment SCADA system attack-graph. The approximation algorithm of [1] took 9 sec to compute the *minimal* critical-attacks set for the 3-hosts system attack-graph, 9 sec to compute the *minimal* critical-attacks set for the 5-hosts system attack-graph, and 16 sec to compute the *minimal* critical-attacks set from the water treatment SCADA system attack-graph.

It can be noted that the proposed SCCiMLC algorithm produced very promising results; the size of the cuts it computed matched the size of the solution computed using the exact algorithm: In the computer network case studies, the SCCiMLC algorithm solution matched that computed using the exact algorithm. For the water treatment system, the identified labels differ, while the size of the label-cut reported was still 3 as in the case of the exact algorithm. However the speed of SCCiMLC is $>$ 65 times faster. Conversely, the size of the label-cut computed by the [1] approximation algorithm was always, although the speed was comparable. In short, in the case-studies examined, the proposed SCCiMLC achieved the accuracy of the exact algorithm and yet the speed of the approximation algorithm.

# CHAPTER 5. SUMMARY AND FUTURE WORK

SCADA/ICS systems control most critical infrastructures and industrial manufacturing processes so compromising such systems could have a tremendous financial effect and possibly even represent danger to human life. There is ongoing research directed toward securing SCADA/ICS systems from different aspects of risk, but there has been no fully secure SCADA/ICS system, so improvement is still possible. To improve SCADA/ICS security, we proposed a framework with supporting components to comprehensively cover SCADA/ICS system vulnerability problems that can be addressed through system component recognition, component vulnerability scan, vulnerability assessment, global vulnerability assessment, and vulnerability resolution.

1. To address system component recognition, We presented a first-of-a-kind SCADA/ICS system device identification approach, based on passive fingerprinting of network data, using a two-stage process. The first stage is a learning stage during which a reference fingerprints database was created. The second stage is a two-step online stage: the first step identifies the control hierarchy based on the SCADA/ICS communication patterns; the second step identifies a device by creating its fingerprint and comparing it to the reference database of the first stage to identify the device type, manufacturer, and model. Identifying the control hierarchy enhances the device recognition capability since it enables further discrimination. Also, we fully implemented our approach in Python and demonstrated the validity of the proposed solution through a real-life Water Treatment SCADA system of the iTrust Lab.

Due to the inherent control-hierarchy within the SCADA/ICS networks, the data from a device does not go beyond its parent. This means that to achieve a complete recognition, the data must be collect at each device or router. The latter results in reduction in data collection points, but requires the analysis of the portion of the data unaffected by the router actions. To this end, we proposed a first-of-a-kind device recognition approach based on analysis of passively collected Modbus-data. We also fully implemented this approach in Python and demonstrated the validity of the proposed solution through a simulated water treatment SCADA system.

We encapsulated these implementations in our ANDVI (Automated Network Device and Vulnerability Identification) tool and further integrated the tool with the existing vulnerability databases to automatically map the discovered devices to their known vulnerabilities.

2. To address global vulnerability assessment,We presented a first general model-based automated attack graph generator and visualizer algorithm and its C-based implementation tool A2G2V, and also illustrated it through three examples. The key to automation is the employment of an architectural description language to capture the security-related details of a networked system, an automated encoding of the latest counterexample to relax the current specification, and an iterative adjustment of the search depth of a bounded model-checker to identify all the acyclic counterexamples. Our algorithm formally models the system using AADL, iteratively model-checks the system with JKind model-checker to generate attack paths, and combines the attack paths into an attack-graph using GraphViz.

   Our algorithm is sound and complete, works for any security property, and terminates for any finite state-space. The latter is to be expected for practical applications, where number of dynamic state-variables is finite and those evolve over finitely many values. From mathematical perspective, one may consider infinite state-space, but then in that setting, finding even a single counterexample is undecidable.

3. Finally, to address component vulnerability We presented a linear complexity, automated critical-attacks set identification approach, by introducing the notion of a strongly connected components-induced min label-cut (SCCiMLC) to approximate a solution to the MLC problem. For this, we used the graph over the vertex set of strongly connected components to obtain an abstracted attack-graph (a tree graph), and next iteratively identified, starting for the terminal node, its backward reachable nodes of higher and higher numbers of hops, and used their outgoing edges to form the cuts and theirs labels to form the label-cuts. The smallest of these labels then yielded a SCCiMLC.

   We extended our tool-chain ANDVI (Automated Network Device and Vulnerability Identification) [90] and A2G2V (Automated Attack Graph Generation and Visualization) [89, 63] (seeFigure 2.1) to analyze an automatically generated attack-graph (constructed from passive observations of network packets to identify devices and their connectivity), to then obtain a SCCiMLC. This was done by automatically constructing the abstracted attack-graph over the SCCs, and performing the backward reachability over the abstracted graph to find a SCCiMLC. The complexity of finding SCCs, constructing the abstracted graph over the SCCs, and backward reachability over those are all of *linear-time complexity* in the size of the original attack-graph. We implemented our algorithm and demonstrated the validity of the proposed approach through its application to two realistic computer networks and a real-world water treatment SCADA system testbed from the iTrust Lab. The results showed that only a fraction of the attacks among all possible ones formed a critical set (e.g., in case of the water treatment SCADA, 3 out of 96), implying that identifying a critical attacks-set can be hugely beneficial in securing complex networked systems while faced with limited resources (budget and as well as the amount of downtime for maintenance).

   To measure the proposed algorithm performance, we implemented an exact algorithm and a state-of-art approximation algorithm taken from the literature, and compared their performance against the SCCiMLC algorithm. The size of the cut computed by the SCCiMLC matched that computed by the exact algorithm, while the speed matched that of the approximation algorithm. In contrast, the approximation algorithm cut size was always larger that that of the SCCiMLC (or the exact algorithm). In the examples considered, ¿65 times speedup was observed, without any loss of accuracy.

### 5.0.1 Future Work

1. ANDVI: Automated Network Device and Vulnerability Identification in SCADA/ICS by Passive Monitoring

   Discovering the devices included in SCADA/ICS systems is an essential first step toward improving their overall cybersecurity. A next step would be to implement the proposed approach along with systems security analysis and mitigation tools, to enhance their overall security and defense strategies against potential cyberattacks.

2. A2G2V: Automatic Attack Graph Generation and Visualization and its Ap-plications to Computer and SCADA Networks

   The availability of the system model is a crucial to the generation of the attack-graphs. In general, it requires a certain one-time modeling effort to obtain the system description for components, connectivity, services, and their vulnerabilities. For example to encode the water treatment CPS in AADL and the AGREE Annex, it took about an hour. It may be possible to automate this by providing a graphical support (where a user will draw the architecture and specify the atomic attacks for each of the components, along with their pre- and post-conditions, and the overall security property), while a tool will automatically generate the AADL model along with the AGREE Annex. Thus, a future direction is to develop a graphical model capturing tool.

   Our attack-graph generation algorithm is written in C-code, which interacts with the model-checker JKind for the generation of the attack-paths one at a time, and with another tool Graphviz for visual display of the attack-graph. Much time is consumed in interaction among the tools, in reading the outputs, parsing and processing, and feeding back as inputs. If all tools are integrated into a single code, the computation will become more efficient, and may be pursued as a future direction.

3. Critical-Attacks Set Identification in Attack-Graphs for Computer and SCADA Networks

   Identifying the critical-attacks set in SCADA/ICS/Computer-networks systems is an essential first step toward improving their overall cybersecurity in helping system administrators optimally allocate their resources for enhancing security defenses. A next step would be to integrate the proposed approach with a runtime security defense-patch implementation tool to achieve an optimized resources based and defense implementation for enhancing the overall defense strategies and security against potential cyberattacks.

# BIBLIOGRAPHY

[1] T. Dutta, L. S. Heath, V. A. Kumar, and M. V. Marathe, "Labeled cuts in graphs," *Theoretical Computer Science*, vol. 648, pp. 34 – 39, 2016.

[2] A. Sajid, H. Abbas, and K. Saleem, "Cloud-assisted iot-based scada systems security: A review of the state of the art and future challenges," *IEEE Access*, vol. 4, pp. 1375–1384, 2016.

[3] A. Hassanzadeh, S. Modi, and S. Mulchandani, "Towards effective security control assignment in the industrial internet of things," in *Internet of Things (WF-IoT), 2015 IEEE 2nd World Forum on*, pp. 795–800, IEEE, 2015.

[4] N. ICS-CERT, "Ics-cert year in review (2015)," 2015.

[5] S. Huang, C.-J. Zhou, S.-H. Yang, and Y.-Q. Qin, "Cyber-physical system security for networked industrial processes," *International Journal of Automation and Computing*, vol. 12, no. 6, pp. 567–578, 2015.

[6] A. Cardenas, S. Amin, B. Sinopoli, A. Giani, A. Perrig, and S. Sastry, "Challenges for securing cyber physical systems," in *Workshop on future directions in cyber-physical systems security*, p. 5, 2009.

[7] U. NewsTrack, "Cyber malfunction halts nuclear plant." link.galegroup.com/apps/doc/A179774603/BIC1?u=iastu_main&xid=42ee4dce, 2008. Accessed: 2017-15-05.

[8] C. G. Chittester and Y. Y. Haimes, "Risks of terrorism to information technology and to critical interdependent infrastructures," *J Homel Secur Emerg Manag*, vol. 1, no. 4, p. 402, 2004.

[9] M. R. Permann and K. Rohde, "Cyber assessment methods for scada security," in *15th annual joint ISA POWID/EPRI controls and instrumentation conference, Nashville, TN*, 2005.

[10] M. A. McQueen, W. F. Boyer, M. A. Flynn, and G. A. Beitel, "Quantitative cyber risk reduction estimation methodology for a small scada control system," in *System Sciences, 2006. HICSS'06. Proceedings of the 39th Annual Hawaii International Conference on*, vol. 9, pp. 226–226, IEEE, 2006.

[11] Y. Jiaxi, M. Anjia, and G. Zhizhong, "Vulnerability assessment of cyber security in power industry," in *Power Systems Conference and Exposition, 2006. PSCE'06. 2006 IEEE PES*, pp. 2200–2205, IEEE, 2006.

[12] D. I. Gertman, R. Folkers, and J. Roberts, "Scenario-based approach to risk analysis in support of cyber security," tech. rep., American Nuclear Society, 555 North Kensington Avenue, La Grange Park, IL 60526 (United States), 2006.

[13] S. C. Patel, J. H. Graham, and P. A. Ralston, "Quantitatively assessing the vulnerability of critical information systems: A new method for evaluating security enhancements," *International Journal of Information Management*, vol. 28, no. 6, pp. 483–491, 2008.

[14] M. H. Henry, R. M. Layer, K. Z. Snow, and D. R. Zaret, "Evaluating the risk of cyber attacks on scada systems via petri net analysis with application to hazardous liquid loading operations," in *Technologies for Homeland Security, 2009. HST'09. IEEE Conference on*, pp. 607–614, IEEE, 2009.

[15] F. Baiardi, C. Telmon, and D. Sgandurra, "Hierarchical, model-based risk management of critical infrastructures," *Reliability Engineering & System Safety*, vol. 94, no. 9, pp. 1403–1415, 2009.

[16] M. H. Henry and Y. Y. Haimes, "A comprehensive network security risk model for process control networks," *Risk analysis*, vol. 29, no. 2, pp. 223–248, 2009.

[17] E. LeMay, W. Unkenholz, D. Parks, C. Muehrcke, K. Keefe, and W. H. Sanders, "Adversary-driven state-based system security evaluation," in *Proceedings of the 6th International Workshop on Security Measurements and Metrics*, p. 5, ACM, 2010.

[18] C.-W. Ten, G. Manimaran, and C.-C. Liu, "Cybersecurity for critical infrastructures: Attack and defense modeling," *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 40, no. 4, pp. 853–865, 2010.

[19] J.-G. Song, J.-W. Lee, C.-K. Lee, K.-C. Kwon, and D.-Y. Lee, "A cyber security risk assessment for the design of i&c systems in nuclear power plants," *Nuclear Engineering and Technology*, vol. 44, no. 8, pp. 919–928, 2012.

[20] S. Kriaa, M. Bouissou, and L. Piètre-Cambacédès, "Modeling the stuxnet attack with bdmp: Towards more formal risk assessments," in *Risk and Security of Internet and Systems (CRiSIS), 2012 7th International Conference on*, pp. 1–8, IEEE, 2012.

[21] J. Yan, M. Govindarasu, C.-C. Liu, and U. Vaidya, "A pmu-based risk assessment framework for power control systems," in *Power and Energy Society General Meeting (PES), 2013 IEEE*, pp. 1–5, IEEE, 2013.

[22] R. Hewett, S. Rudrapattana, and P. Kijsanayothin, "Cyber-security analysis of smart grid scada systems with game models," in *Proceedings of the 9th Annual Cyber and Information Security Research Conference*, pp. 109–112, ACM, 2014.

[23] P. S. Woo and B. H. Kim, "A study on quantitative methodology to assess cyber security risk of scada systems.," *Advanced Materials Research*, 2014.

[24] S. Gordeychik, "Scada strangelove or: How i learned to start worrying and love nuclear plants," 2013.

[25] J. C. Matherly, "Shodan the computer search engine," *Available at [Online]: http://www. shodanhq. com/help*, 2009.

[26] J. B. Radvanovsky and J. Brodsky, "Project shine: What we discovered and why you should care," *10th Sans ICS Security Summit, Orlando, FL*, 2015.

[27] T. Kiravuo, S. Tiilikainen, M. Särelä, and J. Manner, "Peeking under the skirts of a nation: finding ics vulnerabilities in the critical digital infrastructure," in *European Conference on Cyber Warfare and Security*, p. 137, Academic Conferences International Limited, 2015.

[28] J. François, H. Abdelnur, R. State, and O. Festor, "Ptf: Passive temporal fingerprinting," in *12th IFIP/IEEE International Symposium on Integrated Network Management (IM 2011) and Workshops*, pp. 289–296, IEEE, 2011.

[29] A. R. Beyah, I. David Formby, and P. Srinivasan, "Device fingerprinting for cyber-physical systems," Feb. 15 2018. US Patent App. 15/556,136.

[30] S. Jeon, J.-H. Yun, S. Choi, and W.-N. Kim, "Passive fingerprinting of scada in critical infrastructure network without deep packet inspection," *arXiv preprint arXiv:1608.07679*, 2016.

[31] M. Caselli, D. Hadžiosmanović, E. Zambon, and F. Kargl, "On the feasibility of device fingerprinting in industrial control systems," in *International Workshop on Critical Information Infrastructures Security*, pp. 155–166, Springer, 2013.

[32] R. Bodenheim, J. Butts, S. Dunlap, and B. Mullins, "Evaluation of the ability of the shodan search engine to identify internet-facing industrial control devices," *International Journal of Critical Infrastructure Protection*, vol. 7, no. 2, pp. 114–123, 2014.

[33] R. Vigo, F. Nielson, and H. R. Nielson, "Automated generation of attack trees," in *Computer Security Foundations Symposium (CSF), 2014 IEEE 27th*, pp. 337–350, IEEE, 2014.

[34] J. B. Hong, D. S. Kim, and T. Takaoka, "Scalable attack representation model using logic reduction techniques," in *Trust, Security and Privacy in Computing and Communications (TrustCom), 2013 12th IEEE International Conference on*, pp. 404–411, IEEE, 2013.

[35] C.-S. Cho, W.-H. Chung, and S.-Y. Kuo, "Cyberphysical security and dependability analysis of digital control systems in nuclear power plants," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 46, no. 3, pp. 356–369, 2016.

[36] M. G. Ivanova, C. W. Probst, R. R. Hansen, and F. Kammüller, "Transforming graphical system models to graphical attack models," in *International Workshop on Graphical Models for Security*, pp. 82–96, Springer, 2015.

[37] O. Gadyatskaya, "How to generate security cameras: towards defence generation for socio-technical systems," in *International Workshop on Graphical Models for Security*, pp. 50–65, Springer, 2015.

[38] Q. Zhang, C. Zhou, N. Xiong, Y. Qin, X. Li, and S. Huang, "Multimodel-based incident prediction and risk assessment in dynamic cybersecurity protection for industrial control systems," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 46, no. 10, pp. 1429–1444, 2016.

[39] P. Ammann, D. Wijesekera, and S. Kaushik, "Scalable, graph-based network vulnerability analysis," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pp. 217–224, ACM, 2002.

[40] P. Ammann, J. Pamula, R. Ritchey, and J. Street, "A host-based approach to network attack chaining analysis," in *Computer Security Applications Conference, 21st Annual*, pp. 10–pp, IEEE, 2005.

[41] K. Ingols, R. Lippmann, and K. Piwowarski, "Practical attack graph generation for network defense," in *Computer Security Applications Conference, 2006. ACSAC'06. 22nd Annual*, pp. 121–130, IEEE, 2006.

[42] J. Ma, Y. Wang, J. Sun, and X. Hu, "A scalable, bidirectional-based search strategy to generate attack graphs," in *Computer and Information Technology (CIT), 2010 IEEE 10th International Conference on*, pp. 2976–2981, IEEE, 2010.

[43] K. Kaynar and F. Sivrikaya, "Distributed attack graph generation," *IEEE Transactions on Dependable and Secure Computing*, vol. 13, no. 5, pp. 519–532, 2016.

[44] R. W. Ritchey and P. Ammann, "Using model checking to analyze network vulnerabilities," in *Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on*, pp. 156–165, IEEE, 2000.

[45] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. M. Wing, "Automated generation and analysis of attack graphs," in *Security and privacy, 2002. Proceedings. 2002 IEEE Symposium on*, pp. 273–284, IEEE, 2002.

[46] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. M. Wing, "Automated generation and analysis of attack graphs," in *Proceedings 2002 IEEE Symposium on Security and Privacy*, pp. 273–284, IEEE, 2002.

[47] S. Jha, O. Sheyner, and J. Wing, "Two formal analyses of attack graphs," in *Computer Security Foundations Workshop, 2002. Proceedings. 15th IEEE*, pp. 49–63, IEEE, 2002.

[48] P. Zhang, J.-Y. Cai, L.-Q. Tang, and W.-B. Zhao, "Approximation and hardness results for label cut and related problems," *Journal of Combinatorial Optimization*, vol. 21, no. 2, pp. 192–208, 2011.

[49] S. Noel, S. Jajodia, B. O'Berry, and M. Jacobs, "Efficient minimum-cost network hardening via exploit dependency graphs," in *19th Annual Computer Security Applications Conference, 2003. Proceedings.*, pp. 86–95, IEEE, 2003.

[50] L. Wang, S. Noel, and S. Jajodia, "Minimum-cost network hardening using attack graphs," *Computer Communications*, vol. 29, no. 18, pp. 3812–3824, 2006.

[51] C.-S. Cho, W.-H. Chung, and S.-Y. Kuo, "Cyberphysical security and dependability analysis of digital control systems in nuclear power plants," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 46, no. 3, pp. 356–369, 2015.

[52] R. E. Sawilla and X. Ou, "Identifying critical attack assets in dependency attack graphs," in *European Symposium on Research in Computer Security*, pp. 18–34, Springer, 2008.

[53] R. Sawilla and C. Burrell, *Course of action recommendations for practical network defence.* Defence R&D Canada-Ottawa, 2009.

[54] M. Alhomidi and M. Reed, "Finding the minimum cut set in attack graphs using genetic algorithms," in *2013 International Conference on Computer Applications Technology (ICCAT)*, pp. 1–6, IEEE, 2013.

[55] R. Hassin, J. Monnot, and D. Segev, "Approximation algorithms and hardness results for labeled connectivity problems," *Journal of Combinatorial Optimization*, vol. 14, no. 4, pp. 437–453, 2007.

[56] P. Zhang and B. Fu, "The label cut problem with respect to path length and label frequency," *Theoretical Computer Science*, vol. 648, pp. 72–83, 2016.

[57] T. Nelso and M. Chaffin, "Common cybersecurity vulnerabilities in industrial control systems," *Control Systems Security Program. Washington DC: Department of Homeland Security (DHS), National Cyber Security Division*, 2011.

[58] S. A. Oliva and B. Crowe, "Network system and method for automatic discovery of topology using overhead bandwidth," Nov. 25 2003. US Patent 6,654,802.

[59] iTrust, "Secure water treatment." https://itrust.sutd.edu.sg/testbeds/secure-water-treatment-swat/, 2018. accessed:6/5/2018.

[60] L. Hansson, "Capture files from 4sics geek lounge." https://www.netresec.com/index.ashx?page=PCAP4SICS, 2018. accessed:9/4/2018.

[61] T. Yardley, "Ics pcaps." https://github.com/ITI/ICS-Security-Tools/tree/master/pcaps, 2018. accessed:9/4/2018.

[62] K. Stouffer, J. Falco, and K. Scarfone, "Guide to industrial control systems (ics) security," *NIST special publication*, vol. 800, no. 82, pp. 16–16, 2011.

[63] A. T. Al Ghazo, M. Ibrahim, H. Ren, and R. Kumar, "A2G2V: Automatic attack graph generation and visualization and its applications to computer and scada networks," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, pp. 1–11, 2019.

[64] A. T. A. Ghazo and R. Kumar, "Identification of critical-attacks set in an attack-graph," *2019 IEEE 10th Annual Ubiquitous Computing, Electronics Mobile Communication Conference (UEMCON)*, 2019.

[65] T. Bangemann, S. Karnouskos, R. Camp, O. Carlsson, M. Riedl, S. McLeod, R. Harrison, A. W. Colombo, and P. Stluka, "State of the art in industrial automation," in *Industrial Cloud-Based Cyber-Physical Systems*, pp. 23–47, Springer, 2014.

[66] Wireshark, "Oui lookup tool," 2018. accessed:9/4/2018.

[67] PLCopen, "International standard iec 61131 applies to programmable controllers (plc)." `http://www.plcopen.org/pages/tc1_standards/iec61131-1/`, 2013. accessed:9/4/2018.

[68] I. E. C. (IEC), "Iec 62264-2 enterprise-control system integration." `http://www.plcopen.org/pages/tc1_standards/iec61131-1/`, 2016. accessed:9/4/2018.

[69] Wireshark, "tshark dump and analyze network traffic." `https://www.wireshark.org/docs/man-pages/tshark.html`, 2018. Accessed:9/4/2018.

[70] A. Keliris and M. Maniatakos, "Remote field device fingerprinting using device-specific modbus information," in *2016 IEEE 59th international Midwest symposium on circuits and systems (MWSCAS)*, pp. 1–4, IEEE, 2016.

[71] M. T. LLC, "Modbus register mapping for the plc io interface for ge genius io." `https://www.mynah.com/content/ge-legacy-rio-interface-module-memory-map`, 2017. Accessed:10/17/2019.

[72] Schneider, "Modbus protocol and register map for ion devices." `https://www.ccontrol.com/support/dp/ION_Meter_Modbus.pdf`, 2011. Accessed:10/17/2019.

[73] Siemens, "S7 - open modbus / tcp communication." `https://w3.siemens.com/mcms/topics/en/siplus/ric-telecontrol/Documents/ric-docu/modbus-tcp_funktionsbeschreibung_en.pdf`, 2008. Accessed:10/17/2019.

[74] Siemens, "Simatic s7-plcsim v5.4 sp8." `https://support.industry.siemens.com/cs/document/109750064/trial-software-simatic-s7-plcsim-v5-4-sp8?dti=0&lc=en-WW`, 2019. Accessed:10/22/2019.

[75] T. Wiens, "Nettoplcsim network extension for plcsim." `https://usermanual.wiki/Document/NetToPLCsimManualen.1468207178/view`, 2018. Accessed:10/22/2019.

[76] O. Sheyner and J. Wing, "Tools for generating and analyzing attack graphs," in *International Symposium on Formal Methods for Components and Objects*, pp. 344–371, Springer, 2003.

[77] D.-G. Feng, Y. Zhang, and Y.-Q. Zhang, "Survey of information security risk assessment," *Journal-China Institute of Communications*, vol. 25, no. 7, pp. 10–18, 2004.

[78] C. Chai, X. Liu, W. Zhang, R. Deters, D. Liu, D. Dyachuk, Y. Tu, and Z. Baber, "Social network analysis of the vulnerabilities of interdependent critical infrastructures," *International journal of critical infrastructures*, vol. 4, no. 3, p. 256, 2008.

[79] A. M. Haidar, A. Mohamed, and A. Hussain, "Vulnerability assessment of power system using various vulnerability indices," in *Research and Development, 2006. SCOReD 2006. 4th Student Conference on*, pp. 224–229, IEEE, 2006.

[80] H. Ren, J. Huang, S. Jiang, and R. Kumar, "A new abstraction-refinement based verifier for modular linear hybrid automata and its implementation," in *Networking, Sensing and Control (ICNSC), 2014 IEEE 11th International Conference on*, pp. 30–35, IEEE, 2014.

[81] H. Ren, J. Huang, s. Jiang, and R. Kumar, "Verification using counterexample fragment based specification relaxation: case of modular/concurrent linear hybrid automata," *IET Cyber-Physical Systems: Theory & Applications*, vol. 2, no. 2, pp. 65–74, 2017.

[82] S. E. I. SEI, "Architecture analysis and design language.." http://standards.sae.org/as5506/, 2004. accessed:1/11/2018.

[83] Rockwell-Collins and Uof-Minnesota, "The assume guarantee reasoning environment.." http://standards.sae.org/as5506/, 2016. accessed:1/11/2018.

[84] A. Gacek, "Jkind." http://loonwerks.com/tools/jkind.html, 2015. accessed:1/11/2018.

[85] Carnegie-Mellon-University, "Open source aadl tool environment for the sae architecture analysis and design language (aadl)." http://osate.org/about-osate.html, 2016. accessed:1/11/2018.

[86] P. Schnoebelen, "The complexity of temporal logic model checking.," *Advances in modal logic*, vol. 4, no. 393-436, p. 35, 2002.

[87] CVE, "Siemens common vulnerabilities." https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=siemens, 2018. accessed:5/25/2018.

[88] D. of Homeland Security, "ICS-CERT." https://search.usa.gov/search?utf8=us-cert-ics&sort_by=&query=siemens, 2018. accessed:5/25/2018.

[89] A. T. Al Ghazo, M. Ibrahim, H. Ren, and R. Kumar, "A2G2V: Automated attack graph generator and visualizer," in *Proceedings of the 1st ACM MobiHoc Workshop on Mobile IoT Sensing, Security, and Privacy*, p. 3, ACM, 2018.

[90] A. T. A. Ghazo and R. Kumar, "Ics/scada device recognition: A hybrid communication-patterns and passive-fingerprinting approach," *2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pp. 19–24, 2019.

[91] R. Gillmann, "0/1-polytopes: typical and extremal properties," 2007.

[92] F. Stallmann, "masters-thesis-ellipsoid." https://github.com/mrflory/masters-thesis-ellipsoid, 2015. accessed:9/18/2019.