Systematic Vulnerability Evaluation of Interoperable Medical Device System using Attack Trees

by

Jian Xu

A Thesis

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the requirements for the

Degree of Master of Science

in

Computer Science

by

December 2015

APPROVED:

Professor Krishna Kumar Venkatasubramanian, Major Thesis Advisor

Professor Craig C. Shue, Thesis Reader

Professor Craig E. Wills, Head of Department

Abstract

Security for medical devices has gained some attractions in the recent years following some well-publicized attacks on individual devices, such as pacemakers and insulin pumps. This has resulted in solutions being proposed for securing these devices, usually in stand-alone mode. Medical devices are however becoming increasingly interconnected and interoperable as a way to improve patient safety, decrease false alarms, and reduce clinician cognitive workload. Given the nature of interoperable medical devices (IMDs), attacks on IMDs can have devastating consequences. This work outlines our effort in understanding the threats faced by IMDs, an important first step in eventually designing secure interoperability architectures.

A useful way of performing threat analysis of any system is to use attack trees. Attack trees are conceptual, multi-leveled diagrams showing how an asset, or target, might be attacked. They provide a formal, methodical way of describing the threats to a system. Developing attack trees for any system is however non-trivial and requires considerable expertise in identifying the various attack vectors. IMDs are typically deployed in hospitals by clinicians and clinical engineers who may not posses such expertise. We therefore develop a methodology that will enable the automated generation of attack trees for IMDs based on a description of the IMD operational workflow and list of safety hazards that need to be avoided during its operation. Additionally, we use the generated attack trees to quantify the security condition of the IMD instance being analyzed. Both these pieces of information can be provided by the users of IMDs in a care facility. The contributions of this paper are: (1) a methodology for automated generation of attack trees for IMDs using process modeling and hazard analysis, and (2) a demonstration of the viability of the methodology for a specific IMD setup called Patient Controlled Analgesia (PCA- IMD), which is used for delivering pain medication to patients in hospitals.

Acknowledgments

I would like to express my deepest gratitude to my advisor, Prof. Krishna Venkatasubramanian, for his support and guidance throughout the research. Without him, I would not have begun the journey through graduate school and would not be where I am today. In times of peril, his presence and expertise always helped guide me and is truly appreciated. I will forever be grateful for this life changing experience.

I would also like to extend my appreciation to my reader, Prof. Craig Shue. His time and effort has given me opportunities and experiences that I would not have otherwise had. His perspective has also helped to provide me with a better understanding of my research.

Both the PEDS and ALAS research groups have provided useful and intellectually stimulating information and feedback. Our regular meetings have helped cultivate a truly beneficial and unique learning experience.

Without my parents, I would not be where or who I am today. They have always provided a positive attitude and support throughout all my endeavors. Finally, I would like to thank my best friend Hang Cai. Whenever I got frustrated and exhausted, I felt much better after asking him “How is your Ph.D. thesis going?” Big comfort, buddy!

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Recent years have seen rapid growth in the field of interoperable medical devices (IMDs) [2]. IMDs are medical cyber-physical systems that enable effective patient care by coordinating patient side medical devices in a clinically meaningful manner. IMDs have the potential to provide many clinical benefits such as a decrease in false alarms and real-time medication interaction checking [2]. Given the safety-critical nature of the IMDs, understanding the security threats that IMDs can be subjected to is essential. In the IMD context each threat has the potential to cause physical harm to the patients either in the short-term (untimely actuation) or long-term (incorrect diagnosis and treatment).

A standard useful way to performing security analysis of any system is to use attack trees. Attack trees are a systematic way of characterizing the security of a system based on varying attacks [30]. They describe the attacks on a system by first identifying the goal of the attack as a root node of a tree. The way the adversary reaches this goal interactively and incrementally is expressed using the intermediate leaf nodes in the three. Each path in the attack tree from the leaf node all the way to the root node describes a unique attack on the system [23]. The advantage of attack trees is that it provides a graphical view of the threats on the system that can be easily understood and acted upon by engineers. However, identifying the security threats to any system using attack trees requires considerable expertise in threat analysis. IMDs are typically deployed in hospitals by clinicians and clinical engineers who may not posses such expertise. Therefore, it is imperative to develop solutions that can generate attack trees for IMDs *automatically.*

Methods for automatically generating attack trees has been studied before, mostly in the context of network security [28], [31]. The typical approach is to develop a model of the system using formal methods tools (e.g, SPIN) and to generate attack trees from the resultant specification. These approaches were designed for security experts who manage large-scale enterprise networks and require considerable knowledge of formal methods. Using such approaches for IMDs is not very practical, as it essentially replaces the need for area of expertise (i.e, threat analysis) with the need to develop expertise in another area (i.e., formal methods). As IMDs are deployed on-demand and managed by clinicians and clinical engineers in a care facility, existing approaches for attack tree generation may not be viable. We therefore need a methodology that is easy for clinical engineers and clinicians to understand use.

In this work, we present a methodology for generating attack trees for IMDs that takes two inputs: operational workflow of the IMD and a hazard analysis of the IMD in question. To describe the operational workflow of the IMD, we use a process modeling tool. Hazards can be thought of as system states that are inherently unsafe for the user. Hazard analysis involves identifying system states that will eventually lead to physical harm for the patient. Both these inputs can be easily described by the clinical staff, as opposed to an abstract mathematical representation of the workflow that formal methods based approaches require. Once the process model is built and hazards identified, we generate a number of faults trees for the system with the goal of enabling the hazards. A fault tree is a top down deductive failure analysis in which an undesired state of a system is analyzed using Boolean logic to combine a series of lower level events. Each fault tree generated in the last step indicates a path to one specific hazard. We therefore combine the hazards in various ways to create an attack tree that meets a specific attack goal. Attack trees are similar to fault trees except they connect one or more attack (instead of faults) toward an attack goal (i.e., over-infusion of medication). Once the attack tree is available, we quantify the IMD's security condition that describes how vulnerable the IMD is. This helps us identify the most vulnerable component(s) throughout the whole IMDs system. In order to validate the effectiveness of our methodology, we apply it to a specific IMD setup for performing pain management using patient controlled analgesia (PCA). There are couples of reasons that make us choose PCA as our case-study. First, PCA is responsible for a large number of treatment errors in real hospital settings. One study estimated that there are between 600,000 to 2,000,000 million adverse events in U.S. hospitals every year related to PCA [40]. Second, another research indicates that patient controlled analgesia pumps are one of the most insecure medical devices ever seen [8]. It is therefore necessary to understand PCA operations and find a way to improve its security. Third, security has to be accurately guaranteed in medical device environment. Without security, patient safety might be severely affected. Finally, PCA-IMD security has never systematically been explored.

In addition, we also developed an IMD-Security Analysis Tool (IMD-SAT). IMD-SAT is an application we developed to help a user understand and go through our methodology. IMD-SAT simulates our methodology and is integrated into a software application for other people to use.

Given the potential importance of IMDs, they have never been systematic analyzed in terms of security. IMD security is very important for overall reasons. First, IMD transfer and process sensitive patient health information. If the data from within IMDs gets disclosed, patients' privacy is in jeopardy. Second, without security, IMD operation might inadvertently cause patient harm. For example, consider on IMD for delivering insulin to Type-1 diabetics. However, if adversaries could alter the infusion rate at the insulin pump in the IMD, an episode of diabetic ketoacidosis might occur due to lack of insulin [37]. This can lead to severe health issue including death. Furthermore there are additional security issues related to medical device connectivity. Homogeneous device environments facilitate rapid spread of computer malware; medical devices often operate with commercial central processing units, operating systems, or off-the-shelf software, which place them at risk of cyber threats [38]. Therefore it is reasonable to believe that IMD security is very significant and has to be understood and enforced.

1.1 IMDs Architecture

Before we delve into the details of this work, it is useful to present a short overview of interoperable medical devices and their basic deployment architecture. The predominant standard in enabling medical device interoperability is the integrated clinical architecture (ICE) standard, which was created to enable diverse medical devices to talk to one another [11]. ICE was designed to act as a middleware to enable interaction of legacy, stand-alone medical devices and the applications using the medical devices. It has the potential to provide anything from data aggregation to closed-loop control over the patients health. The architecture of ICE typically consists of three entities (see Figure 1.1):

- A collection of Medical Devices on or around a single patient that can perform monitoring and actuation.

- The Supervisor which receives data from the various medical devices, processes it, and initiates action from the medical devices. The Supervisor runs clinical applications (referred to as *apps* from now on) that use the connected devices to support a clinical scenario selected by the caregiver.
- The Network Controller which interfaces with one or more medical devices and the supervisor. It is responsible for collecting data from the individual devices. It also connects the entire setup to an external network, such as the Healthcare Information System (HIS). The network controller also records all the actions of the entire system in a data logger (not shown) for future analysis.

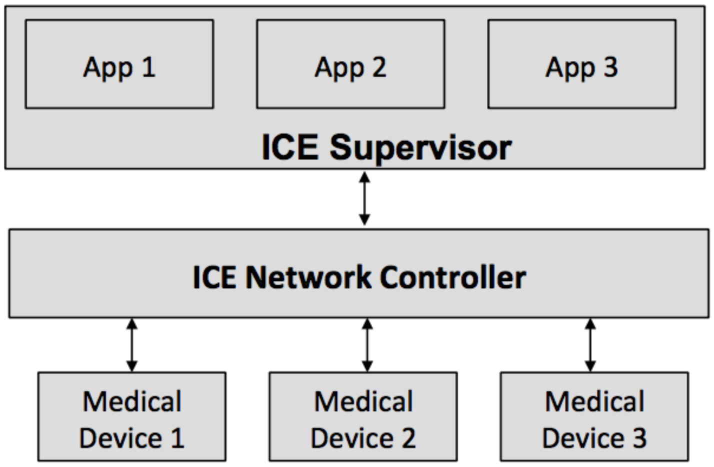


Figure 1.1: **ICE Architecture**

Normally ICE supervisor and network controller are combined together to a central processing unit called the *coordinator*. This coordinator has following functionalities: 1) receive medical data from medical devices, 2) process received data and detect if anything within IMD goes wrong, 3) send command to control each individual medical device, and 4) raise alarm to inform clinicians under emergent circumstances.

1.2 IMDs for Patient Controlled Analgesia

Patient Controlled Analgesia (PCA) uses an infusion pump to allow patients in pain to administer their own pain relief. The infusion pumps are programmable by the clinicians. If they are programmed and functioning as intended, the pumps are designed to prevent overdose of the pain medication. Figure 1.2 shows a example is real hospital PCA. IMDs could be used to automate PCA safety management by setting up a PCA-IMD. After PCA-IMD setup is successfully deployed, clinician types in relevant parameter (medication does needed, patient previous record etc.) to start infusion pump. During this treatment process, all medical devices keep sending data to central coordinator. Coordinator processes received data to monitor that patient. However, if the coordinator finds out the patient is not behaving well, it will immediately command infusion pump to stop, raise an alarm and wait for clinician. During this entire process, IMD guarantees PCA safety by real-time checking. It does not involve with any clinicians activities. The following shows detail about three medical components.

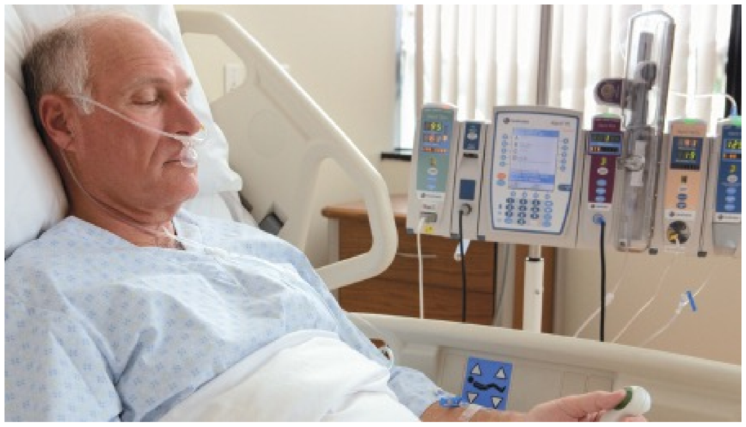


Figure 1.2: **Patient Controlled Analgesia Setting Example**

For a typical PCA-IMD hospital setting, it consists of three major components: an analgesic infusion pump, a capnography and a pulse-oximeter. Here are some brief discussions about those medical devices.

Analgesic infusion pumps (often referred to as PCA pumps) are electronic micro processing machines which can be programmed to deliver a prescribed amount of medication on demand, at specified intervals, by activation (pressing) of a button. Figure 1.3 shows an example of PCA pump.

Capnography is a medical device that helps monitor the concentration or partial pressure of carbon dioxide (CO_2) in the respiratory gases. Its main development is to be used as a monitoring tool for use during anesthesia and intensive care. It is usually presented as a graph of respiratory CO_2 (measured in millimeters of mercury, “mmHg”) plotted against time, or, less commonly, but more usefully, expired volume. The capnography is a direct monitor of the inhaled and exhaled

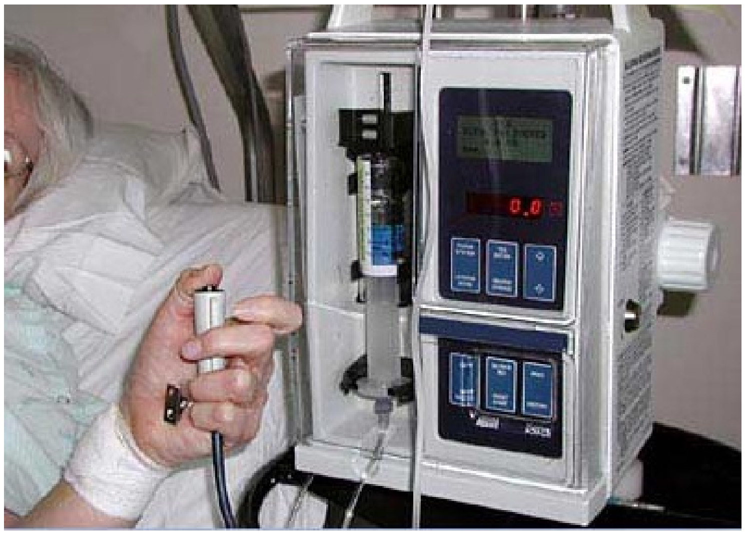


Figure 1.3: **Patient Controlled Analgesia Pump Example**

concentration or partial pressure of CO_2, and an indirect monitor of the CO_2 partial pressure in the arterial blood. Figure 1.4 shows a example of capnography in reality.

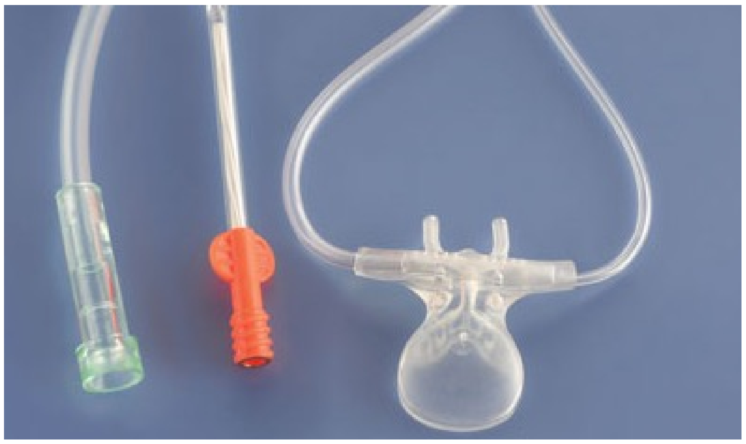


Figure 1.4: **Patient Controlled Analgesia Capnography Example**

Pulse-oximetry is a non-invasive device to monitor human's oxygen O_2 saturation. It is a sensor device to attach on a thin part of the patient's body, usually a fingertip or earlobe, or in the case of an infant, across a foot. The device passes two wavelengths of light through the body part to a photo detector. It measures the changing absorbance at each of the wavelengths, allowing it to determine the absorbance due to the pulsing arterial blood alone, excluding venous blood, skin, bone, muscle, fat, and (in most cases) nail polish. Figure 1.5 shows a example of pulse-oximeter.

When the whole system is successfully deployed, clinician programs above the three medical

Figure 1.5: **Patient Controlled Analgesia Pulse Oximeter Example**

devices with patient-related information (medication dose needed, patient previous record etc.). Then the clinician starts to load safety *app* on *coordinator*, this *app* contains work logic of *coordinator*. During this treatment process, all medical devices keep sending data to central coordinator. Pulse-oximeter shows the oxygen level, capnography indicates carbon dioxide level in blood while infusion pump sends medication does injected so far. The coordinator receives and triggers the safety *app* to process these data in order to monitor that patient. However, if coordinator finds out the patient is not behaving well, it will immediately command the infusion pump to stop, raise an alarm and wait for clinician. During this entire process, IMD guarantees PCA safety by real-time checking.

1.3 Contributions

To the best of our knowledge, this is the first effort in providing attack tree generation and quantification for IMDs. The main contributions of this thesis are as follows:

1. We develop a methodology for automatically generating attack trees in the context of interoperable medical devices (IMDs) used for PCA. We use process modeling, hazard analysis, and fault tree analysis to generate a comprehensive attack tree for IMDs.

2. We then use the attack trees generated to quantify the security condition of the IMD. The quantification enables us to compare the security condition of different configuration of the same IMD.

3. We also implement a tool called IMD-Security Analysis Tool (IMD-SAT). This tool enables users to construct their own attack trees for their target systems, and quantify security condition for that system.

1.4 Thesis Structure

This thesis report will be organized as follows: Chapter 2 discusses about all prerequisite knowledge of this work, including techniques and tools being used in our work. Chapter 3 presents the system model, adversary model for PCA-IMD. Also we discuss related work about automated attack tree generation and process modeling etc. In Chapter 4 we talk about our methodology, and introduce each step in detail. In order to make the discussion of the methodology concrete, we describe it by illustrating how it works for an IMD setup for enabling PCA-IMD. In Chapter 5 we analyze the results and share the lessons from our approach. Chapter 6 shows our conclusions as well as some future work.

Chapter 2

Background

In this chapter we provide an overview of the principal concept plus tools that we use in this thesis work. In particular we discuss process modeling, fault trees and attack trees.

2.1 Process Modeling

Process modeling is a technique that has been widely used to describe real-world processes. A model is an abstract representation of reality that excludes much of the world's infinite detail. The purpose of a model is to reduce the complexity of understanding or interacting with a phenomenon by eliminating details that do not influence it much. Therefore a model reveals what its creator believes is important in understanding the system. Mapping steps from a real system to a model is called *process modeling* [7].

From a theoretical point of view, process modeling explains the key concepts needed to describe what happens in the development process, on what, when it happens, and why. From an operational point of view, process modeling is aimed at providing guidance for system designers and application developers. In general, process modeling has many uses [7]:

- *Facilitate human understanding of a phenomena*: It requires that a group of people be able to share a common representational format.
- *Process improvement*: It defines the desired process and how they should/could/might be performed. If the behaviors of desired process are different from expected, then it demonstrates some aspects within the process should be improved.

- *Process management*: It keeps track of what actually happens during a process run.

- *Automated execution*: It provides automated tools for manipulating the process translations.

Many process-modeling tools are available, such as little-JIL, ARIS Express and BPMN. Little-JIL is the visual process modeling language we choose to use for our work [26]. Little-JIL is chosen as the process definition language in the implementation of our process analysis framework because it is well suited to capture the full complexity inherent in human-intensive processes. It provides support for abstraction, composition, and restricted, well-formed control flow constructs that are able to capture the rich control models needed for human behaviors. It also has extensive support for exception handling, a major aspect of many processes, particularly human-intensive processes.

2.1.1 Process Modeling with Little-JIL

In our research, we choose Little-JIL process definition language [4] to define processes. Little-JIL provides supports for the complex behaviors that often arise in processes, such as concurrency and exception handling, and has well-defined semantics that facilitate the kinds of analysis that we want to apply. In addition, Little-JIL has a visual representation that can be relatively easily understood by domain experts, including those outside of computer science.

Little JIL is a graphic language used to define real-world processes [15]. It is extremely expressive in capturing real-world processes and has well-defined semantics. We will provide a detailed introduction to little-JIL, a visual process modeling language in this section [26].

A Little-JIL process definition consists of three components, an artifact specification, a resource specification, and a coordination specification. The *artifact specification* contains the items that are the focus of the activities carried out by the process. The *resource specification* specifies the agents and capabilities that support performing all activities. The *coordination specification* ties these together by specifying which agents and supplementary capabilities perform which activities on which artifacts at what time. A Little-JIL coordination specification has a visual representation, but is precisely defined by finite state automata, which makes it amenable to definitive analyses. Several features of Little-JIL that make it suitable compared with other process languages are (1) using abstraction to support scalability and clarity, (2) using scope to make step parameterization clear, (3) faciliting for specifying parallelism, (4) capabilities for dealing with exceptional conditions, and (5) clarity in specifying iteration.

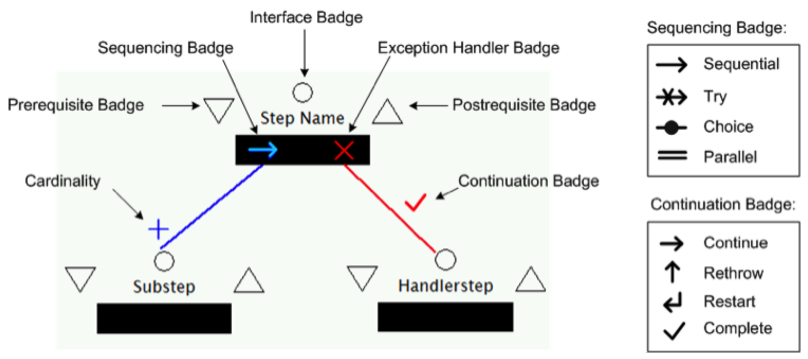


Figure 2.1: **Little-JIL Step Example**

A coordination specification consists of hierarchically decomposed steps (see Figure 2.1), where a step represents a task to be done by an assigned agent [5]. Each step has an interface specifying the input/output artifacts, the required resources and the thrown exception. A step with no sub-steps is called a leaf step and represents an activity to be performed by an agent, without any guidance from the process. Non-leaf Little-JIL steps can be decomposed into two kinds of sub steps, ordinary sub steps and exception handlers. Ordinary sub steps define how the steps are executed and connected to their parents by edges that may be annotated by specifications of the artifacts that flow between parents and sub steps and also by cardinality specifications. Cardinality specifications define the number of times the sub step is instantiated, and may be a fixed number, a Kleene * (greater than or equal to 0), a Kleene + (greater than or equal to 1), or a Boolean expression (indicating whether the sub step is to be instantiated). Exception handlers define how exceptions thrown by the step descendants are handled [5].

A non-leaf step has a sequencing badge (an icon on the left of the step bar; e.g., the right arrow in Figure 2.1) that defines the order of sub step execution. For example, a sequential step (right arrow) indicates that sub steps should execute from left to right. A parallel step (equal sign) indicates that sub steps execute in any (possibly interleaved) order, although the order may be constrained by such factors as the lack of needed inputs. A choice step (circle slashed with a horizontal line) indicates a choice among alternative sub steps. A try step (right arrow with an X on its tail) indicates the sequence in which sub steps are executed as alternatives. A Little-JIL step can be optionally preceded or succeeded by a prerequisite, represented by a down arrowhead to the left of the step bar, or a post-requisite, represented by an up arrowhead to the right of the step bar. Pre-requisites check if the step execution context is appropriate before execution of the step, and post-requisites check if the completed step execution satisfied its goals. The failure of a requisite triggers a corresponding exception [4].

Channels are message passing buffers, directly connecting specified source step(s) with specified destination step(s). Channels are used to synchronize and pass artifacts among concurrently executing steps. In a Little-JIL process, each step is defined exactly once. A step, however, may be used multiple times. These uses are represented by step references. A step reference is represented as a step bar without arrowheads that represent prerequisite and post-requisite.

Although originally developed to define software development and maintenance processes, Little-JIL can also be used to define process in other domains. For example, it has been applied to define medical processes [6], labor-management negotiation processes [24], and scientific data processing processes [20]. The work in each domain has exposed several inadequacies of Little-JIL, such as the lack of good support for specifying timing constraints and transactions, but has confirmed the general applicability of this language.

2.2 Hazard Analysis

A human-intensive process consists of tasks that are performed by various agents, including hardware devices and human agents. The process modeling technique discussed in the previous section primarily describes the way a real-world system operates. It assumes that the processes are done correctly and is concerned with the ordering of events occurred at some events, as specified by the given temporal properties. Due to hardware failure or human error, however, faults may be introduced during the execution of a task. If not detected and corrected, such a fault may propagate to the successors without violating those temporal properties and eventually lead to hazards. *Hazard* is a term used in the system safety analysis community. It is defined as "a state or set of conditions of the system that, together with certain other conditions in the environment, will lead inevitably to an accident" [20]. The accident usually results in harm to people or damage to property. For example, a hazard for IMD process could be "the alarm unit to alert clinician is broken." This hazard could prevent clinician from knowing patients' situations and eventually lead to patients' deaths. Developing such a safety critical process also requires to prevent or control potential hazards. This can be achieved by incorporating various mechanisms to prevent or control faults that could lead to the hazards. For instance, a failure-resistant agent could be assigned to some tasks where major faults could occur. Additionally, consistency checks could be added to well-chosen places in the process to stop the propagation of faults. To add such mechanisms into the process, the process developers need to identify the potential hazards that could occur in the process, as well as the faults that could lead to those hazards. Due to resource limitations or other constraints, the process developers are usually only able to apply the fault prevention mechanisms to the most important faults, which requires assessing and prioritizing the faults. This kind of analysis (i.e. identifying and assessing hazards and faults) is called *hazard analysis* in the system safety analysis community.

Hazard analysis is used as the first step in a process used to assess risk. The result of a hazard analysis is the identification of different type of hazards. Seldom does a single hazard cause an accident or a functional failure. More often an accident or operational failure occurs as the result of a sequence of causes. The main goal of hazard analysis is to provide the best selection of means of controlling or eliminating the risk. The term is used in several engineering specialties, including avionics, chemical process safety, safety engineering, reliability engineering and food safety [5].

Hazards may be realized or unrealized. A realized hazard has happened in the past and can therefore be identified from experience. An unrealized hazard is a potential for a hazardous situation that has not happened in the past but can be recognized by analyzing the characteristics of an environment or failure modes of equipment items. Hazard analysis techniques include:

- *Function Failure Analysis*
- *Event Tree Analysis*
- *Failure Modes and Effects Analysis*
- *Fault Tree Analysis*

In our work, we choose fault tree analysis as our hazard analysis tool. We will talk more about fault tree analysis tool in the next section.

2.3 Fault Tree Analysis (FTA)

During execution, faults might occur in components of a system due to many possible reasons: software failures, human errors, etc. Those faults might propagate through the system and eventually cause hazards to occur. In the context of security analysis, a hazard refers to an unsafe/unexpected state of the system that will inevitably lead to a serious accident if certain conditions in the environment are present. In order to prevent the potential hazards from happening, one needs to understand what kind of hazards could potentially occur in the system and how they could happen. A variety of hazard analysis techniques have been developed to identify potential hazards, assess their effect, and identify and evaluate the causal factors related to the hazards [20].

Fault Tree Analysis is a hazard analysis technique used to systematically identify and evaluate all possible causes of a given hazard. It is a top down, deductive analysis in which an undesired state of a system is analyzed using Boolean logic to combine a series of lower-level events.

This analysis method has been widely used in many different industries. It is mainly used in the fields of safety engineering and reliability engineering to understand how systems can fail, to identify the best ways to reduce risk or to determine (or learn about) event rates of a safety accident or a particular system level (functional) failure. FTA is used in the aerospace, nuclear power, chemical and process, pharmaceutical, petrochemical and other hazard prone industries; but is also used in fields as diverse as risk factor identification relating to social service system failure. FTA is also used in software engineering for debugging purposes and is closely related to cause-elimination technique used to detect bugs.

2.3.1 Fault Tree

Given a hazard, a fault tree is produced to show all the parallel and sequential combinations of events that could lead to the hazard. The basic elements of a fault tree are events and gates.

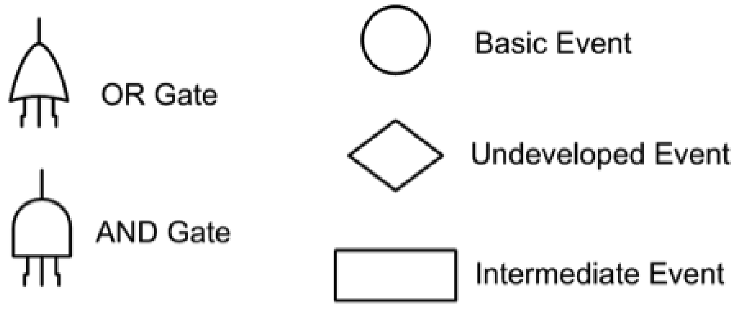


Figure 2.2: **Fault Tree Elements**

Events are used to represent faults, such as component failures, human errors, or other pertinent conditions in the system or environment. Figure 2.2 shows symbols of several commonly used events and gates. *Basic events* are basic initiating faults or conditions. *Undeveloped events* are events that are not developed any further, either because necessary information for deriving the fault tree leading to these events is unavailable or because these events are considered to have insignificant consequence. Basic events and undeveloped events are also called *primary events* because they require no further development. As opposed to primary events, intermediate events are events that need to be developed. Investigating the system to identify the immediate, necessary, and sufficient events that cause this event, and then connecting those events to it via a proper gate develop an intermediate event. In a fault tree, events are connected using gates. Each gate connects one or more input events to a single output event. The output event of an AND gate occurs if all of the input events occur. While the output event of an OR gate occurs if any of the input events occurs. Figure 2.3 shows an example fault tree. Complete explanations of three different gates could refer to the following:

- AND gates: the output event occurs if and only if all the input events occur which implies that the occurrence of all the input events causes the output event.
- OR gates: the output event occurs if and only if at least one of the input events occur which implies that the occurrence of any of the input events causes the output event.
- NOT gates: the output event occurs if and only if the (only) input event does not occur.

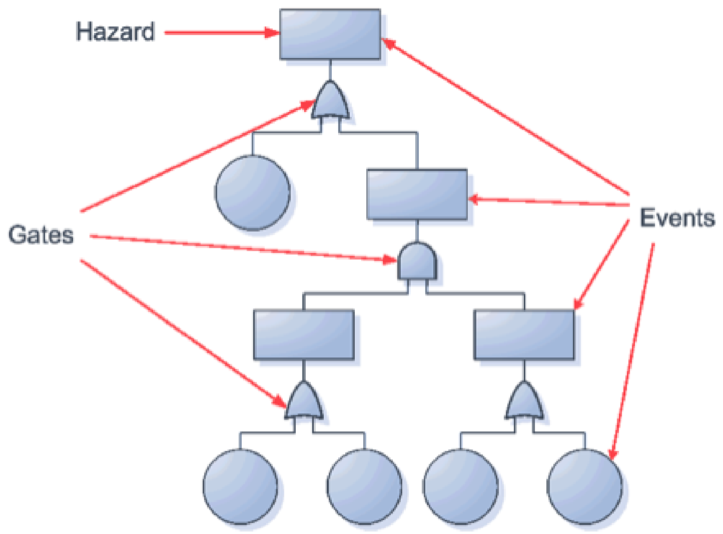


Figure 2.3: **Fault Tree Example**

2.4 Attack Tree

Attack trees are conceptual, multi-leveled diagrams showing how an asset, or target, might be attacked. It provides a formal, methodical way of describing the security of a system.

Similar to many other types of trees (e.g., fault trees), the attack trees are usually drawn inverted, with the root node at the top of the tree and branches descending from the root. The top or root node represents the attacker's overall goal. The nodes at the lowest levels of the tree (leaf nodes) represent the activities performed by the attacker. Nodes between the leaf nodes and the root node depict intermediate states or attacker sub-goals. Although the attacker may gain benefits (and the victim suffer impacts) at any level of the tree, the impacts usually increase at higher levels of the tree. Non-leaf of sub-goal nodes in an attack tree are designated as either AND or OR nodes, and usually represented by the familiar Boolean Algebra AND/OR shapes. AND nodes represent processes or procedures. All of the activities or states represented by the nodes immediately beneath an AND node must be achieved to attain the goal or state represented by the AND node. OR nodes represent alternatives. If any of the nodes directly beneath an OR are attained then the OR state is also attained. Certain combinations of leaf level events will satisfy the tree's AND/OR logic and result in one or more paths leading to the root goal of the tree. These sets of events are known as attack scenarios [29].

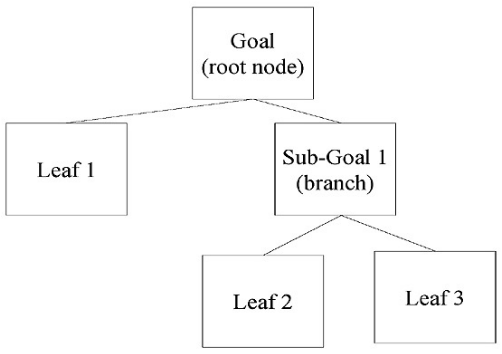


Figure 2.4: **Attack Tree Example**

Figure 2.4 is about classic attack tree model, however in our work, we might need to expand or update the attack tree to meet our requirements. The biggest change is that each attack leaf may include one or more "defense nodes" that are direct successors of the attack leaf. Defense nodes provide descriptions of countermeasures. In Figure 2.5 and 2.6, the box labeled c1 is a countermeasure for attack leaf on the left side. An attack leaf can be an element of different

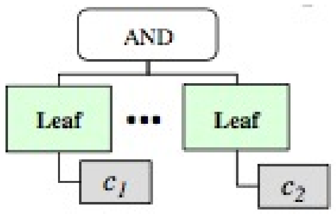


Figure 2.5: **Attack Leaf With "AND"**

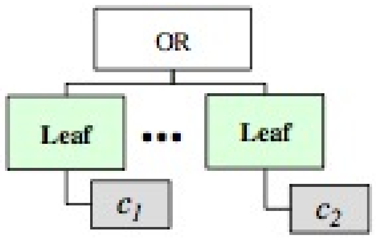


Figure 2.6: **Attack Leaf With "OR"**

intrusion scenarios, depending on the node connectivity associated with it.

Chapter 3

System Model and Related Work

In order to make the discussion of the methodology concrete, we describe it by illustrating how it works for an IMD setup for enabling patient controlled analgesia, referred to as PCA-IMD (see Figure 3.1). In this section, we describe the system model of the PCA-IMD setup followed by a description of the adversary who is expected to attack this system. The adversary model essentially scopes the eventual attack tree that is generated by our methodology.

3.1 PCA-IMD System Model

The aim of PCA-IMD is to allow patients to inject themselves with pain medication (e.g., morphine) in a safe manner. PCA-IMD consists of an infusion pump programmed to infuse pain medication (e.g., morphine) to the patient at a specific (basal) rate in a hospital or care-facility. As pain medications tend to suppress respiration, we also have a pulse-oximeter measures level of O2 in the blood) and a capnograph (measures level of CO2 in the blood) to determine how the patient is responding to the pain medication. The pulse-oximeter and the capnograph are collectively referred to as *sensors*, in the rest of the paper (see Figure 3.1). The details of the network controller and supervisor are abstracted out into a *coordinator entity* in our analysis to keep the discussion simple. The coordinator (through the network controller) interfaces with the hospital electronic health record (EHR) system. It can update and query the EHR when needed. For example, a medical application running on the coordinator can be used to perform a sanity check on the nurse's programming of the infusion pump based on medication orders in the EHR.

PCA-IMD is configured for each patient according to his or her individual needs. This means: (1) deploying the pump, oximeter and capograph on the patient, (2) scanning the patient id for indexing the patient data into hospital EHR system, (3) connecting the three medical devices to the network controller and the supervisor, (4) deploying an app on the supervisor for enabling safe delivery of pain medication, and (5) monitoring the patient's well-being during the treatment. The caregiver monitors the patient through the patient display (dashed arrow in Figure 3.1). The coordinator receives status updates from the individual medical devices, and it displays the information to the caregiver via the patient display. If the blood oxygen level of the patient goes below a certain threshold, a medical application on the coordinator will raise an alarm to the caregivers. The infusion pump and the sensors are programmed directly by the caregiver using a computer-on-wheels PC based on the patient status information on the patient display, which the coordinator provides.

In this typical setting, the ultimate threat we would like to consider is patient being over-infused. As long as patient does not receive excessive doses of medicine, patient will not be killed by an attacker, thus over infusion should be considered to be the highest priority attack.

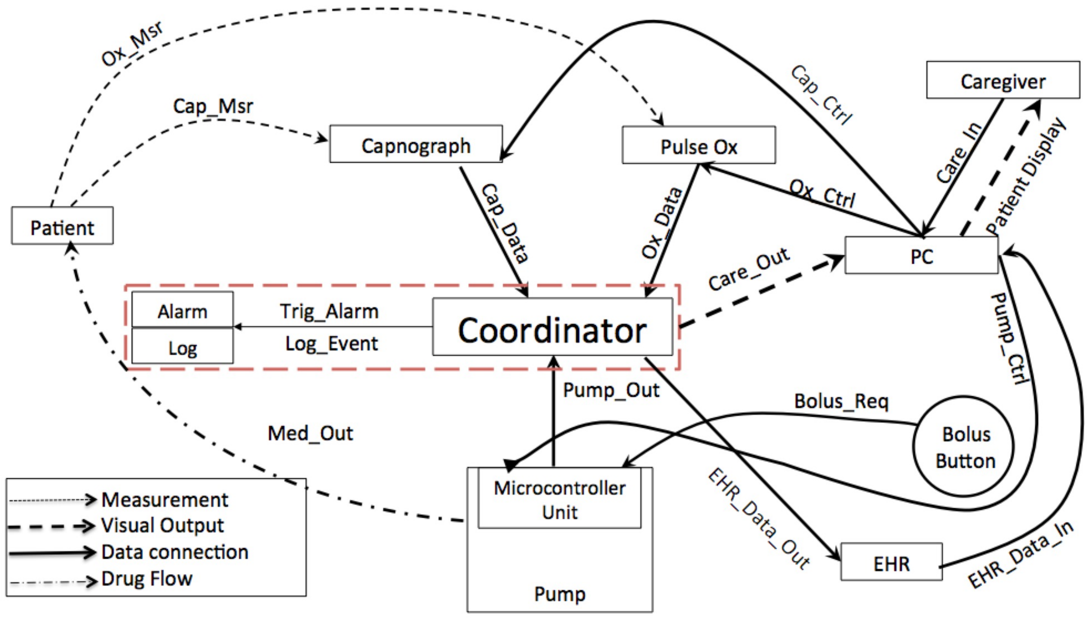


Figure 3.1: **System Model for PCA-IMD Setup**

3.2 PCA-IMD Adversary Model

In our interoperability setup, we consider the coordinator and the associated logging and alarms to be the only members of the trusted computing base (TCB). These components are *trusted* (they do not have malicious intent) and *trustworthy* (they will operate as expected). The dashed box in Figure 3.1 signifies the TCB in our system model. Further, we assume that the caregiver is not necessarily trustworthy, in that the caregiver can make mistakes in programming the devices, but does not have malicious intent. We further assume that the infusion pump in our system model is verifiably safe as described in [14].

For our work, we consider *active adversaries* who may interfere with communication links, as per the Yao-Dolev model of an adversary [9]. In addition, the adversary may also physically alter the infusion pump, the coordinator, the pulse oximeter, and capnograph, and their individual settings, respectively. Another possible situation is that attackers can impersonate the caregiver to send malicious commands to kill the patient if attackers have caregivers' credentials. Note that, while adversaries may simply inject the patient directly and induce a medical emergency, we consider such attacks outside the scope of interoperable medical device security. Finally, we only consider adversaries whose attack goal is over-infusion (for pain medication under-infusion does not hamper patient safety) through the infusion pump in the PCA-IMD setup.

3.3 Related Work

Before we present our work, it is necessary to take a look at the achievements that have been done and those that have not been done in all the domains our work uses. In that case, we could learn from other researches' work to enhance ours.

3.3.1 Security Analysis with Attack Tree

Using attack trees for analyzing the security has been studied before for a variety of systems. Prominent examples include Supervisory Control And Data Acquisition (SCADA) [34], Cyber-Physical Systems [19], and interoperable medical devices [33]. However, none of these approaches provide any systematic means of generating the attack trees. Their focus rather is on using the generated attack trees for detailed analysis of the system.

3.3.2 Automated Attack Tree Quantification

Chee-Wooi Ten et al. [34] came up with a quantification mechanism based on attack trees and applied it to SCADA systems to validate the effectiveness of the approach. However, there are several issues with this approach. First of all, this paper provides an attack tree without demonstrating the particular process of generating that attack tree. Readers have no clues how to get an attack tree like that simply based on a SCADA system. Secondly, the approach proposed by this paper, only helps assess the vulnerability conditions for one particular SCADA system. The author does not demonstrate whether this approach has the ability to be extended to other similar systems. However, in this work, our methodology will help eliminate the above issues.

What's more, Xie et al. [41] used attack trees to focus on Cyber-Physical System security analysis. However, through the whole paper, they neither helped prove their methodology nor provided any validation examples. Therefore, the results of this work are not quite convincing.

Robert et al. [1], their work tried to consider how to identify, monitor and estimate risk impact and probability for different smart grid stakeholders. There are many other research papers that are very similar to this paper; all these approaches require considerable knowledge of formal methods and model checking which IMD users, who are domain experts in healthcare systems, may not possess.

3.3.3 Automated Attack Tree Generation

Stephane Paul et al. [25] find their own way to construct attack tree based on given system model. They also validate their own approach by running a example. However, the process is manually and is very unlikely to be replicated in practice. There are a few other commercial products related to automated attack trees construction. The two most significant ones are SecurITree [22] and AttackTree+ [18]. However, none of these tools support the automated construction of attack trees. The scientific community is engaged in the construction of attack graphs [21] and [3]. However, attack graphs essentially focuses on attacker attempts to penetrate well-defined systems, rather than addressing medical device systems, especially when poorly or partially defined.

Some research papers do address this similar issue. For example, [36] and [27] propose formal approaches to generate attack trees, respectively based on system goals, and security policies. These studies can be seen as upstream complements, but they also require specific frameworks. Our approach attempts to base most of its attack tree extraction on an industrially used framework.

3.3.4 Process Modeling and Model Checking

Huong et al. [26] described a systematic approach for incrementally improving the security of election processes by using a model of the process to develop attack plans and then incorporating each attack plan into the process model. This could help determine whether or not the attack plan can be completed successfully. This paper takes advantage of formal methods to improve system security in a more consistent way. The tools and technologies associated with this paper turn out very useful and could be incorporated into our work. Unfortunately, this paper has several limitations as well. The authors did not quantify the security condition for the voting process, therefore there is no way for users to value the efficiency of this approach. In other words, they failed to measure security conditions.

In [28], model checking is applied to the analysis of system security for a long time. This paper uses network system as an example. Known vulnerabilities on network hosts, connectivity between hosts, initial capabilities of the attacker are described as states and exploits as transitions between states. This model is given to a model checker as its input and the reachability in terms of given goal states is given as a query. The model checker then produces a counterexample if a sequence of exploits can lead to goal states. In [31] and [12], model checking is used for a different purpose, that is to enumerate all attack paths. A modified model checker is used to take as input the finite state machine created from network information. The model checker provides all counterexamples to a query about the safety of the goal states. Those are essentially the possible attack paths.

Chapter 4

Methodology

An overview of the methodology for attack tree generation is shown in Figure 4.1. It has four elements. First, the user (deployer) of the IMD takes an abstract description of the workflow of IMD and develops a process modeling representation for it. This representation essentially describes the operation of the IMD, which in our case is PCA-IMD. Once the process model is developed, the IMD user identifies the various hazards that can occur due to the operation of the system. The hazards are essentially unsafe states of the IMD, which can harm the system itself or the user. *In our case the unsafe states of interest with respect to hazard analysis are those that eventually lead to over-infusion of pain medication when using PCA, our attack goal.* Given the hazards that lead to over-infusion and the process model of the IMD, we derive several fault-trees for the system. The fault-trees essentially provide a representation of the means by which the hazards can be realized due to faults within the system. Once the fault-trees have been extracted, we combine them in a meaningful fashion to generate a global attack tree for the IMD setup such that the root node of the tree is the attack goal. Finally, we construct a countermeasure database to prevent all kinds of attacks that could happen in IMD specifically. We then use the attack trees generated to quantify the security condition of the IMD given countermeasure database. The quantification enables us to compare the security condition of different configuration of the same IMD. The rest of the section describes these steps in more detail.

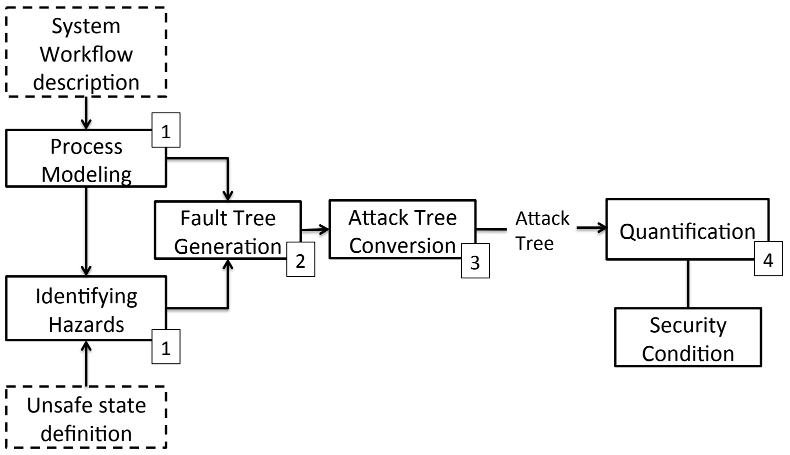


Figure 4.1: **Attack Tree Generation Methodology**

4.1 IMD Workflow Description

The first step in our approach is to use process modeling to describe the workflow of the IMD. To model the workflow we use a tool called Little-JIL [5]. Little JIL provides a graphical language used to define real-world processes. It is very expressive in capturing the nuances of the workflow and has well-defined semantics. Figure 4.2 shows the coordination specification of our PCA-IMD setup defined in Little-JIL. A Little-JIL process definition consists of three components, an artifact specification, a resource specification, and a coordination specification. The *artifact* specification contains the items that are the focus of the activities carried out by the process. In our case this means the information exchanged between the various entities PCA-IMD e.g., Ox_Ctrl (the settings for pulse-oximeter) or Cap_Data (the output of the capnograph). The *resource* specification describes the agents and capabilities they have to execute the workflow. For example, the caregiver, coordinator, capnograph etc. Finally, the *coordination* specification ties everything together by specifying which agents and supplementary capabilities perform which activities on which artifacts at which time(s). This is what is essentially represented in Figure 4.2.

In describing the operational workflow of the PCA-IMD using process modeling we tend to be end-to-end, beginning at the time when the devices in the IMD are deployed around the patient all the way to dismantling the PCA-IMD. For space reason we do not describe the entire specification,

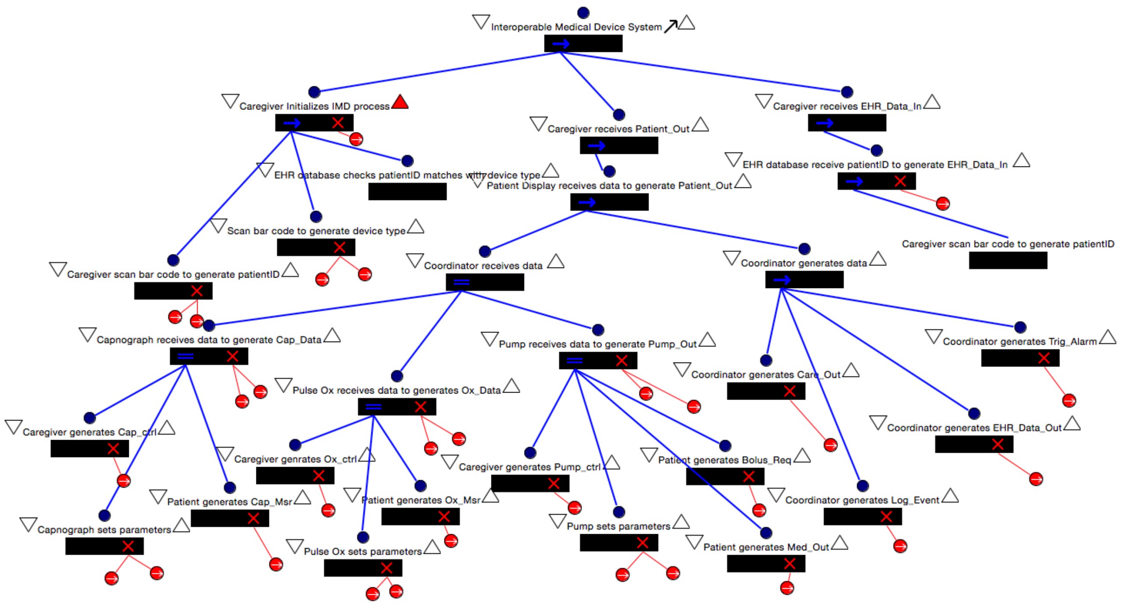

Interoperable Medical Device System
Caregiver Initializes IMD process
Caregiver receives Patient_Out
Caregiver receives EHR_Data_In
EHR database checks patientID matches with device type
Patient Display receives data to generate Patient_Out
EHR database receive patientID to generate EHR_Data_In
Scan bar code to generate device type
Caregiver scan bar code to generate patientID
Coordinator receives data
Coordinator generates data
Caregiver scan bar code to generate patientID
Capnograph receives data to generate Cap_Data
Pump receives data to generate Pump_Out
Coordinator generates Trig_Alarm
Pulse Ox receives data to generates Ox_Data
Coordinator generates Care_Out
Caregiver generates Cap_ctrl
Coordinator generates EHR_Data_Out
Caregiver genrates Ox_ctrl
Caregiver generates Pump_ctrl
Patient generates Bolus_Req
Patient generates Cap_Msr
Patient generates Ox_Msr
Coordinator generates Log_Event
Capnograph sets parameters
Pump sets parameters
Pulse Ox sets parameters
Patient generates Med_Out

Figure 4.2: **Little-JIL-Based Process Modeling of PCA-IMD Workflow**

rather only some of the important modeling elements. The root step *Interoperable Medical Device System* is a sequential step containing three sub-steps that need to be carried out in an order from left to right. This means that the left-most sub-step *Caregiver Initializes IMD Process* is executed first. *Caregiver Initializes IMD Process* is also a sequential step (as indicated by the unidirectional blue arrow in the box representing the step), which means each of its sub-steps *Caregiver scans barcode to generate Patient-ID*, *Caregiver scans barcode to generate Device Type* and *EHR database checks Patient_ID matches Device_Type* have to execute in order for it to complete. The step *Capnograph ... generates Cap_Data*, on the other hand, is a parallel step ((as indicated by the bidirectional blue arrow in the box representing the step) where each of its three sub-steps are expected to execute concurrently. During the execution, each Little-JIL step has an artifact declaration defining the artifacts it will be accessing or providing. Artifacts are passed through the coordination hierarchy between steps and their sub-steps. For example, *Caregiver ... generates Ox_Ctrl* step will produce an artifact named *Ox_Ctrl*. *Caregiver Receives Patient-Out* will receive an artifact named *Pump-Out*. Finally, for each of these steps, exception handlers are defined (as indicated by the red "X" in the box representing the step), which are thrown when the inputs to a certain step are not defined. For example, in *Caregiver scans barcode to generate*

Patient_ID, if *Patient_ID* is not in the EHR, then exception *Patient_ID Unavailable* is thrown.

4.2 Hazard Analysis and Fault Tree Extraction

Once the process modeling is complete, we derive fault-trees for it. The fault-trees represent the various ways in which the system being modeled can fail. In our context, a *hazard* is the occurrence of unsafe/unexpected states within the system that will inevitably lead to patient harm. *In the PCA-IMD setup all the hazards essentially focus on preventing over-infusion of pain medication to a patient.* Table 4.1 shows the list of possible hazards considered for the PCA-IMD that may cause over-infusion. We derive a fault-tree for each of these hazards. Fault-trees are essentially a systematic way to identify and evaluate all possible causes for them. A fault-tree consists of two basic elements events and gates. At the top (root) of the fault-tree is the hazard. In the fault-tree, intermediate events are expanded further, and leaf events are not. Events are connected to each other by Boolean-logic (AND, OR and NOT) gates [5].

Table 4.1: Possible Hazards in PCA-IMDs Leading to Over-infusion

No.	**Possible Hazard Description**
1	Patient_ID scanned is Erroneous
2	Device_Type scanned is Erroneous
3	Device_Type match withPatient_ID Erroneous
4	Cap_Data Erroneous
5	Ox_Data is Erroneous
6	Pump_Out is Erroneous
7	Patient_Out is Erroneous
8	Care_Out is Erroneous
9	EHR_Data_In is Erroneous
10	EHR_Data_Out is Erroneous

We again use the Little-JIL modeling tool, which takes in as inputs a list of hazards along with the process model to generate the fault-trees [5]. To derive a fault-tree, a given hazard is represented as an intermediate event called the TOP event. Starting with a fault-tree that only has the TOP event, the fault-tree derivation procedure expands the fault-tree by developing intermediate events in it. An intermediate event is developed by analyzing the process model to identify the immediate, necessary, and sufficient events that cause this event, and then connecting those events to it via a proper gate. The new events may also be intermediate events that need to be developed further. The derivation procedure terminates when no intermediate events exist in the fault-tree that can be described further [5].

Consider the following example of the aforementioned fault-tree extraction process. One class of the requirements is that patient medical data collected by the sensors should never be modified, as this might mislead the caregiver who can then incorrectly program the infusion pump leading to over-infusion. In this regard, we define a hazard *EHR_Data_In is Erroneous*, which describes the situation when the EHR data sent to the caregiver detailing the current and the past state of the patient is incorrect. Using this hazard definition, we wish to describe various scenarios during the operation of the PCA-IMD that may lead to the EHR being modified or the data being corrupted during transit. Figure 4.3 shows the fault-tree generated for the hazard *EHR_Data_In is Erroneous*. Here, the *EHR_Data_In* is the artifact of interest, which is the label for the data provided by the EHR to the caregiver who uses it to program the pump. The hazard in the fault-tree (which is shown as the root node) indicates that the artifact *EHR_Data_In* is incorrect. The error is caused when the EHR receives the *Patient_ID* it produces the wrong *EHR_Data_In* value associated with the patient as indicated by the OR-gate underneath root node. The reason for producing the wrong *EHR_Data_In* given *Patient_ID* can be one of three conditions as shown in the partial fault-tree in Figure 4.3: (1) the data associated with the patient EHR is erroneous, (2) *Patient_ID* doesn't exist in the EHR database, or (3) the *Patient_ID* scanned was erroneous. As the *EHR_Data_In* is transmitted over a communication medium, we actually have additional conditions to consider where the data is modified during transit, which we don't show for keeping the discussion simple.

4.3 Attack Tree Generation

The availability of fault-trees allows us to convert it to an attack tree, which can be thought of fault-trees where the faults are caused deliberately by adversaries. The attack trees indicate different attack paths to make different hazards happen during system execution. It is important to note that fault-trees, in additional to describing how hazards manifest themselves; provide us with description of the relationship between the various artifacts. For instance, *EHR_Data_In* is related to *Ox_Data*, therefore if *Ox_Data* is modified, the value of *EHR_Data_In* may not be accurate any more. This incorrect EHR data may eventually lead to over-infusion. In generating the attack tree from fault-trees we preserve such associations between artifacts, when we analyze the effects of attacks mounted by adversaries in terms of the hazards they may eventually cause. We use a

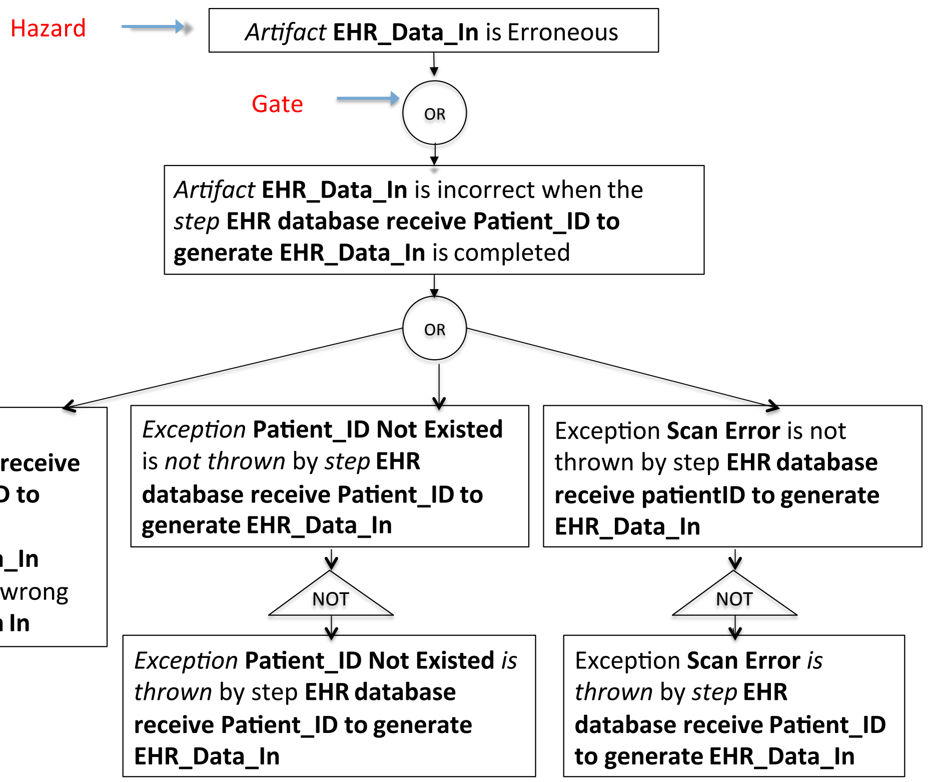

Hazard
Artifact EHR_Data_In is Erroneous
Gate
OR
Artifact EHR_Data_In is incorrect when the step EHR database receive Patient_ID to generate EHR_Data_In is completed
OR
Step EHR database receive Patient_ID to generate EHR_Data_In produces wrong EHR_Data In
Exception Patient_ID Not Existed is not thrown by step EHR database receive Patient_ID to generate EHR_Data_In
Exception Scan Error is not thrown by step EHR database receive patientID to generate EHR_Data_In
NOT
NOT
Exception Patient_ID Not Existed is thrown by step EHR database receive Patient_ID to generate EHR_Data_In
Exception Scan Error is thrown by step EHR database receive Patient_ID to generate EHR_Data_In

Figure 4.3: **Example Partial Fault Tree for a Specific Hazard (EHR Data Incorrect)**

three-step process in converting a fault-tree into an attack tree:

- **Step 1:** Classify the different fault-trees based on whether their artifacts have associations. For example, if artifact A is associated with artifact B, then fault-trees using A and B should be classified into one group. For example consider the hazard *Care_Out is Erroneous*. The artifact *Care_Out*, the value sent by the coordinator to the caregiver's PC representing the state of the patient and her treatment, is made up of three other artifacts namely *Cap_Data*, *Ox_Data*, and *Pump_Out*. Therefore, the fault-trees that affect *Cap_Data*, *Ox_Data*, and *Pump_Out* are combined in the attack tree generated.

- **Step 2:** Concatenate different groups with appropriate logical AND or OR operators. If two fault-trees share a root node then the OR operation is used to connect them, otherwise an AND operator is used, and a new parent node is created for the groups in the attack tree.

- **Step 3:** The node descriptions are modified from faults that materialize in the system to attacks where they are deliberately induced by the adversary.

Algorithm 1 shows the pseudo code we used to convert all fault trees into one attack tree. We now walk through one example to demonstrate how this attack tree conversion algorithm works. The fault tree we are looking at is called "Care_Out in Caregiver Receives Patient_Out Has Wrong Output." The original fault tree contains three child nodes. However, Care_out consists of Cap_data, Ox_data, and Pump_out. So these three artifacts are combined together to generate care_out, they should be connected. According to our algorithm, since those artifacts are associated, we should merge these fault trees. We concatenate all three Cap_data, Ox_data and Pump_out fault trees into Care_out with logic OR-gate. Also we paraphrase some of nodes if needed. Originally we have four individual fault trees, after conversion, we get only one fault tree.

Algorithm 1 Attack Tree Convertion Algorithm

```
 1: procedure ConvertionProcess
 2:     Map<K,V> map = new HashMap<K,V>()          ▷ K = Artifact, V = list of fault trees
 3:     for each fault tree $F_i \in F$ do
 4:         if map.contains($F_i$.artifact) == true then
 5:             map.get($F_i$.artifact).add($F_i$)
 6:         else
 7:             map.put($F_i$.artifact,$F_i$)
 8:         end if
 9:     end for
10:     for each parentTree in map do
11:         for each other childTree in map do
12:             if childTree.K belongs to parentTree.K then
13:                 Connect parentTree.V with childTree.V with logic "AND"
14:             else
15:                 Create a new branch, connect with other branch with logic "OR"
16:                 Paraphase parentTree.V to make description based on attacks
17:             end if
18:         end for
19:     end for
20: end procedure
```

Figure 4.4 shows the entire attack tree of our PCA-IMD system. Instead of using explicit AND and OR operator symbols in the tree as is customary, we use the shape of the node to describe the operation. A rounded rectangle node indicates that all its children are ORed, while a regular (unfilled) rectangle node indicates that all its children are ORed. The leaf nodes are denoted by the filled rectangle boxes and numbers G1 to G22. Each path in the tree from the leaf node all the way to the root node denotes a way to attack the PCA-IMD, which needs to be protected. In general all attacks on PCA-IMD can be divided into two general categories: (1) those where the adversary directly manipulates the pump settings provided by the caregiver *Pump_Ctrl* by mounting attacks on the pump or the communication between the PC and the pump (G1-G3), and (2) those where the adversary indirectly causes incorrect *Pump_Ctrl* settings in the pump by delaying, tampering, or loosing sensor measurements, pump status information, or EHR data (G4-G22).

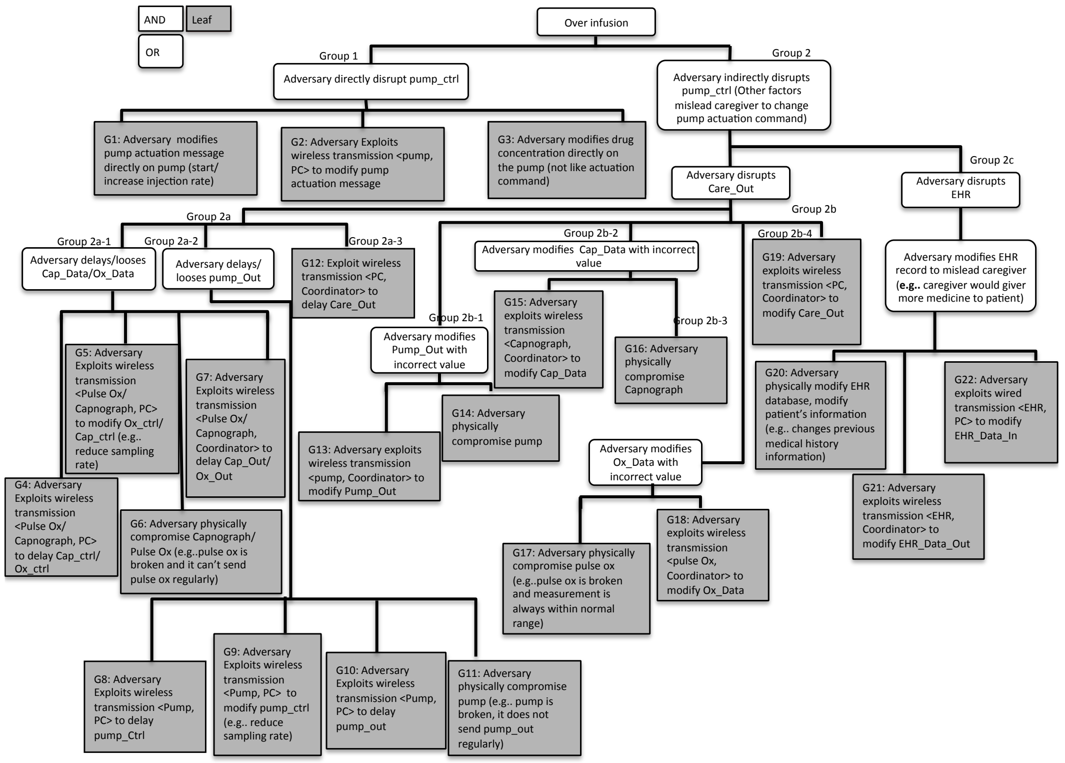

AND
Leaf
OR
Over infusion
Group 1
Group 2
Adversary directly disrupt pump_ctrl
Adversary indirectly disrupts pump_ctrl (Other factors mislead caregiver to change pump actuation command)
G1: Adversary modifies pump actuation message directly on pump (start/ increase injection rate)
G2: Adversary Exploits wireless transmission <pump, PC> to modify pump actuation message
G3: Adversary modifies drug concentration directly on the pump (not like actuation command)
Group 2c
Adversary disrupts Care_Out
Adversary disrupts EHR
Group 2b
Group 2a
Group 2a-1
Group 2a-2
Group 2a-3
Group 2b-2
Group 2b-4
Adversary delays/looses Cap_Data/Ox_Data
Adversary delays/ looses pump_Out
G12: Exploit wireless transmission <PC, Coordinator> to delay Care_Out
Adversary modifies Cap_Data with incorrect value
G19: Adversary exploits wireless transmission <PC, Coordinator> to modify Care_Out
Adversary modifies EHR record to mislead caregiver (e.g.. caregiver would giver more medicine to patient)
Group 2b-1
Adversary modifies Pump_Out with incorrect value
G15: Adversary exploits wireless transmission <Capnograph, Coordinator> to modify Cap_Data
Group 2b-3
G16: Adversary physically compromise Capnograph
G5: Adversary Exploits wireless transmission <Pulse Ox/ Capnograph, PC> to modify Ox_ctrl/ Cap_ctrl (e.g.. reduce sampling rate)
G7: Adversary Exploits wireless transmission <Pulse Ox/ Capnograph, Coordinator> to delay Cap_Out/ Ox_Out
G14: Adversary physically compromise pump
G13: Adversary exploits wireless transmission <pump, Coordinator> to modify Pump_Out
G20: Adversary physically modify EHR database, modify patient's information (e.g.. changes previous medical history information)
G22: Adversary exploits wired transmission <EHR, PC> to modify EHR_Data_In
Adversary modifies Ox_Data with incorrect value
G21: Adversary exploits wireless transmission <EHR, Coordinator> to modify EHR_Data_Out
G4: Adversary Exploits wireless transmission <Pulse Ox/ Capnograph, PC> to delay Cap_ctrl/ Ox_ctrl
G6: Adversary physically compromise Capnograph/ Pulse Ox (e.g..pulse ox is broken and it can't send pulse ox regularly)
G18: Adversary exploits wireless transmission <pulse Ox, Coordinator> to modify Ox_Data
G17: Adversary physically compromise pulse ox (e.g..pulse ox is broken and measurement is always within normal range)
G8: Adversary Exploits wireless transmission <Pump, PC> to delay pump_Ctrl
G9: Adversary Exploits wireless transmission <Pump, PC> to modify pump_ctrl (e.g.. reduce sampling rate)
G10: Adversary Exploits wireless transmission <Pump, PC> to delay pump_out
G11: Adversary physically compromise pump (e.g.. pump is broken, it does not send pump_out regularly)

Figure 4.4: **Attack Tree for PCA-IMD**

4.4 Quantification

In this section, we will discuss the quantification mechanism we used to calculate the security condition for PCA-IMDs. Similar approach can be used for other systems. Our quantification approach has the following steps: (1) develop a definition of security condition, a variable that defines how secure or insecure the system is; (2) list a set of countermeasures that can be applied; (3) attach the countermeasures to the attack tree leaf nodes; (4) compute the value of the security condition based on the countermeasures attached on the attack tree.

4.4.1 Security Condition

A system security condition is a score to define how secure or insecure the system is. It ranges from 0 to 1, from the most insecure (0 value) to the most secure (1 value). This variable enables us to compare security situation among different systems. The higher the system security condition is, the more secure the system is behaving against all possible attacks.

4.4.2 Construct Countermeasure Data Set

To complete the security condition of an attack tree, we first come up with a set of countermeasures for various attacks possible in our systems. In our case, we divide our countermeasure data set into five types. Table 4.2 demonstrates those five types. We select countermeasures from different types to each attack leaf, depending on what level of security we want to achieve as well as whether this countermeasure is related. In order to compare security conditions, we make an assumption here that all of these countermeasures are equally effective, which means if a countermeasure is deployed then the attack specified by the attack leaf is completely mitigated.

Table 4.2: 5 General Types of Countermeasures

Access	**Protection**	**Backup**	**Cryptography**	**Detection**
Authentication	Update Secure Patches	Recovery	Cryptography	Auditing
Access Control	Anti-virus		Integrity	Intrusion Detection
Physical Access				Physical Detection
Firewall				

From Table 4.2, one can see that the countermeasures we consider are of five general types. Under each general type, we have distinct subtypes. Furthermore, under each subtype, we have specific countermeasures. The countermeasures in each sub type share some features in common. Appendix E.1 provides the list of countermeasures for each general type and sub type combination. For example: "Access" is one general type, under which has four subtypes: "Authentication", "Access Control", "Physical Examination" and "Fire Wall". Under subtype "Authentication", there are several countermeasures: such as (1) Use user ID and password to authenticate clinic staff, (2) Implement a biometric technology like fingerprint for authentication and others. Below Figure 4.5 shows the complete attack tree with corresponding countermeasures attached. Those countermeasure numbers and their meaning could be found in the Appendix E.1. The more countermeasures an attack leaf has, the higher its initial attack potential on the system.

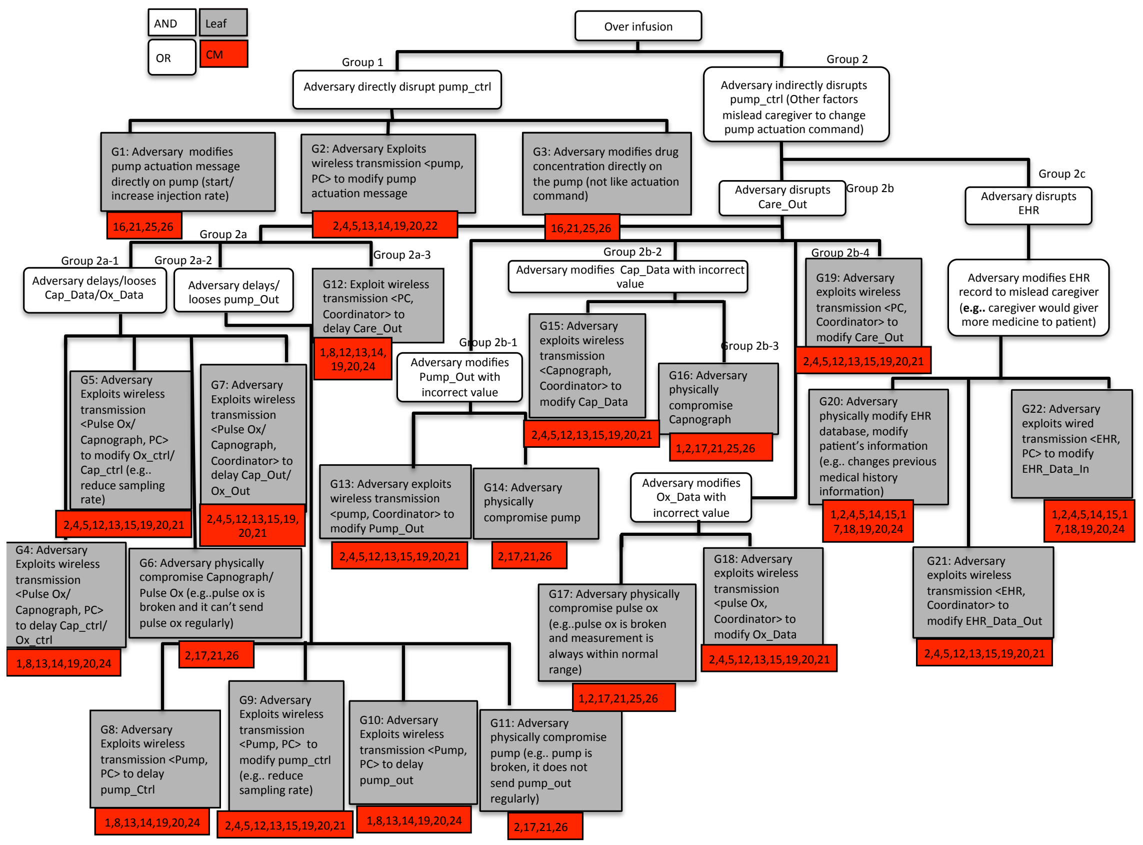

AND
Leaf
OR
CM
Over infusion
Group 1
Adversary directly disrupt pump_ctrl
Group 2
Adversary indirectly disrupts pump_ctrl (Other factors mislead caregiver to change pump actuation command)
G1: Adversary modifies pump actuation message directly on pump (start/ increase injection rate)
16,21,25,26
G2: Adversary Exploits wireless transmission <pump, PC> to modify pump actuation message
2,4,5,13,14,19,20,22
G3: Adversary modifies drug concentration directly on the pump (not like actuation command)
16,21,25,26
Adversary disrupts Care_Out
Group 2b
Group 2c
Adversary disrupts EHR
Group 2a
Group 2a-1
Group 2a-2
Group 2a-3
Adversary delays/looses Cap_Data/Ox_Data
Adversary delays/ looses pump_Out
G12: Exploit wireless transmission <PC, Coordinator> to delay Care_Out
1,8,12,13,14, 19,20,24
Group 2b-2
Adversary modifies Cap_Data with incorrect value
Group 2b-4
G19: Adversary exploits wireless transmission <PC, Coordinator> to modify Care_Out
2,4,5,12,13,15,19,20,21
Adversary modifies EHR record to mislead caregiver (e.g.. caregiver would giver more medicine to patient)
Group 2b-1
Adversary modifies Pump_Out with incorrect value
G15: Adversary exploits wireless transmission <Capnograph, Coordinator> to modify Cap_Data
2,4,5,12,13,15,19,20,21
Group 2b-3
G16: Adversary physically compromise Capnograph
1,2,17,21,25,26
G5: Adversary Exploits wireless transmission <Pulse Ox/ Capnograph, PC> to modify Ox_ctrl/ Cap_ctrl (e.g.. reduce sampling rate)
2,4,5,12,13,15,19,20,21
G7: Adversary Exploits wireless transmission <Pulse Ox/ Capnograph, Coordinator> to delay Cap_Out/ Ox_Out
2,4,5,12,13,15,19, 20,21
G13: Adversary exploits wireless transmission <pump, Coordinator> to modify Pump_Out
2,4,5,12,13,15,19,20,21
G14: Adversary physically compromise pump
2,17,21,26
Adversary modifies Ox_Data with incorrect value
G20: Adversary physically modify EHR database, modify patient's information (e.g.. changes previous medical history information)
1,2,4,5,14,15,1 7,18,19,20,24
G22: Adversary exploits wired transmission <EHR, PC> to modify EHR_Data_In
1,2,4,5,14,15,1 7,18,19,20,24
G4: Adversary Exploits wireless transmission <Pulse Ox/ Capnograph, PC> to delay Cap_ctrl/ Ox_ctrl
1,8,13,14,19,20,24
G6: Adversary physically compromise Capnograph/ Pulse Ox (e.g..pulse ox is broken and it can't send pulse ox regularly)
2,17,21,26
G17: Adversary physically compromise pulse ox (e.g..pulse ox is broken and measurement is always within normal range)
1,2,17,21,25,26
G18: Adversary exploits wireless transmission <pulse Ox, Coordinator> to modify Ox_Data
2,4,5,12,13,15,19,20,21
G21: Adversary exploits wireless transmission <EHR, Coordinator> to modify EHR_Data_Out
2,4,5,12,13,15,19,20,21
G8: Adversary Exploits wireless transmission <Pump, PC> to delay pump_Ctrl
1,8,13,14,19,20,24
G9: Adversary Exploits wireless transmission <Pump, PC> to modify pump_ctrl (e.g.. reduce sampling rate)
2,4,5,12,13,15,19,20,21
G10: Adversary Exploits wireless transmission <Pump, PC> to delay pump_out
1,8,13,14,19,20,24
G11: Adversary physically compromise pump (e.g.. pump is broken, it does not send pump_out regularly)
2,17,21,26

Figure 4.5: **Complete Attack Tree With Attached Countermeasures**

4.4.3 Quantification and Security Condition

After adding relevant countermeasures into attack leaves, the attack tree is complete. Now we could start to quantify security condition for PCA-IMDs. This section describes the procedure to evaluate the security condition.

In this regard, we first compute a security coefficient ω. **The security coefficient** is based on how many relevant types of countermeasures attack leaf implements divided by the total number of relevant countermeasure types available. The security coefficient ranges from 0 to 1. The value 1 indicates the highest security level while value 0 represents the lowest security level. There are five general countermeasure types in total; however, not all general types are applicable to all leaves. Therefore when we consider relevant countermeasures, we only consider those that apply to the attack leaf could help secure the system. For instance, a countermeasure installing anti-virus software is not suitable for medical devices like pulse oximeter, and hence is not relevant. In the attack tree in Figure 5.2, attack leaf 19 is "G19: Adversary Exploits Wireless Transmission <PC,

Coordinator> to Modify Care_Out." This leaf describes attacks on the wireless communications between PC and Coordinator. In this case, use "Access" to help prevent unauthorized access, install "Detection" tools to monitor abnormal circumstances, take "Protection" actions to defend against attacker, and we can use "Cryptography" to secure content delivered in between. In our case, we implement all of these 4 types, so the security coefficient, ω, for this leaf is 4 / 4 = 1.

A vulnerability index is a measure of the difficulty faced by hackers in compromising an attack leaf given implemented countermeasures. The vulnerability index ranges from 0 to 1, from the most vulnerable (0 value) to the most invulnerable (1 value). The whole procedures to evaluating the vulnerability index for each attack leaf are as follows:

- Calculate the percentage (P) of countermeasures implemented for each general type on an attack leaf. For each general type $P^k = 1 / \mathrm{T}_i$ (A_{i1} / B_{i1} + A_{i2} / B_{i2} $\ldots \mathrm{A}_{in}$ / B_{in}). T_i represents the total number of one subtype. A_{in} stands for the number of countermeasures implemented under that subtype while B_{in} shows total number of countermeasures available under that subtype. Finally K is the index of the attack leaf.

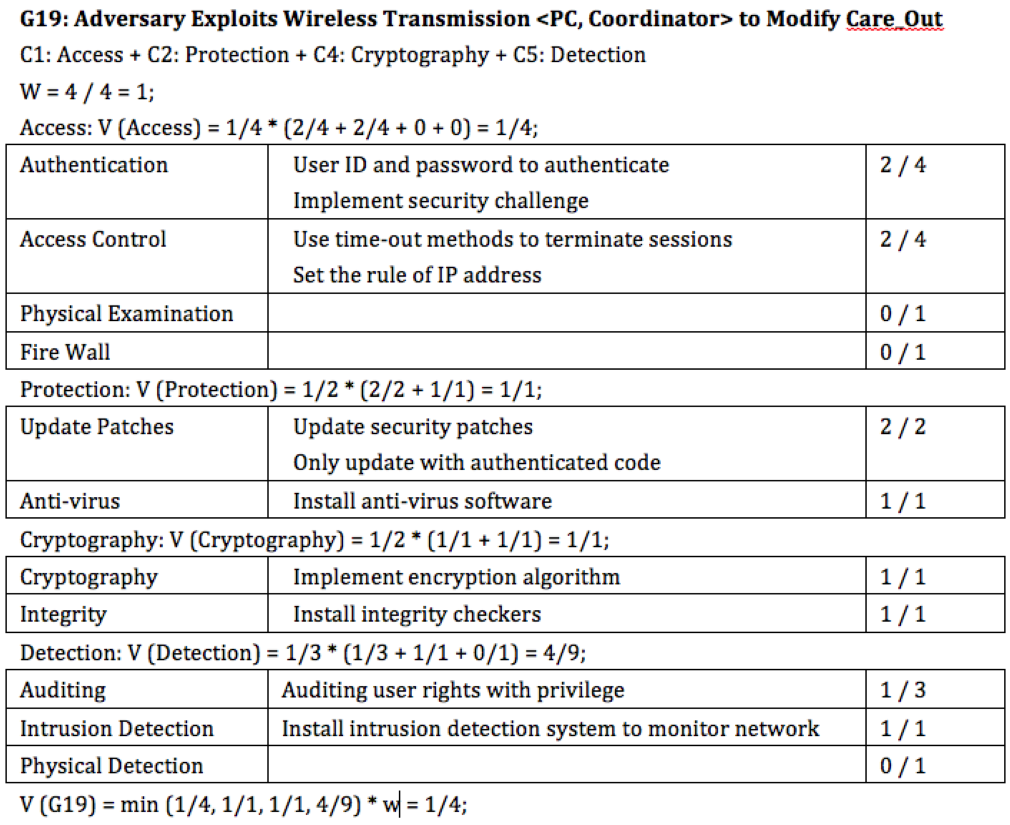
- Choose the minimum P_{min} so that P_{min} = min { P^1, $P^2 \ldots p^n$ }, where n is the total number of attack leaves in the tree.

- We define vulnerability index of a lead as $V^i = P_{min} * \omega$, where i is the index of attack leaf.

Figure 4.6 shows a process of computing security condition for G19 attack leaf. General type "Access" has 4 subtypes. G19 implements two countermeasures from subtype "Authentication", two countermeasures from subtype "Access Control" and nothing from other subtypes. So the percentage P_19 "Access" P_19 (Access) is 1/4 * (2/4 + 2/4 + 0 + 0) = 1/4. It works exact the same for other general types. In the end, vulnerability index for G19 leaf is P_{min} = min (1/4, 1/1, 1/1, 4/9) * ω = 1/4 (As we discussed earlier ω is 1).

From the previous steps, each attack leaf has already associated with one vulnerability index. This index shows how difficulty of compromising an attack leaf. The system security condition should take the least secure leaf node into consideration, so system security condition would be determined by the minimum vulnerability index. V_{min} = min { V^1, $V^2 \ldots V^n$ }, where n is the total number of attack leaves.

Algorithm 2 shows the pseudo code we used to quantify IMDs security conditions. It tells almost the same story as the steps above.



G19: Adversary Exploits Wireless Transmission <PC, Coordinator> to Modify Care Out

C1: Access + C2: Protection + C4: Cryptography + C5: Detection

W = 4 / 4 = 1;

Access: V (Access) = 1/4 * (2/4 + 2/4 + 0 + 0) = 1/4;

Authentication	User ID and password to authenticate Implement security challenge	2 / 4
Access Control	Use time-out methods to terminate sessions Set the rule of IP address	2 / 4
Physical Examination		0 / 1
Fire Wall		0 / 1

Protection: V (Protection) = 1/2 * (2/2 + 1/1) = 1/1;

Update Patches	Update security patches Only update with authenticated code	2 / 2
Anti-virus	Install anti-virus software	1 / 1

Cryptography: V (Cryptography) = 1/2 * (1/1 + 1/1) = 1/1;

Cryptography	Implement encryption algorithm	1 / 1
Integrity	Install integrity checkers	1 / 1

Detection: V (Detection) = 1/3 * (1/3 + 1/1 + 0/1) = 4/9;

Auditing	Auditing user rights with privilege	1 / 3
Intrusion Detection	Install intrusion detection system to monitor network	1 / 1
Physical Detection		0 / 1

V (G19) = min (1/4, 1/1, 1/1, 4/9) * w = 1/4;

Figure 4.6: **Quantification Example**

Algorithm 2 Quantification Algorithm

```
function QUANTIFICATION(List<Node> nodes)
    Double result = 0.0
    List<Node> pivotalNodes
    for each node in nodes do
        Double temp = 0.0
        Calculate Security Coefficient $\omega$
        $\omega$ = node.ImplementedCMT / node.TotalRelevantCMT        ▷ CMT : CounterMeasureType
        for each general Type under node do
            Initialize Calculate percentage (P) of countermeasures implemented
            P = 0
            P += node.generalType.implmented / node.generalType.total
        end for
        P = P / node.generalTypeNumber
        temp = temp > P ? P : temp        ▷ Choose minimal
        if result > temp * $\omega$ then
            pivotalNodes.add(node)
            result = temp * $\omega$
        end if
    end for
    return result,pivotalNodes
end function
```

Chapter 5

Results

In this section, we would like to discuss the results we obtained when applying our methodology to analyze PCA-IMDs security.

5.1 Experiment Description

In our experiment, we choose two PCA-IMDs scenarios for validation purpose to illustrate the comparative analysis capability of our approach. Both scenarios have same types and numbers of medical devices. The only difference here is the intelligence level of coordinator and how caregiver programs medical devices as well as the way caregiver receives feedback from each medical device during the operation. More specifically:

- **Human-Involved Loop**; This is the standard system model of PCA-IMD that we have assumed thus far. In this scenario, a caregiver uses a PC and uses that PC to send/receive information about decision in PCA-IMD. The caregiver sends a request to retrieve the patients' health records if exist. Based on patients' health information, the caregiver programs the infusion pump, the capnograph and the pulse ox though the PC. All medical devices involved are connected to the coordinator via wines. The coordinator keeps receiving status updates from the individual medical device. The coordinator itself is connected wirelessly to PC. Coordinator collects all medical data and sends them back to PC upon request. Caregivers monitor the PC to observe whether patients are doing well or not. If anything unexpected happens, caregiver can send commands to execute the weak therapy provided.

Below 5.1 is the system description for this test case.

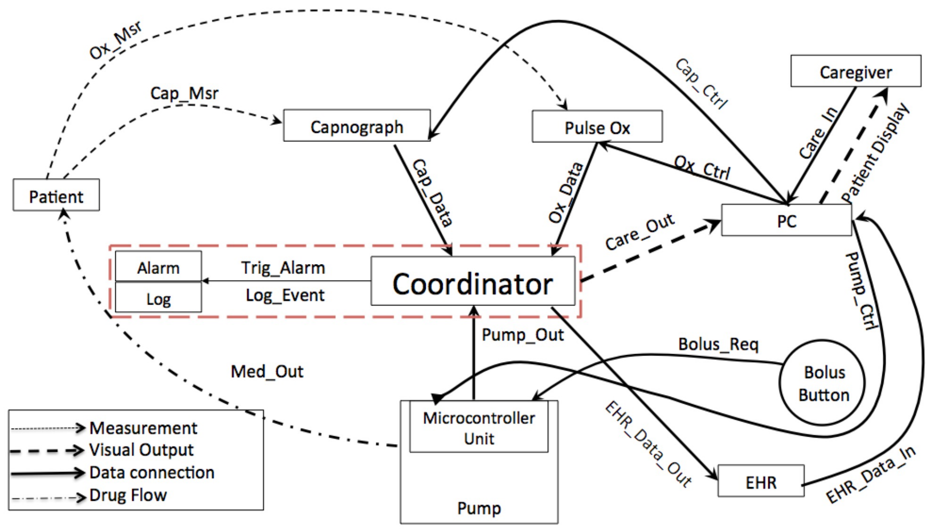


Figure 5.1: **System Model for Human-Involved Loop**

Based on this model, we generate the corresponding attack tree. Since we have already discussed that in the previous chapter, we will skip the generation process description here. In Figure 5.2 shows the attack tree for this particular setting.

- **Closed Loop**: This scenario involves a very intelligent coordinator. In this case, caregiver still uses PC to send/receive information, but all change in therapy of device is done by the coordinator. Coordinator processes the message and sends commands to each medical device. At the same time, coordinator gathers all medical data from three medical devices. Every so often the coordinator sends all information back to PC that the caregiver observes. Figure 5.3 is the system description for this scenario. And Figure 5.4 shows the detailed attack tree for closed loop setting.

5.2 IMDs-Security Analysis Tool (IMD-SAT)

In order to make our methodology more accessible, we provide a tool to automatically generate security condition reports based on attack tree. Here is a brief introduction of the tool we implemented to compute the system security condition. IMD-SAT consists of three main modules:

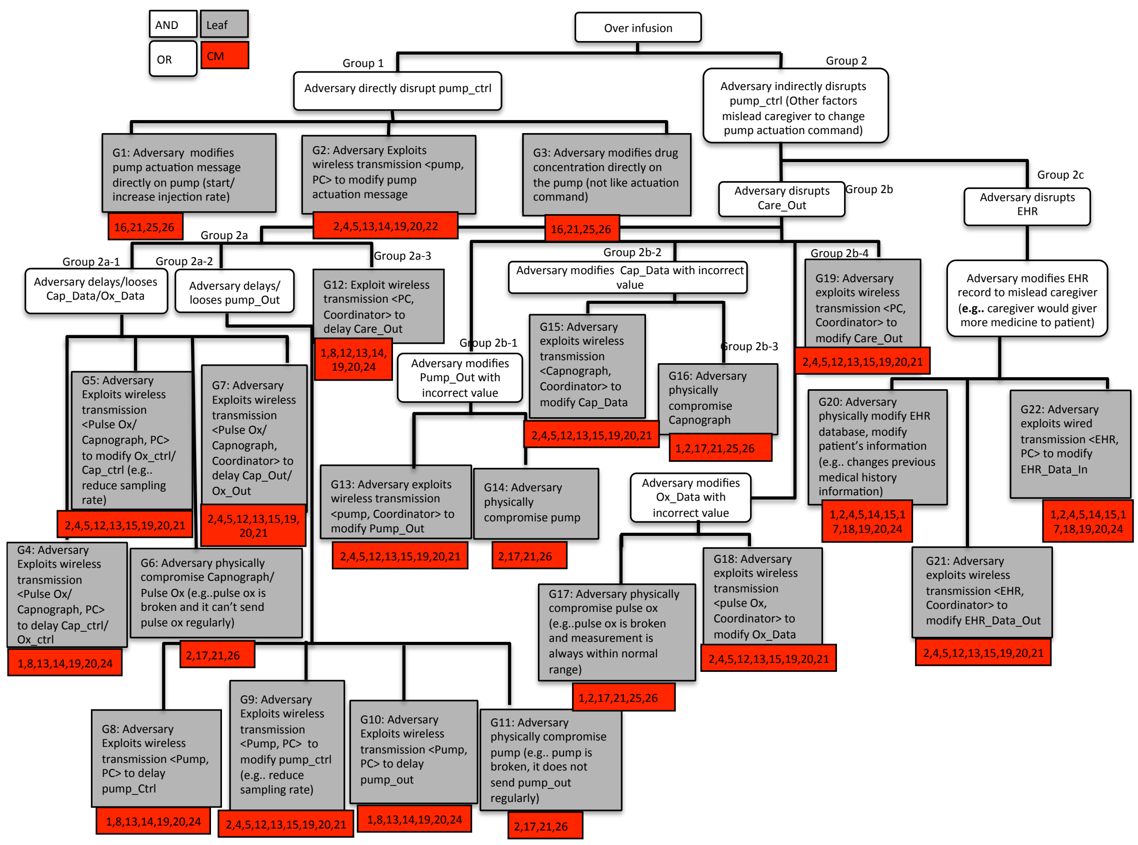

AND
Leaf
OR
CM
Over infusion
Group 1
Group 2
Adversary directly disrupt pump_ctrl
Adversary indirectly disrupts pump_ctrl (Other factors mislead caregiver to change pump actuation command)
G1: Adversary modifies pump actuation message directly on pump (start/ increase injection rate)
16,21,25,26
G2: Adversary Exploits wireless transmission <pump, PC> to modify pump actuation message
2,4,5,13,14,19,20,22
G3: Adversary modifies drug concentration directly on the pump (not like actuation command)
16,21,25,26
Group 2a
Group 2b
Group 2c
Adversary disrupts Care_Out
Adversary disrupts EHR
Group 2a-1
Group 2a-2
Group 2a-3
Adversary delays/looses Cap_Data/Ox_Data
Adversary delays/ looses pump_Out
G12: Exploit wireless transmission <PC, Coordinator> to delay Care_Out
1,8,12,13,14, 19,20,24
Group 2b-1
Group 2b-2
Group 2b-3
Group 2b-4
Adversary modifies Cap_Data with incorrect value
Adversary modifies Pump_Out with incorrect value
G15: Adversary exploits wireless transmission <Capnograph, Coordinator> to modify Cap_Data
2,4,5,12,13,15,19,20,21
G16: Adversary physically compromise Capnograph
1,2,17,21,25,26
G19: Adversary exploits wireless transmission <PC, Coordinator> to modify Care_Out
2,4,5,12,13,15,19,20,21
Adversary modifies EHR record to mislead caregiver (e.g.. caregiver would giver more medicine to patient)
G20: Adversary physically modify EHR database, modify patient's information (e.g.. changes previous medical history information)
1,2,4,5,14,15,1 7,18,19,20,24
G22: Adversary exploits wired transmission <EHR, PC> to modify EHR_Data_In
1,2,4,5,14,15,1 7,18,19,20,24
G5: Adversary Exploits wireless transmission <Pulse Ox/ Capnograph, PC> to modify Ox_ctrl/ Cap_ctrl (e.g.. reduce sampling rate)
2,4,5,12,13,15,19,20,21
G7: Adversary Exploits wireless transmission <Pulse Ox/ Capnograph, Coordinator> to delay Cap_Out/ Ox_Out
2,4,5,12,13,15,19, 20,21
G13: Adversary exploits wireless transmission <pump, Coordinator> to modify Pump_Out
2,4,5,12,13,15,19,20,21
G14: Adversary physically compromise pump
2,17,21,26
Adversary modifies Ox_Data with incorrect value
G4: Adversary Exploits wireless transmission <Pulse Ox/ Capnograph, PC> to delay Cap_ctrl/ Ox_ctrl
1,8,13,14,19,20,24
G6: Adversary physically compromise Capnograph/ Pulse Ox (e.g..pulse ox is broken and it can't send pulse ox regularly)
2,17,21,26
G17: Adversary physically compromise pulse ox (e.g..pulse ox is broken and measurement is always within normal range)
1,2,17,21,25,26
G18: Adversary exploits wireless transmission <pulse Ox, Coordinator> to modify Ox_Data
2,4,5,12,13,15,19,20,21
G21: Adversary exploits wireless transmission <EHR, Coordinator> to modify EHR_Data_Out
2,4,5,12,13,15,19,20,21
G8: Adversary Exploits wireless transmission <Pump, PC> to delay pump_Ctrl
1,8,13,14,19,20,24
G9: Adversary Exploits wireless transmission <Pump, PC> to modify pump_ctrl (e.g.. reduce sampling rate)
2,4,5,12,13,15,19,20,21
G10: Adversary Exploits wireless transmission <Pump, PC> to delay pump_out
1,8,13,14,19,20,24
G11: Adversary physically compromise pump (e.g.. pump is broken, it does not send pump_out regularly)
2,17,21,26

Figure 5.2: **Complete Attack Tree for Human-Involved Loop**

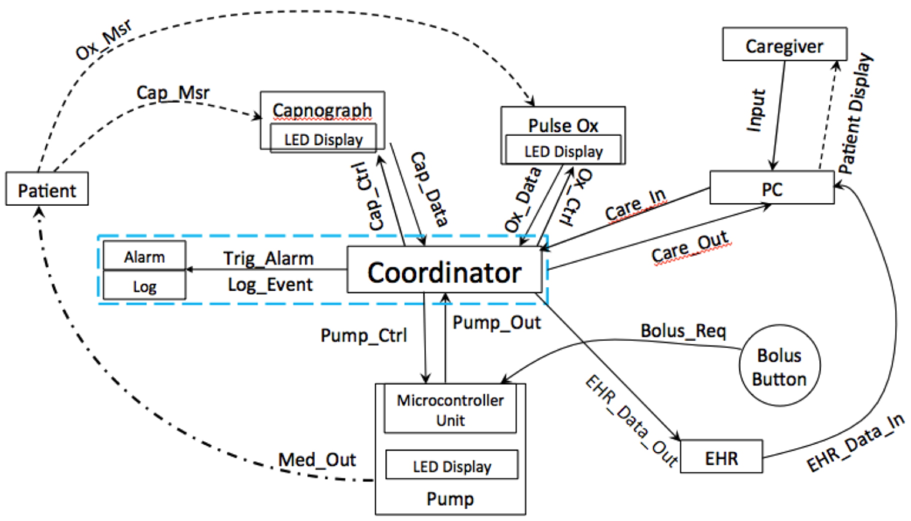


Figure 5.3: **System Model for Closed Loop**

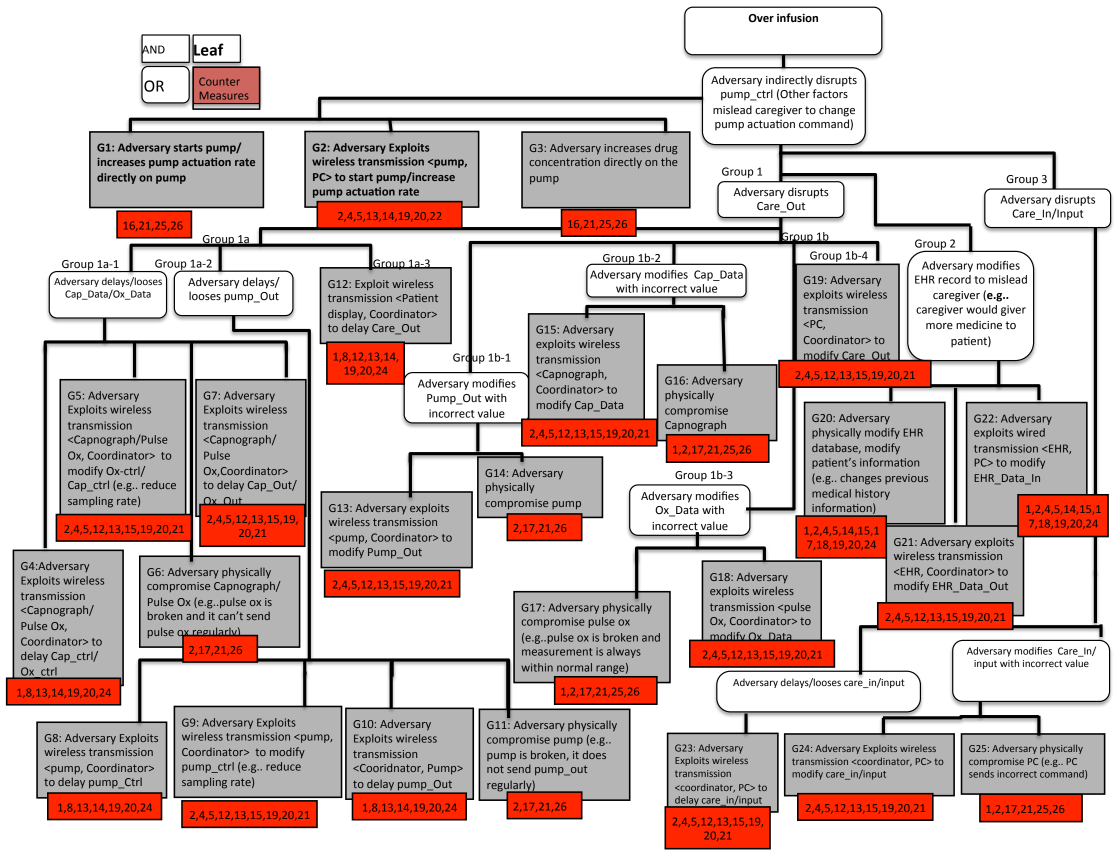

Over infusion
AND
Leaf
OR
Counter Measures
Adversary indirectly disrupts pump_ctrl (Other factors mislead caregiver to change pump actuation command)
G1: Adversary starts pump/ increases pump actuation rate directly on pump
16,21,25,26
G2: Adversary Exploits wireless transmission <pump, PC> to start pump/increase pump actuation rate
2,4,5,13,14,19,20,22
G3: Adversary increases drug concentration directly on the pump
16,21,25,26
Group 1
Adversary disrupts Care_Out
Group 3
Adversary disrupts Care_In/Input
Group 1a
Group 1b
Group 2
Group 1a-1
Adversary delays/looses Cap_Data/Ox_Data
Group 1a-2
Adversary delays/ looses pump_Out
Group 1a-3
G12: Exploit wireless transmission <Patient display, Coordinator> to delay Care_Out
1,8,12,13,14, 19,20,24
Group 1b-2
Adversary modifies Cap_Data with incorrect value
Group 1b-4
G19: Adversary exploits wireless transmission <PC, Coordinator> to modify Care_Out
2,4,5,12,13,15,19,20,21
Adversary modifies EHR record to mislead caregiver (e.g.. caregiver would giver more medicine to patient)
G15: Adversary exploits wireless transmission <Capnograph, Coordinator> to modify Cap_Data
2,4,5,12,13,15,19,20,21
Group 1b-1
Adversary modifies Pump_Out with incorrect value
G16: Adversary physically compromise Capnograph
1,2,17,21,25,26
G5: Adversary Exploits wireless transmission <Capnograph/Pulse Ox, Coordinator> to modify Ox-ctrl/ Cap_ctrl (e.g.. reduce sampling rate)
2,4,5,12,13,15,19,20,21
G7: Adversary Exploits wireless transmission <Capnograph/ Pulse Ox,Coordinator> to delay Cap_Out/ Ox_Out
2,4,5,12,13,15,19, 20,21
G14: Adversary physically compromise pump
2,17,21,26
G13: Adversary exploits wireless transmission <pump, Coordinator> to modify Pump_Out
2,4,5,12,13,15,19,20,21
Group 1b-3
Adversary modifies Ox_Data with incorrect value
G20: Adversary physically modify EHR database, modify patient's information (e.g.. changes previous medical history information)
1,2,4,5,14,15,1 7,18,19,20,24
G22: Adversary exploits wired transmission <EHR, PC> to modify EHR_Data_In
1,2,4,5,14,15,1 7,18,19,20,24
G21: Adversary exploits wireless transmission <EHR, Coordinator> to modify EHR_Data_Out
2,4,5,12,13,15,19,20,21
G4:Adversary Exploits wireless transmission <Capnograph/ Pulse Ox, Coordinator> to delay Cap_ctrl/ Ox_ctrl
1,8,13,14,19,20,24
G6: Adversary physically compromise Capnograph/ Pulse Ox (e.g..pulse ox is broken and it can't send pulse ox regularly)
2,17,21,26
G17: Adversary physically compromise pulse ox (e.g..pulse ox is broken and measurement is always within normal range)
1,2,17,21,25,26
G18: Adversary exploits wireless transmission <pulse Ox, Coordinator> to modify Ox_Data
2,4,5,12,13,15,19,20,21
Adversary delays/looses care_in/input
Adversary modifies Care_In/ input with incorrect value
G8: Adversary Exploits wireless transmission <pump, Coordinator> to delay pump_Ctrl
1,8,13,14,19,20,24
G9: Adversary Exploits wireless transmission <pump, Coordinator> to modify pump_ctrl (e.g.. reduce sampling rate)
2,4,5,12,13,15,19,20,21
G10: Adversary Exploits wireless transmission <Cooridnator, Pump> to delay pump_Out
1,8,13,14,19,20,24
G11: Adversary physically compromise pump (e.g.. pump is broken, it does not send pump_out regularly)
2,17,21,26
G23: Adversary Exploits wireless transmission <coordinator, PC> to delay care_in/input
2,4,5,12,13,15,19, 20,21
G24: Adversary Exploits wireless transmission <coordinator, PC> to modify care_in/input
2,4,5,12,13,15,19,20,21
G25: Adversary physically compromise PC (e.g.. PC sends incorrect command)
1,2,17,21,25,26

Figure 5.4: **Complete Attack Tree for Closed Loop**

- **Attack Tree Editor:** Like other editors, IMD-SAT attack tree editor has most of the functions as an editor, such as "Create a node," "Move a node" or "Connect two nodes." This feature could enable users to edit attack trees.

- **Countermeasures Editor:** This countermeasure editor does all this logic, users could use this to customize countermeasure as they want. Also this editor could add or delete the corresponding countermeasures to or from a leaf node. For PCA-IMDs, we produce 5 general types and several sub types. Five general types are: "Access", "Protection", "Backup", "Cryptography" and "Detection." In the appendix Table D.1, we have clearly stated all those countermeasures in detail. We divide all nodes into several types, they are: (1) Compromise Medical Device, (2) Compromise PCs and EHR database, (3) Exploit Wireless Connection Between Two components, (4) Delay/Loose data, and (5) Incorrect data. Basically these 5 types can cover all the nodes in attack trees. Therefore, we traverse all countermeasures available, and if that countermeasure is applicable, we add that countermeasure to that leaf node. In Appendix Table D.1, we also have clear descriptions for countermeasure for each node type.

- **System Security Analysis:** This analysis function implements the quantification mechanism. Once it is done with analyzing, IMD-SAT will generate a security report, informing users of system security conditions.

More information about this tool, the design, UML diagram and implement details are available in the Appendix C.1.

5.3 Result Analysis

In this section, we describe the security condition computation for our two PCA-IMD scenarios using IMD-SAT. Table 5.1 shows a example of what the content looks like for this scenario:

Table 5.1: Security Condition Analysis Results for Human-Involved Loop

Node	**Score**
Increase Flow(Basal) Rate	0.4375
Decrease SPO2 Sampling Rate	0.4375
Increase Drug Concentration	0.4375
Increase Bolus Flow Rate	0.4375
Delay/Loose Cap_msr/Ox_msr	0.4375
Delay Measurement of Pump(VTBI)	0.1875
Physical Compromise EHR Database	0.375
Exploit Wireless Transmission <EHR, Coordinator>	0.375
Delay/Loose EHR_query	0.1875
Exploit Wireless Transmission <PC, Coordinator>	0.25
Incorrect Pump_ctrl	$0.\dot{3}$
Delay/Loose Pump_ctrl	$0.\dot{3}$
Exploit Wireless Transmission <Pump, Coordinator>	0.1875
Incorrect Measurement of Pump(VTBI)	$0.1\dot{6}$
Exploit Wireless Transmission <Cap/Ox, Coordinator>	0.375
Incorrect Cap_msr/Ox_msr	$0.\dot{3}$
Physical Compromise EHR Database	0.375
Incorrect EHR_query	0.125
Exploit Wireless Transmission <EHR, Coordinator>	$0.\dot{3}$
Exploit Wireless Transmission <PC, Coordinator>	$0.\dot{3}$
System Security Condition is 0.125	
(With 0 to be the most insecure, while 1 to be the most secure)	
The least secure node is Incorrect EHR_query	

We can see from above table (dot above the number represents infinite decimals), the highest vulnerable index of this human-involved loop scenario is 0.4375 which means it is most secure. However, the most vulnerable leaf is 0.125, which is "Incorrect EHR_query". EHR database is the most vulnerable part of this scenario. Please note that vulnerability indexes for the wireless connection are very low, thus if we are able to enhance security condition for EHR database and wireless connection, transmission, then the whole system will become more secure. The reason to make wireless communication and EHR database related leaves have low vulnerability index values is that these two components are very complicated. Complicated means there are so many ways to break into the components. The more complicated one component is, the more difficult to secure one component. Even though we attach as many countermeasures to those leaves, there still have some negligence.

In the closed loop setting, the coordinator plays a very important role. This is a more complex model compared with the previous model. Table 5.2 shows a example of the IMD-SAT output for closed loop setting:

One can see from above table, the highest vulnerable index of this closed loop setting could only reach 0.41667 (not as high as previous model 0.4375). Since this scenario has higher intelligent level, then the requirement for each medical device has also increased in order to guarantee security. Thus almost all leaves vulnerabilities decreases more or less. And the most vulnerable leaves are only 0.1, which are "Exploit Wireless Transmission <EHR, Coordinator>" and "Incorrect EHR_query". EHR database and wireless connection again become the most vulnerable parts for closed loop setting. The reason is similar to the human-involved loop setting.

5.4 Comparative Analysis

Below Table 5.3 compares the security condition and vulnerability of the two scenarios. We can clearly see the dynamic changes depending on the functionality of the system given the same countermeasures:

From the Table 5.3, we see the security condition decreases when system becomes more intelligent. From those results, we conclude:

- **Trade-off between functionality and security**. Human-involved loop is slightly more secure compared with a closed loop setting. It is true that people want to design a more intelligent system to avoid mistakes by humans. However, the sophisticated system is still

Table 5.2: Security Condition Analysis Results For Closed Loop Setting

Node	**Score**
Increase Flow(Basal) Rate	$0.\dot{3}$
Decrease SPO2 Sampling Rate	$0.\dot{3}$
Increase Drug Concentration	$0.\dot{3}$
Increase Bolus Flow Rate	$0.\dot{3}$
Delay/Loose Cap_msr/Ox_msr	$0.41\dot{6}$
Exploit Wireless Transmission <Capnograph /Pulse Ox, Coordinator>	$0.1\dot{6}$
Exploit Wireless Transmission <Pump, Coordinator>	0.1875
Delay Measurement of Pump(VTBI)	0.1875
Physical Compromise EHR Database	0.3125
Exploit Wireless Transmission <EHR, Coordinator>	0.375
Delay/Loose EHR_query	$0.1\dot{6}$
Exploit Wireless Transmission <PC, Coordinator>	$0.\dot{3}$
Incorrect Pump_ctrl	$0.\dot{3}$
Delay/Loose Pump_ctrl	$0.\dot{3}$
Exploit Wireless Transmission <Pump, Coordinator>	$0.1\dot{6}$
Incorrect Measurement of Pump(VTBI)	$0.41\dot{6}$
Exploit Wireless Transmission <Cap/Ox, Coordinator>	0.375
Incorrect Cap_msr/Ox_msr	$0.\dot{3}$
Physical Compromise EHR Database	0.375
Incorrect EHR_query	0.1
Exploit Wireless Transmission <EHR, Coordinator>	0.1
Exploit Wireless Transmission <PC, Coordinator>	$0.\dot{3}$
Exploit wireless Transmission <PC, Coordinator>	$0.\dot{3}$
Physically Compromise PC	$0.\dot{3}$
Physically Modified Input	0.1875
System Security Condition is 0.1	
(With 0 to be the most insecure, while 1 to be the most secure)	
The least secure node is Exploit Wireless Transmission <EHR, Coordinator>	
The least secure node is Incorrect EHR_query	

designed by humans. And people are not machines, and can make errors which could turn out to be a disaster [13]. Meanwhile, all the internal connections or associations between any two components within the system end up being more complex. Therefore, the more complicated (human-involved loop is simpler than closed loop) the system, the more functionality for it to have, and the more likely for it to have security vulnerabilities.

- **Database and secure communication within the IMD setup is paramount**. For the least vulnerable nodes, we can see all of them are more or less associated with EHR database or wireless connections and transmissions. In this case, this should inform us that EHR database as well as wireless transmission should be treated seriously in order to enhance security condition for particular PCA-IMDs.

- **IMD security is always decided by its most vulnerable part**. When designers are working on constructing a new IMD setup, they have to make sure they are protecting every component.

- **Individual medical device security does not equate to interoperability security**. A device can be formally defined as "safe" if and only if none of its execution paths invoke a particular set of negative actions. However, the safety of any particular medical device and the coordinator are insufficient to ensure that it remains safe during the entire execution on the interoperable setting. In our system model, adversary induced misinformation or bad input can cause an infusion pump to over-infuse medication, endangering patient safety. This condition can occur even if the infusion pump is guaranteed to be secure.

- **Attacks from compromised entities in the interoperability are difficult to prevent**. If any of the three main types of entities in the interoperability setup, namely the sensors, the caregiver, and pump can be compromised, then the traditional information security solutions described for securing the inputs are rendered moot. One can use redundancy to attempt to detect events of compromise, but this requires at least one uncompromised device.

Table 5.3: Comparative System Security Condition Analysis Results

Scenario	**Security Score**	**Least Vulnerable Node**
Human-involved loop	0.125	Incorrect EHR_query
Closed loop	0.1	Incorrect EHR_query
		Exploit Wireless Transmission <EHR and Coordinator>

Chapter 6

Conclusions and Future Work

Security for medical devices has gained some attention in recent years following some well-publicized attacks on individual devices, such as pacemakers and insulin pumps. This has resulted in solutions being proposed for securing these devices, usually in a stand-alone mode. However, medical devices are becoming increasingly interconnected and interoperable as a way to improve patient safety, decrease false alarms, and reduce clinician cognitive workload. Given the nature of interoperable medical devices (IMDs), attacks on IMDs can have devastating consequences. This work shows the importance of securing IMDs system and outlines our effort in understanding the threats faced by IMDs, an important first step in eventually designing secure interoperability architectures.

In this work, we develop a methodology for generating attack trees based on PCA-IMD workflow modeling and hazard analysis. In general this brand new process contains four steps: (1) process modeling, (2) fault tree analysis (FTA), (3) attack tree generation and (4) quantification. First, the user (deployer) of the PCA-IMD takes an abstract description of the workflow of PCA-IMD and develops a process modeling representation for it. This representation essentially describes the operation of the PCA-IMD. Once the process model is developed, the IMD user identifies the various hazards that can occur due to the operation of the system. The hazards are essentially unsafe states of the PCA-IMD, which can harm the system itself or the user. In our case, the unsafe state of interest with respect to azard analysis are those that eventually lead to over-infusion of pain medication, our attack goal. Given the hazards that lead to over-infusion and the process model of the IMD, we derive several fault-trees for the system. The fault-trees essentially provide a representation of the means by which the hazards can be realized due to faults within the system.

Once the fault-trees have been extracted, we combine them in a meaningful fashion to generate a global attack tree for the PCA-IMD setup such that the root node of the tree is the attack goal. With the attack tree available, we construct the countermeasure data set and attach some of the countermeasures to attack leaf element. We calculate the security condition for PCA-IMD setup to get an overall understanding of IMDs security conditions.

Finally, we develop a IMD-Security Analysis Tool (IMD-SAT). IMD-SAT simulates our methodology and is integrated into one software application, so anyone could use this application to help test and verify the security condition of an IMD setup.

6.1 Future Work

We demonstrate that this work has its own value. However, there are still some limitations we can address in the near future to make it more applicable. In general, we plan to extend our work as follows:

- **Automatic attack tree generation**. Currently, we have pseudo code to convert fault trees into an attack tree. However, it is still a manual process to generate the attack trees. In the future, one could develop a tool that automatically helps convert the process model and hazard description into attack trees.

- **Quantification**. Currently quantification is based on how many countermeasures an attack leaf implements. In the future, one could come up with a more rigorous way to quantify our attack trees. This quantification method has to be based on mathematics model of the countermeasure efficacy.

- **Generalize the methodology**. Currently, our approach is only applicable for IMDs systems, but we may be able to extend our methodology to other domains.

Bibliography

[1] Robert K Abercrombie, Frederick T Sheldon, Katie R Hauser, Margaret W Lantz, and Ali Mili. Risk assessment methodology based on the nistir 7628 guidelines. In *System Sciences (HICSS), 2013 46th Hawaii International Conference on*, pages 1802–1811. IEEE, 2013.

[2] David Arney, Sebastian Fischmeister, Julian M Goldman, Insup Lee, and Robert Trausmuth. Plug-and-play for medical devices: Experiences from a case study. *Biomedical Instrumentation & Technology*, 43(4):313–317, 2009.

[3] Henk Birkholz, Stefan Edelkamp, Florian Junge, and Karsten Sohr. Efficient automated generation of attack trees from vulnerability databases. In *Working Notes for the 2010 AAAI Workshop on Intelligent Security (SecArt)*, pages 47–55, 2010.

[4] Aaron G Cass, AS Lerner, Eric K McCall, Leon J Osterweil, Stanley M Sutton Jr, and Alexander Wise. Little-jil/juliette: a process definition language and interpreter. In *Software Engineering, 2000. Proceedings of the 2000 International Conference on*, pages 754–757. IEEE, 2000.

[5] Bin Chen, Lori A Adviser-Clarke, and George S Adviser-Avrunin. *Improving processes using static analysis techniques.* University of Massachusetts Amherst, 2011.

[6] Stefan Christov, Bin Chen, George S Avrunin, Lori A Clarke, Leon J Osterweil, David Brown, Lucinda Cassells, and Wilson Mertens. Rigorously defining and analyzing medical processes: An experience report. In *Models in software engineering*, pages 118–131. Springer, 2008.

[7] Bill Curtis, Marc I Kellner, and Jim Over. Process modeling. *Communications of the ACM*, 35(9):75–90, 1992.

[8] Cory Doctorow. Drug pump is most insecure devices ever seen by researcher.

[9] Danny Dolev and Andrew C Yao. On the security of public key protocols. *Information Theory, IEEE Transactions on*, 29(2):198–208, 1983.

[10] eclipse. eclipse.

[11] JM Goldman. Medical devices and medical systems-essential safety requirements for equipment comprising the patient-centric integrated clinical environment (ice). *ASTM final F-2761-2009*, 2009.

[12] Somesh Jha, Oleg Sheyner, and Jeannette Wing. Two formal analyses of attack graphs. In *Computer Security Foundations Workshop, 2002. Proceedings. 15th IEEE*, pages 49–63. IEEE, 2002.

[13] Jesper M. Johansson. The fundamental tradeoffs.

[14] BaekGyu Kim, Anaheed Ayoub, Oleg Sokolsky, Insup Lee, Paul Jones, Yi Zhang, and Raoul Jetley. Safety-assured development of the gpca infusion pump software. In *Proceedings of the ninth ACM international conference on Embedded software*, pages 155–164. ACM, 2011.

[15] UMass University LASER Lab. Little jil.

[16] UMassAmherst LASER. Laser.

[17] UMassAmherst LASER. Laser.

[18] Isograph Ldt. Attacktree+ for windows, version 1.0, attack tree analysis.

[19] Insup Lee, Oleg Sokolsky, Sanjian Chen, John Hatcliff, Eunkyoung Jee, BaekGyu Kim, Andrew King, Margaret Mullen-Fortino, Soojin Park, Alex Roederer, et al. Challenges and research directions in medical cyber–physical systems. *Proceedings of the IEEE*, 100(1):75–90, 2012.

[20] Nancy G Leveson. *Safeware: system safety and computers*. ACM, 1995.

[21] Richard Paul Lippmann and Kyle William Ingols. An annotated review of past papers on attack graphs. Technical report, DTIC Document, 2005.

[22] Amenaza Technologies Ltd. Securitree, attack tree modelling.

[23] Andrew P Moore, Robert J Ellison, and Richard C Linger. Attack modeling for information security and survivability. Technical report, DTIC Document, 2001.

[24] Leon J Osterweil, Norman K Sondheimer, Lori A Clarke, Ethan Katsh, and Daniel Rainey. Using process definitions to facilitate the specifications of requirements. *Department of Computer Science, University of Massachusetts, Amherst, MA*, 1003, 2006.

[25] Stéphane Paul. Towards automating the construction & maintenance of attack trees: a feasibility study. *arXiv preprint arXiv:1404.1986*, 2014.

[26] Huong Phan, George Avrunin, Matt Bishop, Lori A Clarke, and Leon J Osterweil. A systematic process-model-based approach for synthesizing attacks and evaluating them. In *Proc. of the international conference on Electronic Voting Technology/Workshop on Trustworthy Elections (EVT/WOTE). USENIX Association*, 2012.

[27] Wolter Pieters, Trajce Dimkov, and Dusko Pavlovic. Security policy alignment: A formal approach. *Systems Journal, IEEE*, 7(2):275–287, 2013.

[28] Ronald W Ritchey and Paul Ammann. Using model checking to analyze network vulnerabilities. In *Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on*, pages 156–165. IEEE, 2000.

[29] Bruce Schneier. Modeling security threats.

[30] Bruce Schneier. *Secrets and lies: digital security in a networked world*. John Wiley & Sons, 2011.

[31] Oleg Sheyner, Joshua Haines, Somesh Jha, Richard Lippmann, and Wing. Automated generation and analysis of attack graphs. In *Security and privacy, 2002. Proceedings. 2002 IEEE Symposium on*, pages 273–284. IEEE, 2002.

[32] Graphviz Graph Visualization Software. Graphviz.

[33] Curtis R Taylor, Krishna Venkatasubramanian, and Craig A Shue. Understanding the security of interoperable medical devices using attack graphs. In *Proceedings of the 3rd international conference on High confidence networked systems*, pages 31–40. ACM, 2014.

[34] Chee-Wooi Ten, Chen-Ching Liu, and Manimaran Govindarasu. Vulnerability assessment of cybersecurity for scada systems using attack trees. In *Power Engineering Society General Meeting, 2007. IEEE*, pages 1–8. IEEE, 2007.

[35] UMassAmherst. Welcome to laser.

[36] Axel Van Lamsweerde, Simon Brohez, Renaud De Landtsheer, David Janssens, et al. From system goals to intruder anti-goals: attack generation and resolution for security requirements engineering. *Proc. of RHAS*, 3:49–56, 2003.

[37] LYNNE WALLIS. How insulin pumps could give a fatal overdose to diabetics: The 'foolproof' alternative to daily injections.

[38] Axel Wirth. Cybercrimes pose growing threat to medical devices. *Biomedical Instrumentation & Technology*, 45(1):26–34, 2011.

[39] A Wise. Little-jil 1.5 language report department of computer science, university of massachusetts. Technical report, Amherst UM-CS-2006-51, 2006.

[40] Michael Wong. How often do errors with patient-controlled analgesia (pca) occur?, 2011.

[41] Feng Xie, Tianbo Lu, Xiaobo Guo, Jingli Liu, Yong Peng, and Yang Gao. Security analysis on cyber-physical system using attack tree. In *Intelligent Information Hiding and Multimedia Signal Processing, 2013 Ninth International Conference on*, pages 429–432. IEEE, 2013.

Appendices

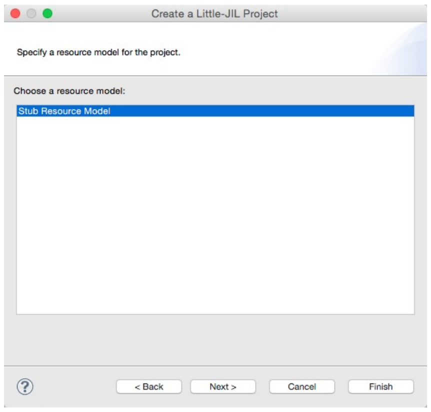
Appendix A

Little-JIL

A.1 Little-JIL

If users want to use this visual Little-JIL tool, they could refer to [16] to download and install. Visual Little-JIL is built upon the Eclipse IDE for Java Developers [10]. If you do not have this version of Eclipse (or a superset), you must install it first. Visual Little-JIL is distributed using the Eclipse Update Manager.

Once we download and install visual Little-JIL, we need to create a new Little-JIL project. Above is exactly one example of how to create a Little-JIL project. We set “Resource Model” as “Stub Resource Model” and “Stub Artifact Model” as artifact model, which is based on LASER [35] tutorial. Once we create a new Little-JIL project, we could create a new module with extension of “.ljx” under each module. Inside of each module, they should be able to create individual diagram to simulate target systems. User could click and editor that diagram. Below Figure A.3 shows how it looks like. More details about little-JIL can be found in the Little-JIL language report [39].

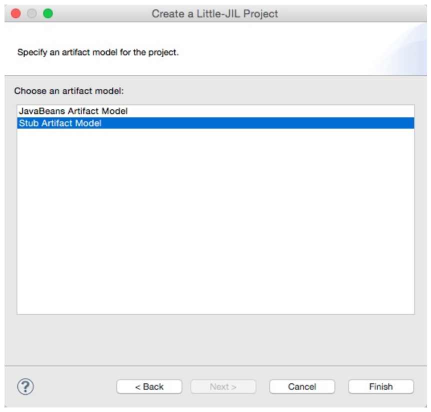


Specify a resource model for the project.

Choose a resource model:

Stub Resource Model

Figure A.1: **Little-JIL Initialization I**

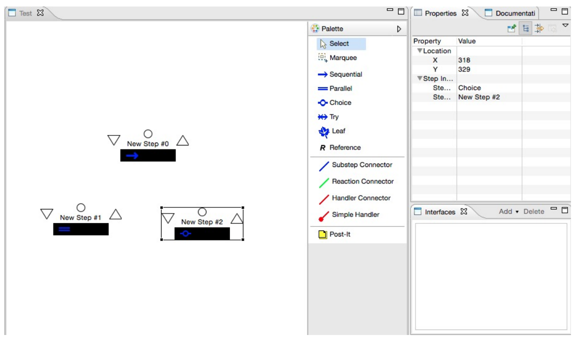


Specify an artifact model for the project.

Choose an artifact model:

JavaBeans Artifact Model
Stub Artifact Model

Figure A.2: **Little-JIL Initialization II**



Property	Value
▾Location	
X	318
Y	329
▾Step In…	
Ste…	Choice
Ste…	New Step #2

Figure A.3: **Little-JIL User Interface**

Appendix B

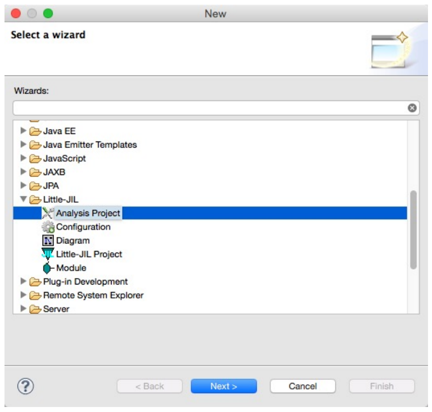
Fault Tree

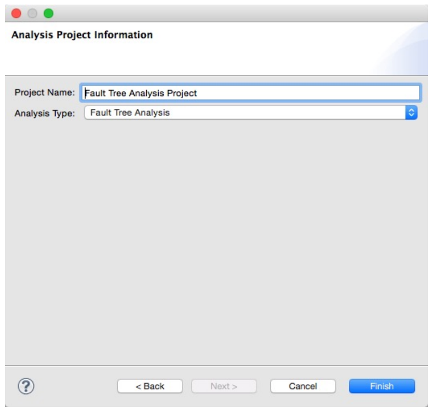
B.1 Fault Tree Derivation

The automation tool to derive fault tree we used is developed by [17], it is called “Visual Little-JIL Analyzer”. The visual Little-JIL Analyzers are built upon Visual-JIL and the Eclipse IDE for Java Developers [10]. If you have not installed Visual Little-JIL you will have to install it first. The visual Little-JIL Analyzers are distributed using the Eclipse Update Manager.

Once we download visual Little-JIL analyzer, we need to create a new fault tree analysis project like the above example Figure B.1. And then you click “next” and enter “Project Name” and “Analysis Type”. A new analysis project is set, and now user should expand the project, fill in “Process Information” with you target Little-JIL project, module and root step, then submit by clicking “Save” button down.

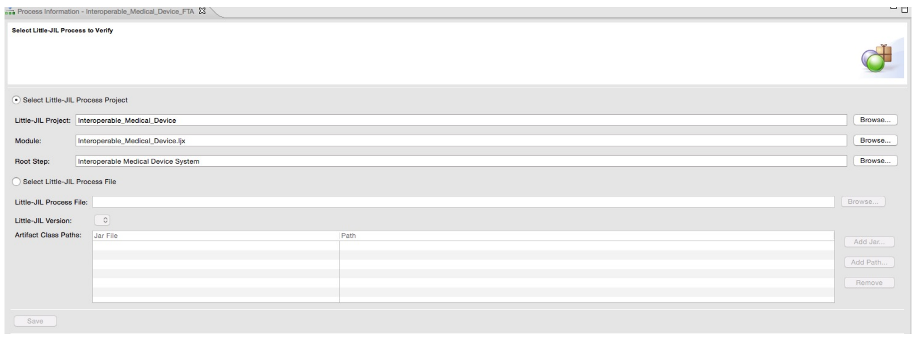
To derive a fault tree specific to one hazard, the user should just right click “Top Event Settings”, and choose “New Setting”. This top new event is defining what hazard user wants to know about the system. A new window like Figure B.4 will pop out. For this “Top Event Definition”, we need to specify “Step Pattern”, “Artifact Name” and “Error Type”. “Step Pattern” is exactly the name of the step, “Artifact Name” tells what artifact you want to examine, and “Error Type” has two options: “Wrong Output” or “Wrong Input”. Fill out proper information for your system and save the changes. After this, user completes defining a hazard, so now user simply just click “Derive Fault Tree” button, then little-JIL analyzer will return you one fault tree demonstrating how that hazard will happen. These fault trees are files with “dot” extensions.





Project Name: Fault Tree Analysis Project
Analysis Type: Fault Tree Analysis

Figure B.1: **Visual Little-JIL Analysis I**Figure B.2: **Visual Little-JIL Analysis II**


Process Information - Interoperable_Medical_Device_FTA

Select Little-JIL Process to Verify

(●) Select Little-JIL Process Project

Little-JIL Project: Interoperable_Medical_Device

Module: Interoperable_Medical_Device.ljx

Root Step: Interoperable Medical Device System

( ) Select Little-JIL Process File

Little-JIL Process File:

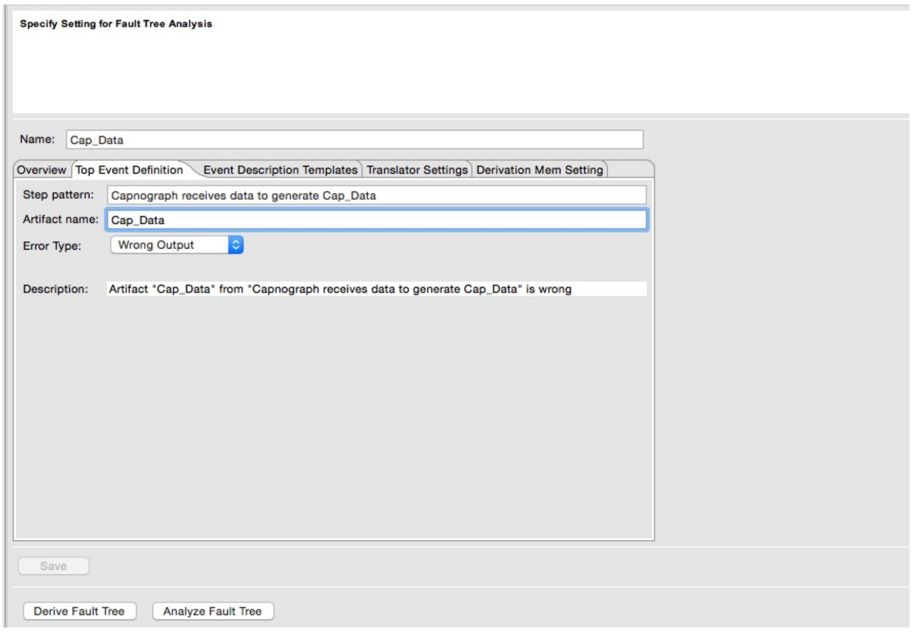
Little-JIL Version:

Artifact Class Paths:

Jar File	Path

Figure B.3: **Fill in Process Information**

There is a recommended application that could help us to open and view those dot files in a diagram format: Graphviz [32].


Specify Setting for Fault Tree Analysis

Name: Cap_Data

Step pattern: Capnograph receives data to generate Cap_Data

Artifact name: Cap_Data

Error Type: Wrong Output

Description: Artifact "Cap_Data" from "Capnograph receives data to generate Cap_Data" is wrong

Figure B.4: **Hazard (Top Event) Definition**

Appendix C

Tool

C.1 IMDs-Security Analysis Tool (IMD-SAT)

In order to make our tool more and more accessible, we would provide an automated tool called IMDs-Security Analysis Tool (IMD-SAT). Here is a brief introduction of the tool we implemented to simulate our methodology of assessing system security condition. For the default countermeasures in IMD-SAT, they are exactly the same as shown in our case study.

C.2 Application Design

IMD-SAT consists of three main modules: (1) attack tree editor, (2) countermeasures editor and (3) system security analysis.

C.2.1 Attack Tree Editor

The paramount functionality IMD-SAT should have is attack tree editor. All of the subsequent events have to be done based on an attack tree model, thus attack tree editor becomes prerequisites in this case. Originally we plan to integrate visual Little-JIL, fault tree analysis and attack tree conversion into IMD-SAT, however due to licenses and other copyright issues, we decided to move on to attack tree directly. Our application could perform attack tree editor function, so users could use IMD-SAT to build their own attack trees directly.

Initialize Attack Tree

When user opens IMD-SAT, it should pop up a select menu asking user which attack tree model they want to choose and edit. If user has never used it before, just clicks drop-down menu and selects "Others". If not, the menu should list all previous saved attack tree models available for user to pick, user could either click saved model or click "Others" to create a new one. Figure C.2 and Figure C.1 shows two initializing menu examples. Second example, user has already got one model (System Model I).

Attack Tree Editor

Like other editors, IMD-SAT attack tree editor has most of the functions as an editor. Let's briefly go through those features:

- Create Nodes: User clicks button "Create" in the interface, a menu will pop up and ask user what is new node's name and what type is the new node. For the type, user could choose

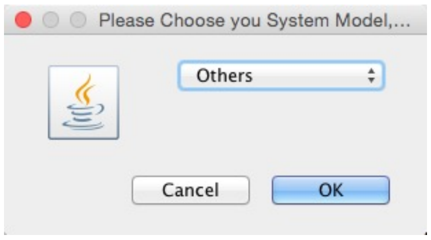


Figure C.1: **Menu Initialization I**

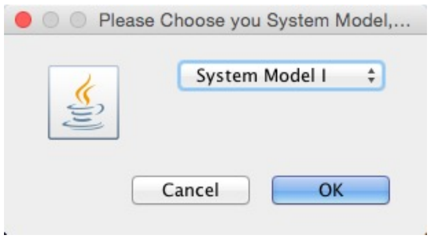


Figure C.2: **Menu Initialization II**

four relations “AND”, “OR”, “NOT” and “LEAF”. The first three relations represent this is an intermediate node, while “LEAF” tells this node is a leaf node. Also different relations are all linked to different colors to help user distinguish which node belongs to which type. Figure C.4 shows what create a new node looks like.

- Move Nodes: After that node is created, system will place that node in a default location of the interface, user is able to move that node to wherever the user want the node to stay. The way to do this is: user first clicks to select one particular node and drag the selected word to target place and stop. During user is dragging the whole, the whole canvas will auto-adjust the size dynamically.

- Connect Nodes. Once user clicks “Connect”, and then user clicks two nodes continuously, system will connect two nodes with a line. From system’s perspective, the first clicked node is going to be treated as parent node, and second clicked one will stay as child node. If user clicks to connect two nodes in reversed order (Trying to connect intermediate node to a leaf node), system will pop out an exception indicating this action is not allowed. Below Figure C.3 shows an example of connecting two nodes.

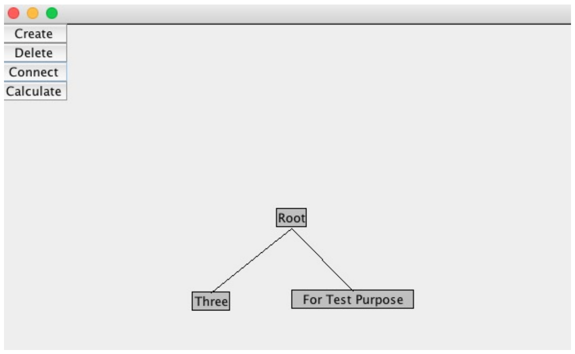


Figure C.3: **Connecting Two Nodes**

- Delete Nodes: User clicks button “Delete” in the interface, a jump-out menu will pop out and let user select which node the system is going to remove. Once user submit the information and system will wipe out that node from the interface. If user tries to delete a node before creating it, system will throw an exception saying this action is not allowed. Figure C.5 shows the menu window for deleting a node.

- Save Model: If all of sudden, user has to stop editing the current model, but user doesn’t want to discard all those changes, just click exit on the left corner, system will automatically save whatever current model to the directory, then user should be able to see one configuration file with extension “.storage”. This configuration is auto-saved model, next time when users want to load back, what they need to do is to choose select under initialize drop-down menu.

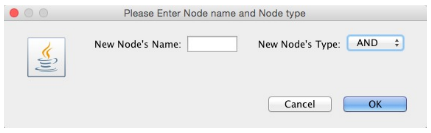


Figure C.4: **Create a New Node**

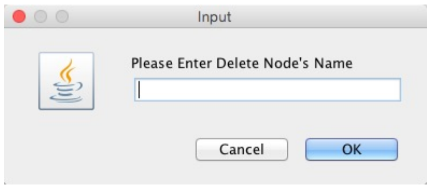


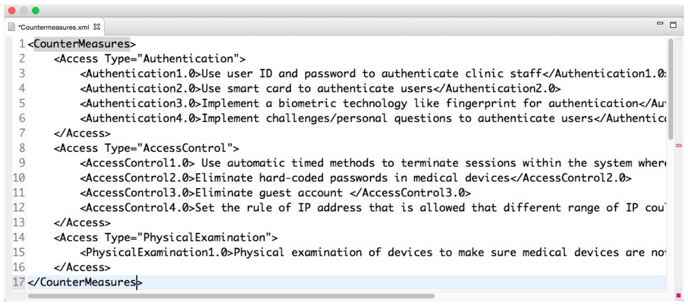
Figure C.5: **Delete an Existing Node**

C.2.2 Countermeasure Editor

The next functionality is countermeasure editor. After we have our attack tree model, what we need is to attach applicable countermeasures to each attack leaf. However, before this, we have to construct a countermeasure database for the target system.

Construct countermeasures

IMD-SAT reads all countermeasures from an XML configuration file. Thus as long as users finish researching about countermeasures, they could just simply insert those entries into a file called "Countermeasures.xml". Below Figure C.6 shows one "Countermeasures.xml" example we used for our case study PCA-IMDs. Different systems could have different sets of countermeasures.



```xml
<CounterMeasures>
    <Access Type="Authentication">
        <Authentication1.0>Use user ID and password to authenticate clinic staff</Authentication1.0
        <Authentication2.0>Use smart card to authenticate users</Authentication2.0>
        <Authentication3.0>Implement a biometric technology like fingerprint for authentication</Au
        <Authentication4.0>Implement challenges/personal questions to authenticate users</Authentic
    </Access>
    <Access Type="AccessControl">
        <AccessControl1.0> Use automatic timed methods to terminate sessions within the system wher
        <AccessControl2.0>Eliminate hard-coded passwords in medical devices</AccessControl2.0>
        <AccessControl3.0>Eliminate guest account </AccessControl3.0>
        <AccessControl4.0>Set the rule of IP address that is allowed that different range of IP cou
    </Access>
    <Access Type="PhysicalExamination">
        <PhysicalExamination1.0>Physical examination of devices to make sure medical devices are no
    </Access>
</CounterMeasures>
```

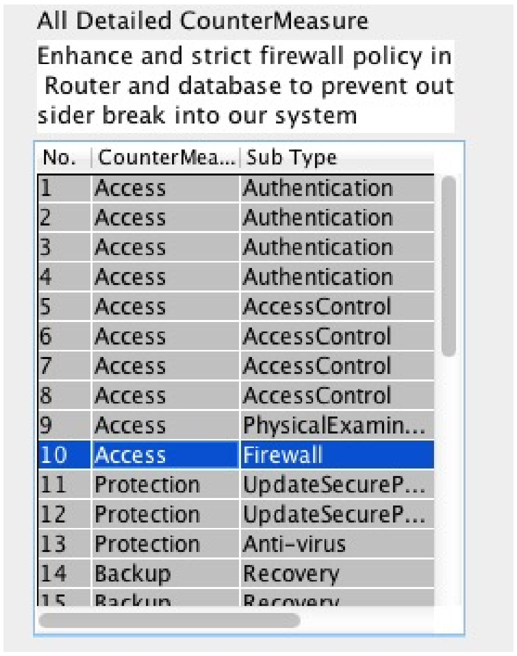
Figure C.6: **Countermeasures.xml example**

Display Countermeasures

After our application starts, it will find "Countermeasures.xml" file in current directory, if that file exists, then IMD-SAT will import all data in "Countermeasures.xml" file and show it to the user with the following format. As we can see from Figure C.7, there are two parts: one is a text field showing all detailed description of a countermeasure; the other one is a table indicating countermeasure type, sub type and value (value equals to the description here). If user clicks one of the entries, then that entry's description will show up automatically in the text field for user to see all information.

Attach/Detach Countermeasures

As we discussed in methodology chapter, before we tries to assess system security, what we need to do is to calculate security condition "ω". In order to calculate that number, we have to set relevant countermeasure types first. Figure C.8 shows the window to set relevant countermeasures. User double clicks one node, there should have two noticeable changes. One is text-field with value "name" will update to show node's name, second one is the window should automatically be updated to show what is current setting for relevant countermeasures is. If user wants to reset,



All Detailed CounterMeasure

Enhance and strict firewall policy in Router and database to prevent out sider break into our system

No.	CounterMea...	Sub Type
1	Access	Authentication
2	Access	Authentication
3	Access	Authentication
4	Access	Authentication
5	Access	AccessControl
6	Access	AccessControl
7	Access	AccessControl
8	Access	AccessControl
9	Access	PhysicalExamin...
10	Access	Firewall
11	Protection	UpdateSecureP...
12	Protection	UpdateSecureP...
13	Protection	Anti-virus
14	Backup	Recovery
15	Backup	Recovery

Figure C.7: **Countermeasures Display Window**

user could choose to tick or unpick corresponding boxes, and click "Set Relevant Countermeasures" button, and those information will be entered into IMD-SAT for future usage.

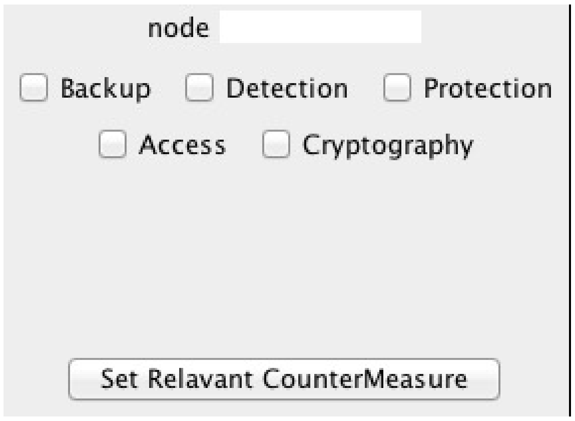


Figure C.8: **Relevant Countermeasures Edition Window**

According to our quantification, after user decides relevant countermeasures, then user could start to add/delete countermeasures to a specific attack leaf. User could accomplish this by right click any leaf nodes, it will pop out a menu. Since countermeasure is multi-level architecture, the menu shown here shares the same feature. Click one menu then this specific countermeasure will be added into that leaf, if user clicks that specific countermeasure again, then it will be removed from that node. Figure C.9 shows how that multi-level menu looks like.

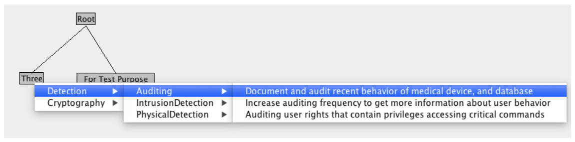
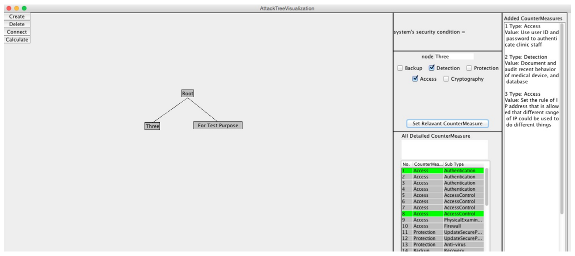

Root
Three
For Test Purpose
Detection
Cryptography
Auditing
IntrusionDetection
PhysicalDetection
Document and audit recent behavior of medical device, and database
Increase auditing frequency to get more information about user behavior
Auditing user rights that contain privileges accessing critical commands

Figure C.9: **Attach/Detach Countermeasures**

View Added Countermeasures

User could double click attack leaf node to see how many countermeasures have already been added and what are those countermeasures. There will be 2 noticeable changes here to inform user above information. Firstly, the countermeasures display window, all countermeasure entries that have been added will change background color from gray to green, so user could click those entries to examine more detailed messages. Second one, user could see in the rightmost part of interface, we have a window called "Added Countermeasures". Basically, this window could tell user what countermeasures have been added. Figure C.10 is one example of hot to check all added countermeasures.


AttackTreeVisualization
Create
Delete
Connect
Calculate
Root
Three
For Test Purpose
system's security condition =
node Three
☐ Backup ☑ Detection ☐ Protection
☑ Access ☐ Cryptography
Set Relavant CounterMeasure
All Detailed CounterMeasure

No.	CounterMea...	Sub Type
1	Access	Authentication
2	Access	Authentication
3	Access	Authentication
4	Access	Authentication
5	Access	AccessControl
6	Access	AccessControl
7	Access	AccessControl
8	Access	AccessControl
9	Access	PhysicalExamin...
10	Access	Firewall
11	Protection	UpdateSecureP...
12	Protection	UpdateSecureP...
13	Protection	Anti-virus
14	Backup	Recovery

Added CounterMeasures

1 Type: Access
Value: Use user ID and password to authenticate clinic staff

2 Type: Detection
Value: Document and audit recent behavior of medical device, and database

3 Type: Access
Value: Set the rule of IP address that is allowed that different range of IP could be used to do different things

Figure C.10: **View Implemented Countermeasures**

C.2.3 System Security Analysis

After all these operations are done, we are ready to quantify and assess systems security conditions.

System Security Calculation

Once we attach all suitable countermeasures to each attack leaf, we should be able to quantify system. User simply just clicks "Calculate" button on the interface, IMD-SAT will automatically calculate that score based on our quantification mechanism (Again this could be changed to the most suitable quantification for different systems). You will see that score in the window, with 0 standing for most insecure and 1 for perfect secure.

Also you should see the most vulnerable nodes of all attack nodes will be emphasized with a bold face font. So user could clearly see which one threats current system most, then user could take relevant measures to enhance it, so that system will become more and more robust.

Report Generation

Also after user gets all data available and clicks "Calculate" button, and an alert window will appear and inform user that a security report has been generated under the current directory. For the security report, it contains vulnerability indexes for all attack nodes. We will give descriptions of security report later.

C.3 Implementation

The architecture is shown in Figure C.11. We follow Model-View-Controller (MVC for short) pattern principle and apply it to IMD-SAT designs and implementations. MVC divides a given software application into three interconnected parts, so as to separate internal representations of information from the ways that information is presented to or accepted from the user.

MVC Pattern

The central component of MVC, the model directly manages the data, logic and rules of the application. A view can be any output representations of information, such as a chart or a diagram; multiple views of the same information are possible, such as a bar chart for management and a tabular view for accountants. The third part, the controller, accepts inputs and converts them to commands for the model or view.

In addition to dividing the application into three kinds of components, the MVC design defines the interactions between them:

- Controller: A controller can send commands to the model and update the model's state (e.g., editing a document). It can also send commands to its associated view to change the view's presentation of the model (e.g., by scrolling through a document). Thus a controller is the link between a user and the system. It provides the user with input by arranging for relevant views to present themselves in appropriate places on the screen. It provides means for user output by presenting the user with menus or other means of giving commands and data. The controller receives user output, translates it into the appropriate messages and passes these messages on to one or more of the views.

- Model: Models represent knowledge. A model stores data that is retrieved to the controller and displayed in the view. Whenever there is a change to the data it is updated by the controller. A model could be a single object (rather uninteresting), or it could be some structure of objects.

- View: A view requests information from the model that it uses to generate an output representation to the user. It is a (visual) representation of its model. It would ordinarily highlight certain attributes of the model and suppress others. It is thus acting as a presentation filter. It may also update the model by sending appropriate messages. All these questions and messages have to be in the terminology of the model, the view will therefore have to know the semantics of the attributes of the model it represents.

MVC Design

Figure C.11 shows MVS models for IMD-SAT. At the top of Figure C.11, we have "User Interface" box. The user interface consists of three major parts: attack tree editor view, countermeasure view and calculation view. The user interfaces are responsible for delivering the state information of the system. It might not necessary for user to understand what is happening behind the screen, however users should be able to observe the information and changes from these interfaces directly. In our case, we are trying to make our user interface convey as much information as possible in a clear way. Also the user interfaces are waiting and receiving users' interactions. These interactions are users' commands to change or update the system. All our three views keep waiting and listening for some events, like button click event, mouse click event and others events. And based on the received events, the system itself will react with the corresponding functions. Once the systems finished the executions, all information gets updated and user interfaces should capture the renewed data and reflect back to users.

In the middle of Figure C.11, we have four different controllers as long as their related logic functions. The process for controller is as follows: based on user request, one controller gets

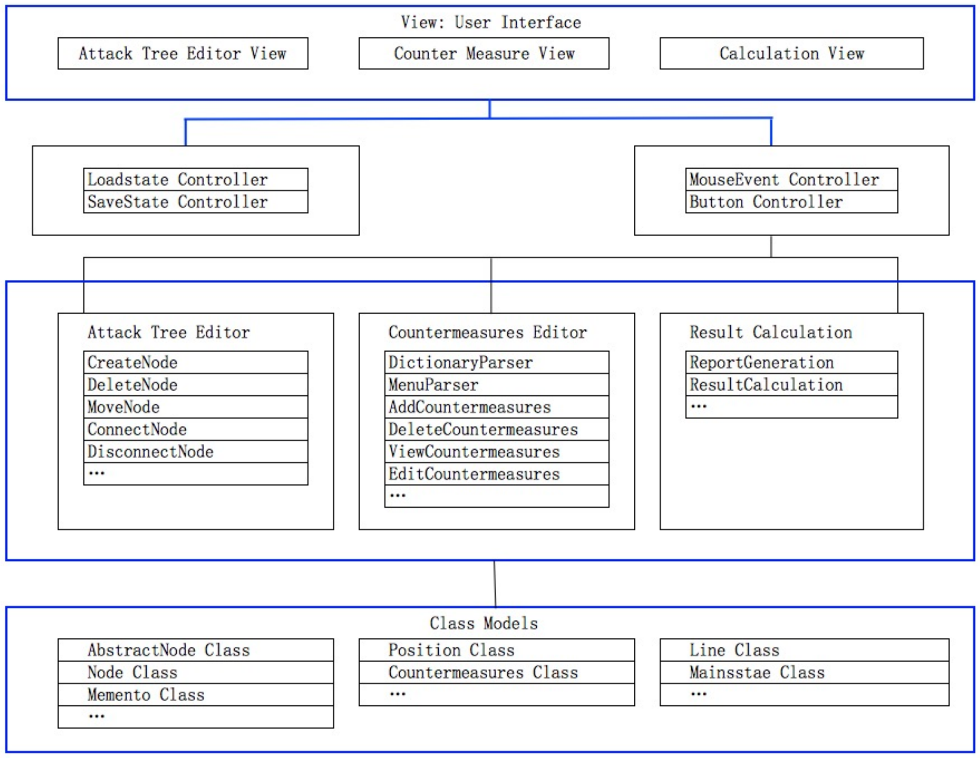


Figure C.11: **MVC Design of IMD-SAT**

activated, finds suitable functions to execute, update models with updated data and renew the user interface for users to understand the latest information. In our case, four controllers are “Loadstate Controller”, “Savestate Controller”, “MouseEvent Controller” and “Button Controller”. When the user first time uses this application, the system will rely on “Loadstate Controller” to load all user interfaces for users. “Loadstate Controller” will call “DictionaryParser” function to analyze state configuration file. The function finds state configuration empty and knows this user never use this software before, it will create a brand new state, initialize all models and show a clean user interface.

Eventually at the end of Figure C.11, we have the class models. Class models keep track of all information at the run time. In our case, we have plenty of class models, “Node Class”, “Position Class” and “Countermeasure Class” etc. During run time, if any state gets changed, a relevant message will send by the controller to update the related models. For instance, user moves one node, the current location of that node changes. System needs to replace the old position with the current one, so system could know the location for that particular node after date gets renewed. Meanwhile models could have dependency on other models if one contains the other. A node should know its own location as well as the countermeasures implemented therefore “Node Class” is the top-level class here. It contains “Position Class” and “Countermeasure Class”.

Appendix D

Countermeasure Type

D.1 Countermeasure Types

This section shows the details of all countermeasures for five general types and some other subtypes. For 6 general types, they are access, protection, backup, cryptography and detection.

Table D.1: Access

No.	SubType	Countermeasure
1	Authentication	Use user ID and password to authenticate clinic staff
2	Authentication	Use smart card to authenticate users
3	Authentication	Implement a biometric technology fingerprint to authenticate
4	Authentication	Implement personal challenges to authenticate users
5	Access Control	Use automatic timed methods to terminate sessions within the systems where appropriate for the use environment
6	Access Control	Eliminate hard-coded passwords in medical devices
7	Access Control	Eliminate guest accounts
8	Access Control	Set the rule of IP address that is allowed that different range of IP could be used to do different things
9	Physical Access	Physical examination of devices to make sure medical devices are not damaged or broken
10	Physical Access	Control physical access to areas around devices
11	Physical Access	Secure device and information physically include securing machines in locked rooms
12	Firewall	Enhance firewall policy to prevent outsider break in system

Table D.2: Protection

No.	SubType	Countermeasure
13	Update Security Patch	Implement patch management system to update security patches on time
14	Update Security Patch	Restrict software, hardware security patches to authenticated code
15	Anti-virus	Install antivirus software to detect/eliminate malwares

Table D.3: Backup

No.	**SubType**	**Countermeasure**
16	Recovery	Install a redundant system in case of urgent need to switch
17	Recovery	Implement a back-up policy for a particular period of time
18	Recovery	Database backup is physically secured (locked) and remains unreadable

Table D.4: Cryptography

No.	**SubType**	**Countermeasure**
19	Cryptography	Implement an encryption/decryption algorithm to protect data
20	Integrity	Implement integrity checker to protect data from being compromised

Table D.5: Detection

No.	**SubType**	**Countermeasure**
21	Auditing	Document/audit the behaviors of medical device and database
22	Auditing	Increase auditing frequency to get more data
23	Auditing	Auditing user rights that contain critical privileges
24	Intrusion Detection	Install intrusion detection system to monitor abnormal traffic
25	Physical Detection	Physically secure machines in locked rooms
26	Physical Detection	Control physical access to areas around devices

Appendix E

Countermeasure Association Table

E.1 Countermeasure Association Table

This section shows details concerning all countermeasures associated with different types of nodes. The different types are compromise medical devices, compromise PCs and EHR database, Exploit Wireless Connection Between Two components, delay/loose data and incorrect data.

Table E.1: Compromised Medical Devices

No.	**Type**	**Countermeasure**
1	Detection	Increase auditing frequency to get more information about user behavior
2	Detection	Auditing user rights that contain privileges accessing critical commands
3	Detection	Document and audit recent behavior of medical device, and database
4	Detection	Secure devices and information physically include securing machines in locked rooms
5	Access	Use smart card to authenticate users
6	Access	Control physical access to areas around devices
7	Access	Eliminate guest account
8	Access	Eliminate hard-coded passwords in medical devices
9	Access	Use user ID and password to authenticate clinic staff
10	Access	Physical examination of devices to make sure medical devices are not damaged or broken
11	Backup	Install a redundant system in case of urgent need to switch

Table E.2: Compromise PCs and EHR Data

No.	**Type**	**Countermeasure**
1	Detection	Increase auditing frequency to get more information about user behavior
2	Detection	Auditing user rights that contain privileges accessing critical commands
3	Detection	Document and audit recent behavior of medical device, and database
4	Detection	Secure devices and information physically include securing machines in locked rooms
5	Access	Use smart card to authenticate users
6	Access	Control physical access to areas around devices
7	Access	Eliminate guest accounts
8	Access	Use user ID and password to authenticate clinic staff
9	Access	Use automatic timed methods to terminate sessions within the system where appropriate for the use environment
10	Backup	Back-up schedule is regular and timely, implement a back-up policy for a particular period of time
11	Backup	Install a redundant system in case of urgent need to switch
12	Backup	Database backup is physically secured (locked) and use cryptography to make it unreadable

Table E.3: Exploit Wireless Connection Between Two components

No.	**Type**	**Countermeasure**
1	Access	Set the rule of IP address that is allowed that different range of IP could be used to do different things
2	Access	Enhance and strict firewall policy in Router and database to prevent outsider break into our system
3	Access	Use user ID and password to authenticate clinic staff
4	Protection	Implement patch management system to update security patches once available
5	Protection	Restrict software and hardware security patches update to authenticated code
6	Detection	Document and audit recent behavior of medical device and database
7	Detection	Install intrusion detection system to monitor abnormal traffic within network
8	Detection	Implement challenges/personal questions to authenticate users
9	Cryptography	Implement a encryption/decryption algorithm to protect data transmitted in the whole system
10	Cryptography	Install integrity checkers to monitor any alternations done to database system

Table E.4: Data Gets Delay or Loose

No.	Type	Countermeasure
1	Detection	Increase auditing frequency to get more information about user behavior
2	Detection	Auditing user rights that contain privileges accessing critical commands
3	Detection	Secure devices and information physically include securing machines in locked rooms
4	Detection	Document/audit recent behavior of medical device and database
5	Detection	Control physical access to areas around devices
6	Access	Use smart card to authenticate users
7	Access	Eliminate guest account
8	Access	Eliminate hard-coded passwords in medical devices
9	Access	Use user ID and password to authenticate clinic staff
10	Access	Physical examination of devices to make sure medical devices are not damaged or broken
11	Backup	Install a redundant system in case of urgent need to switch

Table E.5: Data Gets Manipulated

No.	Type	Countermeasure
1	Access	Set the rule of IP address that is allowed that different range of IP could be used to do different things
2	Access	Eliminate guest account
3	Access	Implement challenges/personal questions to authenticate users
4	Access	Eliminate hard-coded passwords in medical devices
5	Access	Use automatic timed methods to terminate sessions within the system where appropriate for the use environment
6	Access	Use user ID and password to authenticate clinic staff
7	Protection	Implement patch management system to update security patches once available
8	Protection	Restrict software and hardware security patches update to authenticated code
9	Detection	Document/audit recent behavior of medical device and database
10	Dtection	Install intrusion detection system to monitor abnormal traffic within network
11	Cryptography	Implement an encryption/decryption algorithm to protect data transmitted in the whole system
12	Cryptography	Install integrity checkers to monitor any alternations done to database system