A Methodology for Systematic Attack Trees Generation for Interoperable Medical Devices

J. Xu, K. K. Venkatasubramanian
Dept. of Computer Science
Worcester Polytechnic Institute
Worcester, MA, 01609
Email: {jxu3,kven}@wpi.edu

Vasiliki Sfyrla
Unaffiliated
Grenoble, France
Email: vasiliki.sfyrla@gmail.com

***Abstract*—Security for medical devices has gained some traction 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 often requires considerable expertise in identifying the various attack vectors. IMDs are typically deployed in hospitals by clinical engineers who may not possess 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. 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.

I. Introduction

Recent years have seen rapid growth in *interoperable medical devices* (IMDs) [1]. 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 [1]. 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 useful way to perform threat 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 [10]. 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 as lower nodes in the tree. Each path in the attack tree from the leaf node all the way to the root node describes a unique attack on the system [7]. The advantage of attack trees is that it provides a graphical view of the threats of the system that can be easily understood and acted upon. However, generating these attack trees requires considerable expertise in threat analysis. IMDs are typically deployed in hospitals by *clinical engineers* who may not posses such expertise. Therefore, it is imperative to develop solutions that can generate attack trees for IMDs in a systematic manner.

Methods for systematically generating attack trees has been studied before, mostly in the context of network security [9], [11]. The typical approach has been to develop a model of the system in question using formal methods tools (e.g, SPIN) and generate attack trees from the resultant specification. These approaches were designed for security experts who manage large-scale enterprise networks and requires 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 of expertise (i.e., formal methods and static analysis). As IMDs are deployed on-demand and managed by clinical engineers in a care facility, who are more focused on practicalities of operating medical systems rather than mathematical modeling, existing approaches for attack tree generation may not be viable. We therefore need a methodology that is easy for clinical engineers to understand and use.

In this paper, 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 to describe our IMD system as it is deployed, initialized, and operated by the caregiver. Hazards can be thought of as system states that are inherent 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 representations of the workflow that formal methods based approaches require. Once the process model is built and hazards identified, our

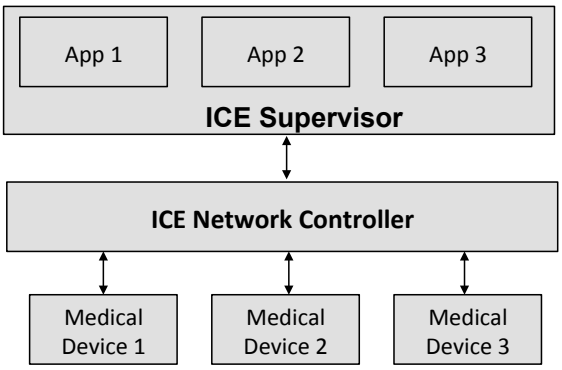

Fig. 1: **ICE Architecture**

methodology generates a number of fault-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. Attack trees are similar to fault-trees except they connect one or more attacks (instead of faults) toward an *attack goal* (e.g., over-infusion of medication). Each fault-tree generated in the last step indicates a path to one specific hazard. Finally, we combine the hazards, and the associated fault-tree, that lead to patient harm in the exact same way (e.g., over infusion) under a common root node (i.e., attack goal). The fault tree node descriptions are modified from faults caused within the system to attacks that are deliberately induced by an adversary to create the final attack tree.

Note that the proposed approach for generating attack trees is designed to complement mainstream security analysis such as risk management approach. Traditional risk management is defined as a four step approach: (1) defining the context of risk, (2) assessing the extent of risk present, (3) responding to the risk and (4) finally monitoring the vulnerable elements of the system for future risk [5]. Attack trees provide a means to assess the risk present in a workflow/system and help in determining which elements of the system are at risk and need to be addressed. The leaf nodes particularly describe the specific points within the system that need to be secured to ensure overall system security. The leaf nodes also describe the threats and therefore automatically suggest countermeasures as a way of responding to the risk identified. Finally, the attack trees help in determining which elements of the system to monitor as a way of mitigating future risk.

We make the following **contributions** in this paper. (1) We develop a methodology for generating attack trees in the context of interoperable medical devices. (2) We demonstrate 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. Even though we focus on the PCA-IMD setup, the proposed methodology and tool can be used for any IMD system whose security we wish to analyze using attack trees

The rest of the paper is organized as follows. Section II presents the background. Section III presents the PCA-IMD system model, and our adversary model. Section IV presents the methodology with examples focusing on the PCA-IMD setup. Section V presents the related work and Section VI concludes the paper.

II. Background

Before we delve into the details of the attack tree generation, it is useful to present a short overview of interoperable medical devices and their basic architecture. The predominant standard in enabling medical device interoperability is the Integrated Clinical Environment (ICE) standard, which was created to enable diverse medical devices to talk to one another [4]. 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 patient's health. The architecture of ICE consists of three entities (see Figure 1):

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

III. System Model

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 (PCA-IMD) (see Figure 2). 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.

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 2). 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 nurses programming of the infusion pump based on medication orders in the EHR.

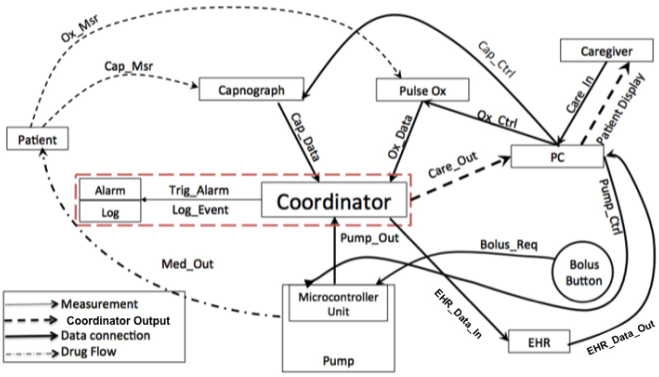

Fig. 2: **PCA-IMD Setup**

PCA-IMD is configured for each patient according to their 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 2). 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.

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 2 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 [6].

For our work, we consider *active adversaries* who may interfere with communication links, as per the Yao-Dolev model of an adversary [3]. In addition, the adversary may also physically alter the infusion pump, the coordinator, the pulse oximeter, and capnograph, and their individual settings, respectively. 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.

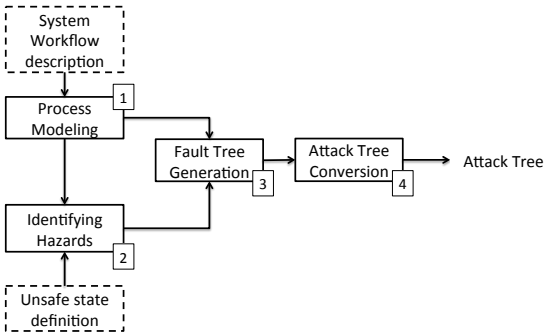

Fig. 3: **Attack Tree Generation Methodology**

IV. Attack Trees Generation Methodology

An overview of the methodology for attack tree generation is shown in Figure 3. It has four elements. First, the user (i.e., clinical engineer) 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 state of interest w.r.t. hazard 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 IMD setup such that the root node of the tree is the attack goal of over-infusion of medication. The rest of the section describes these steps in more detail.

A. 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 [2]. 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 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 their capabilities in 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.

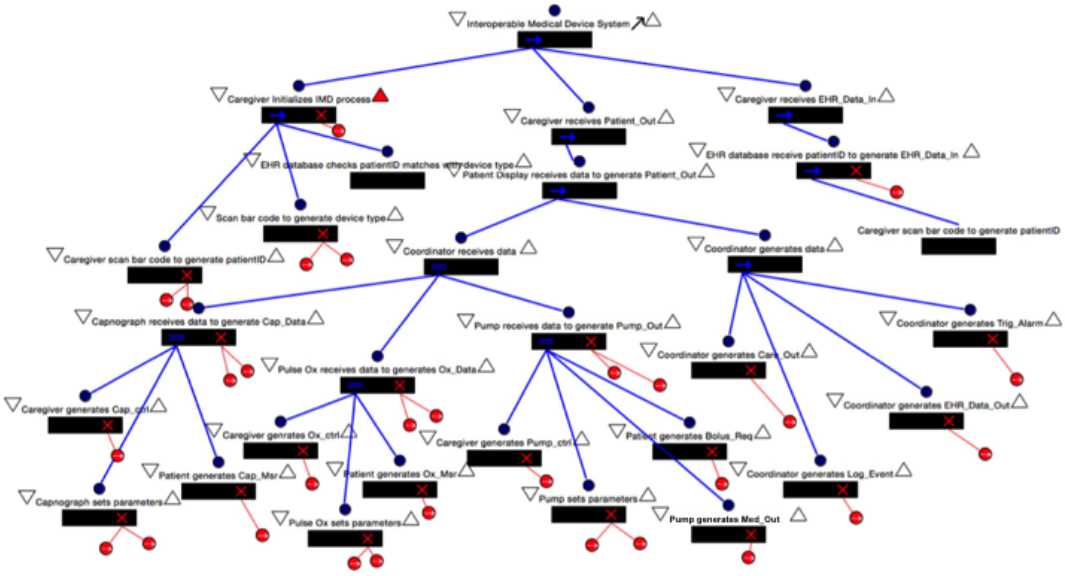

Fig. 4: **Little-JIL-based Process Modeling of PCA-IMD Workflow**

In describing the operational workflow of the PCA-IMD using process modeling we begin 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, 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 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 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 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. Finally, for each of these steps, exception handlers are defined (as indicated by the "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.

B. 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 or in our case harm the patient. 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 I 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 [2].

TABLE I: 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 [2]. 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

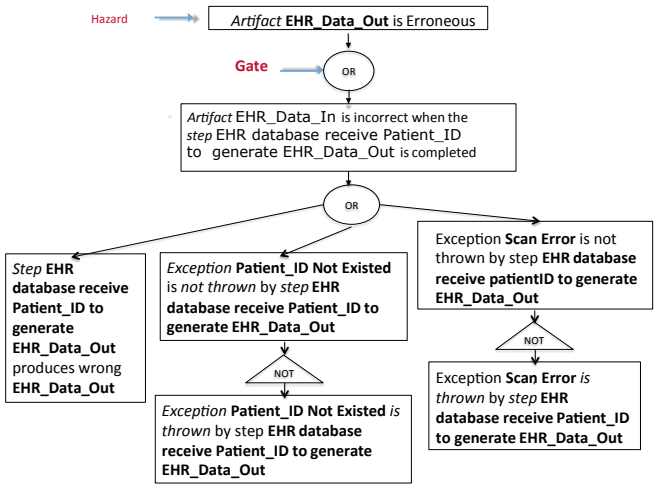

Fig. 5: **Example partial fault-tree for a specific hazard (EHR data incorrect)**

to be developed further. The derivation procedure terminates when no intermediate events exist in the fault-tree that can be described further [2].

Consider the following example of the aforementioned fault-tree extraction process. One class of 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_Out 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 5 shows the fault-tree generated for the hazard *EHR_Data_Out is Erroneous*. Here, the *EHR_Data_Out* 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_Out* is incorrect. The error is caused when the EHR receives the *Patient_ID* it produces the wrong *EHR_Data_Out* value associated with the patient as indicated by the OR-gate underneath root node. The reason for producing the wrong *EHR_Data_Out* given *Patient_ID* can be one of three conditions as shown in the partial fault-tree in Figure 5: (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_Out* 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 to the discussion simple.

C. Attack-Tree Conversion

The availability of fault-trees allow us to convert it to an attack tree, which can be thought of as 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_Out* is related to *Ox_Data*, therefore if *Ox_Data* is modified, the value of *EHR_Data_Out* 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 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.

Figure 6 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 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).

V. Related Work

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) [13], cyber-physical systems [15], and interoperable medical devices [12]. However, none of these approaches provide any systematic means of generating the attack trees. Their focus

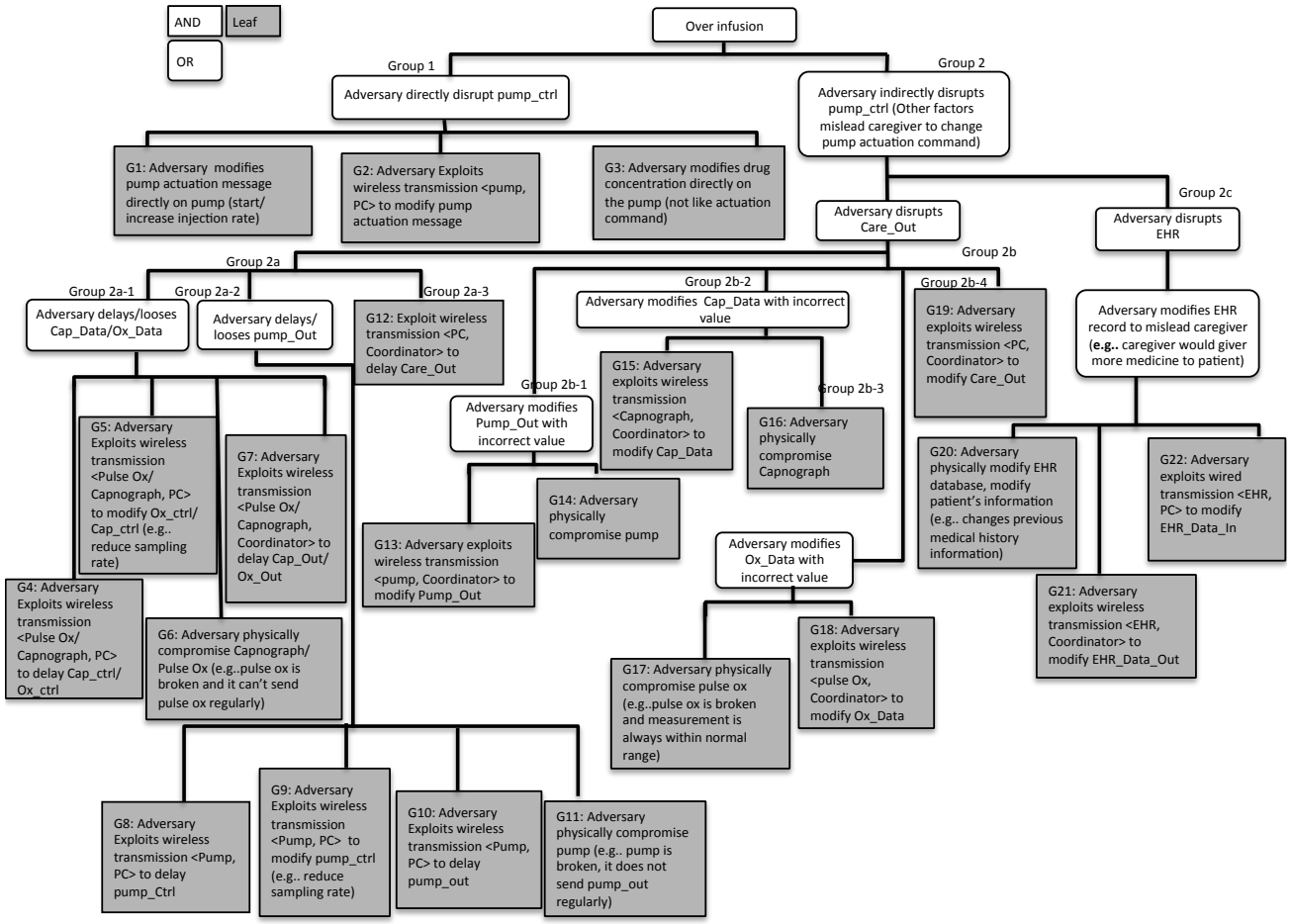

Fig. 6: **Final Attack Tree**

rather is on using the generated attack trees for detailed analysis of the system. Methods for systematically generating attack trees based on system description have been studied before. In [11] the authors propose a methodology that uses formal methods to model a network and generate attack trees for it. In [8], [9] the authors analyze the security of a network through the use of model checking. The approaches are very similar to generating attack trees as a way to verify the security of a system against threats. Finally, in [14], the authors propose a static analysis based approach where attack trees are generation from process algebraic specification of a specific process. All these approaches require considerable knowledge of formal mathematical tools and methods and model checking which IMD users, who are typically clinical engineers interested in the practical operation of medical systems, may not possess. Thus applying these approaches in the IMD context may not be effective and we need a different approach for attack tree generation for IMDs.

VI. Conclusions

In this paper we developed a methodology for generating attack trees based on IMD workflow modeling and hazard analysis. We also illustrated how the methodology can be used in the interoperability context by applying it to generate attack trees for interoperable medical device setup for patient controlled analgesia (PCA-IMD) with respect to preventing over-infusion. In the future we plan to extend our methodology by adding quantification to the attack trees that computes the security condition of the IMDs. Such an analysis would allow us to compare multiple instances of an IMD setup to determine which one is more secure.

References

[1] D. Arney, S. Fischmeister, J. M. Goldman, I. Lee, and R. Trausmuth. Plug-and-play for medical devices: Experiences from a case study. *Biomedical Instrumentation & Technology*, 43(4):313–317, 2009.

[2] B. Chen, L. A. Adviser-Clarke, and G. S. Adviser-Avrunin. *Improving processes using static analysis techniques*. University of Massachusetts Amherst, 2011.

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

[4] J. 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.

[5] Joint Task Force Transformation Initative. *Managing Information Security Risk Organization, Mission, and Information System View*. NIST, 2011.

[6] B. Kim, A. Ayoub, O. Sokolsky, I. Lee, P. Jones, Y. Zhang, and R. 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.

[7] A. P. Moore, R. J. Ellison, and R. C. Linger. Attack modeling for information security and survivability, 2001.

[8] J. Powell and D. Gilliam. Model checking for network security requirements via a flexible modeling framework. In *5th IEEE International Symposium on Requirements Engineering*, 2001.

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

[10] B. Schneier. *Secrets and Lies: Digital Security in a Networked World*. Wiley, Jan. 2004.

[11] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. Wing. Automated generation and analysis of attack graphs. In *Security and Privacy, 2002. Proceedings. 2002 IEEE Symposium on*, pages 273–284, 2002.

[12] C. R. Taylor, K. Venkatasubramanian, and C. 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.

[13] C.-W. Ten, C.-C. Liu, and M. Govindarasu. Vulnerability assessment of cybersecurity for scada systems using attack trees. In *Power Engineering Society General Meeting, 2007. IEEE*, pages 1–8. IEEE, 2007.

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

[15] F. Xie, T. Lu, X. Guo, J. Liu, Y. Peng, and Y. 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.