Protecting interoperable clinical environment with authentication

Liang Cheng
University of Pennsylvania/Institute of Software, Chinese Academy of Sciences
liacheng@cis.upenn.edu

Zhangtan Li
Institute of Software, Chinese Academy of Sciences
lizhangtan@tca.iscas.ac.cn

Yi Zhang
Center for Devices and Radiological Health, US Food and Drug Administration
Yi.Zhang2@fda.hhs.gov

Yang Zhang
Institute of Software, Chinese Academy of Sciences
zhangyang@tca.iscas.ac.cn

Insup Lee
University of Pennsylvania
lee@cis.upenn.edu

ABSTRACT

The Integrated Clinical Environment (ICE) is a standard dedicated to promote open coordination of heterogeneous medical devices in a plug-and-play manner. This carries the potential to radically improve medical care through coordinating, cooperating devices, but also to undermine the patient safety by giving rise to security vulnerabilities in the cyber world. In this paper, we propose an authentication framework as the first step to build an ICE security architecture. This framework is designed in a three-layered structure, allowing it to fit in the variety of authentication requirements from different ICE entities and of networking middleware from ICE instantiations. We implement the authentication framework on OpenICE, an open source ICE instantiation. Our experiments shows that the framework can help OpenICE mitigate the vulnerabilities caused by forged identity with negligible performance overload.

Keywords

Integrated Clinical Environment; Authentication; Medical cyber physical system

1. INTRODUCTION

Emerging interoperable medical systems indicate a promising future for the healthcare domain: coordinating medical devices from different vendors together to accomplish a clinical mission. Many case studies [1, 3, 11, 12, 13, 19] have shown that enabling the interoperability of medical systems can reduce medical errors and improve the productivity of medical care, as compared to the traditional practices that rely upon disconnected, standalone devices. An exemplar effort of promoting medical device interoperability is the ASTM F2761 standard [4], which defines a model-based system architecture to capture the general design principles of integrating cross-manufacturer medical devices to create an Integrated Clinical Environment (ICE). The ICE architecture defined in the ASTM F2761 standard, as illustrated in Figure 1, consists of two major components: the supervisor and the network controller. The supervisor hosts ICE applications that interact with medical devices and communicates with external information systems such as an electronic health record (EHR) system. The network controller facilitates communications between the supervisor and the medical devices. Whenever a device connects to an ICE system, the network controller discovers the device and sets up the communication channel for it. Since its publication, the ASTM F2761 standard has been adopted by many stakeholders, such as Massachusetts General Hospital, Draeger Medical Systems and Kansas State University, to develop interoperable medical systems compliant to the ICE architecture. We refer to such systems as ICE systems in the rest of the paper.

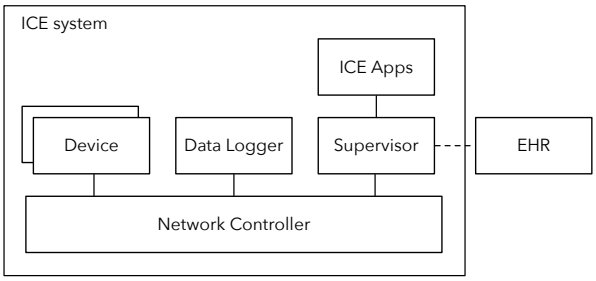


Figure 1: ICE architecture, where ICE Apps refer to software applications running on the ICE system, and EHR is short for an Electronic Health Record system.

Most industrial and academic effort thus far focuses on prototyping various instantiations of ICE systems, while little attention has been given to the cyber security properties of these systems. This does not mean that security is not a priority for ICE systems. Instead, poor security can cause ICE entities (devices, applications, and other equipments) to fail in coordinating with each other and fulfilling their shared clinical missions as expected. For example, ICE systems with security breaches may cause the connected medical devices to act upon information crafted by an adversary and pose unacceptable risk to patient safety. Furthermore, when healthcare organizations have doubt on the security of ICE systems, the adoption of such systems in realistic clinical practices can be hin-

dered. It is thus critical to equip ICE systems with effective and fairly comprehensive security mechanisms.

Unfortunately, the unique engineering characteristics of ICE systems pose several challenges in designing and implementing such a security mechanism:

- The ICE architecture integrates devices from different manufacturers through a collection of device interfaces. While these device interfaces endow manufacturers the flexibility of using their own methods and techniques to develop devices with similar functionalities, they also cover the engineering details of the devices from ICE system developers. This makes it difficult for developers to conduct a comprehensive security analysis over the entire ICE systems, down to the device level. Thus, ICE system developers have to restrict the security analysis to the architectural level, focusing on security properties like authentication of ICE components or confidentiality of the information communicated among ICE components. Meanwhile, they need to make ICE systems secure enough to shield the security exploitation in connected devices from the rest of the system.
- The dynamic composition of an ICE system makes it subject to different security threats at different stages of the clinical processes. For example, adding a patient physiology monitor to an ICE system in the middle of a procedure may cause the system to be vulnerable to man-in-the-middle attacks, which fool the system with adulterated patient monitoring data. In the mean time, an ICE system may be deployed in different clinical scenarios throughout its lifecycle. Therefore, the security mechanism in such a system should be adaptive, in order to align with the changing environment and rebalance security with the evolving clinical needs. To address these issues, an effective security mechanism for ICE systems should not only offer essential security functions, but also allow users to easily re-configure the mechanism to satisfy the changing security needs.

Previous research (e.g., [9, 14, 8, 21]) has laid down some high-level security requirements for ICE systems, but no security mechanism has yet been proposed for realistic ICE instantiations. This is partly because there is lack of consensus among stakeholders on what essential security features such a mechanism should realize and how. We argue that one practical way to reconcile this disagreement among stakeholders is to establish a baseline security mechanism on top of a realistic, exemplar ICE instantiation, so that the community can discuss the security of ICE systems over a concrete basis.

We have seen several efforts towards this direction. For example, Salazar et al. [17] implemented a modularized prototype for ICE device authentication and communication encryption on top of the Medical Device Coordination Framework (MDCF) [1], an open-source development/simulation environment of ICE systems. They also designed a preliminary access control model with a break-the-glass feature in the MDCF framework [18].

These efforts provide evidence for the feasibility of retrofitting an authentication mechanism in ICE systems. However, there are still gaps that need to be bridged before an effective security mechanism can be developed for realistic ICE instantiations. First of all, it needs to consider all aspects of ICE systems. For instance, authentication should cover all devices, equipments, applications, and even human users that may involve in ICE systems.

Secondly, ICE security mechanisms should be designed as independent of the network middleware utilized by the host ICE systems. Network middleware is commonly used in ICE systems to provide communication between ICE components. Existing ICE instantiations adopt different network middleware, such as DDS-based OpenICE [2] and Web Service-based OpenSDC[2]. Thus, a security mechanism specially designed for a certain network middleware is not applicable in ICE systems where different middleware is utilized, while a security mechanism independent of any specific network middleware can be migrated across different ICE instantiations with little re-configuration effort.

In this paper, we propose a middleware-independent authentication scheme for ICE systems, as the first step towards establishing the envisioned baseline ICE security mechanism. Due to the complexity of ICE systems, a practical way of designing an ICE security mechanism is to design it using a module-based development process. That is, security measures for mitigating different security threats are categorized and grouped into different functional modules, which are designed and validated separately. We choose authentication as the first module of our envisioned ICE security mechanism to implement, because authentication provides the fundamental security functionality for other security measures. For example, access control relies on authentication to confirm the identity of an entity before allowing it to access, within its privileges, resources in ICE systems.

The key feature of our authentication module is its three-layer architecture, which logically separates authentication interfaces, authentication workflow, and authentication protocol implementation in different levels. This enables ICE system developers to easily re-customize the module to fit in use with different network middleware. Then we implemented our authentication module on OpenICE, an open source ICE instantiation. This implementation not only realizes ICE device authentication, but also covers authentication of human users and ICE applications. Our experiments demonstrated that the authentication module could avoid device replacement and impersonation attacks for OpenICE without causing significant performance sacrifice.

2. GOALS

A primary goal of ICE systems, as for other medical systems (standalone or interoperable), is to protect patient safety. That is, ICE systems should protect patient safety from being harmed by unexpected or undesirable accidents, including security breaches. This requires ICE systems to adequately mitigate their security risks and establish acceptable security.

Security risks in an ICE system can be categorized as: 1) security vulnerabilities in individual entities in the ICE system, which allow an adversary to steal valuable information from these entities, cause them to behave unexpectedly, or disrupt their normal operation; and 2) security vulnerabilities pertain to the interaction among entities in the system. Note that the information (data and commands) communicated across an ICE system is often privacy-sensitive or critical to the system's mission. Vulnerabilities associated with the interaction of ICE entities can subject such critical information to malicious attacks, or cause the system to operate unexpectedly. For example, an adversary can compromise the infusion settings communicated from the ICE supervisor to the infusion pump connected to the system, posing over-infusion risk to the patient who receives medication from the pump.

It has been widely studied on how to mitigate security vulnerabilities in single medical devices or medical software applications. In this paper, we target at the second type of security risks in ICE systems and, as the first step, present a universal solution to entity

[1]Available at http://mdcf.santos.cis.ksu.edu/

[2]Available at http://opensdc.sourceforge.net.

authentication for ICE systems.

We choose to start with authentication for two reasons. Firstly, a significant portion of security risks to ICE systems can be mitigated if strong authentication is in place to allow only legitimate, authorized entities to operate in these systems. A common starting step to attack ICE systems is for an adversary to gain unauthorized access to these systems, which can cause the following security risks:

1. Spoofing: gain access to the ICE system by using a false identity. After successfully acquiring the access as a legitimate user or host, the entity can elevate or abuse its privileges to compromise the system.
2. Tampering: modify data beyond its authorization by using a counterfeit identity of a privileged entity.
3. Repudiation: deny its actions or transactions even if its behavior is audited.
4. Information Disclosure: disclose private data beyond its authorization by using a counterfeit identity of a privileged entity.
5. Elevation of privilege: gain unauthorized access to privacy-sensitive data or applications.

Authentication serves to confirm the identity of any entity requesting for access to ICE systems by verifying the validity of its credential. Thus, enforcing strong authentication in ICE systems helps to substantially mitigate the above threats by making it more difficult, sometimes impractical, for an adversary to gain such unauthorized access to these systems.

One may argue that authentication is not absolutely necessary for ICE systems (at least for devices and applications used in them), because healthcare organizations who run these ICE systems can ensure that only trustworthy devices and applications operate in these systems. However, this argument is invalid, because it bases on an invalid assumption that ICE system developers always have understanding of the full security profile of the devices and applications they trust, and fails to consider the situation where trustworthy devices and applications can be hacked or tampered (e.g. via online software update) during their use and can no longer be trusted.

Secondly, authentication constitutes the foundation of other security mechanisms needed by ICE systems. For example, access control, auditing and encryption all depend on a unique, unforgeable identifier being assigned to any entity throughout its presence in the ICE system, which is accomplished through authentication. For security threats like Tampering and Information Disclosure that require a collection of security measures to mitigate, authentication can also be used as the trust base for these measures.

Establishing an effective and efficient authentication module for ICE systems is more than just providing basic authentication functions. It should also address the challenges arising from real-world clinical practices in which the ICE systems are to be deployed. The first challenge is to accommodate the complicated authentication requirements of different types of ICE entities. According to the ASTM F2761 standard, the operation of a typical ICE system involves three types of entities:

1. Human users, including operators of the ICE system or a patient undertaking a clinical procedure provided by the ICE system;
2. ICE Application, which is a piece of software running on the ICE supervisor that interacts with human users and ICE-compatible equipments to accomplish the shared clinical mission. An example ICE application is the safety interlock app described in the PCA pump safety interlock scenario (see section 5.1.1), which coordinates the infusion pump and pulse oximeter devices to reduce the likelihood of over-/under-dose incidents during medical care; and
3. ICE-compatible equipment, which is a medical device or electrical equipment with an ICE equipment interface to interact with other parts of the ICE system.

Different types of ICE entities pose different requirements for their authentication. For example, authentication of human users typically occurs in the ICE supervisor before human users log into the system and invoke any ICE application. This is different than authentication of ICE equipment, which typically occurs at a different architectural layer (e.g., in the ICE network or ICE platform) before the equipment is allowed to connect to the ICE system. The authentication module should acknowledge such difference among different ICE entities, and provide a universal solution for validating their identities and managing their credentials.

The second challenge lies in the diversity of network middleware components that may be deployed in arbitrary ICE systems. Different ICE systems might use different network middleware components to facilitate the communication among their entities. For example, the subject ICE platform of our research, OpenICE, utilizes the Data Distribution Service (DDS) middleware to facilitate the network communication among entities running on the platform. In comparison, OpenSDC, a communication library that facilitates the data exchange between ICE devices and applications, is built on Web Service. In order to achieve maximum applicability in more than one class of ICE systems, an authentication module should be designed in a middleware-independent manner and be compatible with the specific network middleware deployed in the host ICE systems. Our authentication module is designed to be independent of network middleware and can be easily re-configured to work with different network middleware components in arbitrary ICE instantiations, even though it is currently configured to work with the DDS middleware for the demonstration purpose.

Moreover, legacy medical devices, which typically do not support any authentication protocols, are widely used in clinical practices. To allow these devices to be used in ICE systems, the authentication module should provide a mechanism to accommodate these devices, so that the adoption and usability of its host ICE system are not likely impacted.

Lastly, the authentication module should not impose unacceptable computational overhead on its host ICE systems. Otherwise, the function, performance, and quality of service of the host ICE systems will be significantly affected, which may not only hinder the adoption and use of these ICE systems but also, in worst cases, pose risks to patient safety. For example, computationally expensive authentication may delay a physiological monitoring device to be used in the ICE system, which in turn may lead to delay of diagnosis and treatment to the patient.

We elaborate the following requirements to clearly define the functionality, usability, and performance expectations for our authentication module and, more importantly, to address the challenges aforementioned.

- It should provide a set of middleware-independent authentication mechanisms to establish the trust between the ICE system and all legitimate entities requesting to enter into the system [Functionality].
  - ⋆ It should assign every authenticated entity with a unique ICE identifier, which should be the only identifier used by other security modules to identify the entity.
  - ⋆ It should revoke the ICE identifier of an entity when the entity is exiting, disconnected or uninstalled.

- ⋆ It should ensure secure storage and transfer of ICE identifiers assigned to all entities.

- It should audit all security-related events during the authentication process [Functionality].
- Its authentication interface should support mainstream authentication protocols and different identity formats mandated by these protocols [Usability].
- It should be able to deal with the connection requests from stale devices that might not support any authentication protocol [Usability].
- It should not compromise the function and performance of other services running on the ICE platform [Performance].

3. DESIGN OF THE ICE AUTHENTICATION MODULE

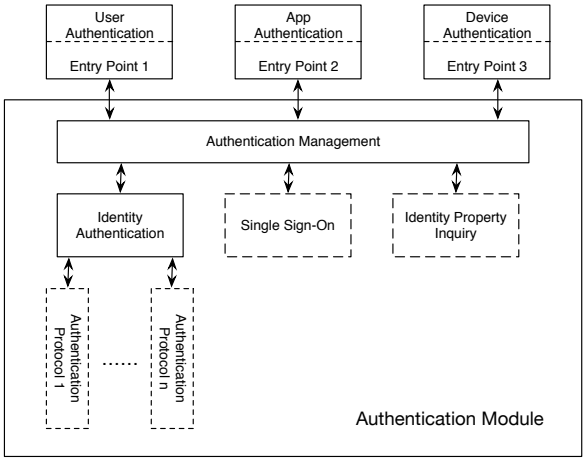


Figure 2: Architecture of the ICE authentication module. Solid boxes represent mandatory components and dotted boxes denote optional functionalities.

Our authentication module consists of three layers, as illustrated in Figure 2. Entry points, residing on the top level in Figure 2, intercept the connection requests from all ICE entities and initiate the corresponding authentication processes. Note that a specific ICE system may customize its protocol of establishing connection with its entities. Thus, we design our authentication module as the location of entry points can also be customized, in order to be compatible with the entity connection protocol of its host ICE system.

The middle (and core) level of Figure 2 encapsulates all functionalities related to authentication, and includes a management component to schedule these functionalities and collect relevant information for them. This level is usually deployed in the ICE supervisor, because the supervisor takes most responsibility of managing ICE entities.

The bottom level of our authentication module is an assembly of authentication protocol implementations utilized in different ICE systems. These authentication protocol implementations can be provided by the network middleware or the ICE network controller of the host ICE system, and are invoked by the core level using a series of predefined interfaces. Notably, the middle and bottom levels of our authentication module separate the specific implementation of authentication protocols utilized by the host ICE system and the interface to invoke it. This enables our authentication module to operate in its host ICE system, irrespective of which network middleware is used by the host ICE system.

Since the top and bottom levels of our module are essentially a collection of interfaces that are not to be configured based on the specific host ICE system, we focus on the design of the core level in the rest of this section.

When an entry point requests an authentication-related task, the Authentication Management (AM) component in the core level classifies the request and forwards it to the corresponding functionality component(s) in the module. In addition, the AM component is also responsible for initializing the authentication module upon the startup of the host ICE system. The AM component provides the following four functions, among which `AM_AUTH()` and `AM_REVOK()` can be invoked by entry points at the top level to start authentication:

1. `AM_AUTH()` for launching the authentication process and invoking the Identity Authentication (IA) component to complete authentication. It has three parameters: the request's type, the entity's credential, and the session ID. The request's type specifies which operation the entity wants to invoke. The entity's credential, such as the certificate or token of the entity, is the evidence to prove that the entity is who it claims to be. The session ID is used to identify the session of its request. Depending on the authentication result, the function also updates the list of authenticated entities, and returns the entity's ICE identifier or an error code indicating that the entity is refused for access.
2. `AM_NEGO()` for establishing an agreement between the ICE supervisor and the requesting entity on which authentication protocol the authentication process should follow. The function has only one parameter, the name of the authentication protocol to be used. Invoking this function causes the IA component to check whether or not the authentication module supports the indicated authentication protocol. If yes, the authentication process proceeds and `IA_INTF()` is then invoked; otherwise an error code is returned and the authentication process is aborted.
3. `AM_AUDIT()` for logging the information related to the authentication process for future auditing. It has four parameters: entity name, request type, authentication protocol, and authentication status. The function assembles these parameters into a record structure and sends it to the ICE's logging module. It returns a status code indicating if the logging succeeds or not.
4. `AM_REVOK()` for revoking the ICE identifier possessed by an ICE entity. It has one parameter: entity name. This interface destroys the data structure related to the ICE identifier, adds it to a revocation list, and informs other security modules (such as the access control module) of the revocation.

The three components below the AM component in Figure 2 are introduced to respectively realize each of the three primary authentication functionalities: the IA component for verifying the identity of an entity; the Single Sign-On (SSO) component for managing the single sign-on of an entity; and the Identity Property Inquiry (IPI) component for querying the property of an entity when explicitly required.

The IA component is a mandatory component, which conducts the authentication process by invoking standard authentication protocol implementations (libraries) and returns the authentication result. These libraries need to be registered with the IA component beforehand. The IA component provides two interfaces: the `IA_INTF()` interface for interacting with the AM component; and the `IA_IMPL()` interface for invoking a specific authentication protocol library. More specifically, `IA_INTF()` has two

parameters: the name of the authentication protocol and the entity's credential. It checks if the specified authentication protocol implementation has been registered with the IA component and loads it. If yes, the corresponding protocol is executed by invoking `IA_IMPL()`, the return value of which is forwarded to the AM component; otherwise, the authentication process fails and an error code is returned.

The `IA_IMPL()` interface, on the other hand, takes only the entity's credential as input. It transforms the entity's credential to a format acceptable to the specified authentication protocol, and then invokes the protocol to finish the authentication process. The authentication result, including the information generated from the entity's credential and the public/private keys (if necessary), is returned to `IA_INTF()`.

3.1 Authentication workflow

Figure 3 describes the workflow of the components at the core level coordinating to finish an authentication process. The workflow starts with invoking the `AM_AUTH` interface. Upon receiving the request for identity validation, the AM component first negotiates with the entity (`AM_NEGO()`) on which authentication protocol to use. If the negotiation succeeds, the IA component (`IA_INTF()`) is invoked with the protocol name and the request as parameters. The IA component conducts the authentication process (`IA_IMPL()`) by invoking standard authentication protocol implementations (libraries) and returns the authentication result. Upon receiving the result, the AM generates the entity's ICE identifier by attaching the result with necessary system-wide information, such as timestamp, session id and the entry point's IP address, etc. The ICE identifier will be the only identifier for the entity during its presence in the ICE system. When the entity exits the system (e.g. a device is disconnected, a user logs off, and an application terminates its operation), its ICE identifier is revoked (`AM_REVOK()`). All these activities are recorded and sent to the Logging system (`AM_AUDIT()`).

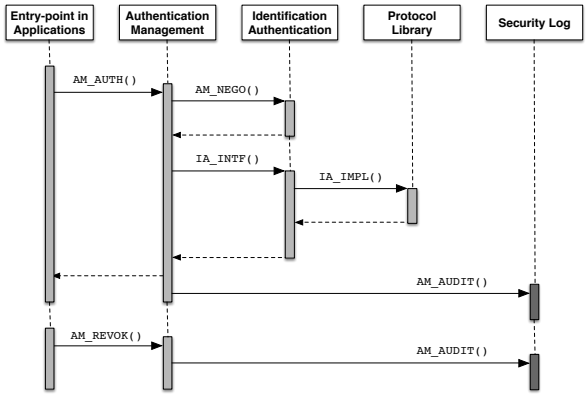


Figure 3: A typical workflow of the authentication module.

3.2 Mitigation to existing security threats

An authentication module fulfilling the expectations listed in section 2 makes the host ICE system more secure against the security threats described in Section 2: the entry point can be placed anywhere before the communication channel between the host ICE system and an ICE entity is established, in order to invoke `AM_AUTH` to start the authentication process. If the identity claimed by an entity doest not match with its credential, the authentication fails, leading to the termination of communication establishment and consequently the rejection of entity's request to access the ICE system. In this way, the threat of *Spoofing* is mitigated. If the authentication succeeds, `AM_AUTH` returns any entity connecting to host ICE system with a unique ICE identifier that indicates the genuine identity of the entity (as evidenced by the entity's credential). With privileges granted to this ICE identifier by the access control module, the attacks of *Tampering*, *Information Disclosure* and *Elevation of privilege* will be much harder to achieve, because all these attacks depend on the success of gaining a privileged ICE identifier. In the meantime, `AM_AUDIT` can be placed where `AM_AUTH` is invoked or the entity's authentication status changes (e.g. its ICE identifier expires after long time idling). `AM_AUDIT` ensures that those authentication related operations are logged along with the corresponding ICE identifier, which disables the attack of *Repudiation*.

4. IMPLEMENTATION OF THE AUTHENTICATION MODULE ON OPENICE

We implement a prototype of the authentication module on top of OpenICE to validate the compliance of the design specified in Section 3 to the requirements specified in Section 2. OpenICE is an open source, standard-based ICE instantiation, which utilizes Data Distribution Service (DDS) as its network middleware. In this section, we discuss in details the implementation and customization of our authentication module on top of OpenICE, with special attention given to taking advantage of DDS utilized in OpenICE.

According to the DDS standard [16], the DDS middleware provides a set of interfaces, called Security Model and Service Plugin Interface (SPI), for DDS-compliant implementations; and a set of built-in security plugins for these SPIs. These DDS plugins and APIs can be used to implement the basic functions of our authentication module. Therefore, we first establish the mapping between the basic authentication functions of the authentication module and the APIs of the corresponding functions provided by the DDS middleware. The rest authentication functions are then implemented based on such mappings.

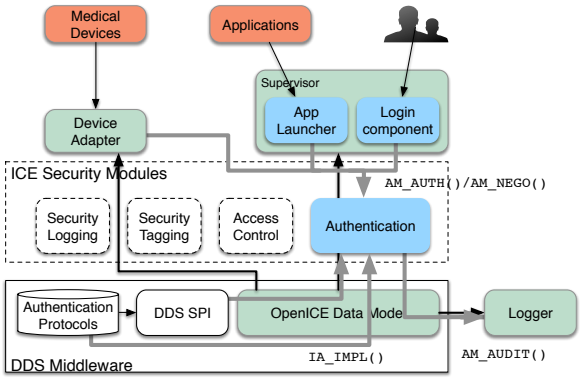


Figure 4: Implementing authentication on OpenICE, which are denoted by green boxes. The authentication module and its supporting functionalities are represented by blue boxes. The grey arrows show the way that authentication module interacts with OpenICE via the interfaces defined in Section 3.

More specifically, Figure 4 illustrates how the authentication module is integrated into the OpenICE platform. In Figure 4, components of the OpenICE platform are colored in green and those of the authentication module in blue. Grey arrowed lines in Figure 4 indicate the interactions between components during the authentication process. The authentication module works between the ICE supervisor/device adapter and the DDS middleware. The App Launcher and Login components, as part of the ICE supervisor, are responsi-

ble for invoking the authentication module to validate applications and human users, respectively.

As described in the following subsections, different types of ICE entities need to undergo different authentication processes, because they interact with the OpenICE platform in different ways and certain parts of their authentication processes cannot map to DDS APIs directly.

4.1 User authentication

The authentication of human users occurs when a user attempts to log into the ICE system. The key question of implementing such authentication functions is to determine the location of authentication entry points for human users. In our implementation, we choose to place these entry points in a login session prior to the user logs into the ICE system.

OpenICE initializes its applications and device adapters during the startup of its ICE supervisor, which does not include a user login session. We hence added a user login session before the applications and device adapters are initialized, which invokes the `AM_AUTH()` interface for user authentication. In this way, every time when the OpenICE system is powered on, a login window is popped up requesting the user to be authenticated. If the authentication succeeds, `AM_AUTH()` returns the user's ICE identifier, and a unique identity (UID) is generated for the user. Otherwise, any unauthenticated user is blocked by `AM_AUTH()` from launching any ICE applications or device adapters.

The login session is responsible for collecting the user's credential information (which could be a username-password combination, user's biometrics, or certificate stored in the user's ID card, depending on the authentication protocol adopted), and transferring it to `AM_AUTH()`. The corresponding authentication protocols for such credential have to be registered with the IA component using the `IA_INTF()` interface.

4.2 Application authentication implementation

Theoretically, the authentication of ICE applications can follow a process similar to the authentication of mobile apps, which is detailed as the following:

Step 1: An ICE application developer applies for a certificate from the certificate authority (CA), such as the ICE industry association, who signs the developer identity information and seals it in his/her certificate.

Step 2: The developer develops an application, signs the application with his/her certificate, and packages the signature, the developer's certificate, and the application as a package.

Step 3: When the application is distributed for installation, the application installer in OpenICE retrieves the application's signature using the developer's certificate, and invokes `AM_AUTH()` to decrypt the developer's information from the certificate using the CA's public key. If the retrieved signature and identity information match with those included in the distributed package, the application is deemed as unadulterated, and manufactured by the claimed developer. Otherwise, the application is untrustworthy and its installation is aborted.

However, applications are not installed by a dedicated installer in current OpenICE implementation. Instead, they are hardcoded in the initialization function of the ICE's supervisor and launched as part of the ICE supervisor's startup process. In particular, OpenICE uses the Java service loader mechanism to dynamically loads the provider of each application from a configuration file, and runs the provider to create the corresponding application. Considering this, we placed the authentication entry point for an application prior to where it is created. For each application, we also defined a data structure `cred` to store its credential, and a function to retrieve the credential from the application's provider. The credential is then delivered to `AM_AUTH()` by the entry point corresponding to the application. Only the applications passing authentication can be created by their providers.

We leveraged the DDS SPI authentication plug-ins to implement `IA_INTF()` for application authentication. OpenICE assigns each application and device with a unique DDS `DomainParticipant` for message exchange, before they are put into operation. DDS SPIs provide a series of security-related data structures and interfaces to protect the concept of `DomainParticipant`, which include an object called `IdentityCredential` to encode the identity information of an application instance, and `validate_local_identity()` and `validate_remote_identity()` operations to encapsulate how a specific authentication protocol verifies `IdentityCredential`. Therefore, the `IA_INTF()` function is implemented as assembling an `IdentityCredential` object based on the `cred` structure in the application provider, and invoking the operations of `validate_*_identity()` to perform the authentication.

4.3 Device authentication implementation

OpenICE utilizes DDS to create communication channels between non-human parties in the system, especially those between ICE devices and the ICE supervisor. DDS implements a publish-subscribe model for sending and receiving data among the network nodes. Publishers (i.e., network nodes producing information) create "topics" (e.g., patient's heartbeat rate) and publish data "samples" (e.g., the reading of a cardiograph) in these topics. DDS delivers the data samples to subscribers that declare an interest in these topics. In other words, once an ICE device has access to a specific topic, it can send data to all subscribers of this topic and receive messages from all publishers of this topic. Therefore, it is obvious that the authentication interface should be placed before the device starts to apply for any DDS topic dedicated to security-sensitive data communication.

OpenICE provides a device adapter for each type of devices that it supports and can be integrated into its network. The device adapters convert the output data from the supported devices, typically in proprietary formats, into the common data fabric used in OpenICE, and manage the exchange of messages between the devices and the OpenICE supervisor. When a device is plugged into the OpenICE system via its device adapter, the adapter first retrieves the device's driver and invokes the `create` method in the driver to initialize the device, which creates all topics required by the device. Then the adapter creates a heartbeat topic and sends heartbeat signals to the OpenICE supervisor. Upon receiving the heartbeat signal from the device adapter, the supervisor updates the device's connection status, which allows the device to exchange with other devices and applications messages on all topics it has required.

This above protocol of establishing connection between a device and the OpenICE system suggests that the device has access to other devices and the supervisor who subscribe the same topics, even before the supervisor receives the connection request from the device. To enforce device authentication without changing this protocol (which may affect the initialization procedure of devices and applications, or the the communication modes among them), we place the authentication entry point of a device before its adapter starts to the initialize procedure. Then we create a public DDS topic for delivering authentication-related information, typically authentication result and entity credentials. The credential of a device is stored in its driver, so that the entry point in its adapter can retrieve

and send the credential to `AM_AUTH()` via the authentication topic to start the authentication process. If a device fails the authentication, its adapter is terminated by the OpenICE supervisor, which disconnects the device from the rest of the system.

Similar to applications, OpenICE allocates a unique `Domain Participant` to each device based on a user-input `DomainID`. Thus, we implement the device-authentication function `AM_AUTH()` in the same way as how`IA_INTF()` is implemented

4.4 OpenICE entity credentials

Although our authentication module intends to support most mainstream authentication protocols, in its current prototype, we chose X.509 certificates as the credentials for human users, ICE applications, and medical devices. The reason is threefold. First, the X.509 standard is widely used in mainstream web browsers, smartcards, and security-sensitive applications such as on-line finance and E-Government, due to its timeliness and scalability. Second, the structure of X.509 certificates includes extension to allows dynamic modification, which can be used to store access control policies. Third, DDS SPI embodies X.509 as part of its mandatory built-in authentication plug-ins, which makes it easier for us to implement the authentication functions.

5. EVALUATION

We conducted a preliminary case study to evaluate the prototype of our authentication module. The evaluation focused on two aspects of the module: its effectiveness in protecting ICE systems from malicious attacks by authenticating ICE entities, and its impact to the performance of ICE systems (in terms of computational overhead caused by authentication).

5.1 Effectiveness

To examine the effectiveness of the authentication module, we conducted a case study upon the PCA safety interlock clinical scenario defined in the the ASTM F2761 standard. Notably, the current OpenICE implementation includes a demo that instantiates this scenario, which makes our case study easier. In the OpenICE demo, no authentication is considered or implemented. Thus, we first conducted a threat analysis on the demo to identify security threats due to the lack of authentication. We then simulated an example attack to compromise the OpenICE system by utilizing the security threats identified in the first step. The same example attack are then applied to the OpenICE platform twice, with our authentication module turned off and on, respectively. If the attack succeeds in the first time, but fails in the second time, then the effectiveness of our authentication module in protecting the OpenICE system (at least against the example attack and alike) is confirmed.

5.1.1 Safety Interlock scenario

The PCA safety interlock scenario describes an ICE-compliant interoperable system setting in the operating room, intended to automatically predict and prevent overdosing incidents caused by Patient Controlled Analgesic (PCA) infusion pumps before such incidents cause harm to the patient. In this scenario, the OpenICE uses a PCA pump to deliver pain-killing medication (such as morphine) to the patient, and uses a pulse oximeter to continuously monitor the vital signs of the patient (such as heart beat rate and blood oxygen saturation level). If the patient's vital signs, as monitored by the pulse oximeter, are below pre-defined thresholds, a safety interlock application running in the system sends instructions to the PCA pump to shut off the medication delivery, so as to avoid overdosing incidents. In the mean time, the safety interlock application issues an alarm to the nurse station via a distributed alarm system, calling for human intervention. The interactions between different parts of the system is shown in Figure 5.

SAFETY-INTERLOCK
1 CONNECT-TO-ICE($PulseOx, Pump, Alarm$)
2 CONNECT-TO-PATIENT($PulseOx, Pump$)
3 $diag$ = ORDER-REQUEST($Patient$)
4 VERIFY-VALUES($diag$)
5 SEND-DATA($Pump, diag$)
6 $bolus = diag.bolus$
7 **while** $bolus > 0$
8 SpO_2 = RECEIVE-DATA($PulseOx$)
9 **if** $SpO_2 <$ THRESHOLD
10 STOP-PUMP($Pump$)
11 RIASE-ALARM($Alarm$, RESP-DISTRESS)
12 EXIT
13 $bolus$ = RECEIVE-DATA($Pump$)
14 STOP-PUMP($Pump$)
15 RIASE-ALARM($Alarm$, MED-COMPLETE)

Figure 5: Safety Interlock Workflow

5.1.2 Threat analysis

For the above safety interlock scenario, ICE entities include all parties that have connection or interaction with the OpenICE platform, namely the PCA pump, the pulse oximeter, the safety interlock application, the medical information system, the alarm system, and the clinicians who has the privilege to operate the system. Our threat analysis focuses only on the security threats coming from the interactions among these ICE entities.

A basic assumption of our threat analysis is that the ICE platform itself is trustworthy. This means that the OpenICE platform does not need to verify the identifies of entities built within the platform. Therefore, security threats considered in the analysis are threats where a malicious entity can forge a fake identification to connect to the ICE platform and then launch a series of attacks:

- **Device replacement.** This threat represents scenarios where an adversary can disguise as a genuine device, such as the pulse oximeter or the PCA pump. For example, a fake pulse oximeter can use the ID of a genuine one to request to connect to the ICE platform, and once connected, can constantly deliver good SpO2 reading to the rest of the ICE system, which disables the alarm system. The logging system only records the ID of the genuine oximeter and has no way to track to the fake pulse oximeter when overdosing occurs.
- **Impersonation.** An attacker can log into (if needed) the ICE platform with a forged clinician id. Then s/he can operate the safety interlock application and modify the alarm threshold with a very high value, so that overdose will not trigger an alarm or a request to stop the infusion process. Forensic analysis can only find the forged id leading to a nonexistent person.

5.1.3 Attacking the OpenICE system

Device replacement. OpenICE implements an ICE instantiation and several applications to demonstrate ICE scenarios defined in the ASTM F2761 standard. It also provides a collection of simulated medical devices, including the PCA infusion pump and the pulse oximeter in the PCA safety interlock scenario. Physiological monitoring readings from the simulated pulse oximeter, such as SpO2 and heart rate, can be adjusted from an application

called *Simulation Control* to simulate the patient's health condition. OpenICE uses DDS topics as the communication channels between the ICE supervisor and its entities, and between the ICE supervisor and a local simulated device.

In its current implementation of the PCA safety interlock scenario, OpenICE does not include any authentication mechanism, but simply assigns a unique device identifier (UDI), created by a random number generator, to any device in the system during its initialization (line 1 in Figure 5). This apparently makes OpenICE vulnerable to Device Replacement attacks. To realize such attacks, we developed a fake pulse oximeter, which duplicates the simulated pulse oximeter device provided in OpenICE but constantly publishes a fixed 98% SpO2 reading. The fake pulse oximeter was registered in the OpenICE's device list as "*Simulated Fake Pulse Oximeter*".

The attack occurs when the fake oximeter instead of the real one is connected to the system. In this case, when the patient's health is deteriorating (i.e., the SpO2 ratio decreases), the fake oximeter still delivers the 98% reading to the safety interlock application (line 8 in Figure 5), preventing it from issuing a respiratory distress alarm at line 11. Although OpenICE can record the UDI of the fake oximeter, it cannot determine whether the UDI is generated for a genuine device or counterfeited by an adversary in forensic analysis.

Our authentication module prevented this type of attacks successfully, because the device authentication interface intercepts a device's request of DDS topics and verifies its authenticity during its initialization. In the above scenario, the simulated fake oximeter could not provide any certificate to the authentication module, leading to its failure in establishing network connection with the safety interlock application at Line 1.

Impersonation. OpenICE does not have a user login session, which allows any person to enter into the system without undertaking any identity check. To address this security threat, we developed a user login component for OpenICE along with the user authentication interface, as well as a database to store the names and credentials of a group of "virtual" valid users. The user login component was placed at the beginning of the OpenICE supervisor's initialization procedure. Thus, an attacker cannot access to the supervisor, as s/he cannot enter a valid pair of (user name, credential).

It is worth noting that authentication by itself cannot protect against all the five types of security risks enumerated in Section 2. For example, even with authentication in place, it is still possible for a malicious device to eavesdrop some communication channel in OpenICE to steal the identify of a genuine device and gain authorized access to the system, if the channel is not protected by proper encryption. Thus, ICE systems cannot establish adequate security, unless they are equipped with a security mechanism that coordinates a collection of security measures (including authentication) to provide comprehensive security protection.

5.2 Efficiency

To estimate the performance tradeoff caused by our authentication module, we compared the performance of the original OpenICE v1.8.0_45[3] in (simulated) clinical scenarios, with authentication enabled and disabled. The experiment was run on a desktop with 4-core intel 3.3Ghz i5 CPUs with 6MB cache and 4GB RAM. We also run the simulated ICE devices on another laptop with 2-core intel 2.1Ghz i7 CPUs with 4MB cache and 8GB RAM to simulate the environment in which these two versions of OpenICE systems operate. Both computers resided in the same local network, so they can communicate with each other.

In the experiment two clinical scenarios were considered: one with a PCA pump connecting to the OpenICE supervisor, and the other with multiple PCA pumps concurrently connecting to the supervisor. The latter was essentially a stress test: up to 20 simulated devices were reliably connected to the supervisor, given the available resources (i.e., the memory size of the experiment computers). The experiment results were illustrated in Figure 6.

In Figure 6, blue curves in all subfigures represent the resource usage when the authentication module was turned off; red curves denote the resource usage with the password authentication on, and green curves illustrate the resource usage with the X.509 certificate authentication on. The starting 3 sharp spikes in Figure 6a are caused by system initialization, where the highest CPU usage ratio is 42%.

In Figure 6 the three curves, representing the resource consumption of the system in three different settings, share a similar shape for all experiment scenarios. For most of the course, the difference among these curves is less than 10%, which suggests that authentication does not cause significant increases in CPU usage.

Moreover, the shape of the first spike for each mode shows more difference than the other two spikes. This is because we enforced authentication before devices and applications are initialized, which causes authentication to pose more influence to system performance at the beginning of system startup. The first spikes in all subfigures also suggest that certificate authentication was less CPU-consuming than password authentication, which probably because the latter needs to search the password database for the password entered by the user. The highest CPU utilization observed after system initialization was 12%, which was caused by password authentication. We consider this level of overhead as acceptable. The CPU usage curves for 3 modes in the stress test are also not significantly distinguishable, while verifying twenty X.509 certificates simultaneously contributes the highest CPU usage spike (37%) in Figure 6c.

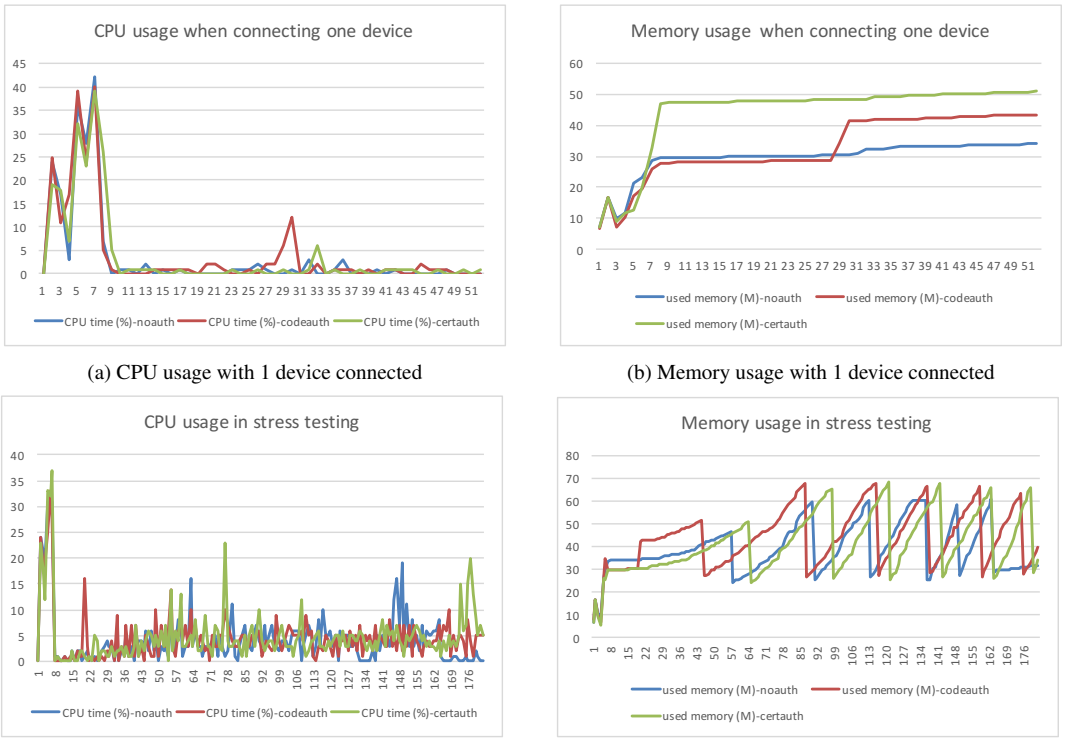
Axis Y in Figure 6b and 6d represents the number of Megabyte of memory consumed by the OpenICE supervisor. When authentication was turned off, the OpenICE supervisor consumed up to 39 MB memory. In comparison, certificate authentication took up to 50 MB memory when only one device was connected to the supervisor, and up to 68 MB memory when 20 devices were connected. The memory consumption of password authentication falls in the middle of the other two modes in both tests, with a maximum memory usage of 67 MB. It is obvious from Figure 6b and 6d that our authentication module with two different authentication mechanisms entails a fixed but moderate level of memory utilization.

6. RELATED WORK

Security of ICE systems has been attracting more and more attentions from stakeholders and the academia. One focus is what security challenges are faced by current ICE technology. For example, Krishna et al. [22, 20] enumerated 5 categories of possible security attacks for the ICE platform. Hatcliff et al. discussed the risk of malicious devices being connected to ICE systems and proposed a solution to establish and reason the trustworthiness of ICE entities [10]. The core concept of their solution is to build a trust chain for verifying an entity's provenance and integrity, by attaching it with a credential issued by authorization bodies.

There is also heated discussion among stakeholders on what constitutes acceptable security in ICE systems. For example, [9] and [14] discussed the high-level goals and challenges of engineering secure ICE systems. [8] and [21] defined a set of ICE security

[3]Downloaded from https://github.com/mdpnp/mdpnp



(a) CPU usage with 1 device connected

(b) Memory usage with 1 device connected

(c) CPU usage with 20 devices connected

(d) Memory usage with 20 devices connected

Figure 6: OpenICE resource consumption with different authentication mechanisms on.

requirements from different perspectives. However, their requirement sets are not assembled together as a whole, so it is hard to implement them in an architectural way.

[19] suggested threat modeling as the first step to design a security architecture for ICE systems. They applied an attack-graph-based analysis to an example interoperable medical system to discover its security threats and discussed possible mitigation measures. However, their analysis focused on threats faced by medical devices in the system, and did not consider ICE security from the system's perspective.

Some research effort (e.g. [6, 5]) focuses on extending access control models with the feature of break-the-glass, which is mandated in many clinical scenarios. In [5], break-the-glass policies were defined as a generic extension to traditional access control models, and were formalized as rules written in the XACML language [15]. Ferreira et al. [6] developed a new BTG-RBAC model to integrate the break-the-glass feature within the standard RBAC model, which provided the third option to break the glass in addition to the standard grant/deny response to the subject's request. Their recent work [7] designed an access control application to facilitate patients' access to their EHR records based on their own access models. Unfortunately, the applicability of these approaches to OpenICE systems remains unclear.

Although there has been plenty of research effort on ICE security, few security mechanisms have been proposed and implemented on realistic ICE systems. In fact, the only security mechanism aware to the authors was proposed by Salazar and Vasserman [18, 17], which implements a modularized prototype for device authentication and communication encryption in the MDCF framework. This mechanism provides a preliminary access control model with a break-the-glass feature, and our authenticate module has the following differences as compared to it:

1. Their mechanism focuses only on device authentication, while we proposed an authentication solution for all types of ICE entities that can be connected to the ICE platform.
2. Their authentication mechanism was designed solely based on the device connection protocol of the MDCF framework, which restricts its applicability to other ICE frameworks. For instance, OpenICE creates all communication channels for the devices before the heartbeat channel is established, which is opposite to the workflow enforced by the MDCF device connection protocol. Hence, their mechanism is not compatible with OpenICE. In contrast, the three-layer design of our authentication module allows system developers to customize the locations of entry points, which makes it suitable for being deployed in a variety of ICE instantiations.

7. CONCLUSION

We have presented a three-layered, middleware-independent authentication framework for ICE-compliant interoperable medical systems: the top level of our framework incorporates a collection of authentication entry points, the locations of which are customizable for specific ICE systems, to intercept the communication between ICE supervisor and its entities; the middle level integrates all functionalities related to authentication, such as authentication protocol management, task scheduling, and auditing; and lastly, the bottom level provides a set of interfaces for the invocation of authentication protocols, which can be provided by networking middleware.

Our authentication framework has been implemented on top of OpenICE to evaluate and demonstrate its effectiveness in enhancing the security of ICE-compliant systems. Our experiments showed that the authentication framework can avoid device replacement and impersonation attacks to the OpenICE framework, with moderate computational overhead. We have to admit that, due to realistic difficulties, we could not deploy our authentication framework in a realistic ICE system setting to fully evaluate its efficiency and other

engineering properties (e.g., scalability) in real clinical practices.

Our future research has two directions. On one hand, we plan to ensure all possible entry points that can be used by potential attacks are protected by appropriate authentication interfaces. To do so, program analysis techniques could be used to verify the locations of authentication interfaces. More importantly, we plan to fully evaluate our authentication module against the requirements defined in Section 2 and its scalability in real clinical use. On the other hand, we plan to develop other security modules (based on the authentication module) for OpenICE, and eventually provide a holistic security solution for ICE-compliant systems.

8. ACKNOWLEDGMENTS

This research was supported in part by NIH 1U01EB012470-01, NSF CNS-1505799, the Intel-NSF Partnership for Cyber-Physical Systems Security and Privacy, the DGIST Research and Development Program of the Ministry of Science, ICT and Future Planning of Korea (CPS Global Center), China Scholarship Council (No. 201304910097), and the National Natural Science Foundations of China (Grant No. 61471344).

Disclaimer. The mention of commercial products, their sources, or their use in connection with material reported herein is not to be construed as either an actual or implied endorsement of such products by the U.S. Department of Health and Human Services.

9. 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] D. Arney, J. Plourde, R. Schrenker, P. Mattegunta, S. F. Whitehead, and J. M. Goldman. Design Pillars for Medical Cyber-Physical System Middleware. In V. Turau, M. Z. Kwiatkowska, R. Mangharam, and C. Weyer, editors, *5th Workshop on Medical Cyber-Physical Systems, MCPS 2014, Berlin, Germany, April 14, 2014*, volume 36 of *OASICS*, pages 124–132. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2014.

[3] D. Arney, S. Weininger, S. F. Whitehead, and J. M. Goldman. Supporting Medical Device Adverse Event Analysis in an Interoperable Clinical Environment: Design of a Data Logging and Playback System. In *ICBO*, 2011.

[4] ASTM International. *ASTM F2761-09(2013), Medical Devices and Medical Systems - Essential safety requirements for equipment comprising the patient-centric integrated clinical environment (ICE) - Part 1: General requirements and conceptual model*, 2009.

[5] A. D. Brucker and H. Petritsch. Extending access control models with break-glass. In *Proceedings of the 14th ACM symposium on Access control models and technologies*, pages 197–206. ACM, 2009.

[6] A. Ferreira, D. Chadwick, P. Farinha, R. Correia, G. Zao, R. Chilro, and L. Antunes. How to securely break into RBAC: the BTG-RBAC model. In *Computer Security Applications Conference, 2009. ACSAC'09. Annual*, pages 23–31. IEEE, 2009.

[7] A. Ferreira, G. Lenzini, C. Santos-Pereira, A. Augusto, M. Correia, and others. Envisioning secure and usable access control for patients. In *Proceeding of 2014 IEEE International Conference on Serious Games and Applications for Health*, 2014.

[8] D. Foo Kune, K. Venkatasubramanian, E. Vasserman, I. Lee, and Y. Kim. Toward a safe integrated clinical environment: a communication security perspective. In *Proceedings of the 2012 ACM workshop on Medical communication systems*, pages 7–12. ACM, 2012.

[9] J. Hatcliff, A. King, I. Lee, A. MacDonald, A. Fernando, M. Robkin, E. Vasserman, S. Weininger, and J. M. Goldman. Rationale and architecture principles for medical application platforms. In *Cyber-Physical Systems (ICCPS), 2012 IEEE/ACM Third International Conference on*, pages 3–12. IEEE, 2012.

[10] J. Hatcliff, E. Vasserman, S. Weininger, and J. Goldman. An overview of regulatory and trust issues for the integrated clinical environment. *Proceedings of HCMDSS 2011*, 2011.

[11] A. King, D. Arney, I. Lee, O. Sokolsky, J. Hatcliff, and S. Procter. Prototyping closed loop physiologic control with the medical device coordination framework. In *Proceedings of the 2010 ICSE Workshop on Software Engineering in Health Care*, pages 1–11. ACM, 2010.

[12] A. King, S. Procter, D. Andresen, J. Hatcliff, S. Warren, W. Spees, R. Jetley, P. Jones, and S. Weininger. An open test bed for medical device integration and coordination. In *Software Engineering-Companion Volume, 2009. ICSE-Companion 2009. 31st International Conference on*, pages 141–151. IEEE, 2009.

[13] A. L. King, S. Chen, and I. Lee. The middleware assurance substrate: Enabling strong real-time guarantees in open systems with openflow. In *Object/Component/Service-Oriented Real-Time Distributed Computing (ISORC), 2014 IEEE 17th International Symposium on*, pages 133–140. IEEE, 2014.

[14] I. Lee, O. Sokolsky, S. Chen, J. Hatcliff, E. Jee, B. Kim, A. King, M. Mullen-Fortino, S. Park, A. Roederer, and others. Challenges and research directions in medical cyber-physical systems. *Proceedings of the IEEE*, 100(1):75–90, 2012.

[15] OASIS Standard. *eXtensible Access Control Markup Language (XACML)*, version 3.0 edition, 2013.

[16] Object Management Group. *Data Distribution Service (DDS)*, v1.4 edition, 2015.

[17] C. Salazar. *A SECURITY ARCHITECTURE FOR MEDICAL APPLICATION PLATFORMS*. PhD thesis, Kansas State University, 2014.

[18] C. Salazar and E. Y. Vassserman. Retrofitting communication security into a publish/subscribe middleware platform. Washington, DC, USA, July 2014.

[19] 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.

[20] E. Vasserman, K. Venkatasubramanian, O. Sokolsky, and I. Lee. Security and Interoperable-Medical-Device Systems, Part 2: Failures, Consequences, and Classification. *IEEE Security Privacy*, 10(6):70–73, Nov. 2012.

[21] E. Y. Vasserman and J. Hatcliff. Foundational Security Principles for Medical Application Platforms. In *Information Security Applications*, pages 213–217. Springer, 2014.

[22] K. K. Venkatasubramanian, E. Y. Vasserman, O. Sokolsky, and I. Lee. Security and Interoperable-Medical-Device Systems, Part 1. *IEEE Security & Privacy*, 10(5):61–63, Sept. 2012.