A Configurable Framework for RBAC, MAC, and DAC for Mobile Applications

Yaira K. Rivera Sánchez
Doctoral Dissertation
Major Advisor: Dr. Steven A. Demurjian
Associate Advisors: Dr. Jinbo Bi, Dr. Bing Wang, Dr. Thomas Agresta

In the last decade, mobile computing as evidenced by the emergence of mobile devices (smartphones, phablets, tablets) has dominated personal and business computing. Users of traditional computing devices (e.g., PCs, laptops, etc.) are transitioning to mobile devices to perform daily tasks such as managing emails, playing games, viewing/editing documents, paying bills, managing healthcare data, etc. The majority of these tasks can be performed through the means of mobile applications, which is a piece of software specifically made to run on a mobile device. Mobile applications (apps) can contain data that ranges from being non-sensitive to highly-sensitive. Specifically, for those apps that contain highly-sensitive data (e.g., banking apps, electronic health records (EHRs), etc.), there is a need to provide authentication and authorization mechanisms in order to secure the application's data. Many mobile apps provide basic user authentication, and, after successful authentication, the user has access to all of its features. Nevertheless, even though there are critical requirements for mobile apps to secure highly-sensitive data, developers have failed to establish sophisticated and multi-faceted authorization mechanisms within the mobile computing design and development process. Specifically, an argument can be made that mobile computing would significantly benefit through the adoption of the three classic access control models:

Role-Based Access Control (RBAC), Mandatory Access Control (MAC), and Discretionary Access Control (DAC).

The overall high-level focus of this dissertation is to propose and realize a configurable framework for RBAC, MAC, and DAC for mobile applications that is capable of supporting access control in different security layers. Security is controlled from three perspectives. The first perspective is for the user interface in terms of which screens and/or their components are accessible to a user under RBAC with optional delegation via DAC. This security layer focuses on modifications to the UI. The second perspective is to control the mobile application's API services in order to define the API services that can be invoked by a particular user based on RBAC and/or MAC permissions with optional delegation via DAC. This security layer between the UI and mobile application API replicates the mobile application's API by creating a mirrored set of services that invoke the original API services so that each call can be intercepted to add RBAC, MAC, and or/ DAC security checks. The third perspective focuses on interactions between the services of the mobile application's API and server-side APIs for the various data servers, to again control whether the user via the mobile application service is authorized to invoke specific server-side APIs by RBAC and/or MAC with optional delegation. This security layer between the two different APIs (mobile app and server-side) is accomplished through the creation of a server interceptor API associated with a cloud computing infrastructure to intercept invocations for RBAC, MAC, and DAC checks. In support of these three perspectives, there is a *unified mobile computing and security model* with RBAC, MAC, and/or DAC can be leveraged to define and enforce UI and service-based permissions in a mobile application. Choosing security features from one or more of these three perspectives provides for the *dynamic combination of access control models and configuration options* to allow for custom security on a mobile-app-by-mobile-app basis. The final step is the ability for the framework to provide human assisted processes and automated algorithms for *access control security enforcement code generation and interceptors*. The end result is that the mobile app can secure the data that can be managed (e.g., inserted, retrieved, updated, deleted) via it's APIs from differing and complementary perspectives, creating multiple additional security layers for RBAC, MAC, and/or DAC that are then adaptable to different mobile apps.

Table of Contents

List of Tables

List of Figures

Chapter 1
Introduction

The increase of capabilities and features of mobile computing devices have changed the way that individuals perform many of their daily activities. Numerous tasks previously performed with a desktop or laptop have transitioned to mobile devices (phones, phablets, and tablets). As we have seen in recent years, mobile devices have become mainstream and begun to serve as a replacement for traditional PC-based computing in numerous and varied consumer and industrial markets. Nevertheless, all of these advances come with critical security risks that can lead to the compromising of confidential data that could affect a user, a group of individuals, and/or an organization. Despite the presence of secure highly-sensitive data, mobile development frameworks and developers have failed to establish sophisticated and multi-faceted authorization mechanisms within the mobile computing design and development process. Access control mechanisms are commonly utilized to secure highly-sensitive data and are able to determine which information each user can access/store in a particular system, with the proviso that disclosing the wrong information could lead to serious consequences (Rindfleisch, 1997). One important dimension of security that has been largely overlooked for mobile applications and that has been dominant in traditional systems and database applications are the three classic access control models: Role-Based Access Control (RBAC) (Ferraiolo & Kuhn, 1992), Mandatory Access Control (MAC) (Sandhu & Samarati, 1994), and Discretionary Access Control (DAC) (Department of Defense, 1985). The other dimension that has been often not adequately considered is that almost all mobile applications in all domains access data not directly but via a wide array of web and cloud application programmer interfaces

(APIs). In fact, in healthcare, the Meaningful Use Stage 3 (Himss, 2016) guidelines require all health information technology (HIT) systems (e.g., electronic health records (EHR), personal health records (PHR), etc.) to have API services to access, modify, and exchange health-related data. This necessitates the consideration of the usage of RBAC, MAC, and DAC to control access to the services that are utilized by a mobile application.

The approach in this dissertation is to explore the different ways or configurations that RBAC, MAC, and/or DAC capabilities can be included as multiple separate and interacting security layers in a mobile application that range from the mobile application's user interface to the server side APIs utilized by the mobile application's API services to access multiple data sources. Mobile applications contain dynamic data and are characterized by a set of interacting components that include: a user interface (UI) to facilitate interactions; a middle layer component that is an Application Programming Interface (API); and a data source (e.g., database, repository, cloud, server, etc.) that are the APIs that the mobile app's API services invoke to interact with multiple data sources. In the process, data is retrieved from a data source (e.g., repository, database, etc.) and/or stored into the source, at varied intervals. To illustrate, Figure 1.1 augments the components of a mobile application with three additional security layers: one layer to control the look-and-feel and content of the UI and two layers to control data that is interchanged between the mobile app UI, the mobile app API, and the data source via server-side API. The two security layers for data interchange involve the invocation of services for the mobile app. In the first of these two layers, the mobile app invokes its own API services, and there is a need to provide RBAC and MAC permissions to intercept these invocations to control which services can be invoked by which user based on RBAC and/or

MAC permissions. In the second of these two layers, the mobile app's API services invoke server-side APIs for multiple data sources, and there is a need to provide another intercept to control which server-side services of data sources the mobile app can invoke based on RBAC and MAC permissions. In the top of Figure 1.1, the different access control models are shown and can be utilized to generate a set of RBAC, MAC, and/or DAC security policies that are enforced in the mobile application. From left to right in Figure 1.1, this includes:

- Defining for each user a role (RBAC) and a sensitivity level (MAC) (i.e., a clearance such as top secret, secret, confidential, unclassified, etc.) that are utilized to define and support permissions and providing the ability for a user to delegate via DAC RBAC and/or MAC permissions to another user.
- Defining user interface (UI) permissions on the mobile app user interface to control which screens and/or their components are accessible to a user under defined roles (RBAC) and are delegable from one user to another by role in support of DAC.
- Defining API permissions that identify which services of the mobile app's API need to be securely controlled using RBAC and/or MAC and creating a mirrored API that replicates the signature of each mobile app's API service of the mobile app's API and servers as an intercepting API to intercept mobile app service invocations in order to embed and perform RBAC and MAC permission checks.
- Defining server interceptor API permissions for the data source/repository/database that provides an additional level of RBAC, MAC, and DAC permission checks on the server-side APIs that are invoked by the mobile app's API services.

The end result is the ability to generate and enforce permissions (shown in the middle part of Figure 1.1) utilizing the RBAC, MAC, and/or DAC models to create a customized version of the user interface (UI), to create a customized version of the mobile app's API, and/or to create a customized version of the server-side API that can check RBAC, MAC and DAC permissions.

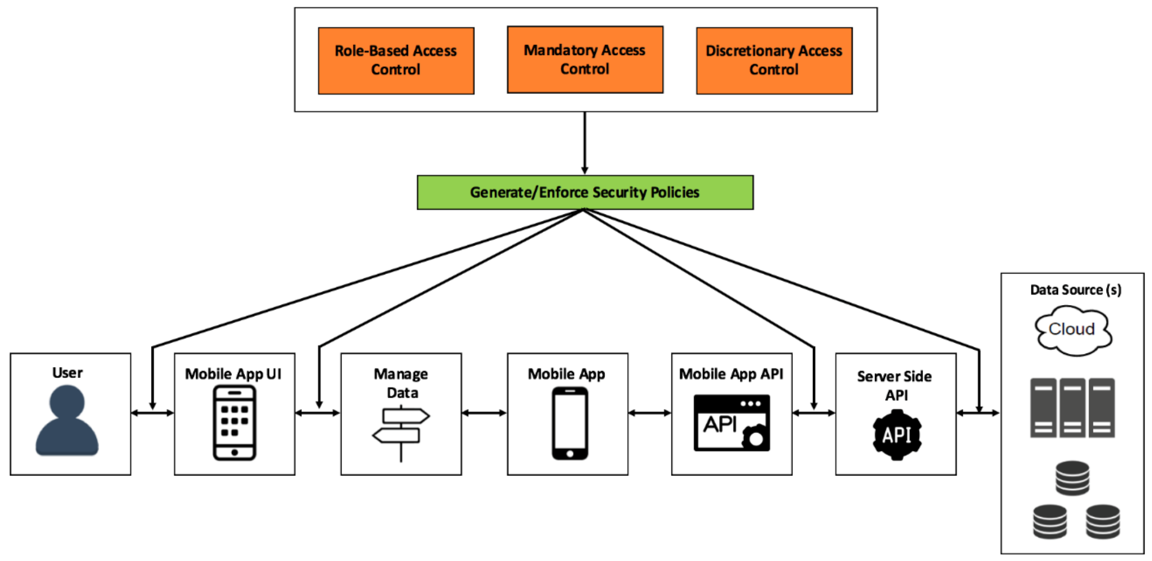


Figure 1.1. Flow of Proposed Configurable Framework.

1.1. Motivation for Access Control for Mobile Applications

Mobile devices are highly portable and can be utilized to perform daily tasks such as reading a document, browsing the internet, and managing emails. In addition, mobile devices contain a wide range of mobile applications including: games, social media, health & fitness, ebooks, banking, email, music, etc. According to 'The 2015 U.S. Mobile App Report' (Lella, Lipsman, & Martin, 2015), mobile application usage is rapidly increasing among mobile device users, surpassing the time they spend on their mobile device web browser as well as the time they spend utilizing a PC/laptop. For both personal and business usage, there is a need to protect secure information in mobile applications ranging from personally identifiable information (PII) to protected health information (PHI) to confidential work product that is displayed, accessed, modified, and stored. Commonly, developers of software and mobile applications focus on applying typical authentication mechanisms (e.g., passwords, PINs, fingerprints, etc.) in order to protect a user's data (Rivera Sánchez & Demurjian, 2016). In addition, an authenticated user often receives all or nothing; successful authentication means the ability to access (read, write, or modify) all of the resources of the application, which may not always be desired on an application-by-application and user-by-user basis. This mentality has led the developers to not take into consideration the fact that they need to verify the user's identity each time a user performs an action. As a result, this can lead to the possibility of having malicious attacks in the system since an unauthorized user could attempt to become over privileged or could obtain improper access to resources.

In addition, many mobile applications do not support fine-grained security policies that are able to specify which resources/features a specific user has access to. For instance, suppose that we have a mobile application utilized by personnel at a pharmacy to fill and process prescriptions for customers with a user interface (UI) that has five screens to: look up the status of a prescription (Screen 1); enter a new prescription to be filled (Screen 2); fill and dispense the prescription with the appropriate medication (Screen 3); look up to see if a medication is in inventory (Screen 4); and, order medications for inventory (Screen 5). The five screens could be linked by next and back buttons or could be five different tabs on one screen. There are two types of users: *pharmacy technicians* that interact with the customer to receive and enter the prescription; and, *licensed pharmacists* that have the legal authority to fill and dispense the prescription. A pharmacy technician would be limited to Screens 1, 2, and 4, while a licensed pharmacist would have access to all five screens. To achieve this in practice, access control policies can be adapted in order to grant them the necessary permissions to view only the resources they need to work with in order to avoid improper disclosure of information. Users that have access to data that does not pertain to them could benefit from this. For example, in 2013, a billing technician at a hospital spent several months looking for people that had recently been in car accidents and then sold that information to an attorney. The attorney would then contact the individuals who were involved in the car accident and offered them legal representation (Wiech, 2013). This issue highlights the need for access control in systems that contain highly sensitive information, such as hospitals and health insurance companies.

Mobile devices and computing are being improved on a daily basis in terms of hardware and software, increasing capabilities, features, and capacity. This in turn has resulted in the rise of new security risks. In the worst-case, a mobile device may be lost or stolen; if there are available techniques to control and securely access highly sensitive data, then damage can be mitigated. For example, healthcare data stored in a mobile device is being created, retrieved, and manipulated from multiple sources and by varied applications and this sensitive information must be protected from disclosure. This security requirement is juxtaposed against a recent survey (West & Miller, 2009) where a strong majority of people wanted to manage their healthcare electronically, including: email access with providers (74%), diagnostic test results electronically (67%), and access to their EMR (64%). These tasks require a great amount of security as the information to be shared is highly sensitive and pertains to specific people from multiple sources and ultimately resides in a patient's mobile device.

1.2. Motivation for Mobile Healthcare Applications

Mobile computing devices and applications have exploded in the marketplace, with the Gartner group forecasting worldwide shipments in 2015 (Gartner Newsroom, 2015) of 1.9 billion mobile phones and 230 million tablets, which is outpacing PC/laptop sales significantly (300 million estimate). In the United States, a PEW Research Center report of smartphone usage (Smith, 2015; PEW Research Center, 2015) found that as of October 2014, 64% of American adults own a Smartphone while as of January 2014, 42% own a Tablet, and 32% own an e-reader. Predictive statistics project that tablet users will surpass 1 billion in 2015 worldwide (eMarketer, 2015) while the total of mobile devices will exceed 12.1 billion by 2018 (Radicati, 2014). In addition, Cisco reported (Cisco, 2014) in 2014 that 497 million mobile devices were added that year, and 88% of that growth is accounted to smartphones and, predicted that by 2019, there will be approximately 1.5 mobile devices per capita giving a total of 11.5 billion mobile devices around the world. As mobile devices become more mainstream, they have begun to serve as a replacement for traditional PC-based computing in major consumer and industrial markets.

One such domain that is exploding is healthcare, where there is a growing desire for an individual seeking to utilize his/her mobile device to monitor and track health conditions and fitness that includes both protected health information (PHI) and personally identifiable information (PII). For example, consider the proliferation of health and fitness applications on multiple mobile platforms for: pharmacies and organizing medications (myCVS (CVS Pharmacy, 2015), MEDWatcher (MedWatcher, 2012), Drugs.com Medication guide and Pill Identifier Applications (Drugs.com, 2008), etc.); personal health record (PHR) applications (CAPZULE PHR (Capzule, 2012), MTBC PHR (MTBC PHR, 2011), suite of WebMD Applications (WebMD, 2016), etc.); a wide array of fitness applications that work with phones and wearables (Duffy, 2016; Cohen, 2015); Apple's HealthKit app (iOS 9 Health, 2014) and the Google Fit fitness tracker (Google Play, 2013), where both companies have pushed strongly into the smartwatch market to track activity, heart rate, blood pressure, etc. (Kelly, 2014); and, Apple's ResearchKit, which is an open source framework for mobile applications to support medical research (Apple, 2015). Patients also seek to have access via their mobile devices to the electronic medical records (EMRs) utilized by their medical providers, as well as various health information technology (HIT) systems that contain medical testing results (Care360, 2014) or results from imaging testing (My Imaging Records App, 2013). All of these systems must adhere to the Health Insurance Portability and Accountability Act (HIPAA) (HHs.gov, 2013) for the security, availability, transmission, and release of a patient's medical information.

In this dissertation, our approach for a Configurable Framework for RBAC (Ferraiolo & Kuhn, 1992), MAC (Sandhu & Samarati, 1994), and DAC (Department of Defense, 1985) for Mobile Applications is presented and discussed by utilizing a healthcare setting as a means to illustrate and demonstrate the concepts and ideas of this work. We have chosen the healthcare domain since improper disclosure of data (both PHI and PPI) can have serious impact on patients, which can include: personal embarrassment, prejudice, ostracization from family and community groups, death, and issues with insurability (Rindfleisch, 1997). Moreover, a report by Ponemon Institute (2009) revealed that the cost of data breaches in the majority of the industries (e.g., communications, retail) per lost or stolen record averages approximately $154 per record as shown in Figure 1.2 (Ponemon Institute, 2015). This cost is even higher in the healthcare industry, as high as $363 per record, which is more than double the cost to manage a data breach in comparison to other industries. While we utilize a healthcare setting as an appropriate and informative manner to present the work in this dissertation, note that our proposed approach is generalized to any mobile application as was illustrated in Figure 1.1.

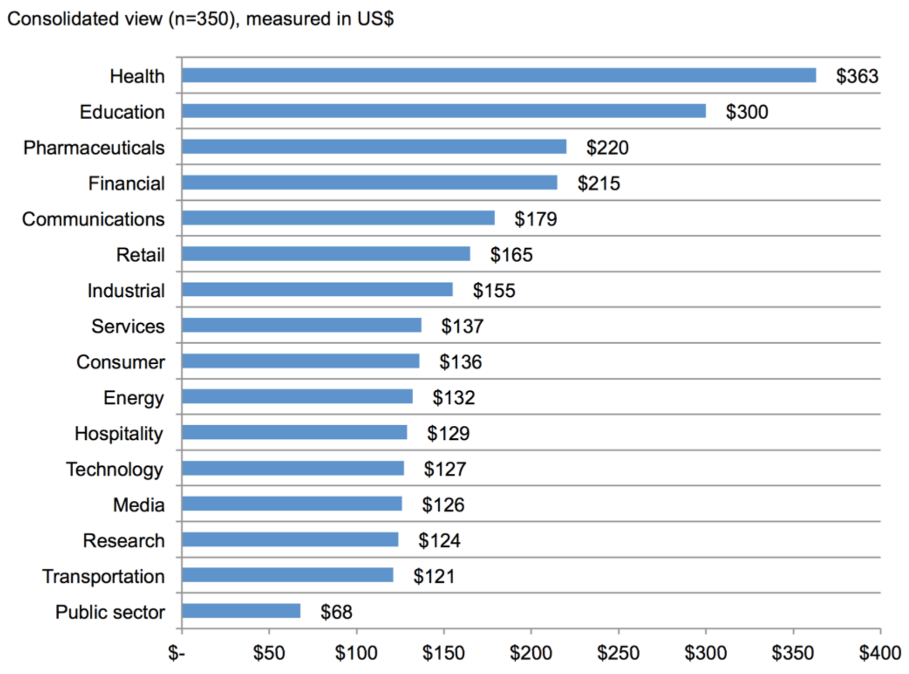


Figure 1.2. Per Capita Cost by Industry Classification.

1.3. Mobile Healthcare Applications Requirements and Challenges

Mobile applications span a broad spectrum of complexity, including games, social networking, email, web browsing, financial management, health and fitness, pharmaceutical, etc. For both personal and business usage, there is a need to insure that access to secure information is controlled, ranging from protected health information (PHI) and personally identifiable information (PII) to confidential work product that is displayed, accessed, modified, and stored. In support of such a scenario of usage for healthcare, one motivation and justification factor for this dissertation involves the transition from paper-based to electronic health records (EHRs) systems which has greatly increased in the past decade with eight out of ten physicians in the U.S. utilizing EHRs in their practices (Heisey-Grove & Patel, 2015). Despite this progress, there is still a need for a significant next step to allow patients and medical providers to easily access healthcare data that is distributed across multiple EHRs and other health information technology (HIT) systems. To support these actions, health information exchange (HIE) for the interoperation across sources has the potential to reduce healthcare data expenses where healthcare institutions could save up to $77.8B in the U.S. (Walker et al., 2005). In addition, the Office of the National Coordinator issued a report (Health and Human Services Department, 2015) in 2015 on certification rules for EHRs that has required that HIT vendors develop RESTful APIs for EHRs and other systems so that patients and medical providers using mobile health (mHealth) applications (apps) can easily access their healthcare data from multiple sources.

Specifically, there is a diverse collection of stakeholders who are interested in healthcare and medical data. From the patient side, stakeholders that directly interact on a day-to-day basis include: patient (him/herself), family members (child care, elder care, spousal care), nutritionists, personal trainers, therapists (physical, occupational, pulmonary), home healthcare aides, etc. Accompanying mobile health (mHealth) applications in healthcare and fitness for patients are numerous and diverse including: tracking medications (myCVS (CVS Pharmacy, 2015), MedWatcher (MedWatcher, 2012), etc.); personal health records (PHR) (CAPZULE PHR (Capzule, 2012), MTBC PHR (MTBC PHR, 2011), etc.); fitness applications that work with phones and wearables (Cohen, 2015); Apple's HealthKit app (iOS 9, 2014) and the Google Fit fitness tracker (Google Play, 2013), to track activity, heart rate, blood pressure, etc. (Kelly, 2014); Apple's

ResearchKit (Apple, 2015), an open source framework for mobile applications to support medical research, etc. Patients also seek to have access via their mobile devices to the electronic health records (EHRs) utilized by medical providers and health information technology (HIT) systems that contain medical testing results (Care360, 2014) or results from imaging testing (My Imaging Records App, 2013).

From the medical provider side, stakeholders include: for daily care, internist, family medicine, nurse practitioner, physician assistants, and pediatrics; for periodic care specialists, cardiologists, ENTs, orthopedic surgeons, etc.; for mental healthcare physiatrist, phycologist, therapist, etc.; and, for medical services, individuals at laboratories, imaging centers, pharmacies, etc. The interest in all of these stakeholders in mobile applications is evidenced by a report that found 43,700+ medical applications in the Apple application store, with approximately 54% targeting healthcare with 69% of the applications targeting consumers/patients and 31% for use by medical providers (Aitken, 2013); this was further summarized with emerging mobile devices and applications for healthcare professionals (HIT Consultant, 2014). The aforementioned healthcare professionals utilize diverse mobile apps and mHealth apps for: administrative purposes (information management (e.g., write notes, organize information) and time management (e.g., schedule appointments, schedule meetings, etc.)), health record maintenance and access (e.g., EHRs, medical imaging apps), communications and consulting (e.g., multimedia messaging, social networking), reference and information gathering (e.g., medical textbooks, medical journals), patient monitoring (e.g., clinical decision-making, monitor patient health, collect clinical data) and, for medical education (e.g., case studies, continuing medical education) (Ventola, 2014). All of these systems must adhere to the

Health Insurance Portability and Accountability Act (HIPAA) (HHS.gov, 2013) for the security, availability, transmission, and release of a patient's medical information.

Most of these health and fitness mobile applications that are centric to an individual are accompanied by the desire by patients to be able to dictate and define the way that such information can be shared with other individuals. One such effort has surveyed patients to ascertain the degree that they wish to exert regarding the attainment of privacy control at varying levels of granularity over their health and fitness information, which may be present in electronic form in various locations (Caine & Hanania, 2013). For a given patient, this effort highlights the potential recipients of the information (e.g., primary physicians, spouse, family, emergency medical providers, etc.) and the type of information to be controlled (e.g., contact info, current conditions, medications, recent test results, genetic information, etc.). In such a setting, patients are also interested in actually defining specific fine-grained access control (Sujansky et al., 2010) by designating access by role, for example: a family member may view my medication list (but not all of them), a medical provider may view my medication list and history of hospital visits (but not modify), my personal physician may both view and modify my health care and fitness data, etc. These efforts highlight a strong need to achieve fine grained role-based level of security to allow patients to define who can see and/or modify what portions of their health/fitness data other individuals can view/modify using mobile applications for health care, where the mobile application itself can be customized based on role to meet the permission definition provided by the patient (Peleg et al., 2008). In addition, one of the main challenges that healthcare providers face when utilizing an EHR mHealth app is patient data privacy. EHRs can contain patient data such as past medical history, medications, conditions, and insurance information. Depending on their area of work, users of the EHR could be limited to only accessing certain parts of the patient data. For instance, a doctor has access to all the data shown in the EHR but an administrative professional can only schedule appointments and, view a patient's demographics and a patient's insurance information.

To fulfill these actions, we propose to protect the data of a user of a mHealth application by applying the RBAC, MAC, and DAC models in order to allow the owner(s) of the information to decide which data can be accessed/modified by other users. In support of our efforts, the three dominant access control models (Sandhu & Samarati, 1994) that could be utilized to secure highly-sensitive data are: role-based access control (RBAC); discretionary access control (DAC); and, mandatory access control (MAC). RBAC has a strong history in healthcare; a literature review (Fernández Alemán, Señor, Lozoya, & Toval, 2013) identified access control models deployed by EHRs, where out of 45 articles reviewed, 35 used access control methods, and 27 of these specifically utilized RBAC. DAC has also been studied for EHRs in conjunction with RBAC (Alhaqbani & Fidge, 2008; Khan & Sakamura, 2012) in an attempt to combine the capabilities and advantages of both approaches. There have been some very limited attempts to utilize MAC in the health care domain; one study (Gajanayake, Ianella, & Sahama, 2014) considered the use of MAC in EHRs; another (Hafner, Memon, & Alam, 2007) explored the combination of MAC with RBAC and DAC; and, the HL7 vocabulary (HL7 v3, 2013) where the confidentiality portion (HL7, 2013) defines sensitivity levels of low, moderate, normal, restricted, unrestricted, and very restricted.

In order to provide the sharing and exchange of information, healthcare standards have been developed including: Health Level Seven (HL7) v3 (Health Level Seven

International, 2011) to manage, exchange, integrate, and retrieve electronic health information; and, Digital Imaging and Communications in Medicine (DICOM, 2012) for distributing and viewing medical images. In 2011, HL7 introduced the first draft of the Fast Healthcare Interoperability Resources (FHIR) specification designed to enable interoperability and integration with the newest and adopted technologies by the industry with a particular focus on making healthcare data in different EHRs and other HIT systems easily available to mHealth apps via RESTful APIs in the cloud. FHIR has a set of security requirements (FHIR, 2016) that identifies the major topics (communications security, authentication, authorization, access control, auditing, digital signatures, etc.). However, FHIR lacks concrete mechanisms that would be capable of controlling access to the services of RESTful APIs that manage sensitive healthcare data stored in the cloud.

1.4. A High-Level View of Proposed Approach

The overall high-level focus of this dissertation is to propose and realize a configurable framework for RBAC, MAC, and DAC for mobile applications that is capable of supporting access control in different security layers. Security is controlled from three perspectives. The first perspective is for the user interface in terms of which screens and/or their components are accessible to a user under RBAC with optional delegation via DAC. This security layer focuses on modifications to the UI. The second perspective is to control the mobile application's API services in order to define the API services that can be invoked by a particular user based on RBAC and/or MAC permissions with optional delegation via DAC. This security layer between the UI and mobile application API replicates the mobile application's API by creating a mirrored set of services that invoke the original API services so that each call can be intercepted to add RBAC, MAC, and or/

DAC security checks. The third perspective focuses on interactions between the services of the mobile application's API and server-side APIs for the various data servers, to again control whether the user via the mobile application service is authorized to invoke specific server-side APIs by RBAC and/or MAC with optional delegation. This security layer between the two different APIs (mobile app and server-side) is accomplished through the creation of a server interceptor API associated with a cloud computing infrastructure to intercept invocations for RBAC, MAC, and DAC checks.

In support of the service level security in the second and third perspectives, we evolve RBAC and MAC from permissions on objects/operations to an approach that can control the services of an API that are available for usage, allowing each service to be controlled by role (can a user by role access a service) or by sensitivity level (does a user with a clearance level has the necessary permission to access a service with a classification level). The intent is that for any mHealth app that needs to securely utilize services from multiple HITs, each with their own specific RESTful API, will be controlled so that the user is only allowed to invoke services to which they have permission. For example, a mHealth app for patient data works differently depending on who is using the app. A patient is able to invoke services to read all of his/her patient data, to invoke some services that update basic information (e.g., demographics), but would be restricted from invoking services that order medications or laboratory tests. These latter service invocations would be appropriate for medical providers (e.g., physician, nurse, etc.) that have the authority to change a patient's medical record. Then, we can define a user (with a clearance level) with a role (with permissions to invoke particular services) where each service has a classification level, and have runtime enforcement to insure that a user with a particular role is allowed to invoke a service within the mobile app if the RBAC and/or MAC permissions are met.

In support of RBAC, MAC, and/or DAC at the three perspectives of UI, mobile application API, and server-side API, Figure 1.3 delineates the different ways that security layers can be incorporated within the mobile application's user interface (UI), application programming interface (API), and server-side data source (database, cloud, server, etc.), where permissions can be defined and enforced. The three locations are: the user interface to change the look-and-feel by role; intercepting API calls to alter information delivered/stored to the app; and/or by modifying the mobile app server. This leads to three corresponding options for the inclusion of RBAC, MAC, and DAC corresponding to the aforementioned three perspectives. The first option (perspective), *direct UI modifications*, shown in the left side of Figure 1.3, would be to modify the mobile app itself with RBAC, DAC, and MAC permissions on screens, UI widgets, etc., which would involve code-level changes so that the look-and-feel of the UI would change based on the defined access control security policies. The second option (perspective), *intercepting API calls*, shown in the middle of Figure 1.3, would be to define RBAC, MAC, and DAC permissions on the API (REST, web, cloud) and/or database calls of the mobile app and intercept them in order to include access control permission checks that determine the filtered information returned to the mobile app or control information that can be stored in the mobile application's server. This may require minimal changes on the way that the mobile app calls the backend or the way that the backend calls are intercepted by the access control code. Finally, the third option (perspective), *server interceptor API*, shown in the right side of Figure 1.3, involves making changes to the backend of the original mobile app (e.g., server for database, cloud, web, etc.) in support of RBAC, MAC, and DAC enforcement that would retain the view of the mobile application's API to the mobile app and embed access control policies on the server side. The end result for each of these options (perspectives) is a revised mobile application, a revised mobile application API, and a revised server interceptor API, respectively, as shown in the bottom box of Figure 1.3.

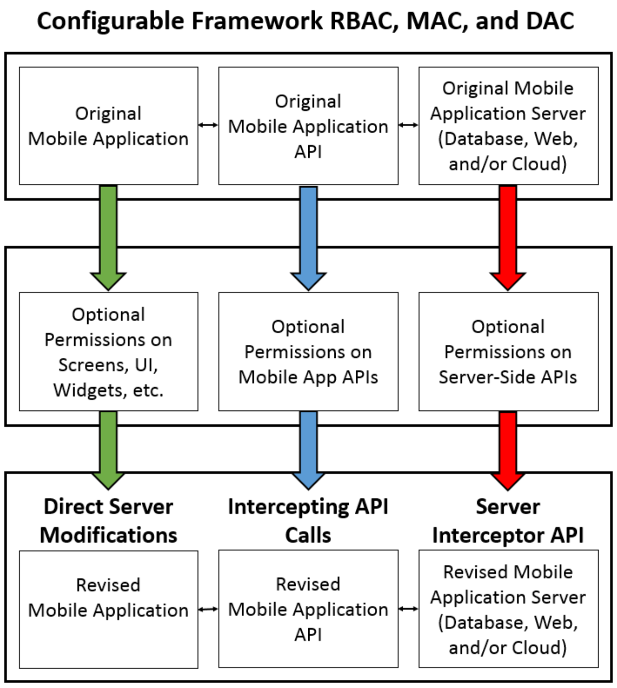

Configurable Framework RBAC, MAC, and DAC
Original Mobile Application
Original Mobile Application API
Original Mobile Application Server (Database, Web, and/or Cloud)
Optional Permissions on Screens, UI, Widgets, etc.
Optional Permissions on Mobile App APIs
Optional Permissions on Server-Side APIs
Direct Server Modifications
Intercepting API Calls
Server Interceptor API
Revised Mobile Application
Revised Mobile Application API
Revised Mobile Application Server (Database, Web, and/or Cloud)

Figure 1.3. Permissions and Three Options for Mobile Security.

Figure 1.4 presents a high-level view of the architecture of the configuration framework, including the concepts of Figure 1.3 into this larger context. There are six major components (outlined black boxes in Figure 1.4). Basically, the *Mobile Application* (topmost component in Figure 1.4) consists of a UI, an API, and a data source (database, cloud, server, etc.). The second component in Figure 1.4, *Mobile Application Clients*, contains a set of users where each user is assigned a clearance (e.g., top secret, secret, confidential, unclassified). In order to determine which resources of the mobile application each user is allowed to access, the third component of Figure 1.4, *Access Control Models* is utilized to allow the assignment to each user of: a role (RBAC approach), a clearance (MAC), or a role (RBAC) and a clearance (MAC extension) in combination; DAC may be optionally included. Access control is defined to involve the mobile application main structure of UI, API, and Data Sources as was shown in the first component. These models are realized against the mobile application user interface, API, and data sources, which allow RBAC, MAC, and/or DAC to be defined on: the screens and components of the UI to control who can do what; the services of the API to control which are secure and which can be called; and the data sources to control the information that can be read/written. This requires the definition of a *unified mobile computing and security model* to define and enforce permissions in complementary and combinable ways.

For the application itself, the fourth component of Figure 1.4, *Permissions and Impact on Mobile App*, contains the permissions on: the UI components of a mobile application, the APIs of a mobile application, and/or the data source/server side. Permissions involve: the look-and-feel of the UI per the allowable screens, their components, and interactions; the ability to involve different services of the API; and, the ability to control access to the data sources. These permissions can be based on a combination of RBAC, MAC, and/or DAC. Assigning roles and clearances to users in a system as well as identifying the permissions that are generated for certain parts of a mobile application is part of creating a unified mobile computing and security model with access control. Bringing together the mobile application's UI, API, and data sources (first component), its clients (second component), and the access control models RBAC, MAC, and DAC (third component) allows for the realization of permissions for the three different options shown in Figure 1.3: *direct UI modifications*, *intercepting API calls*, and *server interceptor API*.

Specifically, the first block in the fourth component, *user interface permissions/direct UI modifications*, provides the ability to define permissions to modify the existing mobile application itself with access control permissions (both RBAC and MAC) on screens, UI widgets, etc., which would also involve code-level changes so that the look-and-feel of the UI would change based on the enforcement of access control policies. Permissions can be defined for a user by role on a UI screen, its components (text fields, drop down, buttons, etc.), and interactions among screens, which may optionally include classification levels (MAC) as well. This can be accomplished through a human-assisted process that outlines the way that mobile application code changes are made. The second block in the fourth component, *API permissions/intercepting API calls*, shown in the middle of the fourth component of Figure 1.4, would be to define access control permissions on the API (REST, web, cloud) and/or database calls of the mobile application and intercept them in order to include access control permission checks that determine the filtered information returned to the mobile application or control information that can be stored in the mobile application's server. Permissions are defined on the mobile application API that is partitioned into secure/unsecure services (RBAC) and labeled/unlabeled services (MAC), with service permission assignment to roles and users. This can be accomplished through the automatic generation of security code. Finally, the third block of the fourth component, *data sources/server interceptor*, involves making changes to the backend or data source(s) of the existing mobile application (e.g., server for database, cloud, web, etc.) that would retain the view of the mobile application's API to the mobile application and embed access control policies on the server side. This can also be accomplished through the automatic generation of security code.

Once the permissions are defined, we can generate access control security policies as shown in the fifth component of Figure 1.4 by combining different aspects of access control models in the components of a mobile application by utilizing a single option or a combination of the options presented in Figure 1.3. This is further discussed in Section 3.2 that covers all the allowable combinations of: access control models (RBAC, MAC, and DAC), the mobile application (UI, API, and data sources), and the options (direct UI modifications, intercepting API calls, and server interceptor API). Collectively, the human-assisted processes and algorithms to automatically generate code results in the fifth component of Figure 1.4, generation of security policies at the UI or between the UI and API or between the API and data sources. The proposed configuration framework allows for the generation of different combinations based on access control models (RBAC, MAC, and DAC), the mobile application (UI, API, and data sources), and the three options (direct UI modifications, intercepting API calls, and server interceptor API). For example, a mobile application for a pharmacy may only have RBAC, UI, and direct UI modifications, while a more complex patient data mobile health app for medical providers (nurse, physician, etc.) may use all capabilities in combination. After the policies are established, we can enforce the security policies (sixth component of Figure 1.4) in the different portions identified of a mobile application which lead us to an end result of a customized mobile application. The final sixth component, *enforcement of security policies*, at the bottom of Figure 1.4, is the resulting enforcement code from the human-assisted processes and automatic algorithm. For example, the modified code of the pharmacy app with RBAC, UI, and direct UI modifications.

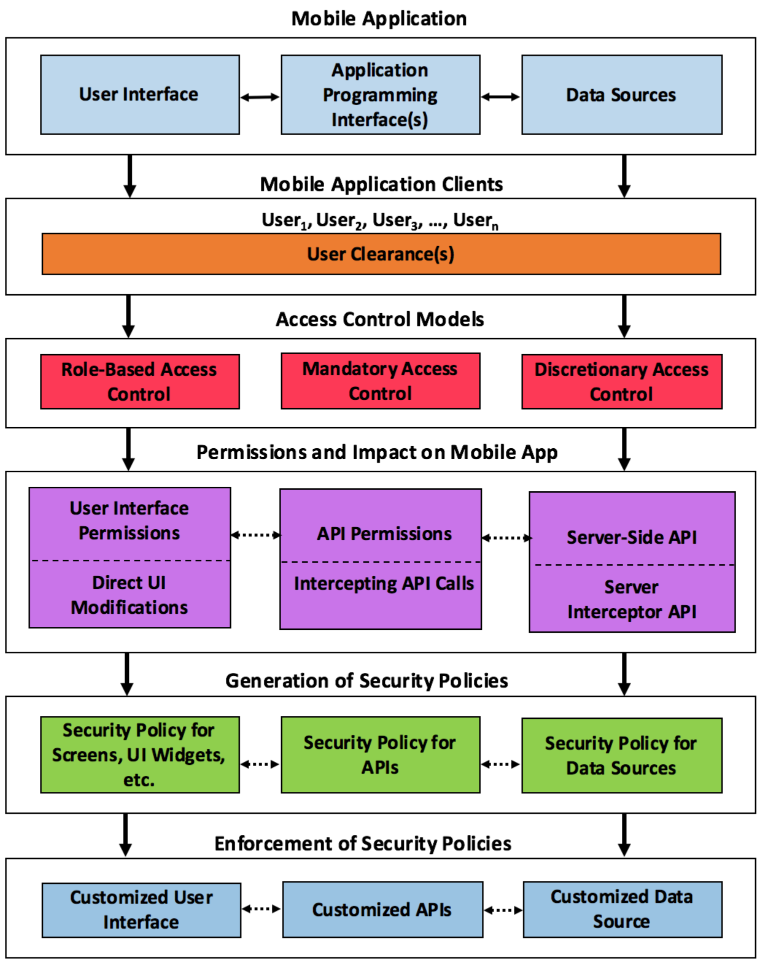

Mobile Application
User Interface
Application Programming Interface(s)
Data Sources
Mobile Application Clients
User$_1$, User$_2$, User$_3$, ..., User$_n$
User Clearance(s)
Access Control Models
Role-Based Access Control
Mandatory Access Control
Discretionary Access Control
Permissions and Impact on Mobile App
User Interface Permissions
Direct UI Modifications
API Permissions
Intercepting API Calls
Server-Side API
Server Interceptor API
Generation of Security Policies
Security Policy for Screens, UI Widgets, etc.
Security Policy for APIs
Security Policy for Data Sources
Enforcement of Security Policies
Customized User Interface
Customized APIs
Customized Data Source

Figure 1.4. High-Level View of Configurable Access Control Framework for Mobile Apps.

1.5. Research Objectives and Expected Contributions

From the research perspective, the proposed Configurable Framework for RBAC, MAC, and DAC for Mobile Applications has the following expected contributions.

A. Software Architecture for a Configurable Access Control Framework for Mobile Applications: The contribution is the specification, design, and description of a software architecture for the configurable access control framework as given in Figure 1.4.

This allows the ability to insert role-based, mandatory, and discretionary access controls at alternate and multiple locations throughout the architecture.

B. Unified Mobile Computing and Security Model with Access Control: The contribution is a unified model (the first three components of Figure 1.4) that contains: generalized structure of a mobile application as a user interface of screens, components (text fields, drop down, buttons, etc.), and interactions among screens; roles, sets of roles, users, and sets of users; allowable permissions defined on screens, components, and screen interactions which includes classification levels (MAC); permission assignments of users and roles on screens, components, and interactions; mobile application API that is partitioned into secure/unsecure services (RBAC) and labeled/unlabeled services (MAC); and service permission assignment to roles and users. This allows the ability to model role-based, mandatory, and discretionary access controls on the mobile application and its API.

C. Dynamic Combination of Access Control Models and Configuration Options: The contribution is the ability to combine different aspects of access control models (RBAC, MAC, and DAC), of the mobile application (UI, API, and Data Source), and of the configuration options (Direct UI Modifications, Intercepting API Calls, and Server Interceptor API) into custom access control solutions for a mobile application. All of the allowable combinations are defined as part of this contribution as shown in the third and fourth components of Figure 1.4.

D. Access Control Security Enforcement Code Generation and Interceptors: The contribution is the generation of processes for the Direct UI Modifications option and algorithms for the different configuration options for the framework that support the interceptors for the Intercepting API Calls, and Server Interceptor API options. Processes for the Direct UI Modifications are often human assisted and involve the need to actually modify limited portions of the mobile application code, API, and/or server database. Algorithms for the Intercepting API Calls, and Server Interceptor API options are defined for those cases where actual code is generated. This is part of the fifth and sixth components in Figure 1.4.

Throughout the remainder of the dissertation, these expected contributions are high-lighted when relevant.

1.6. Research Progress to Date

In support of the work presented in this dissertation, we summarize our 10 publications (8 published and 2 submitted) and their role in support of the material in this dissertation: lead author directly related to the work are 2 published refereed book chapters, 2 published referred full conference articles, and 1 submitted journal article; coauthor of 1 published refereed book chapter and 1 submitted refereed full conference article; and co-author of three other papers as an REU student. First in this area initially focused on authentication requirements for mobile apps (Rivera Sánchez & Demurjian, 2016) that was expanded to define an approach for role-based access control (RBAC) for mobile computing (Rivera Sánchez et al., 2016) that delineates security permissions on the screens and components of a mobile app that customizes both the appearance and functionality based on role which is the foundation of the Direct UI Modifications option.

- **Rivera Sánchez, Y. K.**, & Demurjian, S. A. (2016). Chapter 6: User Authentication Requirements for Mobile Computing. *Handbook of Research on Innovations in Access Control and Management*. IGI Global.

- **Rivera Sánchez, Y. K.**, & Demurjian, S. A., Conover, J., Agresta, T., Shao, X., & Diamond, M. (2016). Chapter 6: An Approach for Role-Based Access Control in Mobile Applications. *Handbook of Mobile Application Development, Usability, and Security*. S. Mukherja (ed.). IGI Global.

Using this as a basis, we expanded RBAC in (Rivera Sánchez, Demurjian, & Gnirke, 2017) to control the services that are accessible by role for each mobile application with intercepting API calls to check permissions before a service can be invoked; this is the foundation of the Intercepting API Calls option. This was generalized in (Rivera Sánchez, Demurjian, and Baihan, 2017a) to apply to services in a cloud computing setting using the FHIR standard and RESTful APIs that facilitate ease of mobile health application development. The combination of RBAC and MAC using the interceptor concepts (Rivera Sánchez, Demurjian, & Baihan, 2017b) and the FHIR standard and its infrastructure allows a mHealth app that is FHIR-compliant to exchange healthcare data that is in the cloud with multiple EHRs/HIT systems. This is achieved by defining on a role-by-role basis and/or on a sensitivity level basis (i.e., classification-clearance), the subset of the FHIR RESTful API services that are available to users of a mHealth app, to intercept calls that are not allowed, thereby prohibiting access to the sensitive healthcare data associated with those calls; these last two are the foundation of the server interceptor API option.

- **Rivera Sánchez, Y. K.**, Demurjian, S.A., & Gnirke, L. (2017). An Intercepting API-based Access Control Approach for Mobile Applications. In *Proceedings of The 13th International Conference on Web Information Systems and Technologies (WEBIST 2017).*

- **Rivera Sánchez, Y. K.**, Demurjian, S.A., & Baihan, M. (2017). Achieving RBAC on RESTful APIs for Mobile Apps using FHIR. In *Proceedings of The 5th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (IEEE Mobile Cloud 2017)*.
- **Rivera Sánchez, Y.K.**, Demurjian, S. A., & Baihan, M. S. (2017). A Service-Based RBAC & MAC Approach Incorporated into the Fast Healthcare Interoperable Resources (FHIR) standard. Submitted to special issue on 2017 IEEE Mobile Cloud Conference submissions, Elsevier journal of Digital Communications and Networks, https://www.journals.elsevier.com/digital-communications-and-networks/call-for-papers/special-issue-on-the-security-privacy-and-digital-forensics.

The final two efforts (one published and one submitted) involve collaboration on the shared HAPI FHIR Infrastructure to support RBAC and MAC interceptors.

- Baihan, M., **Rivera Sánchez, Y. K.**, Shao, X., Gilman, C., Demurjian, S. A., & Agresta, T. (2017). A Blueprint for Designing and Developing an mHealth Application for Diverse Stakeholders Utilizing Fast Healthcare Interoperability Resources. *IGI Global*.
- Baihan, M. S., Demurjian, S. A., Rivera Sánchez, Y. K., Toris, A., Franzis, A., Onofrio, A., Cheng, B., & Agresta, T. (2017). Role-Based Access Control for Cloud Computing Realized within HAPI FHIR. Submitted to *16th International Conference of WWWINTERNET 2017*, 18 – 20 October, Vilamoura Algarve, Portugal.

Other publications that I have participated in are:

- De La Rosa Algarin, A., Ziminski, T., Demurjian, S., & **Rivera Sánchez, Y. K.** (2014). Generating XACML Enforcement Policies for Role-Based Access Control of XML Documents. In *Web Information Systems and Technologies, Revised Selected Papers, Lecture Notes in Business Information Processing, Springer-Verlag*, *189*, 21-36.
- De La Rosa Algarin, A., Demurjian, S., Ziminski, T., **Rivera Sánchez, Y. K.**, & Kuykendall, R. (2013). Chapter 13: Securing XML with Role-Based Access Control: Case Study in Health Care. In *Architectures and Protocols for Secure Information Technology*. A. Ruiz-Martínez, F. Pereñíguez-García and R. Marín-López (eds.). IGI Global. 334-365.
- De La Rosa Algarin, A., Ziminski, T., Demurjian, S., Kuykendall, R., & **Rivera Sánchez, Y. K.** (2013). Defining and Enforcing XACML Role-Based Security Policies within an XML Security Framework. *Proceedings of the 9th Intl. Conf. on Web Information Systems and Technologies (WEBIST2013)*.

1.7. Dissertation Outline

The remainder of the dissertation has six chapters. In Chapter 2, we review background on: the logical architecture of a mobile application, Role-Based Access Control (RBAC) (Ferraiolo & Kuhn, 1992), Mandatory Access Control (MAC) (Bell & La Padula, 1976), Discretionary Access Control (DAC) (Department of Defense, 1985), the application programming interface (API) concept, and the Fast Healthcare Interoperable Resources (FHIR) specification (FHIR DSTU2, 2015) and the HAPI FHIR reference implementation (HAPI FHIR, 2014), Chapter 2 also introduces the Connecticut

Concussion Tracker (CT^2) to be utilized in examples throughout the dissertation. In Chapter 3, we primarily address Contribution B: Unified Mobile Computing and Security Model with Access Control by defining a formal model to represent: the general structure of a mobile application; RBAC concepts including roles, sets of roles, users, and sets of users; MAC concepts including classifications and clearances; permissions related to RBAC and MAC; permission assignment to the user interface and to control the services of the mobile application API; and, delegation for mobile applications. Chapter 3 also addresses Contribution C: Dynamic Combination of Access Control Models and Configuration Options to combine RBAC, MAC, and DAC, the mobile app's UI, API, and Data Source with the Direct UI Modifications, Intercepting API Calls, and Server Interceptor API options (see Section 1.3 and Figure 1.4). Chapters 4, 5, and 6 detail the realization of Contribution D: Access Control Security Enforcement Code Generation via both human-assisted processes that modify limited portions of the mobile application code and algorithms that automatically generate security enforcement code. Chapter 4 focuses on the *Direct UI Modifications* option which supports the inclusion of RBAC to control the screens, components, and interactions among screens on a role-by-role basis. Chapter 5 focuses on the *Intercepting API Calls* option on the interactions between the UI and the mobile applications' API services to control by both RBAC and/or MAC permissions which services are allowed to be invoked for on a user-by-user basis through the generation of an intercepting API that mirrors the original mobile application's API. Chapter 6 focuses on the *Server Interceptor API* option on the interactions between the mobile application's API services and their invocations to server-side APIs of data sources, with a server interceptor API defined using the HAPI FHIR reference implementation. Finally, Chapter

7 summarizes the main points discussed throughout the dissertation and what was achieved with the proposed approach.

Chapter 2
Background

This chapter provides background information on the main concepts and topics that support the discussion and explanation in the remainder of the dissertation in seven sections. We have chosen the healthcare domain to support the explanation of our research since healthcare data is highly sensitive, requires fine-grained security, and involves multiple stakeholders. Section 2.1 presents the logical architecture of a mobile application through a description of its different layers and their interaction. Section 2.2, 2.3, and 2.4 review, respectively, the three access control models that are the basis for our work: Role-Based Access Control (RBAC) (Ferraiolo & Kuhn, 1992), Mandatory Access Control (MAC) (Bell & La Padula, 1976), and Discretionary Access Control (DAC) (Department of Defense, 1985). Section 2.5 reviews the application programming interface (API) concept that is instrumental in our approach that necessitates permissions based on which user is authorized to which API service call. Section 2.6 introduces and explains the Fast Healthcare Interoperable Resources (FHIR) specification (FHIR DSTU2, 2015) and the HAPI FHIR reference implementation (HAPI FHIR, 2014), both of which are utilized to support the proof-of-concept discussion in Chapter 6. Section 2.7 introduces and reviews the Connecticut Concussion Tracker (CT^2) mobile application, a collaboration between the Departments of Physiology and Neurobiology, and Computer Science & Engineering at the University of Connecticut and Schools of Nursing and Medicine in support of a new law passed to track concussions of children from kindergarten through high school in public schools (CT Law HB6722) (Connecticut General Assembly, 2015).

2.1. Logical Architecture of a Mobile Application

In this section, the logical architecture of a client mobile application is explored, as shown in Figure 2.1 (Microsoft Corporation, 2008). The architecture consists of four main layers: the *User Layer* which symbolizes the users of the mobile application; the *Presentation Layer* which consists of the UI components of the mobile application; the *Business Layer* which contains the logic of the mobile application (e.g., libraries, APIs, source code); and, the *Data Layer* which contains all of the data the mobile application manages (e.g., retrieves, inserts). For the purposes of our work in this dissertation, the logical architecture in Figure 2.1 can be organized as a set of higher-level mobile application components, namely: the user interface (UI), the application programming interface (API), and the data source (database, cloud, server, etc.). This was shown in the first component in Figure 1.4 of Chapter 1. The intent of the work presented in this dissertation is to explore the inclusion of security within and between the layers of Figure 2.1 as realized in the mobile application as given in Figure 1.3 of Chapter 1.

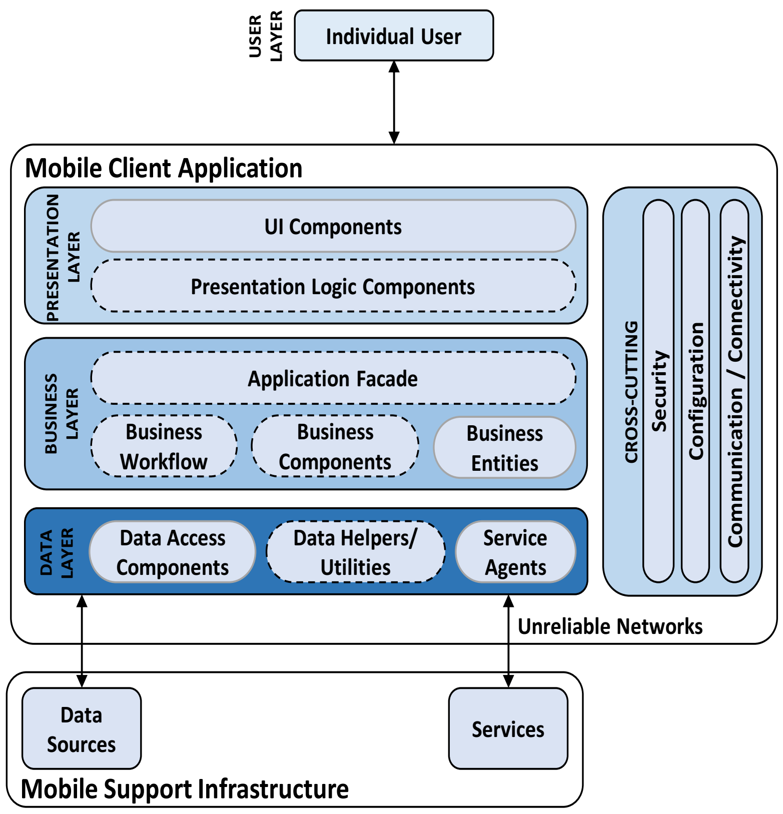


Figure 2.1. Logical Architecture of a Client Mobile Application.

2.2. Role-Based Access Control

Role-Based Access Control (RBAC) was proposed by David Ferraiolo and Richard Kuhn (Ferraiolo & Kuhn, 1992) and transitioned to the National Institute for Standards and Technology (NIST) (Ferraiolo, Sandhu, Gavrila, Kuhn, & Chandramouli, 2001) that was adopted in 2004. The main concepts of the NIST RBAC standard (Sandhu, Ferraiolo, & Kuhn, 2000) are conceptualized in Figure 2.2 (SlideShare, 2012) with four reference models. $RBAC_0$ in the middle portion of Figure 2.2 is comprised of: *users* that perform a specific function within an organization, *roles* that are assigned to users based on their responsibilities, and *permissions* (PRMS) that define which operations (OPS)/objects (OBS) within a system/application a role can have access to. Users can have one or more roles, and roles can contain one or more permissions to objects. $RBAC_1$, shown in the upper middle portion of Figure 2.2, supports the ability of roles to be organized in a hierarchy. $RBAC_2$, shown in the upper left and on the lower right of Figure 2.2, provides the definition of constraints, such as static (SSD) and dynamic (DSD) separation of duty. Lastly, $RBAC_3$, shown at the bottom part of Figure 2.2, captures the concept of sessions that represent the lifetime of a particular user, role, and permissions in a dynamic runtime application.

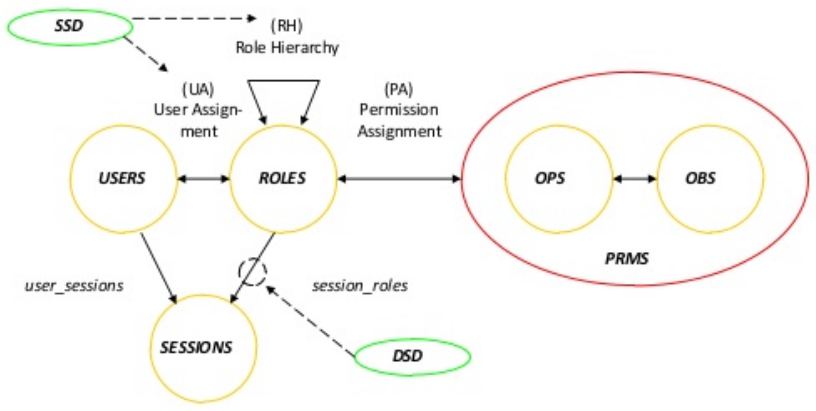


Figure 2.2. General structure of the RBAC model (Slideshare, 2012).

The NIST RBAC standard (Sandhu, Ferraiolo, & Kuhn, 2000) controls the access to system resources based on the roles available in an enterprise that a user can assume. Each role contains different capabilities that allow a user with a particular role to complete his/her tasks within the enterprise and nothing more (Rouse, 2006). Three key concepts are utilized to complete this process (NIST Computer Security Resource Center, 2015): role assignment allocates a role to a user based on what he/she is allowed to see, with each user of the system having at least one role and being able to connect with one role per opened session; role authorization to make sure the user was assigned the role necessary for him/her to complete his/her tasks and nothing more; and transaction authorization where a user can carry out a task if his/her role has permission to do so. Note that in RBAC, permissions as defined are operations on objects, and for the purposes of this dissertation, this must be evolved so that permissions can be defined on services that are invoked (called on objects).

2.3. Mandatory Access Control

Different from RBAC, in the Mandatory Access Control (MAC) model (Bell & La Padula, 1976; Biba, 1977), a security administrator assigns sensitivity levels (Top Secret (TS), Secret (S), Confidential (C), and Unclassified (U)) to objects (classifications) and users (clearances) to control who can see what. As with RBAC, the permissions are defined on objects and the allowable access modes/operations (e.g., read, append, write, or read-write). These levels are ordered hierarchically from most to least secure: TS > S > C > U. Basically, each of the users in a system has a clearance; therefore, if the user has a clearance of secret this means that he/she is allowed to access the files/programs that are either secret, confidential, or unclassified. Notice that he/she can't access the files/programs that are classified as top secret since he/she has a lower clearance status. An example of this model can be seen in Figure 2.3. In the example we have two users: *user A* that has a clearance of secret and *user B* has a clearance of top secret. In this case, both users are trying to access a data server that is classified as top secret, therefore the attempt to access the data for *user A* is not successful while *user B* can retrieve the data successfully.

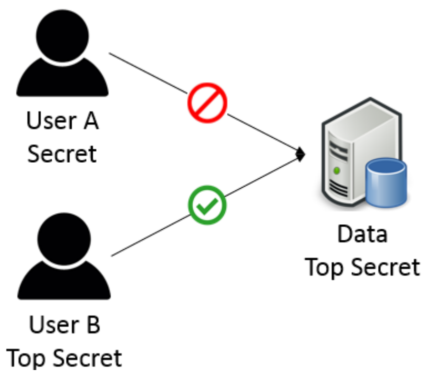


Figure 2.3. A MAC Example.

The allowable interaction of a user with a clearance to an object with a classification are governed by a set of security properties presented in the Bell-LaPadula model (Bell & LaPadula, 1976) and in the Biba model (Biba, 1977): simple security, simple integrity, liberal star, and strict star. The security properties are evaluated based on the access mode/operations (read, append, write, read-write). *Simple security (SS)* allows a user to read elements (read or read-write access mode) with a sensitivity level equal to or lower than their clearance level, but not those elements with a higher sensitivity level; *Simple integrity (SI)* allows a user to write elements (append, write, read-write access mode) of equal or lower sensitivity level when compared to their clearance level, but not to those elements with a higher sensitivity; *Liberal star (LS)* is the permission to write to equal or greater levels; and *Strict Star Write (SSW*) and *Strict Star Read (SSR),* or write (read) equal, is the permission to write (read) only to equal levels. Note also as similar to RBAC, permissions for MAC need to be upgraded from objects and operations to services and invocations.

2.4. Discretionary Access Control

The Discretionary Access Control (DAC) model (Department of Defense, 1985; Sandhu & Samarati, 1994) establishes security policies (e.g., read, write, execute permissions) based on a combination of the objects and on the user's identity and authorization and can be delegated. In other words, a user has the ability to assign permissions on the data that belongs to him/her. To illustrate, the example in Figure 2.4 has the case where User A owns the three files shown and gives read/write permissions to User B and User C for each of the files. In a healthcare setting, a physician often sees patients in his/her office weekdays (9am-5pm) with an on-call physician handling calls from patients at nights and on weekends. In this case, DAC can allow the physician to delegate his/her responsibilities (some or all) to the on-call physician who would not normally have access to those patients for a fixed period of time.

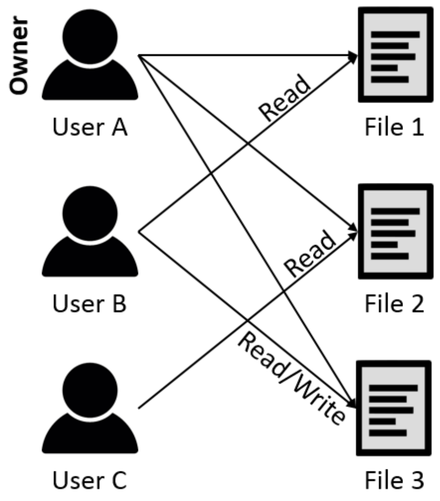


Figure 2.4. A DAC Example.

There exist several alternatives on the way that the owner of the information can grant access to other users (Osborn, Sandhu, & Munawer, 2000). Two of the alternatives that apply to our approach in this dissertation are:

- **Strict DAC:** The owner of an object is the only one who has authorization to grant access of the object to another user and this ownership cannot be transferred.
- **Liberal DAC:** The owner of an object can delegate his/her access to the object with other users. There are three alternatives of liberal DAC:
    - *One Level Grant*: The owner of an object can delegate his/her access to the object to another user, which does not have access to further delegate the permission.

- *Two Level Grant*: The owner of an object can delegate his/her access to the object to another user, which has access to further delegate the permission to another user but not further than this.
- *Multilevel Grant*: The owner of an object can delegate his/her access to the object to another user, this user can delegate the permission to another user, and so on indefinitely.

Again, DAC concepts need to be upgraded in support of this dissertation for the ability to delegate services that can be invoked.

2.5. Application Programming Interface

In order to perform data transactions between a server/database and a mobile application, many developers utilize the Application Programming Interface (API) concept. This consists of different tools or libraries utilized to interface data to an application. Figure 2.5 illustrates a general idea of the way that an API connects a data source to an application. Basically, the client sends a request through the means of a URL, the API receives the URL and interprets it, and then sends this to the data source. The data source then executes the request and sends back a response to the API. The API encodes the response in a human-readable format (e.g., JSON, XML) and sends the response in this format to the client. An example of a JSON response is:

```
[{"state_id":"1","state_name":"Connecticut","state_code":"CT"}]
```

Some of the advantages of APIs is that one API can be utilized in several applications as most of them are modular (e.g., Facebook Graph API) and, they demonstrate their usefulness in applications that contain dynamic data (data that changes in a frequent manner). This concept originated with traditional desktop devices and is now being heavily utilized in mobile applications.

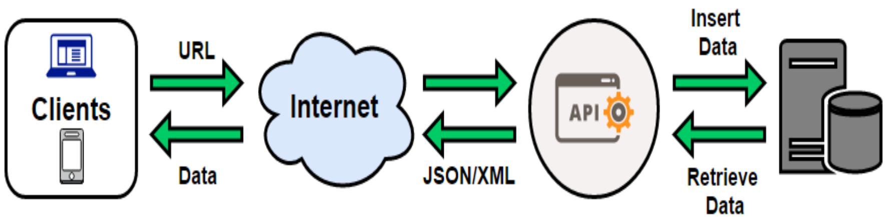


Figure 2.5. General Idea of how APIs work.

One of the most widely utilized architectures to build APIs is REST (Representational State Transfer). A web service that utilizes the REST architecture to build the APIs is referred to as RESTful APIs. The REST architecture usually runs over HTTP and is commonly utilized between the client and the server of a mobile application to manage the requests and responses between both (Rouse, 2014). There are several architectural constraints that characterize REST (WhatIsREST.com, 2012):

- **Client-Server:** A web service needs to have a separation of concerns, which consists of having an unambiguous separation between the consumers of a system (who can request services) and the services the system provides (that returns consumers a response to their request).
- **Stateless:** Each request a user sends to a service must contain all of the necessary information for the service to return a response.
- **Cache:** The client, service, or middleware are able to cache the response for reuse in later requests.
- **Interface/Uniform Contract:** Services and consumers of these services must share a technical interface (e.g., HTTP).

- **Layered System:** The web service must leverage a layered system. This means that the interactions between a consumer and a service must remain consistent regardless of the layer a consumer is communicating with.

Nevertheless, even though the constraints stated above are said to be required in order for a web service to be considered as using a REST architecture, a great amount of services lack at least one of these constrains, therefore, there are not many that are considered to be fully utilizing the REST architecture (Bleigh, 2010).

2.6. Fast Healthcare Interoperability Resources (FHIR)

The Fast Healthcare Interoperable Resources (FHIR) specification (FHIR DSTU2, 2015) is a standards framework created by the health language seven (HL7) organization (HL7, 2011) with the intention of providing easier and quicker implementation of interoperability in healthcare systems to facilitate access of mHealth apps to healthcare data in the cloud as stored in multiple EHRs/HIT systems. One of the main goals of FHIR is to represent the entities and procedures in healthcare as resources (FHIR DSTU2, 2015). There are currently ninety-three resources that can be utilized to map data from a healthcare system and the implementers of these resources claim that more resources are going to be added in a future (FHIR Resources, 2015). Sample resources include: for patients, Patient, FamilyMemberHistory, Condition, Observation, Diagnostic Report, Medication, Immunization, AllergyIntolerance, AdverseEvent, etc.; and, for insurance, Coverage, EligibilityRequest, Claim, PaymentNotice, etc. The available resources can be accessed through the means of a RESTful API, which allows to connect healthcare interfaces with data sources that exist in the cloud. Different from SOAP (Simple Object Access Protocol) (W3C, 2007), which has been the dominant approach to manage web services interfaces over the past years and it is utilized in HL7 v2, RESTful APIs are easier to understand and to implement as they rely on HTTP and on Create, Read, Update, and Delete (CRUD) operations to develop services.

One popular open-source library that implements FHIR specification is the HAPI FHIR reference implementation (HAPI FHIR, 2014). HAPI FHIR was developed in the Java programming language and offers the features of FHIR in addition to other features such as the ability to intercept the server (by using Java servlets (Java, 2013)) that processes the user's requests. HAPI FHIR offers several server interceptor functions (depicted in Figure 2.6) (HAPI FHIR Server Interceptors, 2016) that allow developers to perform actions on the user's request before it is executed and after its execution (before the response is delivered to the user). The main interceptors the library provides are: the *incomingRequestPreProcessed* interceptor which can be called before the request is processed; the *incomingRequestPostProcessed* interceptor which can be called once the request is classified (URL and request headers are examined in order to know this); the *incomingRequestPreHandled* interceptor which can be called once the request has been handled; and, the *outgoingResponse* interceptor, which can be called after the operation is handled but before the response is returned. This intercepting feature is critical to support our interception of RESTful API calls in order to check access control security permissions to prevent unauthorized access to services that have not been assigned to a given role/clearance/delegate for a mHealth app.

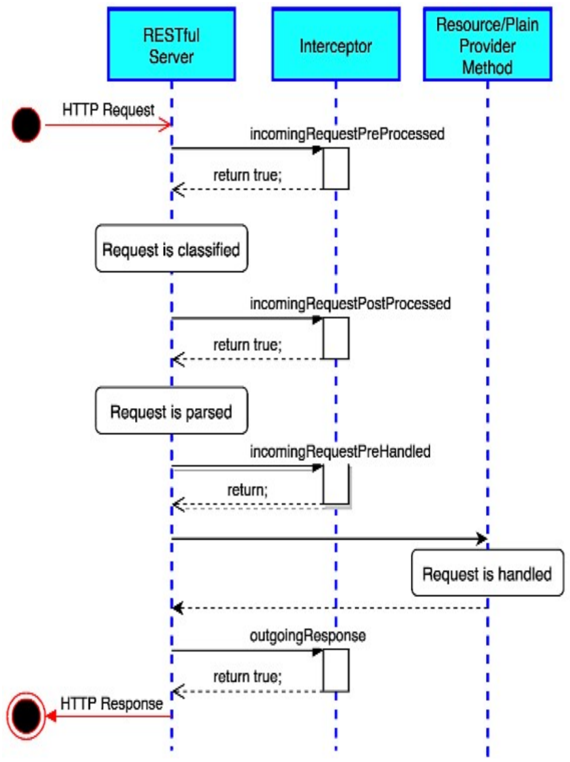


Figure 2.6. HAPI FHIR Server Interceptors

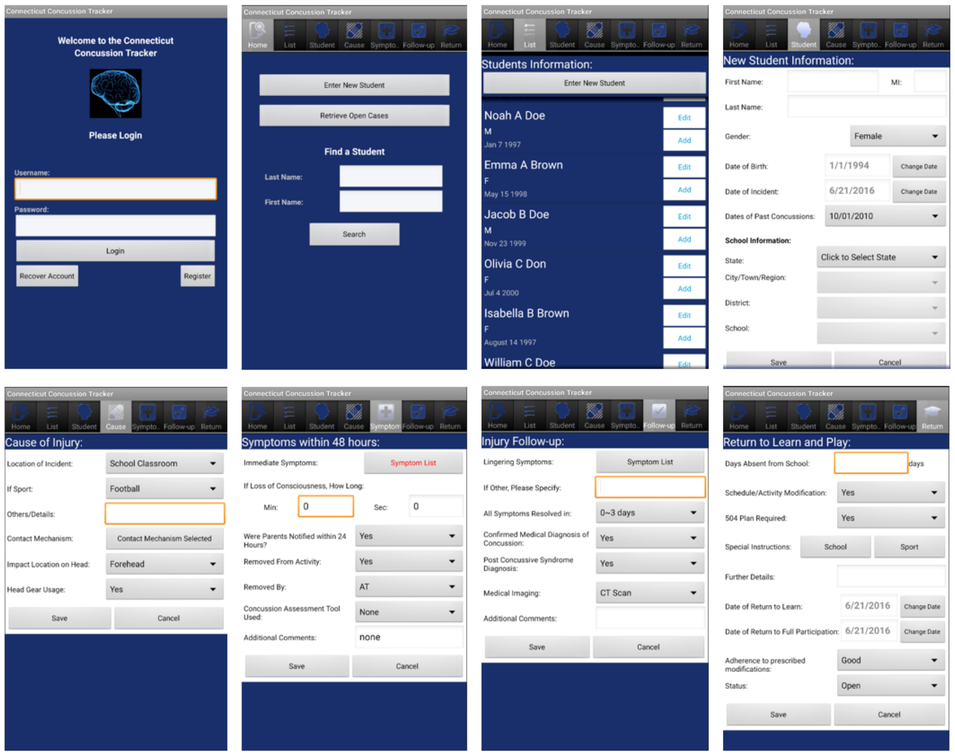
(HAPI FHIR Server Interceptors, 2016).

2.7. Connecticut Concussion Tracker (CT^2) Prototype

To evaluate and demonstrate the proposed approach, we utilize the Connecticut Concussion Tracker (CT^2) mobile application throughout the dissertation. CT^2 is being developed for both Android (Figure 2.7) and iOS (Figure 2.8) platforms and is a collaboration between the Departments of Physiology and Neurobiology, and Computer Science & Engineering at the University of Connecticut and Schools of Nursing and Medicine in support of a new law passed in the state of Connecticut to track concussions of kids between ages 7 to age 19 in public schools (CT Law HB6722) (Connecticut General Assembly, 2015). As shown in Figure 2.7, the Android version of the CT^2 mobile application consists of a UI of 7 tabs after the initial screen: 'Home', 'List', 'Student', Cause', 'Symptoms', 'Follow-up', and 'Return'. As shown in Figure 2.8, the iOS version of the CT^2 mobile application consists of a UI of 10 screens in which 5 of those are linked with navigation actions (left swipe and right swipe to move through the aforementioned five screens): 'Student', 'Cause', 'Symptoms', 'Follow-up', and 'Return' respectively, in the bottom row of the figure. Briefly, we explain each of the 10 screens of the iOS version in Figure 2.8; note that common screens of the Android version have the same functionality. The 'Login' screen (second screenshot, top row) allows users to log in to the application with a valid username/password combination. The 'Registration' screen (third screenshot, top row) allows future users to create an account to be able to use the application. The 'User Information' screen (fifth screenshot, top row), which is accessible by pressing the 'Home' button found at the top of the List screen, allows the user to manage his/her general information. The 'List' screen (fourth screen, top row) allows users to add a new concussion; to search a student by name; and, contains the list of students the user has permission to view and, for each student gives him/her the option to add a concussion, edit the existing concussion, share the concussion, or close the concussion case (if he/she has permission to access these components). The 'Student' screen, first screenshot in the bottom row, allows the user to input the student's general information (e.g., name, birthdate, school). The 'Cause' screen, the second screenshot of the bottom row, allows the user to insert the date that the incident occurred and, allows him/her via drop down options to specify where the injury was caused, with what it was caused, etc. After the user saves the data he/she entered in the 'Cause' screen, he/she can proceed to the 'Symptoms' screen (third screenshot, bottom row), where the symptoms the student had within 48 hours and other pertinent data are entered. To finish, the 'Follow-up' and 'Return' (fourth and fifth screenshot, respectively, bottom row) screens allow users to record the status of the student over time (Follow-up) and when the student can return to various activities at school (Return).

To illustrate the processing within CT^2, Figure 2.9 depicts the general structure of the mobile application and components. The CT^2 mobile app allows users to manage students' concussion data from when the student got the concussion up to its resolution (when the case is closed). The two boxes depicted on the top of Figure 2.9 represent the databases utilized to store the data the mobile app manages. The CT^2 mobile app DB contains user account information, school data (school's name and location), and other non-sensitive data such as drop-downs, select lists, etc., in the mobile app (e.g., location of incident, state, assessment tool used). Different from the CT^2 DB, the OpenEMR DB stores highly-sensitive data that is managed through the means of the CT^2 mobile app. This includes: students' demographic data such as name and date of birth; concussion data such as date of incident, contact mechanism, and symptoms; follow-up data such as lingering symptoms, medical diagnosis, and post concussive syndrome diagnosis; and, return data such as schedule modification, details of return, and date of returning to full participation. In addition to the data sources, the CT^2 mobile app utilizes an API in order to retrieve/insert/update data as depicted in the *CT^2 Mobile Application API* box in Figure 2.9. A subset of the services available in the CT^2 API allow users to: view/insert/edit students' demographic data; manage their account information (name, password); search for an specific student; and manage concussion, follow-up, and return data. While the data that is retrieved/inserted from/to the CT^2 DB is managed solely by the CT^2 API, the data that is retrieved/inserted from/to the OpenEMR DB requires an API (depicted in the

OpenEMR API box in Figure 2.9) that is positioned between the CT^2 API and the OpenEMR DB, which contains the necessary services to handle the retrieval/insertion of demographic data, concussion data, follow-up data, and return data. Finally, the bottom box of Figure 2.9 indicates that there are two versions of the CT^2 mobile app (as shown in Figures 2.7 and 2.8): one for mobile devices that support Android OS and the other one for mobile devices that support iOS.



Students Information:

Noah A Doe, M, Jan 7 1997
Emma A Brown, F, May 15 1998
Jacob B Doe, M, Nov 23 1999
Olivia C Don, F, Jul 4 2000
Isabella B Brown, F, August 14 1997
William C Doe

New Student Information:

First Name:	
MI:	
Last Name:	
Gender:	Female
Date of Birth:	1/1/1994
Date of Incident:	6/21/2016
Dates of Past Concussions:	10/01/2010
School Information:	
State:	Click to Select State
City/Town/Region:	
District:	
School:	

Cause of Injury:

Location of Incident:	School Classroom
If Sport:	Football
Others/Details:	
Contact Mechanism:	Contact Mechanism Selected
Impact Location on Head:	Forehead
Head Gear Usage:	Yes

Symptoms within 48 hours:

Immediate Symptoms:	Symptom List
If Loss of Consciousness, How Long: Min:	0
Sec:	0
Were Parents Notified within 24 Hours?	Yes
Removed From Activity:	Yes
Removed By:	AT
Concussion Assessment Tool Used:	None
Additional Comments:	none

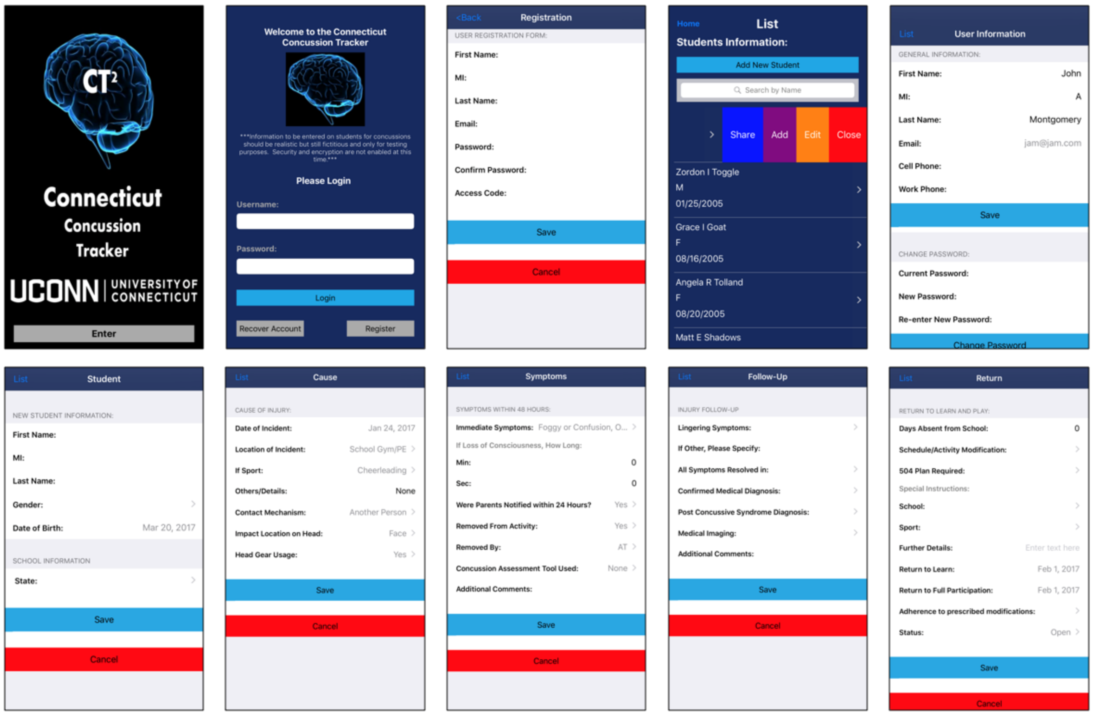
Injury Follow-up:

Lingering Symptoms:	Symptom List
If Other, Please Specify:	
All Symptoms Resolved in:	0~3 days
Confirmed Medical Diagnosis of Concussion:	Yes
Post Concussive Syndrome Diagnosis:	Yes
Medical Imaging:	CT Scan
Additional Comments:	

Return to Learn and Play:

Days Absent from School:	days
Schedule/Activity Modification:	Yes
504 Plan Required:	Yes
Special Instructions:	School / Sport
Further Details:	
Date of Return to Learn:	6/21/2016
Date of Return to Full Participation:	6/21/2016
Adherence to prescribed modifications:	Good
Status:	Open

Figure 2.7. CT^2 Mobile Application - Android Version.


CT2
Connecticut Concussion Tracker
UCONN | UNIVERSITY OF CONNECTICUT
Enter

Welcome to the Connecticut Concussion Tracker

Information to be entered on students for concussions should be realistic but still fictitious and only for testing purposes. Security and encryption are not enabled at this time.

Please Login

Username:

Password:

USER REGISTRATION FORM:

First Name:

MI:

Last Name:

Email:

Password:

Confirm Password:

Access Code:

Students Information:

Zordon I Toggle
M
01/25/2005

Grace I Goat
F
08/16/2005

Angela R Tolland
F
08/20/2005

Matt E Shadows

GENERAL INFORMATION:

First Name:	John
MI:	A
Last Name:	Montgomery
Email:	jam@jam.com
Cell Phone:	
Work Phone:	

CHANGE PASSWORD:

Current Password:

New Password:

Re-enter New Password:

NEW STUDENT INFORMATION:

First Name:	
MI:	
Last Name:	
Gender:	
Date of Birth:	Mar 20, 2017

SCHOOL INFORMATION

State:

CAUSE OF INJURY:

Date of Incident:	Jan 24, 2017
Location of Incident:	School Gym/PE
If Sport:	Cheerleading
Others/Details:	None
Contact Mechanism:	Another Person
Impact Location on Head:	Face
Head Gear Usage:	Yes

SYMPTOMS WITHIN 48 HOURS:

Immediate Symptoms:	Foggy or Confusion, O...
If Loss of Consciousness, How Long:	
Min:	0
Sec:	0
Were Parents Notified within 24 Hours?	Yes
Removed From Activity:	Yes
Removed By:	AT
Concussion Assessment Tool Used:	None
Additional Comments:	

INJURY FOLLOW-UP

Lingering Symptoms:

If Other, Please Specify:

All Symptoms Resolved in:

Confirmed Medical Diagnosis:

Post Concussive Syndrome Diagnosis:

Medical Imaging:

Additional Comments:

RETURN TO LEARN AND PLAY:

Days Absent from School:	0
Schedule/Activity Modification:	
504 Plan Required:	
Special Instructions:	
School:	
Sport:	
Further Details:	Enter text here
Return to Learn:	Feb 1, 2017
Return to Full Participation:	Feb 1, 2017
Adherence to prescribed modifications:	
Status:	Open

Figure 2.8. CT^2 Mobile Application - iOS Version.

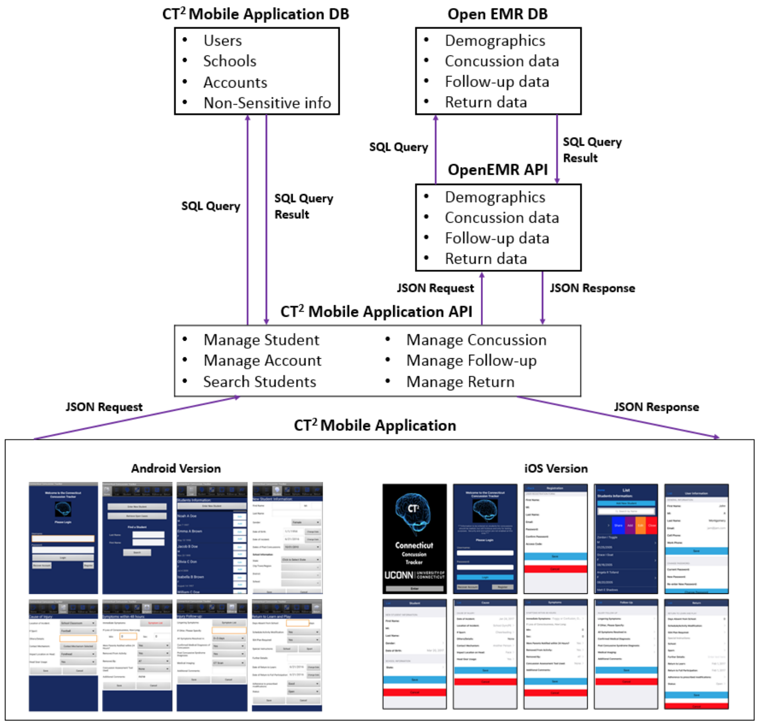


Figure 2.9. CT^2 Structure.

Chapter 3
Unified Model of Access Control for Mobile Applications

This chapter provides a detailed discussion of a unified model of access control for mobile applications in 7 sections. Specifically, we define a Unified Mobile Computing and Security Model with Access Control, contribution B in our work (see Section 1.5, Chapter 1), that is meant to capture the generalized structure of a mobile application, the different access control models (role-based access control - RBAC, mandatory access control - MAC, and discretionary access control - DAC), and the permissions that can be assigned at the user interface, API, and data source of the mobile application; the presentation is logically partitioned into groups. Section 3.1 introduces the first group of definitions for the generalized structure of a mobile application which consists of a user interface of screens, components (text fields, drop down, buttons, etc.), and interactions among screens. Section 3.2 reviews the second group of definitions for RBAC and MAC concepts including: roles, sets of roles, users, sets of users' clearances and classification for MAC. Section 3.3 presents the third group of definitions for RBAC permissions on the user interface, namely: permission assignments of users and roles on screens, components, and interactions (RBAC). Section 3.4 presents the fourth group of definitions for RBAC and MAC permissions on the mobile application API that is partitioned into secure/unsecure services (RBAC) and labeled/unlabeled services (MAC); and discusses service permission assignment to roles and users. Section 3.5 explores the fifth and final group of definitions for DAC that includes the delegation of permissions from one user/group to another user/group for RBAC permissions on the UI of a mobile application and RBAC and/or

MAC permissions on the services of the mobile application API. Collectively, the model presented in Sections 3.1 to 3.5, allows for the ability to model role-based, mandatory, and discretionary access controls on the mobile application and its API and supports contribution B: Unified Mobile Computing and Security Model with Access Control from Section 1.5 of Chapter 1. Section 3.6 discusses the ability to take the model concepts as given in Sections 3.1 to 3.5 and pick-and-choose in order to define and design a unique set of security capabilities for each mobile application; this supports contribution C: Dynamic Combination of Access Control Models and Configuration Options. Section 3.7 contains an entity relationship diagram to store information programmatically from the Unified Security model in Sections 3.1 to 3.5. Finally, Section 3.8 presents related work on access control in mobile computing.

3.1. Generalized Structure of Mobile Application

Table 3.1 contains assertions on mobile computing. The first assertion in Table 3.1 is that the mobile computing field is composed of mobile communication, mobile hardware, and mobile software; corresponding assertions are 2, 3, and 4; the focus of our work is on mobile software assuming that appropriate devices (hardware) exist and that communication between these and the servers and/or databases and/or repositories is secure. This is the fourth assertion in Table 3.1.

Mobile computing is composed by three main fields: mobile communication, mobile hardware, and mobile software.
Mobile communication consists of establishing an infrastructure to assure a seamless and reliable communication (e.g., the use of protocols, services, etc. to assist the communication).
Mobile hardware comprises mobile devices or device components that involve mobility. These could be laptops, smartphones, and tablets.
Mobile software represents the programs that run on a mobile device and handles the characteristics and requirements of mobile applications.

Table 3.1. Mobile Computing Assertions.

Given the assertions in Table 3.1, the first group of Definitions 1 to 6 describe the main content and structure of a mobile application:

Defn. 1: A service $\alpha_j = <\alpha_{ID}, \alpha_{NAME}, \alpha_{CLS}, \alpha_{SIG}>$ has a unique ID, a service name, a classification level (MAC from Section 2.3 of Chapter 2), and a signature defined as $\alpha_{SIG} = <\alpha_{TYPE}, \alpha_{URI}, \alpha_{PARAMS}, \alpha_{RETURN}>$ with $\alpha_{TYPE} \in$ {Create, Read, Update, Delete, GET, POST, PUT, READ}, a unified resource identifier, a set of parameters, and a return value.

Example: The CT^2 mobile application has a detailed REST API that is utilized to store concussion information on students to/from a MySQL database. Sample services include:

- Get the list of students of a specific school:
  - α_{ID} = 26
  - α_{NAME} = */students/school/:schoolId*
  - α_{CLS} = C
  - α_{SIG}:
    - α_{TYPE} = GET
    - α_{URI} = bmi10.engr.uconn.edu:10090/
    - $\alpha_{PARAMS\ (Service\ params)}$ = school id

        - α_{RETURN} = Array

- Update a student's general information (first name, last name, school, student number, etc.):
    - α_{ID} = 34
    - α_{NAME} = */students/update/:studentId*
    - α_{CLS} = TS
    - α_{SIG}:
        - α_{TYPE} = POST
        - α_{URI} = bmi10.engr.uconn.edu:10090/
        - $\alpha_{PARAMS\ (Service\ params)}$ = student id
        - $\alpha_{PARAMS\ (POST\ params)}$ = first name, middle name, last name, suffix, email, student number, school id, date of birth, gender
        - α_{RETURN} = Bool

Defn. 2: A *mobile application*, $MA = <UI_{MA}, \beta_{MA}>$, consists of:

- a *user interface (UI)* UI_{MA} = <UI_{Name}, UI_S> with a name UI_{Name} and a set of *n* screens, $UI_S = \{S^1, S^2, \ldots S^n\}$, where each screen $S^i = < s_{ID}, s_{Name} >$ is defined as a two-pair $< s_{ID}, s_{Name} >$ with unique s_{ID} identifier and name s_{Name}, that are organized as either tabs (users can click among tabs) or a sequence of inter-connected screens which are linked with next and previous buttons, and
- an API $\beta_{MA} = \{\alpha_1, \alpha_2, \ldots, \alpha_k\}$ where each α_j is as given in Defn. 1 and services are either web or cloud APIs.

Examples: The pharmacy application presented in Section 1.1 of Chapter 1 had five screens while the CT^2 mobile application from Section 2.7 of Chapter 2 is composed of eight screens with seven of them organized by tabs in the Android version (see Figure 2.7 again). Meanwhile, the iOS version of the CT^2 mobile application (see Figure 2.8 again) is composed of ten screens and a subset of these are linked through navigation actions (left swipe, right swipe).

Defn. 3: Each *screen*, S^i, has a set of *k screen components*, *SC*, denoted, SC_k^i, that allow a user to select, enter, and manipulate data in a MA.

Example: The 'Cause' screen which is depicted in the second screen shown at the bottom of Figure 2.8 contains ten main screen components: three buttons (List, Save, Cancel), one date picker (Date of Incident), five radio buttons (Location of Incident, If Sport, Contact Mechanism, Impact Location on Head, Head Gear Usage), and one text field (Others/Details).

Defn. 4: A *component*, $C^i = < c_{ID}, c_{Name} >$ is defined as a two-pair $< c_{ID}, c_{Name} >$ with unique c_{ID} identifier and name c_{Name} is a portion of a screen that can be displayed and/or entered by users and includes but is not limited to: a text field (TF) to enter information; a button (BN) to effect the state of the application (save, cancel, next, previous, etc.); a drop down (DD) where one value is chosen; a set of checkboxes (CB) where multiple values can be chosen; a set of radio buttons (RB) to select only one of a number of options; a spinner (SP) to select values; and, a date picker (DP) to enter calendar dates; note that this is not an exhaustive component list.

Example: The 'Student' screen of the iOS version for the CT^2 mobile application in Figure 2.7 has: three buttons (List, Save, Cancel), three text fields (First Name, MI, Last Name), five drop downs (Gender, State, etc.), and one date picker (Date of Birth).

Defn. 5: Each MA has a *screen set*, *SS*, which is classified as either:

- A collection of tabs where each tab is a screen, where there is an order among the tabs in the way that they are displayed left to right within the MA.
- A collection of screens where each screen has an appropriate list of buttons/actions to navigate among screens that is augmented with the *screen interactions*, *SI*, necessary to switch among the various screens.

Example: The Android version of the CT^2 mobile application, depicted in Figure 2.7, is composed of seven tabs while the iOS version of the CT^2 mobile application, depicted in Figure 2.8, is composed of six screens linked by navigation actions (left swipe, right swipe).

3.2. RBAC and MAC Model Definitions

The second group of definitions for the unified model of access control for mobile applications involve the way that RBAC and MAC can be defined for the mobile application. In support of these definitions, the relevant assertions are in Table 3.2. The first assertion focuses on the three parts of an application upon which RBAC, MAC, and optional DAC can be defined, specifically: the user interface, the API, and the data sources. The second assertion defines the locations that permissions on RBAC and/or MAC can be defined. The third and fourth assertions primarily relate to RBAC. The fifth assertion introduces a clearance level for a user in support of MAC. The sixth assertion supports constraint checking in MAC (Bell & La Padula, 1976), via a set of properties that define the conditions under which a user with a CLR can read and/or write an object with a CLS: from Section 2.3 of Chapter 2, recall Simple Security (SS), Simple Integrity (SI), Liberal* (L*), and Strict* (S*) that has both Read and Write capabilities.

A mobile application contains a user interface, an API, and data sources. Each of these can be secured using RBAC and/or MAC with optional delegation (DAC).
The user interface, UI, has RBAC and/or MAC permissions defined on screens, screen components, and their interactions, where relevant.
Each role for a mobile application defines permissions as related to UI, API, and data sources in support of RBAC.
Each mobile application has a set of roles and a set of users, and the assignment of a role to a user in support of RBAC.
Each user has a clearance and resources that need to be secured has a classification chosen from a sensitivity level Unclassified (U), Confidential (C), Secret (S), Top Secret (TS), where U < C < S < TS, meaning that users that have a clearance of TS can view all of the resources (U, C, S, TS) while users that have a clearance of U are limited to only viewing the resources that have a classification of U.
Each user is assigned a read and write property that constrains the conditions under which a user is allowed to view objects.

Table 3.2. RBAC and MAC Assertions.

The usage of MAC to control access to objects has to be upgraded in support of this dissertation to apply to the services of a mobile application. The aforementioned MAC properties are defined to determine under which conditions a user with a CLR level can read or write a given data item with a CLS level. These concepts need to be adapted to the different types of services for read (read, GET) and write (Create, Update, Delete, POST, DELETE, PUT). A user is given both a read and a write property for MAC; suppose we have SS for read and SI for write. For read services, the SS property (or read-down, no read-up) is interpreted as the permission to invoke a read service that has an equal or lower CLS level. That is, a user is allowed to invoke a read service with a CLS level equal to or lower than their CLR level, but not those read services with a higher CLS level. For write services, the SI property (or write-down, no write-up) is interpreted as the permission to invoke a write service that has an equal or lower CLS level. That is, a user can invoke a create, update, or delete service of equal or lower CLS level when compared to their CLR level, but not to those create, update, or delete services with a higher CLS level. From a definition and management perspective, an Information Security Officer (ISO) would set the CLR level of users following the predefined sensitivity levels (e.g., TS, S, C, and U) to establish the levels for both users and services. These levels are then augmented on a user-by-user basis by assigning a read property (via SS or S* Read) to invoke a read service and a write property (via SI, L*, or S* Write) to invoke a write service.

Given the assertions in Table 3.2 and the extension of MAC to be applied to services, the second group of Definitions 6 to 12 support the concepts of roles, clearances, and users with both RBAC and MAC characteristics for a mobile application:

Defn. 6: A role r is defined as a two-pair $r = < r_{ID} , r_{Name} >$ with unique identifier r_{ID} and name r_{Name}.

Example: The CT^2 app has many roles, one of which would be for a parent: $r = <r_{ID4} , Parent>$.

Defn. 7: Let $R_{MA} = \{r_1, r_2, \ldots , r_j\}$ be defined as the set of j roles for a given application MA where $r_j \in R_{MA}$ and $r_j = <r_{IDj} , r_{Namej}>$.

Example: The CT^2 app has four roles: $R_{CT}{}^2 = \{r_1 = <r_{ID1} , AT>, r_2 = <r_{ID2} , Coach>, r_3 = <r_{ID3} , Nurse>, r_4 = <r_{ID4} , Parent>\}$ where AT is short for athletic trainer.

Defn. 8: A user u is defined as a tuple $< u_{ID}, u_{Name}, u_{CLR} >$, with unique u_{ID} identifier, name u_{Name} and optional clearance $u_{CLR} \in \{TS, S, C, U\}$ that signifies that a user is limited to information (UI) in the GUI by RBAC and services (API) that satisfy the established MAC properties (e.g., simple integrity, simple security, liberal *, strict *, etc.).

Example: The CT^2 app has a user with top secret clearance <u_{ID1}, Karen, TS>.

Defn. 9: Let $U_{MA} = \{u_1, u_2, \ldots, u_j\}$ be defined as the set of j users for a given application MA, where $u_j \in U_{MA}$ and $u_j = < u_{IDj}, u_{Namej}, u_{CLRj} >$.

Example: The CT^2 app has three users each with different clearances:

$U_{CT^2} = \{ u_1 = < u_{ID1}, Karen, TS >, u_2 = < u_{ID2}, Carmen, C >, u_3 = < u_{ID3}, Joe, C >, u_4 = < u_{ID4}, Peter, S > \}$.

Defn. 10: A user u that has a clearance u_{CLR} (Defn. 8) assigned has also a read property and a write property assigned to control access to a service α (Defn. 1) as follows:

- Read Properties:
  - Simple Security (SS-r): User u has read access on service α iff $u_{CLR} \geq \alpha_{CLS}$.
  - Strict * (Read) (S*-r): User u has read access on service α iff $u_{CLR} = \alpha_{CLS}$.
- Write Properties:
  - Simple Integrity (SI-w): User u has write access on service α iff $u_{CLR} \geq \alpha_{CLS}$.
  - Strict * (Write) (S*-w): User u has write access on service α iff $u_{CLR} = \alpha_{CLS}$.
  - Liberal *: (L*-w) User u has write access on service α iff $u_{CLR} \leq \alpha_{CLS}$.

Given Defn. 10, we revise Defn. 8 as below:

Defn. 8 v2: A user u is defined as a tuple $< u_{ID}, u_{Name}, u_{CLR}, u_{MACRD}, u_{MACWR} >$, where $u_{MACRD} \in$ {SS, S*} and $u_{MACWR} \in$ {SI, S*, L*}.

Updated Example: The CT^2 app has four users each with different clearances and read/write properties: $U_{\mathrm{CT}}{}^2$ = {u_1=<u_{ID1}, *Karen*, *TS, SS-r, L*-w* >, u_2=< u_{ID2}, *Carmen*, *C, S*-r, S*-w* >, u_3=< u_{ID3}, *Joe*, *C*, SS-r, S*-w>, u_4=< u_{ID4}, *Peter*, S, SS-r, S*-w >}, where: nurse Karen can read down and write up and has the most privileges, the parent Carmen is limited to one level secret; and, the coach Joe an AT Peter can both read down and write equal.

Defn. 11: User Role Assignment (URA): Each *user* $u_j \in U_{MA}$ can be assigned a role $r_j \in R_{MA}$ for a user role assignment *ura* = <u_{IDi}, r_{IDj}> that signifies that a user is limited to playing that and the authorized permissions. Note that a user can be assigned multiple roles but only play one role in any session with the mobile application.

Example: Karen is a nurse at a Connecticut middle school that utilizes the CT^2 mobile application, therefore, she is assigned the role of *Nurse* and she is able to access all of the screens/components that are allowed for that role: *ura* = < u_{ID1}, r_{ID3}>.

Defn. 12: User Role Assignment Set (URAS) is the set of all user role assignments that contains an entry for every user/role combination.

Example: Karen, Joe, Carmen, and Peter are associated with a Connecticut middle school in different ways. Karen is the school's nurse, Joe is the school's coach, Carmen is a parent of a student whom has had a concussion and attends the school, and Peter is the school's athletic trainer. In this case Karen is assigned the role of *Nurse*, Joe is assigned the role of *Coach*, Carmen is assigned the role of *Parent,* and Peter is assigned the role of *Athletic Trainer*. As mentioned before, each of these users is able to see different features of the mobile application depending on the permissions of their assigned role:

$$URAS_{MA} = \{<u_{ID1}, r_{ID3}>, < u_{ID2}, r_{ID4}>, < u_{ID3}, r_{ID2}>, < u_{ID4}, r_{ID1}>\}$$

3.3. RBAC UI Permission Definitions

The third group of definitions for the unified model of access control for mobile applications involve establish the way that RBAC can be defined against the user interface elements of a mobile application. In support of these definitions, the relevant assertions are in Table 3.3. The first assertion involves the fact that UI permissions can be defined in different UI elements of mobile application. The second assertion states the way that RBAC can be assigned to the UI elements of a mobile application.

The UI of a mobile application can have RBAC permissions with optional delegation defined on the screens, their components, and their interactions.
Each screen, component, or interaction of a mobile application can be assigned a set of roles that have permission to access the screen in support of RBAC.

Table 3.3. RBAC and MAC UI Permissions Assertions.

Definitions 13 to 16 formalize the concepts of permissions in a mobile application's user interface that involves RBAC control on screens, components, and screen interactions:

Defn. 13: A *screen permission*, $sp = < s_{ID} , p_s >$, where $s_{ID} \in SS$ is a screen identifier and p_s is a screen permission, is utilized to define whether a screen s in SS as given in Defns. 2 and 5, that is part of a mobile application MA is allowable ($p_s = true$) or not ($p_s = false$).

Example: The role of Coach in the CT^2 mobile application can add a student, add the student's concussion information, and view the list of students the user with the specified role has entered. Nevertheless, the Coach role is not able to add symptoms, add follow up data and, is not able to add return data. Therefore, by looking at these permissions, for the Android version (Figure 2.7 of Chapter 2) a user that has the role of Coach has access to the Home, List, Student, and Cause tabs in the CT^2 mobile application as shown in Figure 2.7 (the role has access to the second, third, and fourth screens at the top row of the figure as well as the first screen of the bottom row of the figure). The Coach role has no access to the Symptoms, Follow up, or Return tabs, which are depicted in Figure 2.7 as the second, third, and fourth screens in the bottom row of the figure. Note that MAC could alternatively be utilized to control access to screens where: Home, List, Student and Cause tabs would be confidential (C), Symptoms tab would be secret (S), and Follow up/Return tabs would be top secret (TS). In this case, the user Joe would have a clearance level of C, to limit to the Home, List, Student, and Cause tabs; Peter would be S to also get the Symptoms tab; and Karen would be TS to get all tabs.

Defn. 14: A *component permission*, $cp = < c_{ID} , p_c >$, where $c_{ID} \in SC_k^i$ is a component identifier (Defn. 3), is utilized to define permissions p_c on various components of each screen S (Defn. 4).

a. *on/off permissions* for button (BN), radio button (RB), drop down (DD), checkbox (CB), date picker (DP), spinner (SP), or text field (TF). The permission values for each component are: p_c = *enabled* or *disabled*.

b. *data permissions* for text fields (TF). The permission values for a text field are: p_c = *view*, *edit*, or *edit once*.

Example: As we described in the example of Defn.13, the role of Coach in the CT^2 mobile application has access to the Home, List, Student, and Cause tabs shown in Figure 2.7. Nevertheless, there are certain components in these screens that a user with the aforementioned role does not have access to or, he/she has limited access to such components. For instance, a user with the role of Coach can add general information about a student as well as add the general information of the student's concussion but, he/she does not have permission to edit this information once it is saved.

Defn. 15: A *screen interaction permission set*, $SIP = < si_1 ,\ldots, si_e >$, defines all permitted screen interactions, where each $si = [s_{IDx} , s_{IDy}]$ is a pair of screens that means that screen s_{IDx} interacts with screen, s_{IDy}.

Example: After a user with the Coach role successfully logs in on the iOS version of the CT^2 mobile application, he/she has access to the fourth and fifth screens of the top row of Figure 2.8 and, to the first and second screens at the bottom row of Figure 2.8. In this case, $SIP = < [s_{ID3}, s_{ID4}], [s_{ID3}, s_{ID5}], [s_{ID3}, s_{ID6}], [s_{ID4}, s_{ID3}], [s_{ID5}, s_{ID3}], [s_{ID5}, s_{ID6}], [s_{ID6}, s_{ID3}], [s_{ID6}, s_{ID5}] >$.

Defn. 16: A *role*, $r \in R_{MA}$, is assigned a set of *role permissions*, $r_p = < \gamma, \chi , \lambda >$ for: a subset $m \leq n$ screens of SS in MA where $\gamma = <sp_1, \ldots , sp_m>$ are the m screen permissions (Defn. 13) assigned to the role r, $\chi = < cp_1, \ldots , cp_j >$ are j component permissions (Defn. 14) for all m screens, and $\lambda = < si_1, \ldots , si_q >$ are the screen interactions (Defn. 15) for non-tabbed UIs. Note that if γ is null then χ and λ must also be null. If γ is not null, then either χ and λ can be null meaning that the MA has only screen permissions. Other combinations are possible.

Example: We can combine the examples given at Defns. 13, 14, and 15 to generate the set of role permissions. The permissions for the Coach role are summarized in Table 3.4 using the notation of our model, while Table 3.5 contains the permissions for the four roles available in the Android version of the CT^2 mobile app (Figure 2.7) along with clearance/classification permissions.

Permissions for <r_{ID2},Coach>	
Screens	{<s$_{ID1}$,Home>,<s$_{ID2}$,List>,<s$_{ID3}$,Student>,<s$_{ID4}$,Cause>,<s$_{ID5}$, Symptoms>,<s$_{ID6}$,Follow Up>,<s$_{ID7}$,Return>}
Screen Permissions	{<s$_{ID1}$,true>,<s$_{ID2}$,true>,<s$_{ID3}$,true>,<s$_{ID4}$,true>, <s$_{ID5}$,false>,<s$_{ID6}$,false>,<s$_{ID7}$, false>}
Components	{<c$_{ID1}$,'Enter Student' BN>,<c$_{ID2}$,'Retrieve Open Cases' BN>,<c$_{ID3}$,'Last Name' TF>,<c$_{ID4}$,'First Name' TF>, <c$_{ID5}$,'Search' BN>,<c$_{ID6}$,'Enter New Student' BN>, <c$_{ID7}$,'View Student Info' BN>, <c$_{ID8}$,'Edit' BN>, <c$_{ID9}$,'Add' BN>,<c$_{ID10}$,'First Name' TF>,<c$_{ID11}$,'Middle Initial' TF>,<c$_{ID12}$,'Last Name' TF>,<c$_{ID13}$,'Gender' DD>, <c$_{ID14}$,'Date of Birth' DP>,<c$_{ID15}$,'Date of Past Concussions' DP>,<c$_{ID16}$,'State' DD>, <c$_{ID17}$,'City/Town/Region' DD>,<c$_{ID18}$,'District' DD>, <c$_{ID19}$,'School' DD>,<c$_{ID20}$,'Save' BN>,<c$_{ID21}$,'Cancel' BN>, <c$_{ID22}$,'Location of Incident' BN>,<c$_{ID23}$,'If Sport' DD>, <c$_{ID24}$,'Others/Details' TF>,<c$_{ID25}$,'Contact Mechanism' DD>, <c$_{ID26}$,'Impact Location of Head' DD>,<c$_{ID27}$,'Head Gear Usage' DD>,<c$_{ID28}$,'Save' BN>,<c$_{ID29}$,'Cancel' BN>}
Component Permissions	{<c$_{ID1}$,Enabled>,<c$_{ID2}$,Enabled>,<c$_{ID3}$,View/Edit>,<c$_{ID4}$,View/ Edit>,<c$_{ID5}$,Enabled>,<c$_{ID6}$,Enabled>,<c$_{ID7}$,Enabled>,<c$_{ID8}$, Disabled>,<c$_{ID9}$,Enabled>,<c$_{ID10}$,View/Edit Once>,<c$_{ID11}$,View/Edit Once>,<c$_{ID12}$,View/Edit Once>,<c$_{ID13}$,View/Edit Once>, <c$_{ID14}$,View/Edit Once>,<c$_{ID15}$,View/Edit Once>, <c$_{ID16}$,View/Edit Once>,<c$_{ID17}$,View/Edit Once>, <c$_{ID18}$,View/Edit Once>,<c$_{ID19}$,'School' DD>,<c$_{ID20}$,Enabled>, <c$_{ID21}$,Enabled>,<c$_{ID22}$,View/Edit Once>,<c$_{ID23}$,View/Edit Once>,<c$_{ID24}$,View/Edit Once>,<c$_{ID25}$,View/Edit Once>, <c$_{ID26}$,View/Edit Once>,<c$_{ID27}$,View/Edit Once>, <c$_{ID28}$,Enabled>,<c$_{ID29}$,Enabled>}

Table 3.4. Permissions for the Coach Role of CT2.

Screens/Components	Permissions for Role			
	Nurse	Athletic Trainer	Coach	Parent
Home Tab	**Show**	**Show**	**Show**	**Show**
‘Enter New Student’ BN	Enabled	Enabled	Enabled	Enabled
‘Retrieve Open Cases’ BN	Enabled	Enabled	Enabled	Enabled
‘Last Name’ TF	View/Edit	View/Edit	View/Edit	View/Edit
‘First Name’ TF	View/Edit	View/Edit	View/Edit	View/Edit
‘Search’ BN	Enabled	Enabled	Enabled	Enabled
List Tab	**Show**	**Show**	**Show**	**Show**
‘Enter New Student’ BN	Enabled	Enabled	Enabled	Enabled
‘View Student Info’ BN	Enabled	Enabled	Enabled	Enabled
‘Edit’ BN	Enabled	Disabled	Disabled	Disabled
‘Add’ BN	Enabled	Enabled	Enabled	Enabled
Student Tab	**Show**	**Show**	**Show**	**Show**
‘First Name’ TF	View/Edit	View/Edit once	View/Edit once	View/Edit once
‘Middle Initial’ TF	View/Edit	View/Edit once	View/Edit once	View/Edit once
‘Last Name’ TF	View/Edit	View/Edit once	View/Edit once	View/Edit once
‘Gender’ DD	View/Edit	View/Edit once	View/Edit once	View/Edit once
‘Date of Birth’ SP	View/Edit	View/Edit once	View/Edit once	View/Edit once
‘Date of Past Concussions’ DD	View/Edit	View/Edit once	View/Edit once	View/Edit once
‘State’ DD	View/Edit	View/Edit once	View/Edit once	View/Edit once
‘City/Town/Region’ DD	View/Edit	View/Edit once	View/Edit once	View/Edit once
‘District’ DD	View/Edit	View/Edit once	View/Edit once	View/Edit once
‘School’ DD	View/Edit	View/Edit once	View/Edit once	View/Edit once
‘Save’ BN	Enabled	Enabled	Enabled	Enabled
‘Cancel’ BN	Enabled	Enabled	Enabled	Enabled
Cause Tab	**Show**	**Show**	**Show**	**Show**
‘Location of Incident’ DD	Enabled	View/Edit once	View/Edit once	View/Edit once
‘If Sport’ DD	Enabled	View/Edit once	View/Edit once	View/Edit once
‘Others/Details’ TF	View/Edit	View/Edit once	View/Edit once	View/Edit once
‘Contact Mechanism’ DD	Enabled	View/Edit once	View/Edit once	View/Edit once
‘Impact Location of Head’ DD	Enabled	View/Edit once	View/Edit once	View/Edit once
‘Head Gear Usage’ DD	Enabled	View/Edit once	View/Edit once	View/Edit once
‘Save’ BN	Enabled	Enabled	Enabled	Enabled
‘Cancel’ BN	Enabled	Enabled	Enabled	Enabled
Symptom Tab	**Show**	**Show**	**Hide**	**Show**
‘Mild and Severe Symptoms’ BN	Enabled	View/Edit once	-	View
‘Hour(s)’ TF	View/Edit	View/Edit once	-	View
‘Minute(s)’ TF	View/Edit	View/Edit once	-	View
‘Second(s)’ TF	View/Edit	View/Edit once	-	View
‘Were Parents Notified?’ DD	Enabled	View/Edit once	-	View
‘Removed From Activity’ DD	Enabled	View/Edit once	-	View
‘Removed by’ DD	Enabled	View/Edit once	-	View
‘Concussion Assessment Tool’ DD	Enabled	View/Edit once	-	View
‘Additional Comments’ TF	View/Edit	View/Edit once	-	View
‘Save’ BN	Enabled	Enabled	-	Disabled
‘Cancel’ BN	Enabled	Enabled	-	Disabled
Follow Up Tab	**Show**	**Show**	**Hide**	**Show**
‘Lingering Symptoms’ BN	Enabled	View	-	View
‘If Other, Please Specify’ TF	View/Edit	View	-	View
‘All Symptoms Resolved in’ DD	Enabled	View	-	View
‘Concussion Diagnosed by’ DD	Enabled	View	-	View
‘Post Concussive Syndrome’ DD	Enabled	View	-	View
‘Medical Imaging’ DD	Enabled	View	-	View
‘Additional Comments’ TF	View/Edit	View	-	View
‘Save’ BN	Enabled	Disabled	-	Disabled
‘Cancel’ BN	Enabled	Disabled	-	Disabled
Return Tab	**Show**	**Hide**	**Hide**	**Hide**
‘Days Absent From School’ TF	View/Edit	-	-	-
‘Schedule/Activity Modification’ DD	Enabled	-	-	-
‘504 Plan Required’ DD	Enabled	-	-	-
‘Date of Return to Learn’ SP	Enabled	-	-	-
‘Date of Return to Full Part.’ SP	Enabled	-	-	-
‘Save’ BN	Enabled	-	-	-
‘Cancel’ BN	Enabled	-	-	-

Table 3.5. Summary of Permissions for Roles in CT^2 mobile app.

To bring the concepts together on permissions, Table 3.5 contains detailed permissions on the four user roles (Nurse, AT, Coach, and Parent) as well as the clearance/classification permissions against all of the tabs of the Android version of the CT^2 mobile app (Figure 2.7 of Chapter 2). The four roles can be defined in terms of their ability to access the UI of CT^2 screens and the components (text fields, spinners, date pickers, drop down boxes, and buttons) on a screen-by-screen basis to establish both the on/off permissions and the data permissions as discussed in the prior section. This information represents the privileges or permissions that are authorized to each role and clearance, which when assigned to a given user, results in CT^2 being customized in terms of the screens that are displayed and the components that are enabled. In terms of permissions, the entire screen can either be shown or hidden as a first level of control. For screens that are shown, the different components can be enabled/disabled (button (BN), radio button (RB), drop down (DD), checkbox (CB), date picker (DP), and/or spinner (SP)) or can be view, edit, or edit once (text field (TF)) via the on/off (values of enable and disable) and data (values of view, edit, and edit once) permissions. The edit once data permission means that the user can input data in the text field one time and, after he/she saves such data, he/she cannot modify it. The edit once option also applies to buttons, drop downs, and spinners, since there are cases where the user selects an option from one of these and it can't be modified later on by him/her. If a screen is hidden from the role, then all of the components of the screen are hidden by default.

3.4. RBAC and MAC API Permission Definitions

The fourth group of definitions for the unified model of access control for mobile applications involve the way that the API of the mobile application is viewed from a security perspective in order to control who can call which service(s) of an API at which times and the way that each service is viewed from a security perspective. In support of these definitions, the relevant assertions are in Table 3.6. From a RBAC perspective, we can partition the services of an API into two broad categories: secure and unsecure services. *Secure services* are a subset of the API that require control from a security perspective and can be assigned to individual roles. Not all of the API services need to be in the secure category; for example, API services to load drop downs, display web content, etc., may not need to be secure. The secure API services are the ones that leads to data that is stored/edited/displayed that must be controlled by role. *Unsecure services* need not be assigned and are available to any user. From a MAC perspective, there may be a subset of the API where the services handle data that has different sensitivity levels (top secret, secret, confidential, and unclassified) that must be controlled. These are referred to as *labeled services* which can be given classifications to enforce the MAC model while *unlabeled services* do not need to be classified due to the fact that they do not contain highly sensitive data. To illustrate the labeling of API services with CLS levels, Table 3.7 lists all of the methods for the CT^2 mobile app and their respective CLS levels.

The API of a mobile application can have RBAC permissions defined on the services of the API in order to control which services can be utilized by which role.
The API of a mobile application can be partitioned into two subsets – those that need to be securely controlled by role and those that do not.
The API of a mobile application can have MAC permissions (classifications) defined on the services of the API in order to control which services can be utilized by which user (by clearance).
The API of a mobile application can be partitioned into two subsets – those that need to be securely labeled by classifications and those that do not.
The data sources of a mobile application are accessed by the services of an API in which we can incorporate access control permissions.

Table 3.6. RBAC and MAC API Permissions Assertions.

Classification	Service Name
Confidential	GET /user/:userId
Confidential	GET /userAccounts/account/:userId
Confidential	GET /useraccounts/:username/:password
Confidential	GET /userRoleSchool/:userid
Top Secret	POST /userAccounts/add
Confidential	GET /students/school/:schoolId
Confidential	GET /student/:studentId
Confidential	GET /students/:firstName/:lastName
Confidential	GET /student/guardians/:studentId
Confidential	POST /students/add
Top Secret	POST /students/update/:studentId
Confidential	POST /students/guardian/add
Confidential	POST /students/guardian/update/:guardianId
Confidential	GET /concussion/:concussionId
Secret	GET /concussion/followups/:concussionId
Secret	GET /concussion/followup/symptoms/:recordId
Confidential	GET /concussions/school/:schoolId
Confidential	GET /concussions/student/:studentId
Confidential	GET /concussions/user/:userId
Confidential	GET /concussions/status/:incidentId/:status
Confidential	POST /concussions/add
Top Secret	POST /concussions/update/:incidentId
Secret	POST /concussions/followup/add/:concussionEventId
Top Secret	POST /concussions/followup/update/:followUpId/:referenceId+
Secret	GET /concussion/symptoms/:referenceId+

Table 3.7. Classifications for Labeled Services of CT^2.

Table 3.6 presents the assertions made related to API permissions in support of RBAC and MAC. The first two assertions are related to the partitioning of the API to allow different services to be assigned by role. The next two assertions are related to the partitioning of the API that assigns classifications to services that are accessed by a user with a clearance. The last assertion is related to the way that the data sources are accessed in a mobile application.

Definitions 17 to 22 formalize the assertions in Table 3.6 and are utilized to control policies to the API of a mobile application for the Intercepting API Calls option and the Server Interceptor option of the configurable access control framework.

Defn. 17: The API β_{MA} of a mobile application *MA* can be partitioned into two disjoint sets Secure API σ_{β} and Unsecure API μ_{β} in regards to the services that are to be assigned by role:

- Secure API $\sigma_{\beta} \subseteq \beta_{MA}$ are the services of MA that need to be controlled.
- Unsecure API $\mu_{\beta} \subseteq \beta_{MA}$ are the services of MA that do not need to be controlled where $\beta_{MA} = \sigma_{\beta} \cup \mu_{\beta}$ and $\sigma_{\beta} \cap \mu_{\beta} = \varnothing$ (e.g., $\mu_{\beta} = \beta_{MA} - \sigma_{\beta}$).

Example: The following services are utilized in the API that provides and stores data for the CT^2 mobile application:

- GET /states – Gets the list of states available

- POST /concussions/followup/add/{concussionEventId} – Inserts follow up data of a student into the database

The first service stated above does not need to be secured since all of the users of the mobile application can view the list of states (this is not confidential data), nonetheless, the second service needs to be secured since only a subset of the roles available are allowed to add students' follow up data. Table 3.8 summarizes the secure/unsecure services that are partitioned from the REST API of CT^2.

Secure/Unsecure	Service Name
Secure	GET /user/:userId
Secure	GET /userAccounts/account/:userId
Secure	GET /useraccounts/:username/:password
Secure	GET /userRoleSchool/:userid
Secure	POST /userAccounts/add
Secure	GET /students/school/:schoolId
Secure	GET /student/:studentId
Secure	GET /students/:firstName/:lastName
Secure	GET /student/guardians/:studentId
Secure	POST /students/add
Secure	POST /students/update/:studentId
Secure	POST /students/guardian/add
Secure	POST /students/guardian/update/:guardianId
Secure	GET /concussion/:concussionId
Secure	GET /concussion/followups/:concussionId
Secure	GET /concussion/followup/symptoms/:recordId
Secure	GET /concussions/school/:schoolId
Secure	GET /concussions/student/:studentId
Secure	GET /concussions/user/:userId
Secure	GET /concussions/status/:incidentId/:status
Secure	POST /concussions/add
Secure	POST /concussions/update/:incidentId
Secure	POST /concussions/followup/add/ :concussionEventId
Secure	POST /concussions/followup/update/ :followUpId/:referenceId+
Secure	GET /concussion/symptoms/:referenceId+

Secure/Unsecure	Service Name
Unsecure	GET /states
Unsecure	GET /regions/:stateId
Unsecure	GET /districts/:regionId
Unsecure	GET /schools/all
Unsecure	GET /schools/:districtId
Unsecure	GET /schools/:schoolId
Unsecure	GET /menu/assessmentTools
Unsecure	GET /menu/eventLocations
Unsecure	GET /menu/contactMechanisms
Unsecure	GET /menu/medicalimaging
Unsecure	GET /menu/diagnosingroles
Unsecure	GET /menu/headLocation
Unsecure	GET /menu/sports
Unsecure	GET /menu/symptoms
Unsecure	GET /menu/symptoms/within
Unsecure	GET /menu/symptoms/lingering
Unsecure	GET /menu/roles

Table 3.8. Secure/Unsecure Services of CT^2.

Defn. 18: The API β_{MA} of a mobile application MA can be partitioned into two disjoint sets Labeled API δ_β and Unlabeled API θ_β in regards to the services that are to be controlled by classifications:

- Labeled API $\delta_\beta \subseteq \beta_{MA}$ are the services of MA that need to be controlled.
- Unlabeled API $\theta_\beta \subseteq \beta_{MA}$ are the services of MA that do not need to be controlled.

where $\beta_{MA} = \delta_\beta \cup \theta_\beta$ and $\delta_\beta \cap \theta_\beta = \varnothing$ (e.g., $\theta_\beta = \beta_{MA} - \delta_\beta$).

Example: The service *GET /concussion/followups/:concussionId* and the service *POST /concussions/followup/add/:concussionEventId* can be placed in both the secure API set and the labeled API set. Table 3.9 summarizes the labeled/unlabeled services that are partitioned from the REST API of CT^2.

Intercepting calls for unsecure and unlabeled services are automatically passed through since there are no required security checks. A given mobile application can have a partitioning of the API into: Secure/Unsecure in support of RBAC, Labeled/Unlabeled in support of MAC, or both. In the CT^2 API, labeled services have classifications as given in Table 3.7. Note that a labeled service can have a sensitivity level of unclassified. In MAC, data often moves from level to level, so what is unclassified today, could be confidential or secret at a later point it time; this could be true of services. Only the labeled services in Table 3.9 have classifications as was shown in Table 3.7. The unlabeled services in Table 3.9 are all related to the display of menu drop down values, selection values, etc.

Labeled/Unlabeled	Service Name
Labeled	GET /user/:userId
Labeled	GET /userAccounts/account/:userId
Labeled	GET /useraccounts/:username/:password
Labeled	GET /userRoleSchool/:userid
Labeled	POST /userAccounts/add
Labeled	GET /students/school/:schoolId
Labeled	GET /student/:studentId
Labeled	GET /students/:firstName/:lastName
Labeled	GET /student/guardians/:studentId
Labeled	POST /students/add
Labeled	POST /students/update/:studentId
Labeled	POST /students/guardian/add
Labeled	POST /students/guardian/update/:guardianId
Labeled	GET /concussion/:concussionId
Labeled	GET /concussion/followups/:concussionId
Labeled	GET /concussion/followup/symptoms/:recordId
Labeled	GET /concussions/school/:schoolId
Labeled	GET /concussions/student/:studentId
Labeled	GET /concussions/user/:userId
Labeled	GET /concussions/status/:incidentId/:status
Labeled	POST /concussions/add
Labeled	POST /concussions/update/:incidentId
Labeled	POST /concussions/followup/add/ :concussionEventId
Labeled	POST /concussions/followup/update/ :followUpId/:referenceId+
Labeled	GET /concussion/symptoms/:referenceId+

Labeled/Unlabeled	Service Name
Unlabeled	GET /states
Unlabeled	GET /regions/:stateId
Unlabeled	GET /districts/:regionId
Unlabeled	GET /schools/all
Unlabeled	GET /schools/:districtId
Unlabeled	GET /schools/:schoolId
Unlabeled	GET /menu/assessmentTools
Unlabeled	GET /menu/eventLocations
Unlabeled	GET /menu/contactMechanisms
Unlabeled	GET /menu/medicalimaging
Unlabeled	GET /menu/diagnosingroles
Unlabeled	GET /menu/headLocation
Unlabeled	GET /menu/sports
Unlabeled	GET /menu/symptoms
Unlabeled	GET /menu/symptoms/within
Unlabeled	GET /menu/symptoms/lingering
Unlabeled	GET /menu/roles

Table 3.9. Labeled/Unlabeled Services of CT^2.

Defn. 19: Secure API Role Permissions: Each role *r* can be assigned *Secure API role permissions* $\varphi = \{ss_1, ss_2, \ldots, ss_j\}$ where each $ss_j \in \sigma_\beta$ represents a subset of the secure services in the Secure API σ_β (Defn. 17) that denote those services that can be invoked by a user playing role *r*.

Example: In Table 3.8, φ corresponds to all of the Secure services on the left side. In the CT^2 mobile application a user with the role of Nurse has access to all of the Secure services as shown in Table 3.8.

Defn. 20: Labeled API Classification Permissions: Each user u with a clearance can be assigned *Labeled API classification permissions* $\psi = \{ls_1, ls_2, \ldots, ls_j\}$ where each $ls_j \in \delta_\beta$ represents a subset of labeled services in the Labeled API δ_β (Defn. 18) that denote those services that can be invoked by a user u playing clearance CLR_i under the read and write property conditions for that user.

Example: In Table 3.9, ψ corresponds to all of the Labeled services on the left side. From the CT^2 mobile app, recall the four users: $U_{CT}{}^2$ = { u_1=< u_{ID1}, *Karen*, *TS, SS-r, L*-w* >, u_2=< u_{ID2}, *Carmen*, *C, S*-r, S*-w* >, u_3=< u_{ID3}, *Joe*, *C*, SS-r, S*-w>, u_4=< u_{ID4}, *Peter*, S, SS-r, S*-w > }. Karen with TS CLR has read property SS-r and write property L*-w (see Defn. 8v2) and as a result can access all read services as governed by SS-r (TS, S, C, and U) and all write services as governed by L*-w (TS) (see Defn. 10). Carmen with C CLR has read property, *S*-r* and write property *S*w* (see Defn. 8v2) and as a result can access all read services as governed by *S*-r* (only C) and all write services as governed by *S*-w* (only C) (see Defn. 10). The other two users (Joe and Peter) at S CLR can access all read services as governed by SS-r (S, C and U) and all write services as governed by *S*-w* (only C) (see Defn. 10).

Given Defns. 19 and 20, a version 2 of Defn. 16 can be defined to include secure API role permissions for roles and Defn. 21 can be defined for user labeled API classification permissions for users.

Defn. 16 v2: A *role*, $r \in R_{MA}$, is assigned a set of *role permissions*, $r_p = < \gamma, \chi, \lambda, \varphi>$ for: a subset $m \leq n$ screens of *SS* in MA where $\gamma = <sp_1,\ldots, sp_m>$ are the *m* screen permissions (Defn. 13) assigned to the role *r*, $\chi = < cp_1, \ldots, cp_j >$ are the *j* component permissions (Defn. 14) for all *m* screens, and $\lambda = < si_1, \ldots, si_q >$ are the screen interactions (Defn. 15) for non-tabbed UIs; and $\varphi = \{ss_1, ss_2, \ldots, ss_j\}$ where each $ss_j \in \sigma_\beta$ are the *Secure API role permissions* (Defn. 19) assigned to the role *r*. Note that if γ is null then χ and λ must also be null. If γ is not null, then either χ and λ can be null meaning that the MA has only screen permissions. Note that if φ is null then there are no Secure API role permissions. Other combinations are possible.

Example: For the role permissions, $r_p = < \gamma, \chi, \lambda, \varphi>$, $< \gamma, \chi, \lambda>$ corresponds to all of the permissions defined in Table 3.5, while φ corresponds to the secure services in Table 3.8.

Defn. 21: A *user*, $u \in U_{MA}$, is assigned a set of *user permissions*, $u_p = <\psi>$ where $\psi = \{ls_1, ls_2, \ldots, ls_j\}$ and each $ls_j \in \delta_\beta$ are the *Labeled API classification permissions* (Defn. 20) assigned to the user *u*.

Example: For the user permissions, $u_p = <\psi>$, ψ corresponds to the labeled services in Table 3.9.

Figure 3.1 conceptualizes the permissions associated with services. The secure/unsecure services are assigned by role with the user acquiring these services when they choose a role for a given session (a user may have multiple roles but is limited to one role per mobile application session). This allows RBAC to be used to control services independent of MAC. The labeled/unlabeled services are accessible based on a user's clearance that dominates the classification of the services under the properties (simple security, liberal-*, etc.). This allows MAC to control services independent of RBAC. In addition, we can control services with both RBAC and MAC. This can be achieved by classifying the services of an API as secure/unsecure in support of RBAC and then extending security by classifying secure services even further as labeled/unlabeled services in support of MAC. Note that secure services can be classified as either labeled/unlabeled but unsecured services can only be classified as unlabeled services (dashed arrows shown in Figure 3.1).

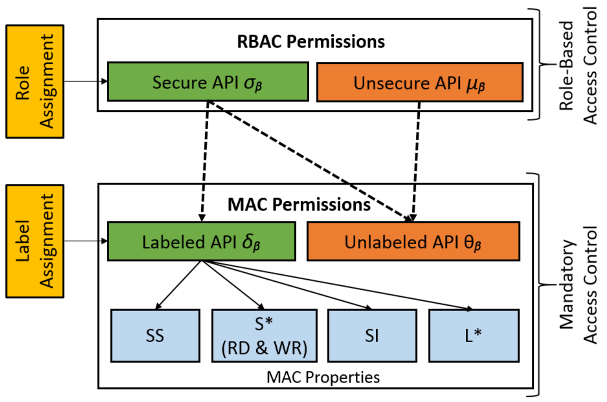


Figure 3.1. Permissions for API Services.

3.5. DAC Model Definitions

Discretionary access control and associated delegation (Department of Defense, 1985; Sandhu & Samarati, 1994) involves the ability of a user (the delegator) to delegate his responsibilities to another user (the delegatee) for a period of time. For the purposes of this dissertation, in support for discretionary access control, we are limiting our approach to *user-directed delegation* where the user decides when and what to delegate to another user. The initial model capabilities for DAC in our model has a two-fold focus in order to delegate permissions defined on the user interface by role and on the secure and labeled services by role and user. Specifically, for RBAC, we control the delegation of permissions on screen, component, and screen interactions (Defns. 13, 14, and 15, respectively) for a role assigned to a user that can be delegated to another user. In this case, the user delegates his/her role to another user and as a result all of the screen, component, and interaction permissions associated with that role are delegated which is termed *Full RBAC UI Delegation.* For RBAC, we also control the delegation of the secure API role permissions assigned to a user by role where a subset of the secure services has been assigned to each role. A user can delegate all of his/her assigned secure services to another user by delegating the role to that user which is called *Full RBAC Service Delegation.* In addition, a user has the option to delegate a subset of his/her assigned secure services to another user which is called *Partial RBAC Service Delegation.* In an analogous manner, for MAC we control the delegation of the labeled API classification permissions assigned to a user by clearance where a subset of the labeled services has been assigned to each role. A user can delegate all of his/her assigned labeled services to another user by delegating to that user which also passes along the clearance which is called *Full MAC Service Delegation*. In addition, a user could delegate a subset of his assigned labeled services to another user which is called *Partial MAC Service Delegation*. In all three delegation possibilities, we support the concepts of delegation authority and pass on delegation of authority.

Delegation authority states that a security officer can delegate the authority to delegate to another user. In our case, a user can delegate his/her role or clearance permissions to another user but cannot delegate the authority to delegate those permissions further to another user. In order to delegate further, *pass-on-delegation authority* can be defined that allows a user to delegate the *delegation authority* along with the delegation of a role or a clearance to a user. In turn, that delegated user can delegate those permissions to another user. In this reminder of the section on the DAC model capabilities for mobile computing we utilize concepts from (Liebrand et al., 2003) on delegation.

In support of this section, the fifth and final group of definitions for the unified model of access control for mobile applications involve the way that Discretionary Access Control (DAC) can be incorporated in a mobile application in order to extend the security provided with RBAC and/or MAC definitions in Sections 3.2, 3.3, and 3.4. In support of these definitions, the relevant assertions are in Table 3.10.

A user with an assigned role and/or clearance can delegate his/her role (in support of RBAC) and/or clearance (in support of MAC) at a given time/situation in support of DAC.
Users with the ability to delegate can pass on their allowed actions to delegable users.
A delegable user can pass his/her delegated permissions further if he/she has authority to do so.

Table 3.10. DAC Assertions.

Defn. 22: An *original user*, $\mathcal{ou}$, is a user that owns a given role.

Defn. 23: An *original role*, $\mathcal{or}$, is the role delegated by an original user $\mathcal{ou}$.

Defn. 24: A *delegated user*, $\mathcal{du}$, is a user to who a role will be delegated.

Defn. 25: A *delegated role*, $\mathcal{dr}$, is the role delegated to the delegated user $\mathcal{du}$.

Defn. 26: *Delegation Authority* (*DA*): A security officer determines which users in an U_{MA} can delegate their permissions to other users in the U_{MA}.

Defn. 27: *Pass On Delegation Authority* (*PODA*) is a Boolean value assigned to a user which determines if he/she can delegate his/her permissions to another user (*poda* = *true*) or not (*poda* = *false*).

Example: Karen in the prior example can be an original user, ou, with the original role, or nurse. If a school needs a substitute nurse Lois to cover for Karen, Karen could delegate her original role or to Lois as the delegated user or.

As previously discussed in Section 3.3, there are the permissions associated with the screens, components, and screen interactions of the UI of a MA. From an RBAC perspective, a particular user might have access by role to certain screens (screen can be hidden from user) and components of screens with the ability to view (component is enabled), edit, edit once, or hide (component is disabled). In this case, *view* allows a user to view the component, *edit* allows a user to modify the contents of the component (applicable to text fields, text boxes, etc.), *edit once* allows a user to add data through the means of a component but after the data is saved the component cannot be edited (component becomes disabled), and *hide* conceals the component from the user so that he/she can't have access to such. The following example is presented to demonstrate the way that DAC can be utilized within the UI screens and components. In CT^2, nurse Karen could delegate all of her UI permissions via Table 3.5 to nurse Lois that is the substitute nurse that day, but decide not to allow Lois to pass on that delegation. In this case, we delegate the entire role's UI permissions from the original user Karen to the delegated user Lois, with the exception of PODA. The following definition supports DAC in the UI of a MA, where the pass-on-delegation authority:

Defn. 28: A *Full RBAC UI (FRUI) Delegation* $d_{FRUI} = < \mathcal{ou}, \mathcal{or}, \mathcal{du}, \mathcal{dr}, < \gamma, \chi, \lambda >, \mathcal{poda}, \mathcal{timePeriod} >$ delegates all UI permissions $< \gamma, \chi, \lambda > \in r_p$ (screens, component, and screen interactions – Defn. 16v2) from an original user, $\mathcal{ou}$, with an original role, $\mathcal{or}$, to a delegated user $\mathcal{du}$ with a $\mathcal{dr} = \mathcal{or}$ with the potential to pass on ($\mathcal{poda}$ is true or false) and $\mathcal{timePeriod}$ = {*startTime*, *endTime*} which represents the period of time in which the $\mathcal{du}$ has access to the delegated permissions of $\mathcal{dr}$.

Example: The original user $\mathcal{ou}$ Karen < u_{ID1}, *Karen*, *TS*, *SS-r*, *L*-w* > seeks delegate her original role nurse $\mathcal{or}$ to the delegated user $\mathcal{du}$ Lois, a substitute school nurse for one day: $\mathcal{del} = < u_{ID1}, r_{ID3}, u_{ID5}, r_{ID3}, < \gamma, \chi, \lambda >$, *false*, {*2017-07-31T09:00:00+00:00*, *2017-12-15T07:00:00+00:00*} >. Note that $< \gamma, \chi, \lambda >$ is as defined in the Nurse column in Table 3.5.

In the case of API permissions, there are two levels of delegations we define to support DAC. The first level involves full and partial RBAC service delegation while the second level involves full and partial MAC service delegation. Note that the services we focus on delegating to DU are those that are either secure or labeled services since all of the users of a MA have access to unsecure/unlabeled services. For RBAC, when we delegate the role from the original user to the delegated user, all or a subset of the authorized secure services are delegated for the given role to the new user. Specifically: DAC delegates all secure services authorized to a user/role for RBAC delegation from original to delegated user in Full RBAC Services delegation and delegates a subset of secure services in Partial RBAC Services delegation:

Defn. 29: A *Full RBAC Services (FRS) Delegation* d_{FRS} = < ou, or, du, dr, φ, poda, $\mathit{timePeriod}$ > delegates all of the assigned secure service permissions $\varphi \in r_p$ (Defn. 16v2) from an original user, ou, with an original role, or, to a delegated user, du, with a $\mathit{dr} = \mathit{or}$ with the potential to pass on (poda is true or false), and $\mathit{timePeriod}$ = {*startTime*, *endTime*} which represents the period of time in which the du has access to the delegated permissions.

Example: The original user ou Karen < u_{ID1}, *Karen*, *TS*, *SS-r*, *L*-w* > seeks to delegate original role nurse or and all of the secure assigned services of her Nurse role the delegated user du Lois, a substitute school nurse for one day: *del* = < u_{ID1}, r_{ID3}, u_{ID5}, r_{ID3}, φ, *false*, {*2017-07-31T09:00:00+00:00*, *2017-12-15T07:00:00+00:00*} >. Note that φ are the labeled services in Table 3.9.

Defn. 30: A *Partial RBAC Services (PRS) Delegation* d_{PRS} = < ou, or, du, dr, φ', poda, $\mathit{timePeriod}$ > delegates a subset of the assigned secure service permissions $\varphi' \in r_p$ and $\varphi' \subseteq \varphi$ (Defn. 16v2) from an original user, ou, with an original role, or, to a delegated user, du, with a $\mathit{dr} = \mathit{or}$ with the potential to pass on (poda is true or false) and $\mathit{timePeriod}$ = {*startTime*, *endTime*} which represents the period of time in which the du has access to the delegated permissions.

Example: The original user ou Karen < u_{ID1}, *Karen*, *TS, SS-r; L*-w* > seeks to delegate original role nurse or and only a subset of the secure services assigned to her Nurse role the delegated user du Lois, a substitute school nurse for one day to only log on and be able to read (GET services) information on students (first nine secure services on left portion of Table 3.8): $del = < u_{ID1}, r_{ID3}, u_{ID5}, r_{ID3}, \varphi',$ *false,* {*2017-07-31T09:00:00+00:00*, *2017-12-15T07:00:00+00:00*} >, where φ'= {GET/user/:userid, … , GET /userRoleSchool/:userid, POST /userAccounts/:add, … , GET /student/guardians/:studentID}.

For MAC, if we choose to delegate, then we are delegating a combination of the the clearance of the user and the read and write properties for the user; this allows the delegated user to access the appropriate labeled services by classification. In *Full MAC Services Delegation*, a user delegates all of labeled services authorized to a user/CLR/read-write properties for MAC delegation from original to delegated user in Full MAC Services Delegation. In *Partial MAC Services Delegation*, a user delegates his read-write properties and a CLR that is less secure that his current level, thereby automatically resulting in a subset of methods that is at most the same of the original clearance level but is more often less.

Defn. 31: A *Full MAC Services (FMS) Delegation* $d_{FMS} = < ou, oclr, oprops, du, dclr, dprops,$ $\Omega, poda, timePeriod >$ delegates all of the assigned labeled service permissions $\Omega \in u_p$ (Defn. 16v2) from an original user, ou, with an original clearance, $oclr$, and original read/write properties, $oprops$, to a delegated user du with a delegated clearance $dclr = oclr$ and delegated read/write properties, $dprops = oprops$ with the potential to pass on ($poda$ is true or false) and $timePeriod$ = {*startTime*, *endTime*} represents the period of time in which the $\mathscr{du}$ has access to the delegated permissions.

Example: The original user $\mathscr{ou}$ Karen < u_{ID1}, *Karen*, *TS*, *SS-r*, *L*-w* > seeks to delegate her MAC privileges (CLR and read/write properties) and all of labeled services from Table 3.9 to the delegated user $\mathscr{du}$ Lois, a substitute school nurse for one day: *del* = < u_{ID1}, r_{ID3}, *SS-r*, *L*-w*, u_{ID5}, r_{ID3}, *SS-r*, *L*-w*, *Ω*, *false*, {*2017-07-31T09:00:00+00:00*, *2017-12-15T07:00:00+00:00*} >. Note that *Ω* are the labeled services in Table 3.9.

Defn. 32: A *Partial MAC Services (PMS) Delegation* d_{PMS} = < $\mathscr{ou}$, $\mathscr{oclr}$, $\mathscr{oprops}$, $\mathscr{du}$, $\mathscr{dclr}$, $\mathscr{dprops}$, *Ω'*, $\mathscr{poda}$, $\mathscr{timePeriod}$ > delegates a subset of the assigned labeled service permissions $Ω'\ u_p \subseteq Ω$ and $Ω' \in u_p$ (Defn. 16v2) from an original user, $\mathscr{ou}$, with an original clearance, $\mathscr{oclr}$, and original read/write properties, $\mathscr{oprops}$, to a delegated user $\mathscr{du}$ with a delegated clearance $\mathscr{dclr} = \mathscr{oclr}$ and delegated read/write properties, $\mathscr{dprops} = \mathscr{oprops}$ with the potential to pass on ($\mathscr{poda}$ is true or false) and $\mathscr{timePeriod}$ = {*startTime*, *endTime*} represents the period of time in which the $\mathscr{du}$ has access to the delegated permissions.

Example: The original user $\mathscr{ou}$ Karen < u_{ID1}, *Karen*, *TS*, *SS-r*, *L*-w* > seeks to delegate her MAC privileges (CLR and read/write properties) and only the GET services from Table 3.9 to the delegated user $\mathscr{du}$ Lois, a substitute school nurse for one day: *del* = < u_{ID1}, r_{ID3}, *SS-r*, *L*-w*, u_{ID5}, r_{ID3}, *SS-r*, *L*-w*, *Ω'*, *false*, {*2017-*

07-31T09:00:00+00:00, 2017-12-15T07:00:00+00:00} >, where *Ω'* = { GET /user/:userid, GET /userAccounts/account/:userid, … , GET /concussion/symptoms/:referenceID + }.

Defn. 33: The *Delegation Collection*, *DC*= <*FRUI*, *FRS*, *PRS*, *FMS*, *PMS*> for a given mobile application, MA, are five sets (possible null) of active delegations for Full RBAC UI, Full RBAC Services, Partial RBAC Services, Full MAC Services, and Partial MAC Services (Defns. 28-32, respectively), where each set contains elements d_i where $i \in$ {FRUI, FRS, PRS, FMS, PMS}.

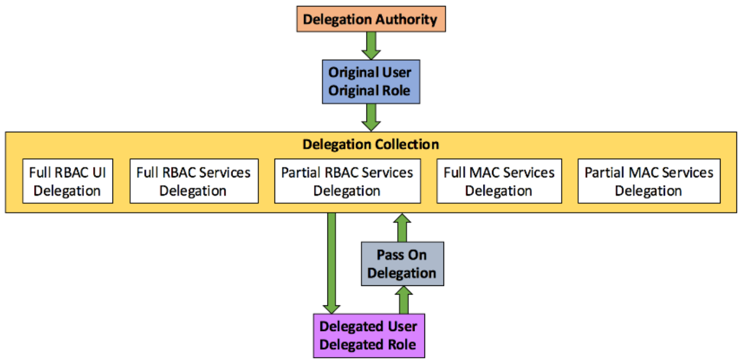


Figure 3.2. DAC Permissions.

Figure 3.2 depicts a summary of Defns. 22-33. Basically, a delegation authority (top box of Figure 3.2) is in charge of assigning which users are allowed to delegate their permissions (second box of Figure 3.2), which are the original users (delegators). Note that for our model, we assume that a delegation authority has already established the delegation permissions and therefore do not go into detail about this process and other security policy definition and administration processes. Depending on which permissions the system has enforced (UI, API), a delegator can either delegate a full set of his/her UI permissions (first box shown in the middle vertical box of Figure 3.2), a full or partial set of his/her RBAC API permissions (second and third boxes shown in the middle vertical box of Figure 3.2), and/or a full or partial set of his/her MAC API permissions (last two boxes shown in the middle vertical box of Figure 3.2) to a delegated user (last box of Figure 3.2). In addition, the delegator can choose whether he/she wants to grant their role/clearance to the delegated user (delegatee) or if they want to grant them a role/clearance with less privileges (delegated role/clearance) and, he/she can choose the period of time the delegatee has access to the delegated permissions. Finally, a delegatee can pass their delegated permissions to another delegatee if he/she has *pass on delegation* value set as true (fourth box of Figure 3.2).

3.6. Combining Access Control Models and Options

In this section, we discuss the different ways in which the access control models (RBAC, MAC, and DAC), the mobile application (UI, API, and data sources), and the options (Direct UI Modifications, Intercepting API Calls, and Server Interceptor API) of the unified model of access control for mobile applications can be combined in meaningful ways in order to define a specific type of security on a mobile-app-by-mobile-app basis. This supports contribution C: Dynamic Combination of Access Control Models and Configuration Options. The different combinations of (RBAC, MAC, and DAC) vs. (UI, API, and Data Sources) vs. (Direct UI Modifications, Intercepting API Calls, and Server Interceptor API) that are chosen by a security engineer are based on the type of security that a subject is seeking to attain and may be dictated by whether the source code of the mobile app, API, and/or data source is available. Combinations are shown in Table 3.11.

Combination	**Access Control Model(s)**	**Mobile App**	**Options**
C1	RBAC	UI	Direct UI Modifications*
C2	RBAC, DAC		
C3	RBAC	API	Intercepting API Calls
C4	MAC		
C5	RBAC, MAC		
C6	RBAC, DAC		
C7	MAC, DAC		
C8	RBAC	Data Source	Server Interceptor API
C9	MAC		
C10	RBAC, MAC		
C11	RBAC, DAC		
C12	MAC, DAC		
C13	RBAC	UI, API	Direct UI Modifications*, Intercepting API Calls
C14	RBAC, MAC		
C15	RBAC, DAC		
C16	RBAC, MAC, DAC		
C17	RBAC	API, Data Source	Intercepting API Calls, Server Interceptor API
C18	MAC		
C19	RBAC, MAC		
C20	RBAC, DAC		
C21	MAC, DAC		
C22	RBAC, MAC, DAC		

Table 3.11. Combinations of Access Control Configurations.

Note that the '*' after Direct UI Modifications signifies that code level changes may be required. This is a preliminary list that is currently under investigation and is expanded and refined over the course of the remainder of the research. To illustrate, we enumerate some combinations of access control for different applications:

- **Combination C1:** A mobile application for the pharmacy as previously described would utilize a RBAC approach to represent capabilities of licensed pharmacist and pharmacy technician roles. The mobile application had different screens for different roles – all screens for the licensed pharmacist and only screens 1, 2, and 4 for the technician. This would necessitate the use of the Direct UI Modifications option to set permissions on which screen is available to which role (see Defn. 13). In addition, for screens 1, 2, and 4 of the technician, all of these screens would be further constrained so that all of the text fields are read only (see Defn. 14.b).

- **Combination C22:** A mobile application for patient data to be used by medical professionals would utilize a combination of access control: RBAC to define different roles for internists, family practitioners, physiatrists, psychologists, etc.; MAC to allow the use of top-secret sensitivity level for all mental health data, secret for the majority of medical data, confidential for certain patient info, and unclassified for demographics/contact info; DAC to allow for the delegation from a physician to the on-call physician for nights and weekends. There would be no screen permissions in this case, since the control is on data delivered to the mobile application. In this case, to control access to data for RBAC and MAC, the API is partitioned into secure and unsecure APIs (see Defn. 17) while all services for patient data is labeled (see Defn. 18) to control the data returned to users. The Intercepting API option would support RBAC, MAC, and DAC in terms of API control, while the Server Interceptor option may be necessary to filter the mental health data from the data source. These are just two examples – in the course of the remainder of this chapter the Connecticut Concussion Tracker (CT^2) app is utilized as well for additional explanation and illustration.

3.7. Relational Database Design for the Unified Security Model

This section presents a relational database design to store the content unified security model as presented in Sections 3.1 to 3.5 realized via an entity-relationship diagram as shown in Figure 3.3. The diagram in Figure 3.3 contains 44 entities and to assist the discussion in Chapters 4, 5, and 6, Table 3.12 contains a list of all 44 entities including:

- Entity Name: Name of the entity.
- Table Content: Tuples stored in the entity.
- Primary Related Entities: Other related entities.
- Primary Definition(s): A mapping to the one or two primary definitions of the unified security model corresponding to the entity.
- Secondary Definition(s): A mapping to the secondary definitions of the unified security model corresponding to the entity.

In Chapters 4, 5, and 6, when discussing the three different options, Direct UI Modifications, Intercepting API Calls, and Server Interceptor API, respectively, a subset of the ER diagram in Figure 3.3 is presented and discussed.

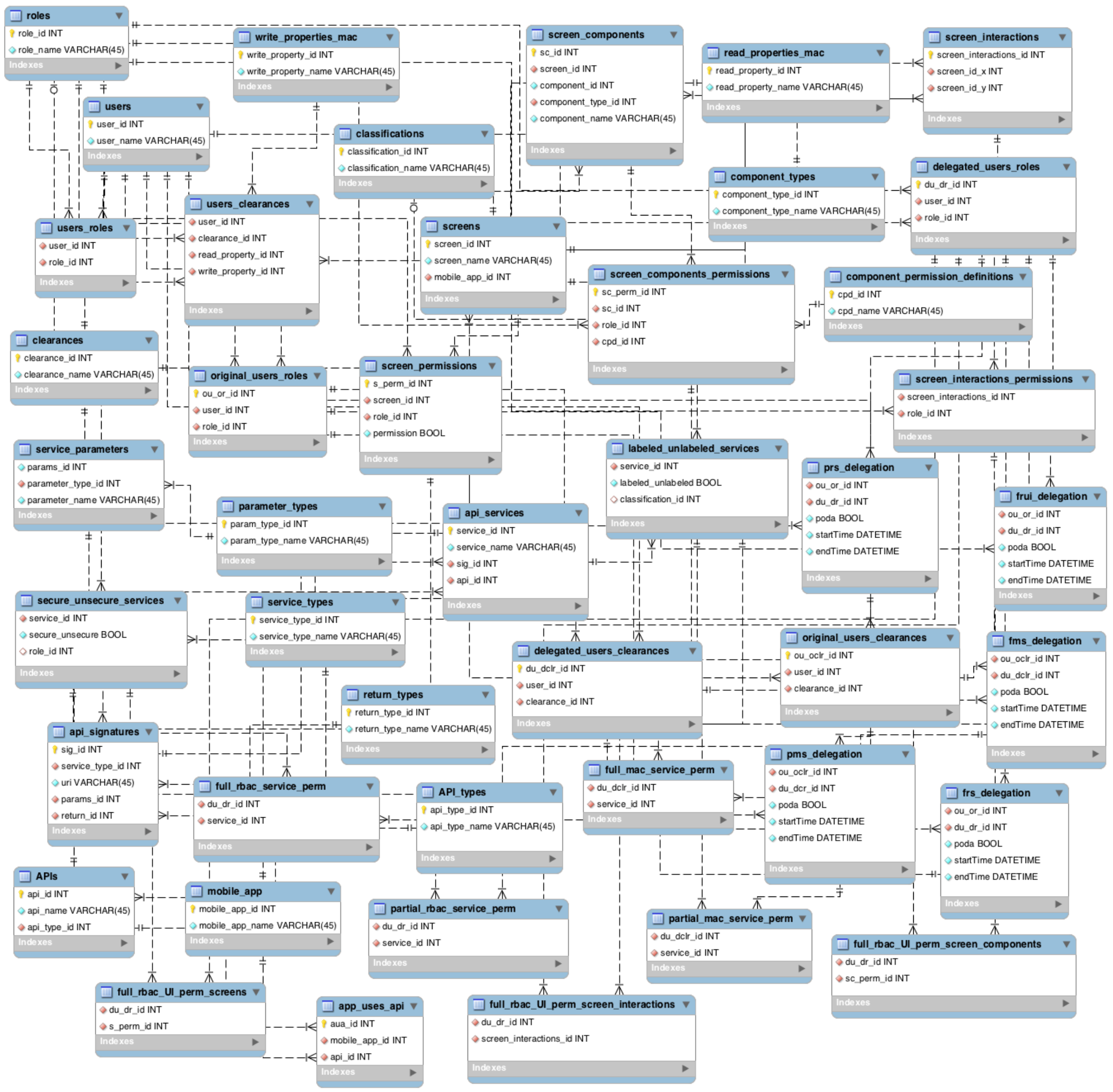


Figure 3.3. Entity-Relationship Diagram for Unified Security Model.

	Entity Name	Table Content	Primary Related Entities	Primary Definition(s)	Secondary Definition(s)
User Definitions	roles	All roles	user_roles, original_user_roles, original_user_roles	Defn. 6	Defns. 7, 11, 12
	users	All users	user_roles, user_clearances	Defn. 8v2	Defns. 9-12
	users_roles	All User/Role Combo	users, roles	Defn. 11	Defns. 6, 8, 9, 11
	clearances	CLR: T, S, C, and U	user_clearances, original_users_clearances, delegated_users_clearances	Defn. 8	Defns. 8v2, 10
	users_clearances	CLR all users	users, clearances, read_properties_mac, write_properties_mac	Defn. 8v2	Defns. 9-12
	classifications	CLS: T, S, C, and U	labeled_unlabeled_services	Defn. 20	Defn. 18
	read_properties_mac	SS-r, S*-r	users_clearances	Defn. 10	Defns. 8v2, 31, 32
	write_properties_mac	SI-w, S*-w, L*-w	users_clearances	Defn. 10	Defns. 8v2, 31, 32
Original & Delegated User Definitions	original_users_roles	Roles of users who delegate	users, roles	Defn. 22, 23	Defns. 28-33
	delegated_users_roles	Roles of users delegated to	users, roles	Defn.24, 25	Defns. 28-33
	original_users_clearances	CLR of users who delegate	users, clearances	Defn. 31,32	Defns. 21,22-27, 33
	delegated_users_clearances	CLR of users delegated to	users, clearances	Defn. 31,32	Defns. 21,22-27, 33
Mobile Applications & APIs Definitions	mobile_apps	Names of all mobile applications	APIs, screens	Defn. 2	Defns. 3-5
	APIs	Names of all APIs	api_services	Defn. 2	Defns. 17-20
	app_uses_api	Mobile apps and their APIs	mobile_apps, APIs	Defn. 1	Defn. 2
	API_types	MA, Server Side, etc.	APIs	Defn. 1	Defn. 2
	api_services	Name, FK API, api_signatures	APIs, api_signatures	Defn. 1	Defn. 2
	api_signatures	Name, Parameters, Return	api_services	Defn. 1	Defn. 2
	service_parameters	Parameters of an API service	APIs, api_services	Defn. 1	Defn. 2
	parameter_types	URL parameters, POST parameters, etc.	api_services	Defn. 1	Defn. 2
	service_types	GET, PUT, CREATE, etc.	api_signatures	Defn. 1	Defn. 2
	return_types	Int, Float, String, Boolean, etc.	api_signatures	Defn. 1	Defn. 2
Serv Perm Defn	secure_unsecure_services	All S/US Services per API	roles, api_services	Defn. 17	Defns. 29,30
	labeled_unlabed_services	All L/UL Services per API	classifications, api_services	Defn. 18	Defns. 31,32
Service Permissions Delegations Definitions	full_rbac_service_perm	Full RBAC Secure Delegation	frs_delegation, secure_unsecure_services	Defn. 29	Defns. 16v2, 22-27, 33
	partial_rbac_service_perm	Partial RBAC Secure Delegation	prs_delegation, secure_unsecure_services	Defn. 30	Defns. 16v2, 22-27, 33
	full_mac_service_perm	Full MAC Labeled Delegation	fms_delegation, labeled_unlabeled_services	Defn. 31	Defns. 21, 22-27, 33
	partial_mac_service_perm	Partial MAC Labeled Delegation	pms_delegation, labeled_unlabeled_services	Defn. 32	Defns. 21,22-27, 33
	frs_delegation	Permissions for Full RBAC	original_users_roles, delegated_users_roles	Defn. 29	Defns. 16v2, 22-27, 33
	prs_delegation	Permissions for Partial RBAC	original_users_roles, delegated_users_roles	Defn. 30	Defns. 16v2, 22-27, 33
	fms_delegation	Permissions for Full MAC	original_users_clearances, delegated_users_clearances	Defn. 31	Defns. 21,22-27, 33
	pms_delegation	Permissions for Partial MAC	original_users_clearances, delegated_users_clearances	Defn. 32	Defns. 21,22-27, 33
UI Permissions Definitions	screens	All Screens for each MA	screen_components, screen_permissions	Defn. 3	Defn. 5
	component_types	Buttons, Drop Downs, etc.	screen_components	Defn. 4	Defns. 16v2, 28
	components_permissions_definitions	View, Edit, Edit Once, etc.	screen_components_permissions	Defn. 14	Defns. 16v2, 28
	screen_components	Components for Each Screen	screens, screen_components_permissions	Defn. 3	Defns. 16v2, 28
	screen_interactions	Interactions among Screens	screens	Defn. 5	Defns. 16v2, 28
	screen_permissions	Permissions for each Screen	screens, roles	Defn. 13	Defns. 16v2, 28
	screen_components_permissions	Permissions for each Component	screen_components, role	Defn. 14	Defns. 16v2, 28
	screen_interactions_permissions	Interactions among screens by role	screen, roles	Defn. 15	Defns. 16, 28
UI Permissions Delegations Definitions	full_rbac_UI_perm_screens	Full RBAC UI Delegation for Screens	original_users_roles, delegated_users_roles, s_perm_id	Defn. 28	Defns. 13, 16, 33
	full_rbac_UI_perm_screen_components	Full RBAC UI Delegation for Screen Components	original_users_roles, delegated_users_roles, sc_perm_id	Defn. 28	Defns. 14, 16, 33
	full_rbac_UI_perm_screen_interactions	Full RBAC UI Delegation for Screen Interactions	original_users_roles, delegated_users_roles, screen_interactions_id	Defn. 28	Defns. 15, 16, 33
	frui_delegation	Permissions for Full UI	original_users_roles, delegated_users_roles	Defn. 28	Defns. 13-16, 22-27, 33

Table 3.12. Entities and Explanations.

3.8. Related Work on Access Control in Mobile Computing

In this section, we compare and contrast the unified model of access control for mobile applications from Sections 3.1 to 3.6 to other related work that utilizes and extends access control models to provide secure authorization in mobile computing. To begin, the work of (Abdunabi, Sun, & Ray, 2014) proposed a spatio-temporal access control framework to enforce spatio-temporal policies in mobile applications. Basically, the authors utilized the RBAC model as a basis for their approach and then extended this to provide the spatio-temporal feature. To apply the proposed approach in a mobile application, there are three modules that are needed: one that needs to be installed in the mobile device and the other two are placed server-side. One drawback in this approach is the fact that users need to install a module on their mobile devices in order to utilize the proposed approach, which is similar to our drawback from code-level changes in the mobile app for the Direct UI Modifications option. In our Intercepting API Calls and Server Interceptor options, users do not need to modify their mobile devices since we are enforcing access control policies server-side. Another approach, which involves utilizing user attributes to provide access control for business processes in mobile computing, consists of utilizing RBAC in combination of context-aware access control mechanisms (Schefer-Wenzl & Strembeck, 2013). Basically, the approach identifies the tasks that are available in a system, assigns roles to the users of that system and, establishes which roles have permissions of which tasks and under what context. Our approach can augment their work by including additional RBAC, MAC, and DAC capabilities. A third approach (Santos-Pereira, Augusto, Correia, Ferreira, & Cruz-Correia, 2012) focuses on securing healthcare data by proposing an architecture that combines the RBAC model with personal/technical characteristics as well as with capabilities of a smartphone in order to deliver patients a way to exercise safe discretionary online access permissions on their EHR. This approach utilizes the mobile device as a token in order to verify a user's identity and give him/her access to the EHR data he/she has access to manage. However, this approach is not for mobile applications but to use with a web-browser. While our approach is targeted at native mobile applications for the Android and iOS platforms, the intercepting API call option could be utilized between the web app and the API. A fourth approach (Fadhel et al., 2016) proposed a model that extends RBAC to generate RBAC conceptual policies. Nevertheless, the aforementioned effort does not provide details of which specific application domains the approach could support.

The next two related efforts, address the way that the mobile application itself (UI, API, Server/Database) is impacted depending on the role that a user assumes for a particular mobile application session or relevant MAC capabilities. The first effort utilizes MAC to provide security in mobile computing (Bugiel, Heuser, & Sadeghi, 2013) by proposing and implementing *FlaskDroid*, a security architecture that provides mandatory access control in both middleware and kernel layers of Android OS. The purpose of this work is to apply fine-grained MAC security policies to Android OS services such as *LocationManager* and *Audio Services*. The end result is that the applications that form part of the device conforms to these finer-grained security policies rather than utilizing the ones the device provides. Our approach contrasts to their work since we do not rely on modifying the default services of a mobile OS and it is mobile application-specific. The second effort that involves modifying the Android OS consists of applying context-based access control restrictions in mobile devices (Shebaro, Oluwatimi, & Bertino, 2015). The intent is to allow a user of a mobile device to create a security policy that establishes which resources/services of his/her mobile device their installed mobile applications should have access to. This is occurring at a much higher level of granularity (i.e., entire mobile application) than our approach which is focusing on specific services of APIs. Both of these efforts are targeted towards modifying Android OS in order to provide finer-grained security for the permissions that mobile devices offer. They operate at a much higher conceptual level than our work which focuses on applying access control security policies to individual mobile applications; their work is on changing device permissions and this determines what mobile applications can access as a whole.

In terms of DAC, to our understanding, there are no approaches that directly implement the access control mechanism in a mobile computing setting. Nevertheless, there are several works that state that DAC mechanisms can be incorporated in a system through the means of RBAC (Hansen & Oleshchuk, 2003; Baracaldo & Joshi, 2013) by referring to (Osborn, Sandhu, & Munawer, 2000). However, these proposed approaches on extending RBAC and only mention that such access control model can be configured to enforce mandatory and discretionary access control mechanisms. In other words, they do not provide details on the way that MAC or DAC can be applied in an RBAC setting. On another note, Android OS uses the traditional Linux Discretionary Access Control to manage filesystem access (Morris, 2013). This is different to our approach since it does not enforce this mechanism on the data that is handled within a mobile application but on the files that a user creates/handles in the storage of the mobile device itself.

Chapter 4
Direct UI Modifications Option

This chapter reviews the security policy definition and generation process for the screens, components, and interactions of the user interface in order to explore and explain the *Direct UI Modifications* option (see Section 1.4 and Figure 1.3 again) to change the look-and-feel of the UI according to RBAC and/or DAC permissions. The specific objective is to allow the capabilities of the mobile application's UI to be dynamically customized based on a user's role and delegation permissions. This both permits a user by role to perform needed tasks using the mobile application while simultaneously limiting and/or disabling and/or removing capabilities and features that are not allowed at certain times or in certain situations to that user/role combination. In our approach, the components of a mobile application UI will be treated as "objects" to which we can apply access control mechanisms to. The main focus of this chapter is to present and discuss an approach for role-based access control (RBAC) with optional delegation (DAC) for the UI of mobile applications that allows permissions established by the information owner to be defined for other authorized users by role, thereby allowing the mobile application to be dynamically customized to deliver only authorized information, and defined view and/or modify capabilities. To demonstrate the feasibility of our work, we utilize the CT^2 mHealth application (see Section 2.7 of Chapter 2).

The chapter provides a detailed discussion of the Direct UI Modifications option in 5 sections. Section 4.1 briefly reviews a subset of the model and permissions from Chapter 3 for the mobile app UI that define which screens and components can be viewed/edited/viewed once/enabled/hidden in order to customize the look-and-feel of the

UI by role. Section 4.2 reviews a subset of the ER diagram for the unified security model in Figure 3.3 of Section 3.7, focusing on the subset of the unified model involving UI, screens, components, screen interactions, roles, and optional delegation in support of the Direct UI Modifications option. The section also provides an example on the way that the CT^2 mobile app will be displayed to users depending on their role. Section 4.3 explains the programmatic changes that must be made to the mobile application itself to allow for the screens and their components to be customized. Section 4.4 provides a guide that states which programmatic changes need to be done in a mobile app in order to apply the Direct UI Modifications option. Finally, Section 4.5 presents related work on the customization of user interfaces via adaptive UIs and the usage of RBAC.

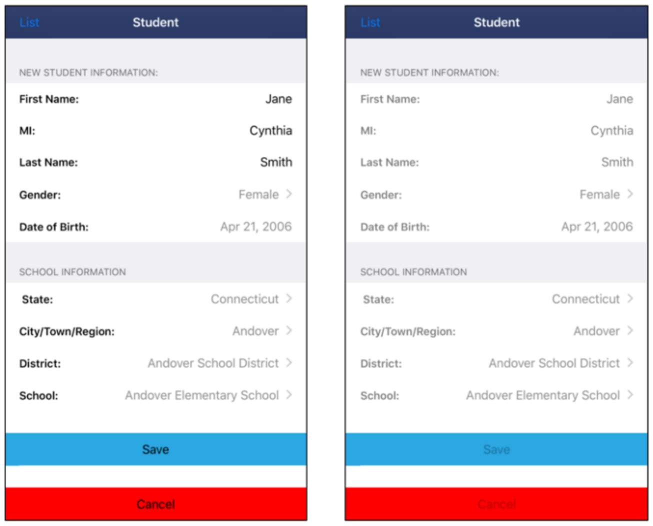
4.1. Reviewing the Unified Model and Permissions

In this section, we discuss the features and characteristics of the UI of a mobile application using the unified model from Chapter 3 in order to define what is viewable and/or modifiable on a role-by-role basis. Recall that, a mobile application's user interface will be comprised of a series of inter-connected screens. Each screen will have a set of different components that could have: information that is displayed (cannot be changed); information that can be entered by a user (text fields, drop downs where one value is chosen; checkboxes where multiple values are chosen, etc.); and, buttons that are utilized to effect the state of the application (save, cancel, next, previous, etc.). Users will be authorized to access a subset of the screens (that may occur in a particular sequence) with defined permissions for the various components that comprise each screen defining access (view/modify) depending on role. For a given mobile application with these capabilities, the RBAC approach that we present in this chapter can enable/disable the components based on a user's role. In the concussion app, a user with a Nurse role has access to the full capabilities of the CT^2 mHealth application, while other users with different roles would be limited; this was illustrated in the examples for screen, components, and screen interactions. These different users with different roles receive a dynamically customized version of the CT^2 mHealth application. The advantage of this is that user permissions can be configured based on role; therefore, the application does not need to be configured for each individual user, but will operate by role against the user instance (e.g., nurse or athletic trainer) that has been authorized.

Given this overview, the main focus of this chapter is to explore the way that permissions on screens, components, and screen interactions (see Defns. 13, 14, and 15, respectively, in Section 3.3) can be defined and enforced. A mobile application's UI will be comprised of a series of inter-connected *screens* where each screen contains a portion of the functionality (Defns. 1 and 5). Each screen of the UI will have a set of different *components* (Defns. 3 and 4) consisting of information that is displayed (cannot be changed) and information that can be entered by a user including: text field (TF), button (BN), drop down (DD), checkboxes (CB), radio buttons (RB), spinner (SP), date picker (DP), etc. A mobile application can have one or more screens and screens can have one or more of the aforementioned components.

Utilizing these definitions as a basis, users will be authorized by role to access a subset of the screens (screen permissions in Defn. 13) with defined permissions for each screen in order to limit and control the access to each screen's components, where our approach can enable/disable the components based on a user's role (component permissions in Defn. 14). In addition, the interactions of screens (flow from one screen to another) can be controlled by a user's role (screen interaction permissions in Defn. 15). As result, a user as owner by his/her role could have full access to the application, say Karen with Nurse role has access to all of the screens, while a user with a parent role is limited to the 'List', 'Home', 'Student', 'Cause', and 'Symptoms' screens of the iOS CT2 app in Figure 2.8. The end result is the ability to control which components of the application's UI users can have access to (view/edit) depending on their role. The advantage of this is that user permissions can be configured based on his/her role; therefore, the application does not need to be configured for each individual user, but will operate by role against the user instance that has been authorized.

Permissions, as presented in Section 3.3, can be defined against a generalized structure of a mobile application's UI screens and their components to customize which screens and their respective components are available in the mobile application, depending on the role a user assumes. The screens and the components are the objects that will be authorized as screen, component (text fields, dropdown box, date picker, radio buttons, check boxes, buttons), and screen interaction permissions to a particular role. This essentially defines what a role can and cannot do in terms of screen, component, and screen interaction permissions and determines whether the user with such role can access and/or view a certain component. As an example, the screens shown in Figure 4.1 from CT^2 (see Figure 2.8) has the 'Student' screen (left) customized on the right side of Figure 4.1 where all of the fields are disabled. In this case, the user with the role would be able to view information but would not be able to make any changes to the aforementioned disabled components. The permissions that are defined on the components of a screen are placed in two main categories: *on/off permissions* that are for components that can be 'on' (enabled) or ‘off’ (disabled); and, *data permissions* that are for components that can be ‘view’, ‘edit’, or ‘edit once’. On/Off permissions are defined for the different components: button (BN), radio buttons (RB), drop down (DD), checkboxes (CB), date picker (DP), spinner (SP), and text fields (TF), while data permissions are defined for text fields (TF). A text field has to be On in order for view/edit/edit once to be defined. As a further example, the three screens shown in Figure 4.2 show full access to all components (first screen) and then the edit button is restricted by disabling the button (second screen) or by hiding the button (third screen).


List Student

NEW STUDENT INFORMATION:

First Name:	Jane
MI:	Cynthia
Last Name:	Smith
Gender:	Female
Date of Birth:	Apr 21, 2006

SCHOOL INFORMATION

State:	Connecticut
City/Town/Region:	Andover
District:	Andover School District
School:	Andover Elementary School

Save

Cancel

List Student

NEW STUDENT INFORMATION:

First Name:	Jane
MI:	Cynthia
Last Name:	Smith
Gender:	Female
Date of Birth:	Apr 21, 2006

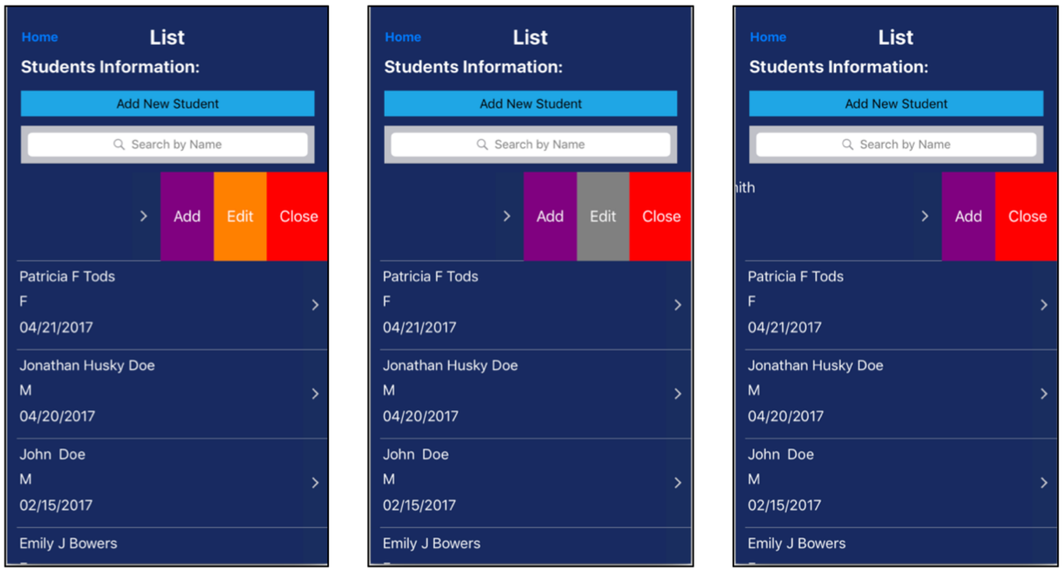
SCHOOL INFORMATION

State:	Connecticut
City/Town/Region:	Andover
District:	Andover School District
School:	Andover Elementary School

Save

Cancel

Figure 4.1. A Screen with Components (left) that are Customized (right) in CT^2.



Patricia F Tods
F
04/21/2017

Jonathan Husky Doe
M
04/20/2017

John Doe
M
02/15/2017

Emily J Bowers

Patricia F Tods
F
04/21/2017

Jonathan Husky Doe
M
04/20/2017

John Doe
M
02/15/2017

Emily J Bowers

Patricia F Tods
F
04/21/2017

Jonathan Husky Doe
M
04/20/2017

John Doe
M
02/15/2017

Emily J Bowers

Figure 4.2. A Screen with the Edit button enabled (screen 1), disabled (screen 2), and hidden (screen 3).

To utilize as an example, the Nurse role permissions as given in Table 3.5 have been augmented with a new Sub_Nurse role as shown in Table 4.1. The Sub_Nurse role is also a role that could be held by a nurse user such as Karen, and is intended to be for those nurses that are substituting at the school for one day. As a result, the role set for CT^2 now contains: $R_{CT}{}^2 = \{r_1 = <r_{ID1}, AT>, r_2 = <r_{ID2}, Coach>, r_3 = <r_{ID3}, Nurse>, r_4 = <r_{ID4}, Parent>, r_7 = <r_{ID7}, Sub\_Nurse>\}$ (see Defn. 7 again) and a new user role authorization can be added for Karen, $ura = <u_{Id1}, r_{ID7}>$, which means that Karen can play either the Nurse or Sub_Nurse role, and is able to delegate both roles. The permissions for the Sub_Nurse role are given in Table 4.1; notice that the role is no longer allowed to modify any information (only View) and has limited buttons enabled. An example of the way that the 'Student' screen would look like to a user with a Sub_Nurse role is depicted on the right screen in Figure 4.1.

Screens/Components	Nurse	Sub_Nurse
Home Tab	**Show**	**Show**
‘Enter New Student’ BN	Enabled	Enabled
‘Retrieve Open Cases’ BN	Enabled	Disabled
‘Last Name’ TF	View/Edit	View
‘First Name’ TF	View/Edit	View
‘Search’ BN	Enabled	Enabled
List Tab	**Show**	**Show**
‘Enter New Student’ BN	Enabled	Enabled
‘View Student Info’ BN	Enabled	Enabled
‘Edit’ BN	Enabled	Disabled
‘Add’ BN	Enabled	Disabled
Student Tab	**Show**	**Show**
‘First Name’ TF	View/Edit	View
‘Middle Initial’ TF	View/Edit	View
‘Last Name’ TF	View/Edit	View
‘Gender’ DD	View/Edit	View
‘Date of Birth’ SP	View/Edit	View
‘Date of Past Concussions’ DD	View/Edit	View
‘State’ DD	View/Edit	View
‘City/Town/Region’ DD	View/Edit	View
‘District’ DD	View/Edit	View
‘School’ DD	View/Edit	View
‘Save’ BN	Enabled	Disabled
‘Cancel’ BN	Enabled	Disabled
Cause Tab	**Show**	**Show**
‘Location of Incident’ DD	Enabled	View
‘If Sport’ DD	Enabled	View
‘Others/Details’ TF	View/Edit	View
‘Contact Mechanism’ DD	Enabled	View
‘Impact Location of Head’ DD	Enabled	View
‘Head Gear Usage’ DD	Enabled	View
‘Save’ BN	Enabled	Disabled
‘Cancel’ BN	Enabled	Disabled
Symptom Tab	**Show**	**Show**
‘Mild and Severe Symptoms’ BN	Enabled	View
‘Hour(s)’ TF	View/Edit	View
‘Minute(s)’ TF	View/Edit	View
‘Second(s)’ TF	View/Edit	View
‘Were Parents Notified?’ DD	Enabled	View
‘Removed From Activity’ DD	Enabled	View
‘Removed by’ DD	Enabled	View
‘Concussion Assessment Tool’ DD	Enabled	View
‘Additional Comments’ TF	View/Edit	View
‘Save’ BN	Enabled	Disabled
‘Cancel’ BN	Enabled	Disabled
Follow Up Tab	**Show**	**Show**
‘Lingering Symptoms’ BN	Enabled	View
‘If Other, Please Specify’ TF	View/Edit	View
‘All Symptoms Resolved in’ DD	Enabled	View
‘Concussion Diagnosed by’ DD	Enabled	View
‘Post Concussive Syndrome’ DD	Enabled	View
‘Medical Imaging’ DD	Enabled	View
‘Additional Comments’ TF	View/Edit	View
‘Save’ BN	Enabled	Disabled
‘Cancel’ BN	Enabled	Disabled
Return Tab	**Show**	**Hide**
‘Days Absent From School’ TF	View/Edit	View
‘Schedule/Activity Modification’ DD	Enabled	View
‘504 Plan Required’ DD	Enabled	View
‘Date of Return to Learn’ SP	Enabled	View
‘Date of Return to Full Part.’ SP	Enabled	View
‘Save’ BN	Enabled	Disabled
‘Cancel’ BN	Enabled	Disabled

Table 4.1. Screen and Component Permissions for Nurse and Sub_Nurse roles.

4.2. ER Diagram Subset for Unified Model

This section discusses a subset of the entity relationship diagram of Figure 3.3 in Section 3.7 that realizes the subset of the unified security model from Sections 3.1 to 3.5.

that supports the Direct UI Modifications option. To support the security enforcement generation process for the Direct UI Modifications option, a subset of Figure 3.3 is shown in Figure 4.3 and is able to represent screen, component, and screen interaction permissions (see Defns. 13, 14, 15, and 16 in Section 3.3 of Chapter 3), for each role (see Defns. 6 and 7 in Section 3.2 of Chapter 3). As needed, revisit Table 3.11 for an explanation of each entity and its relationship to other entities and the unified security model. Figure 3.3 represents the generalized structure of the mobile application (via the entities `mobile_apps`, `screens`, `screen_components`) and the permissions (via the entities `users`, `roles`, `user_roles`, `delegation_permissions`, `screen_permissions`, `component_permissions`, and `screen_interactions`). In support of RBAC and as shown in Figure 4.3, the *`mobile_apps`* entity contains the name and unique identifier of the mobile app, and the *`users,`* entity contain information the name and identity of all users. Next, the *`roles`* entity contains the name of the roles (`role_name` field in the `roles` entity shown in Figure 4.3) that are available as well as a unique `role_id` assigned to each one of these. To keep track of the roles for each user, the *`user-roles`* entity is utilized; remember a user can have multiple roles but be restricted to a single role. This `role_id` is utilized to determine if a role has access to a specific screen of a UI, to a component in that UI screen, and the sequence of screens it is allowed to view. Since screens, components, and screen interactions could be accessed by more than one role we define the `screen_component_roles` entity which maps the subset of role ids that have access to a screen, component, or screen interaction by assigning an id (represented as the `role_mapping_id` field in the `screen_component_roles` entity shown in Figure 4.3) to the subset and adding this id as a foreign key in the `screen_interactions`, `screen_permissions`, and `screen_components_permissions` entities. This `role_mapping_id` would leverage the prior pharmacy example, where the pharmacy technician role would only be allowed to access UI screens 1, 2, and 4. If a role by `role_id` has access to a specific screen (the role is within the subset of roles allowed to access the screen), then the second step of the permission process would be to define components of the screen that such role can access; otherwise, if `role_id` is not assigned to a screen (the role is not within the subset of roles allowed to access the screen), the screen and its components are hidden.

In this model, `users` can only have one role, and roles can have one or more permissions. In order to capture permissions, the entities `screen_permissions`, `screen_components_permissions`, and `screen_interactions` are utilized, where: the *`screen_permissions`* entity supports the definition of the permission for a role with respect to the entire screen (Defn. 13 in Section 3.5 of Chapter 3), the *`component_permissions`* entity supports the definition of the permissions for a role with respect to the components of a screen (Defn. 14 in Section 3.5 of Chapter 3), and the `screen_interactions` entity supports the definition of the permissions for a role with respect of allowable screen sequences (Defn. 15 in Section 3.5 of Chapter 3). To bring the concepts together, Figure 4.4 illustrates the authorization process that assigns a user one or more roles (while limiting a user to one identified role per session) and optional delegation permissions and then defines screens, their components, and their interactions on a role-by-role basis against all mobile application screens/components. Note that both the on/off and data permissions are for the components that are captured in the `screen_components_permissions` entity.

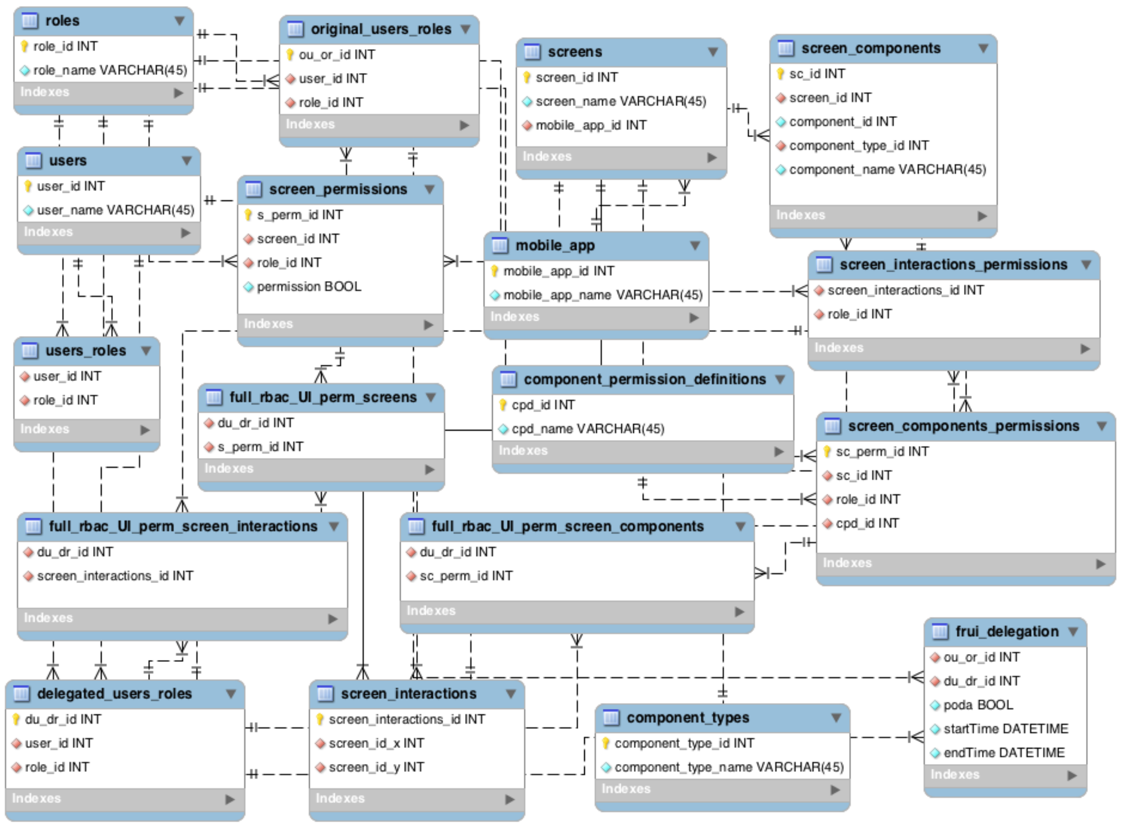


Figure 4.3. A Subset of the ER Diagram from Figure 3.3. for Supporting the Unified Security Model for the Direct UI Modifications Option.

In support of DAC and as shown in Figure 4.3, there are three entities that hold the necessary data required for delegation permissions. First, the `original_users_roles` entity holds the original user's identifier (`user_id`) and the role (`role_id`) the user is allowed to delegate (Defns. 22 and 23 in Section 3.5 of Chapter 3). Second, the `delegated_users_roles` entity holds the delegated user's identifier (`user_id`) and the role (`role_id`) that the delegated user can receive as part of their delegated permissions (Defns. 24 and 25 in Section 3.5 of Chapter 3). Third, the `frui_delegation` entity holds the permission rules for Full RBAC UI (FRUI) Delegation (Defn. 28 in Section 3.5 of Chapter 3). Basically, `ou_or_id` and `du_dr_id` act as foreign keys for the data stored in the `original_users_roles` entity and the `delegated_users_roles` entity, respectively, in order to determine which original user (`ou_or_id`) is delegating his/her role, screen permissions, component permissions, and screen interactions to which delegated user (`du_dr_id`). The `poda` field is a Boolean value that determines whether the delegated user can pass on the delegated permissions to another delegated user (`poda` = true) or not (`poda` = false) (Defn. 27 in Section 3.5 of Chapter 3). The last two fields of the `frui_delegation` entity, represent when the delegated user can start to have access to the delegated permissions (`start_time`) and when the access to such permissions end (`end_time`) the period of time that a delegated user has the access to the delegated permissions. Moreover, the `full_rbac_ui_perm_screen`, `full_rbac_ui_perm_screen_components`, and `full_rbac_ui_perm_screen_interactions` entities hold the delegated screens permissions, delegated screen components permissions, and delegated screen interactions permissions, respectively, for delegated users.

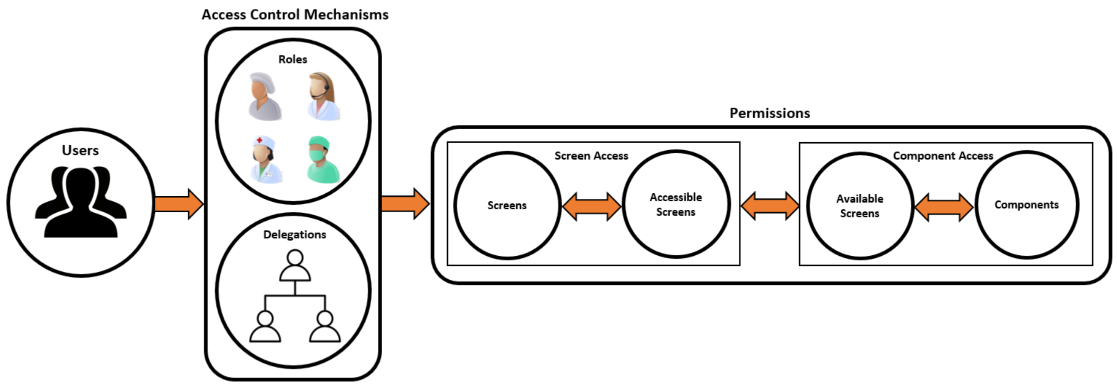
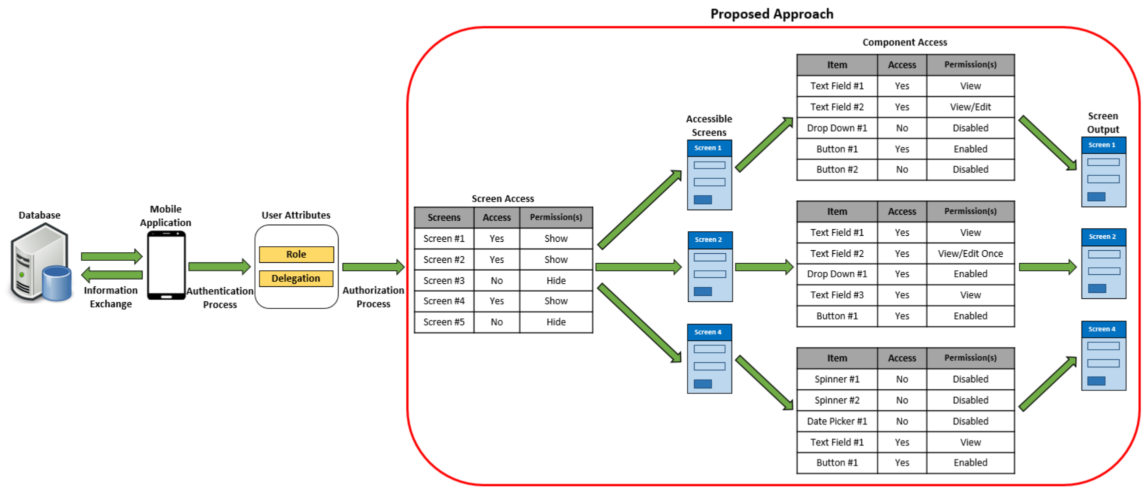


Figure 4.4. Authorization Process with respect to Screens and Components.

The authorization policies for a generalized mobile application as defined via the process in Figure 4.4 are then enforced as shown in Figure 4.5 The enforcement process begins with the mobile application authenticating the user, verifying the credentials, and then retrieving the user's access control attributes to customize the mobile application

(Defns. 11, 12, and 28 in Sections 3.2 and 3.5 of Chapter 3). The right hand side of Figure 4.5 (red box) utilizes the data in the screen permissions, screen components permissions, screen interactions, and delegation permissions entities (see Figure 4.3 again) to determine a custom version of the UI of the mobile application by role and optional delegation permissions (Defns. 16 and 28 in Sections 3.3 and 3.5, respectively). Notice that the screen access instances are shown for a role and delegation with screens 1, 2, and 4 authorized. The *Accessible Screens* part of Figure 4.5 illustrates a basic idea of the screen permissions (Defn. 13 in Section 3.3 of Chapter 3). Also, the *Component Access* table on the right side of Figure 4.5 illustrates the components of the aforementioned screens that have been authorized to a specific role, thereby realizing the component permissions (Defn. 14 in Section 3.3 of Chapter 3). There are no screen interaction permissions in Figure 4.5, since we assume the mobile application consists of a set of tabs.



Screen Access

Screens	Access	Permission(s)
Screen #1	Yes	Show
Screen #2	Yes	Show
Screen #3	No	Hide
Screen #4	Yes	Show
Screen #5	No	Hide

Component Access

Item	Access	Permission(s)
Text Field #1	Yes	View
Text Field #2	Yes	View/Edit
Drop Down #1	No	Disabled
Button #1	Yes	Enabled
Button #2	No	Disabled

Item	Access	Permission(s)
Text Field #1	Yes	View
Text Field #2	Yes	View/Edit Once
Drop Down #1	Yes	Enabled
Text Field #3	Yes	View
Button #1	Yes	Enabled

Item	Access	Permission(s)
Spinner #1	No	Disabled
Spinner #2	No	Disabled
Date Picker #1	No	Disabled
Text Field #1	Yes	View
Button #1	Yes	Enabled

Figure 4.5. Enforcement Process for a Mobile Application.

For RBAC, the permissions as captured in Figure 4.3 support the four API calls of the prior section and are executed in the CT^2 mobile application's code to enforce the permissions of each role to determine the screens and their components for each user by role. CT^2 utilizes a MySQL database to store its data and relies on API commands to retrieve data for display on the UI and to store new concussion incidents (or changes) into the database. Figure 4.6 illustrates the result of the screen and component permissions for the *Nurse*, *Parent*, and *Coach* roles. The *Nurse* role has all of the tabs active (first screen of figure 4.6). The *Parent* role has limited access to the tabs and also, although users with the *Parent* role can view the 'Symptom' and 'Follow Up' tabs, they are not allowed to update that information (second screen of figure 4.6). Finally, the *Coach* role has the first four tabs active and has limited access to the information it can view/modify (third screen of figure 4.6).

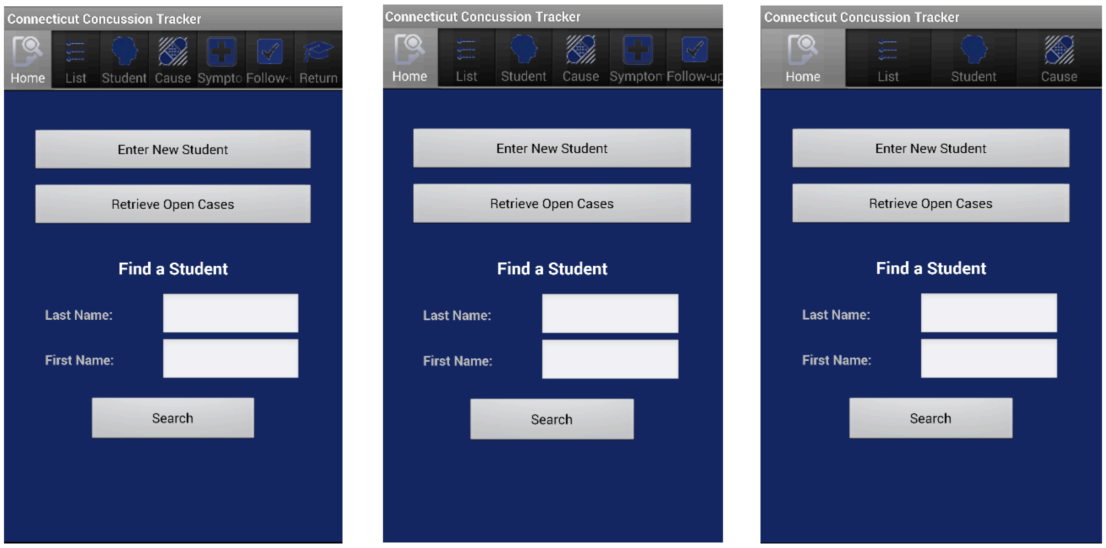


Figure 4.6. Result of RBAC in Connecticut Concussion Tracker (*Nurse*, *Parents*, and *Coach* view).

For DAC, delegation is a process that is initiated by a user in order to pass on credentials to another user for a restricted period of time. From the DAC model definitions in Section 3.5, in support of DAC for the UI, the relevant definitions are: Defns. 22-27 for original user, original role, delegated user, delegated role, delegation authority, and pass on delegation authority, respectively. Only Full RBAC UI Delegation, as given in Defn. 28, is supported, which means that an original user must delegate all of the permissions associated with his/her original role, to a delegated user with delegated role with possible pass on delegation authority for a given time period. Assume that the original user ou Karen $< u_{ID1}, \mathit{Karen}, \mathit{TS}, \mathit{SS}\text{-}r, L^{*}\text{-}w >$ has two user role authorizations ura = $\{<u_{ID1}, r_{ID3}>, <u_{ID1}, r_{ID7}>\}$ for her Nurse and Sub_Nurse role and is interseted to delegating the nurse or to the delegated user du Lois, a substitute school nurse for one day: $\mathit{del} = < u_{ID1}, r_{ID7}, u_{ID5}, r_{ID7}, < \gamma, \chi, \lambda >, \mathit{false}, \{\mathit{2017\text{-}07\text{-}31T09{:}00{:}00{+}00{:}00}, \mathit{2017\text{-}12\text{-}15T07{:}00{:}00{+}00{:}00}\} >$. Note that $< \gamma, \chi, \lambda >$ is as defined in the Sub_Nurse column in Table 4.1. Karen is interested in only giving Lois the ability to review information on current students and not be able to enter new students that have concussions that day at the school.

4.3. Access Control API and Programmatic Changes to UI

The final part of the direct UI modifications option explains the programmatic changes that must be made to the mobile application itself to allow for the screens and their components to be customized. This involves the definition of an API for access control enforcement using the database of Section 4.2 coupled with programmatic changes that are made just one time. Since the permissions are taken from the database of Figure 4.3, these can be changed and the mobile application's UI will adjust the look-and-feel based on the defined role/delegation permissions without any additional code-level changes. The entity-relationship diagram in Figure 4.3 was realized as a relational database using MySQL. To support interactions from the mobile application to the MySQL database of permissions, an Access Control Application Programming Interface (API) as given in Table 4.2 was created which can be utilized to support the application wrapper for the direct UI modifications option as illustrated in Figure 1.4. Moreover, additional API calls were created in order to retrieve general information about the available access control policy as given in Table 4.3. The Access Control API calls are invoked within the mobile application's source code and return data from the queries in JSON format.

There are four main API calls that will constitute the wrapper that has been defined against the entities in Figure 4.3 and realized in the MySQL database, which are briefly described along with the incorporation of their usage within the code of the mobile app. The first API call, `GET /screenaccesses/:roleID/:screenID`, returns a Boolean value true (JSON format: [{"access":"1"}]) if the role represented by `role_id` has the permission to display a particular UI screen `screen_id`, and false otherwise (JSON format: [{“access":"0"}]). This first API call in Table 10 queries the `screen_access` entity of Figure 4.3 which was illustrated in Figure 18, and would return true for screens 1, 2, and 4, and false for screens 3 and 5. This requires a change to the mobile application code to include a conditional statement that only displays a particular screen of the UI based on the `screen_id` and `role_id` if there is permission defined in the `screen_access` entity.

The second API call in Table 4.2, `GET /returnAllowableComponents/: roleID/:screenID`, returns a table of component permissions for the `role_id/screen_id` combination that identifies the on/off and data permissions on buttons, spinners, date spinners, radio buttons, checkboxes, drop downs, and text fields. The API call begins by querying the `component_access` entity of Figure 4.3, which was illustrated in Figure 4.5 by the three permission tables for screens 1, 2, and 4 (*Component Access* table in the figure). Then, the API call retrieves the component permissions for a single screen of the UI that is authorized to that role. As part of the process to display the components of an allowable screen `s`, the mobile application's code is modified with conditional statements for the various components of each screen. Specifically, for the on/off permissions, the button (BN), radio button (RB), drop down (DD), checkbox (CB), date picker (DP), and/or spinner (SP) components are disabled for all non-allowed actions which are the 'no' entries as shown in the *Component Access* table of screen 1 for Figure 4.5. For the data permissions, each text box is set accordingly based on View, Edit, or Edit once. Note that the APIs are not called in sequence rather are utilized in multiple locations throughout the mobile application's source code.

The third API call in Table 4.2, `GET /screensequences/:roleID`, utilizes a `role_id` to look up all of the allowable screens (via a database query to the `screen_access` entity for `role_id`) and using this information, returns the sequences of permissible movement/interactions among all allowable screens of the UI. First, the API call utilizes the `screen_interactions` entity in Figure 4.3 in order to find all of the allowed interactions among screens of the UI for `role_id` and only enable those interactions that occur among the allowable screens. The API call is then utilized to set behavior related information to buttons (BN) on a particular screen. For a button that is enabled, the information in `screen_interactions` for the given role and its allowable screens will allow the button to cause the screen to be reached; a button not enabled will not link to another screen.

The fourth API call in Table 4.2, `GET /delegationPermissions/:userID`, takes the user's id as a parameter in order to determine if the user in session is a delegated user. If the user has been delegated with permissions, then the API call will return the delegated role the user can assume; the screen permissions, component permissions, and screen interaction permissions of the original user (user who delegated his/her permissions to the delegated user); and, the period of time the delegated user is allowed to access the delegated permissions.

Access Control	Service Name	Description
RBAC	GET /screenPermissionsRBAC/:roleID/:screenID	Returns if a role has permission to access a screen or not
	GET /allowableComponentsRBAC/:roleID/:screenID	If the specified role has access to the specified screen then this service returns the components the role has access to of the screen
	GET /screensequences/:roleID	Gets the allowed screen sequences for an specific role in a mobile application
DAC	GET /delegationPermissions/:userID	Checks if user has delegate permissions and, if so, returns the delegation permissions the user has (delegated role, screen permissions, component permissions, and screen interaction permissions).

Table 4.2. API Services for RBAC & DAC Security Enforcement.

Service Name	Description
GET /screens	Gets all of the screens of a mobile application
GET /screens/:screenId	Gets a screen of a mobile application by screen id
GET /screens/:screenname	Gets a screen of a mobile application by screen name
GET /screenobjects	Gets all the components of all screens in a mobile application
GET /screenobjects/:screenId	Gets the components of a screen by screen id
GET /screenobjects/:objectName	Gets a component by component name
GET /screenobjects/:objectID	Gets a component by component id
GET /screenobjectslabels/:screenID	Gets the names of all the components of a screen in a mobile application
GET /screensequences	Gets all of the possible screen sequences in a mobile application
GET /screensequences/:objectID	Gets the allowed screen sequences for an specific component action in a mobile application
GET /screensequences/:screenID	Gets the allowed screen sequences for an specific screen action in a mobile application
GET /delegationPermissions/ou	Gets the user id and role id of all the original users of a mobile application
GET /delegationPermissions/du	Gets the user id and role id of all the delegated users of a mobile application

Table 4.3. Additional API Services for the Direct UI Modifications Option.

To this point, we have shown the different components that our approach contains and the way that these are incorporated in a mobile application in support of the direct UI modifications option via an application wrapper. Next, we review the way that a mobile application maintains its functionality after adding these direct changes. First, we identify the screens and the objects of a screen of the mobile application to which RBAC is to be applied. Then, we assign a unique id to each of these components and store them as tables in a database as shown in Figure 4.3. The role of the user is retrieved and stored in a secured session variable (passed over https) through the means of an API call (part of the application wrapper) and, the identity of the user is verified at each API call the user makes.

Using this as a basis, to enforce the policies established in the database with the mobile application, we create a set of API calls. These calls will return if a component can be shown/edited or if it needs to be disabled/hidden. Each call will check if the component that we are trying to apply the policy to exists in the database; if it doesn't, then the API call will return that the component is enabled for the role that is in session as no RBAC permissions were found in the created policy. In support of DAC, if the user is a delegated user, we assign him/her the role of the original user and grant him/her access to all the permissions of the original user for the specified period of time by the original user. By making these changes, the functionality of the mobile application will not be affected since the API calls that are being added to the source code of the mobile application always return a value regardless if the component does not have a policy stored in the database anymore. In addition, storing the role of users in a secured session variable will prevent a user from tampering its role and it does not require any changes in the source code.

Note that the approach presented in this section supports the Direct UI Modifications option that was part of the Figure 1.4 configurable framework for RBAC. The direct UI modifications option focuses on RBAC (with optional DAC) of the UI of a mobile application's screens and their components, and then customizes the look-and-feel by role that is defined with varied screen, component, and screen interaction permissions stored in a database. When permissions change, only the database needs to be changed, and the mobile application will adjust appropriately. However, one disadvantage of the direct UI modifications option is that programmatic changes are required in the mobile application itself through the addition of condition statements and calls to APIs that return allowable screens and permissions on components in order to adjust the look-and-feel of the mobile application (via the application wrapper).

4.4. A Guide for Programmatic Changes in a Mobile App

In this section, we discuss the programmatic changes that need to be performed in a mobile app that is utilizing our Direct UI Modifications option to include permission checks at the code level. The programmatic changes require:

1) **Identification and storage of security policies in a ER Diagram of Figure 3.3 and subset as given in Figure 4.3**: The first step requires the designer(s)/developer(s) of the mobile app to identify the screen(s) that are to be securely controlled that includes the respective components on each screen. The second step requires the designer(s)/developer(s) to define a set of roles/delegation capabilities that are available in the mobile app and assign one of the defined roles/delegation permissions to each of the users of the mobile app. The third step is for the designer(s)/developer(s) to define and generate the permissions by set of roles for each of the screens/screen components/screen interactions (see Defns. 13-16 in Section 3.3 of Chapter 3) and to store these permissions in the data source accessible to the mobile app.

2) **Implementation of the Access Control API Calls:** Developer(s) need to incorporate the RBAC and DAC API calls server-side as shown in Table 4.2 in order to allow the source code of the mobile app to access the security policies stored in the data source via these calls.

3) **Retrieval of the security policies utilizing the source code of the mobile app:** These programmatic changes require the developer(s) to obtain the security policies in the source code of the mobile app and utilize them throughout the code by adding the API

services into the mobile application code-base so that they may be called by sending requests in JSON format. The response of the JSON request will also be in JSON format.

4) Enforcement of the security policies throughout the source code of the mobile app: After the developer(s) add the code that allows the communication between the security policies stored in the data source and the mobile app, the next step is to enforce the obtained security policies throughout the mobile app. The security policies can contain delegation permissions, screen permissions, screen component permissions, and screen interactions permissions. Each one of these permissions must be carefully inserted in specific locations of the mobile app code:

- *Screen permissions*: These permissions need to be retrieved after a user manages to log in successfully into the mobile app and before the screens of the app are displayed. The screens are obtained using a *for* loop and, if the user does not have access to a screen based on his/her role, then that screen is hidden. If he/she does have access to a screen, then the screen id will be stored in an array to be utilized at a later point to obtain the screen component permissions for those allowable screens.
- *Screen Component permissions*: Once the array of allowable screens is defined, the developer must create a file that maps the components that are found throughout the source code with the components stored in the database by assigning the same id to both. Then, for every component that has a permission, the developer needs to insert if/else conditions that determine what actions a user can do with his/her role. The conditions can be performed as follows:

    - if the user has permission to edit/click the component, then display the component in the mobile application;
    - if the user can edit the component once, then display the component once and afterwards send a flag to the database indicating that the specified user cannot edit the component anymore;
    - if the user can only view the contents of a component, then such components needs to be disabled permitting the user to view but not modify the component;
    - otherwise, if the user does not have permission to access the component, then the developer can either disable the component (and don't show the contents) or hide the component.

- *Screen Interactions permissions*: These type of permissions are utilized when a user is requesting to access another screen through the means of a button. To support screen interactions, the developer needs to call the screen interaction service using the component id of the button and the role of the user as parameters when the user clicks the button (in the function that determines the action to be done when the button is clicked). This will return to which screen the user can move on as a result of clicking a navigation button.
- *Delegation permissions (Optional)*: When a user logs in successfully to the mobile app, we need to verify if he/she is a delegated user. If the user is in fact a delegated user, we assign the delegated role to such user and continue verifying the remaining permissions with this assigned role.

Sample programmatic changes related to screen permissions for the CT^2 mobile app are given in Appendix A.

4.5. Related Research of Customizing UIs

This section reviews related research in the customization of user interfaces in a number of different areas: adaptive UIs (Eisenstein, Vanderdonckt, & Puerta, 2001; Aikiki, Bandara, & Yu, 2012) that are characterized by the ability to change based on the needs of different users, similar to our approach to customize the UI by role; two efforts (Lin & Speedie, 2003; Aikiki, Bandara, & Yu, 2013) that create roles and modify the UI software in an automated process via specialized IDE; and, the work of Stormpath (Hazelwood, 2012) for code level modifications to support RBAC. The two efforts in adaptive UI (Eisenstein, Vanderdonckt, & Puerta, 2001; Aikiki, Bandara, & Yu, 2012) are targeting a versatile modifiable UI. The first work (Eisenstein, Vanderdonckt, & Puerta, 2001) focuses on changing the UI and capabilities depending on three components: on the platform model (computer systems that can run the UI), on the presentation model (visual appearance of the UI), and on the task model (representation of the tasks a user may want to perform in the software) of the mobile device. While the aforementioned proposed approach is able to change the components of a software depending on the three components mentioned, it does not enforce access control meaning that all users of the software have access to all of the features different to our work which changes the look-and-feel of a mobile application depending on a user's role/delegation permissions. The second work (Aikiki, Bandara, & Yu, 2012) attempts to modify the UI per user needs of usability; to achieve this, the work introduces a tool controlled by a developer in which he/she can add code and modify the UI's components. In addition, the approach mentions RBAC to secure resources (which in this case is the UI) but does not go into details about the way that the UI would look like if access control was applied. Our proposed approach is automatic meaning that the developer need to place the security checks throughout the source code of the mobile app only once and after that a security administrator could modify the UI permissions through the means of a separate UI and, our approach addresses the way that the UI of the mobile app will look after access control mechanisms have been enforced.

The next two efforts combine and use adaptive UIs in mobile devices in conjunction with RBAC (Lin & Speedie, 2003; Aikiki, Bandara, & Yu, 2013). The first work (Lin & Speedie, 2003) has proposed an approach that establishes roles and permissions for those roles in order to show a user only the features he/she has access to. Nevertheless, the approach is more concerned about only showing the features that users might use instead of showing the user all of the features he/she has access to like our proposed approach does. The second work (Aikiki, Bandara, & Yu, 2013) is an extension of (Aikiki, Bandara, & Yu, 2012). Basically now, instead of allowing users to see all of the features of the software, the approach focuses on creating roles and modifying the UI of the software according to these through the means of an IDE the authors developed. Apart from this, the approach also focuses on how the components of the software would look like depending on the user's role. For example, a software has two roles: the *Sales Officer* and the *Novice*. Both roles can view the same features but they will have access to them in a different way since one user is more experienced than the other. We consider that while the aforementioned approach contains adaptive UI capabilities based on role it creates additional overhead since they need to determine which UI to show a user at runtime in addition to determining which components the role of the user has access to.

In terms of related work in RBAC in UI on mobile apps, an article presented by Stormpath (Hazelwood, 2012) consists of modifying code in an application in order to decide who has access to a specific resource. For example, suppose we have a mobile app that contains two roles: *teacher* and *student*. An implicit way of saying that these roles have access to a specific resource would be to do an *if* condition that states that if the user has the role of *teacher* or if he/she has the role of *student* then the resource is available, otherwise the resource (API) is hidden. Nevertheless, this approach does not handle the case where the owners of the application decide to add a new role (*parent*) that also has the permission to access that specific resource. In that case, the developers would need to go back to the source code and add to the condition that the role of *parent* is also allowed to view the resource. One solution to this problem would be to create an *if* condition that takes the username/role of the user as well as the id of the resource. This information gets processed against a security policy, and, depending on which permissions are defined, that specific user/role combination will be granted/denied permission to the resource. Therefore, the article argues that utilizing explicit access control is a better approach than to utilize implicit access control. In other words, instead of hard coding which roles have permission to a certain resource, the resource itself will be the one checked. This leads to a number of benefits: the amount of coding can be reduced, the security model is more flexible, the security policies can be modified without making changes to the code, and, the resource is more protected. However, despite of that, the proposed solution intends to basically enforce RBAC capabilities in the UI of a mobile application it does not consider DAC which can be utilized to augment security as we do in our approach.

Chapter 5
The Intercepting API Calls Option

Health Information Exchange (HIE) provides a more complete health record with the aim to improve patient care with relevant data gathered from multiple health information technology (HIT) systems. In support of the emergence of cloud computing, in healthcare, the Meaningful Use Stage 3 (Himss, 2016) guidelines require all health information technology (HIT) systems (e.g., electronic health records (EHR), personal health records (PHR), etc.) to have API services to access, modify, and exchange health-related data. If services are the primary means of access, there must be a way to control who can invoke which service at which time. This necessitates the consideration of the usage of RBAC, MAC, and DAC to control access to the services that are utilized by a mobile application. To address the aforementioned, this chapter presents the Intercepting API Calls option of our configurable framework for controlling access to the API of the mobile app. Remember, from Section 1.4 of Chapter 1, there were two levels of API control: a security layer between the UI and mobile application API replicates the mobile application's API by creating a mirrored set of services that invoke the original API services so that each call can be intercepted to add RBAC, MAC, and or/ DAC security checks as presented in this chapter; and, a second security layer between the two different APIs (mobile app and server-side) is accomplished through the creation of a server interceptor API associated with a cloud computing infrastructure to intercept invocations for RBAC, MAC, and DAC checks, to be presented in Chapter 6. We have evolved RBAC and MAC to support permissions on services (as opposed to the usual object view) at a model level applied to a setting where a mobile application is using RESTful APIs and, by adding delegation permissions to services in support of DAC. The resulting RBAC, MAC, and DAC service-based model as presented in Sections 3.1 to 3.5 of Chapter 3 for the unified security model can be incorporated by creating mirrored APIs can intercepting calls from the mobile app to the mobile app API. The work in this chapter on the interceptor supports Contribution D: Access Control Security Enforcement Code Generation and Interceptors.

The remainder of the chapter has seven sections. Section 5.1 motivates the Intercepting API Calls option by explaining the important role of the API in accessing information, especially PPI and PHI. Section 5.2 presents the high-level processing of the Intercepting API Calls option using the classic architecture of the User Layer, Presentation Layer, Business Layer, and Data Layer. Section 5.3 explores the underlying processing of the Intercepting API Calls option by examining the way that API services are categorized. Section 5.4 examines the interactions and infrastructure for the Intercepting API Calls option. Section 5.5 explores the algorithm generation process for the Intercepting API Calls option. Section 5.6 illustrates the Intercepting API Calls option via the CT^2 mHealth application. Finally, Section 5.7 discusses related work in security and access control mechanisms for mobile applications.

5.1. Motivating the Intercepting API Calls Option

The idea behind the Intercepting API Calls option is to secure highly-sensitive information that is present in mobile applications and is accessible via an API. To support this focus, we assume that data transactions between a mobile app and a server are performed via an API. Through this mobile app API, we seek to provide a means for a user playing a role, and possibly contain a clearance, to be constrained to deliver/store data when utilizing the mobile app via the interception of the API calls. According to Cobb (Cobb, 2014), every API call should be verified to ensure that the user accessing the mobile app has the necessary permissions to manage the requested data. The Intercepting API Calls option makes use of the ability to define permissions on the services of the API in three different ways. First, the API can be partitioned into Secure/Unsecure services (see Defn. 17 in Section 3.4 of Chapter 3) where the Secure services can be assigned on a role-by-role basis (see Defn. 19 in Section 3.4 of Chapter 3), thereby supporting RBAC. Second, the API can be partitioned into Labeled/Unlabeled services (see Defn. 18 in Section 3.4 of Chapter 3) where each Labeled service has a classification and Labeled services can be assigned based on a user's clearance (see Defns. 20 and 21 in Section 3.4 of Chapter 3), thereby supporting MAC. Third, if an API is partitioned by using either RBAC or MAC, an original user (Defn. 22 in Section 3.5 of Chapter 3) or a delegated user with pass-on delegation authority (Defns. 24 and 27 in Section 3.5 of Chapter 3) can delegate a full (Defns. 29 and 31 in Section 3.5 of Chapter 3) or a partial (Defns. 30 and 32 in Section 3.5 of Chapter 3) set of their services to a delegated user, thereby supporting DAC.

The Direct UI Modifications option reviewed in Chapter 4 required custom programmatic changes that include conditional checks (user/role) and an access control security API, and as a result may not be possible in cases when the source code of the mobile app is unavailable. The Intercepting API Calls option requires minimal or no changes to the UI of the mobile application other than for the need to identify a given role for a session being initiated by a user. In this case, we incorporate the functionality of API calls into REST or API services that are utilized to intercept the API calls to disable the delivery of content to the user. Recalling the pharmacy example, the pharmacy technician could see all five screens, but information on screens 3 and 5 would be blocked in the display of data. For the case where the pharmacy technician attempts to utilize screens 3 and 5, if they do attempt to make a positive action to search or insert information, this would be intercepted at the server side to disallow the attempt. Basically, the access control checks on defined permissions that have been discussed for the approach in this chapter would be before the REST/API calls in the case of the Intercepting API Calls option and after the REST/API calls for the Server Interceptor API option (discussed in Chapter 6).

In addition, we acknowledge one of the most recognized options to display (deliver) and manage (store) dynamic data in a mobile app is to utilize the concept of API. However, before attempting to implement an API, one must evaluate their security risks and their effective management (Collet, 2015). For example, consider the recent security breaches in Snapchat and Instagram APIs. Snapchat, a mobile app that enables users to view and send self-destructive pictures and videos (Snapchat, 2011), had a data breach that affected 4.6 million users (Snapchat, 2013). The company quickly posted a statement revealing that the vulnerability allowed individuals to compile a database that contained usernames and phone numbers of users of the mobile app and, that this problem came from their private API. To address this issue, Snapchat is attempting to identify which third-party applications offered in the iTunes store and Google Play store are accessing their private API and any application that uses it is accessing Snapchat’s information without their permission (Zeman, 2015). Instagram, a mobile app that allows users to take pictures and share them with family and friends (Instagram, 2010), had a password breach in 2015 (Dellinger, 2015). The breach allowed a third-party application to steal more than 500,000 usernames and passwords, and used the information to post spam on Instagram accounts without permission. To remedy this, Instagram is now reviewing all of the applications that utilize their API and adding new usage policies (Larson, 2015). Clearly both public and private APIs need to be continuously secured and monitored to prevent disclosure of restricted information from occurring. To address this issue, a number of companies have added security and associated management mechanisms to APIs.

5.2. High-Level Processing of Intercepting API Calls Option

This section explores the high-level processing of the Intercepting API Calls option with an emphasis on the way that calls from the mobile application to the mobile app API are intercepted. The Intercepting API Calls option defines a new API that mirrors the original mobile app API (in terms of signatures) and serves as a wrapper and includes calls to the original mobile app API to proceed based on access control checks that control the data that is displayed (delivered) and managed (stored). In this section, the Intercepting API Calls option is explored in detail; this option offers the versatility of intercepting original API calls that have no impact on the source code of the mobile application. We differentiate between three different APIs in the discussion: the *mobile app APIs* that are used by the mobile app (original mobile app APIs); the *intercepting mobile app API* that has the same signatures as the mobile app APIs to replace these and provide permission checks; and the *renamed mobile app APIs* (former original mobile app APIs) that are wrapped by the intercepting mobile app API.

For the general architecture of a mobile app, we employ a client mobile app (Microsoft Corporation, 2008) augmented with the intercepting API-based approach. We focus on client applications since these are easier to maintain and assume that the app is always fully connected to the Internet. This assumes that all of the data is processed server-side and does not contain cache and local data. The architecture consists of four main layers as shown in the left side of Figure 5.1: the User Layer which symbolizes the users of the mobile application; the Presentation Layer which consists of the UI components of the mobile application; the Business Layer which contains the logic of the mobile app (e.g., libraries, APIs, source code); and, the Data Layer which contains all of the data the mobile app manages (e.g., retrieves, inserts). The right side of Figure 5.1 details the architecture of the intercepting API-based approach across the four layers in three groups. The first group, Role/Clearance/Delegations Assignment, involves the user layer and contains the users of the mobile app and their assigned roles/clearance/delegations. The second group, Define Access Control Permissions on API Services, spans the presentation and business layers and contains the original mobile app API services to retrieve/insert data from/into the data source. This group is utilized to define access control permissions on a role-by-role, clearance, and optional delegation basis on which mobile app API services are authorized to each role/clearance/delegation, which in turn is assigned to different users. Once access control permissions are defined on the mobile app API, our approach can intercept API services utilized by the mobile app in order to perform security and permissions checks. To transition from the second to third group, our intercepting API based approach utilizes the data layer as a pass via the renamed API service calls, and as a result, does not require modifying the source code of the mobile app in order to achieve. Lastly, the third group, Enforce Access Control Permissions on API Services, contains the RBAC, MAC, and DAC policies that need to be incorporated in the original data source(s) so that they can be enforced. This includes a new set of intercepting API services that must be defined and then utilized to replace the original mobile app API services to enforce the defined access control policies to control the data that is displayed (delivered) and managed (stored) on a user/role/clearance combination.

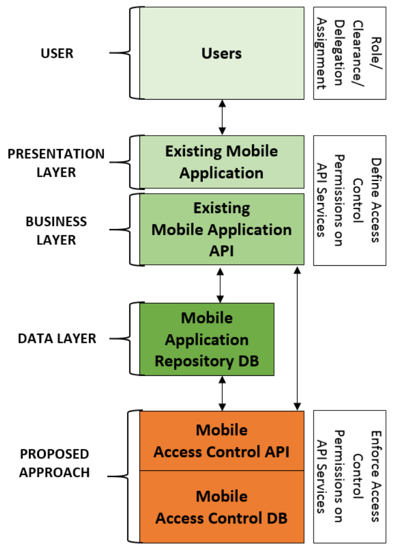
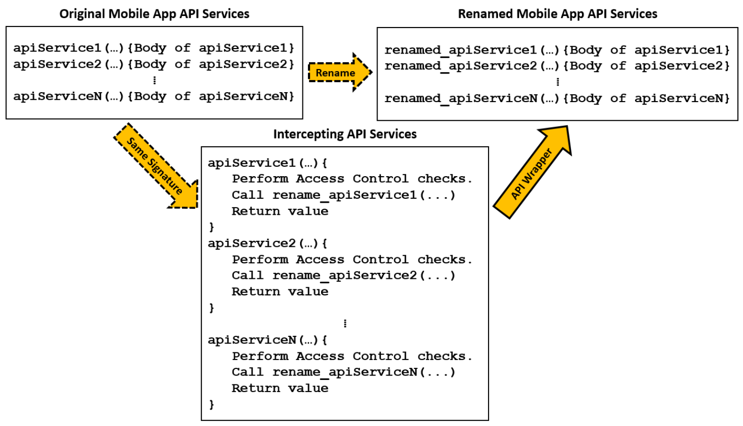

USER
Users
Role/ Clearance/ Delegation Assignment
PRESENTATION LAYER
Existing Mobile Application
BUSINESS LAYER
Existing Mobile Application API
Define Access Control Permissions on API Services
DATA LAYER
Mobile Application Repository DB
PROPOSED APPROACH
Mobile Access Control API
Mobile Access Control DB
Enforce Access Control Permissions on API Services

Figure 5.1. Intercepting API-Based Approach Architecture.

To illustrate the third group, Figure 5.2 details the modifications of the original API services that are needed for interception. Specifically, for a mobile app, there is a set of original mobile app API services, as shown in the left side of Figure 5.2. To maintain the functionality of the mobile app and provide an ability to continue to invoke services by name, the original mobile app API services are renamed (as shown on the right side of Figure 5.2) in order to reuse the original name of the original service for the new intercepting API services so that services from the mobile app remain unchanged (would now be occurring against the intercepting services). For each original mobile app API service, we define a corresponding intercepting API service, as shown in the bottom (middle) part of Figure 5.2, that is able to: perform RBAC, MAC, and DAC security checks for the user/role/clearance/delegation combination; call the corresponding mobile app API service (if it is allowed); and then return either filtered data (retrievals) or success/failure (inserts, updates, or deletes) status.

The mobile app is still able to invoke the same APIs by name and signature, which are now the intercepting API services (with the same signature) that are able to step in and interrupt the process. As a result, the intercepting API services act as a wrapper that adds a security layer to the original API services. The dashed arrows in Figure 5.2 indicate that the process of renaming the original API services as well as the process of creating the intercepting file needs to be done only once. Therefore, the developer only needs to create these files once and after that security administrators can manage the RBAC, MAC, and DAC policies without modifying the server-side portion of the mobile app through the means of a separate user interface. The solid arrow indicates the way that the API behaves when a user makes a request through the mobile app; first, the request is intercepted in order to be evaluated with the pertinent access control policies and then, depending on the result, we either proceed to execute the request or send an error message to the user who sent the request.


Original Mobile App API Services
Rename
Renamed Mobile App API Services
Same Signature
Intercepting API Services
API Wrapper

```
apiService1(…){Body of apiService1}
apiService2(…){Body of apiService2}
        ⁞
apiServiceN(…){Body of apiServiceN}
```

```
renamed_apiService1(…){Body of apiService1}
renamed_apiService2(…){Body of apiService2}
        ⁞
renamed_apiServiceN(…){Body of apiServiceN}
```

```
apiService1(…){
   Perform Access Control checks.
   Call rename_apiService1(...)
   Return value
}
apiService2(…){
   Perform Access Control checks.
   Call rename_apiService2(...)
   Return value
}
                 ⁞
apiServiceN(…){
   Perform Access Control checks.
   Call rename_apiServiceN(...)
   Return value
}
```

Figure 5.2. Conceptual API Process.

5.3. Categorizing Services of APIs

This section discusses the way that the API of a mobile app is viewed from a RBAC and MAC security perspective in order to control who can invoke which service(s) of an

API at which times, and the way that each service is viewed from a security standpoint. In support of this process, we categorize the services on and API in different ways. From a RBAC perspective, we partition the services of an API into two broad categories: secure and unsecure services (Defn. 17 in Section 3.4 of Chapter 3). Secure services are a subset of the API that require control from a security perspective and can be assigned to individual roles (Defn. 19 in Section 3.4 of Chapter 3). Not all of the API services need to be in the secure category; for example, API services to load drop downs, display web content, etc., may not need to be secure. The secure API services are the ones that lead to data that is stored/edited/displayed that must be controlled by role. Unsecure services need not be assigned and are available to any user. On the other hand, from a MAC perspective, we partition the services of an API into two other categories: labeled and unlabeled services (Defn. 18 in Section 3.4 of Chapter 3). Labeled services are a subset of the API that require control from a security perspective and can be assigned to clearances. As mentioned in the RBAC perspective for the secure category, not all of the API services need to be in the labeled category. The labeled API services are the ones that lead to data that is stored/edited/displayed that must be controlled by clearance and MAC properties (Defn. 20 in Section 3.4 of Chapter 3). Unlabeled services need not be assigned and are available to any user. In addition to these categories, the mobile app API services can be delegated by an original user or a delegated user with pass on delegation authority (Defns. 22, 24, 26 in Section 3.5 of Chapter 3) in four different ways: delegate all of his/her allowed Secure services to a delegated user (Defn. 29 in Section 3.5 of Chapter 3), delegate a portion of his/her allowed Secure services to a delegated user (Defn. 30 in Section 3.5 of Chapter 3), delegate all of his/her allowed Labeled services to a delegated user (Defn. 31 in Section

3.5 of Chapter 3) and, delegate a portion of his/her allowed Labeled services to a delegated user (Defn. 32 in Section 3.5 of Chapter 3). To illustrate the aforementioned definitions, Figure 5.3 depicts an extension of Figure 3.1 in Section 3.4 of Chapter 3 by including delegation capabilities.

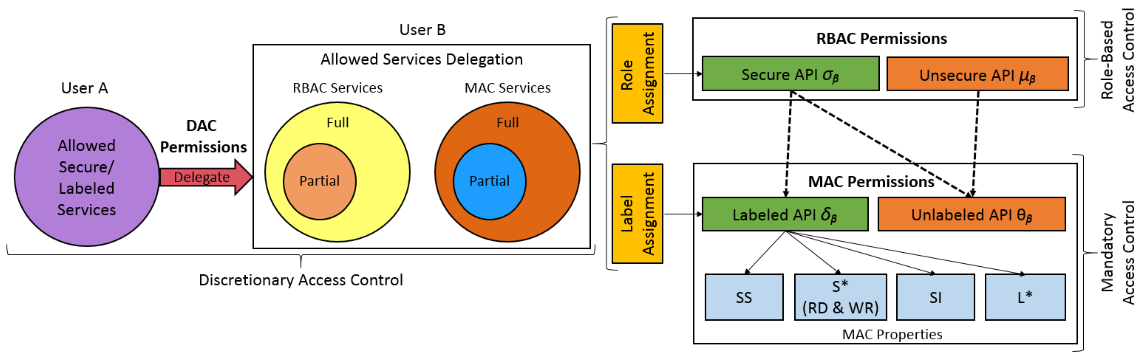
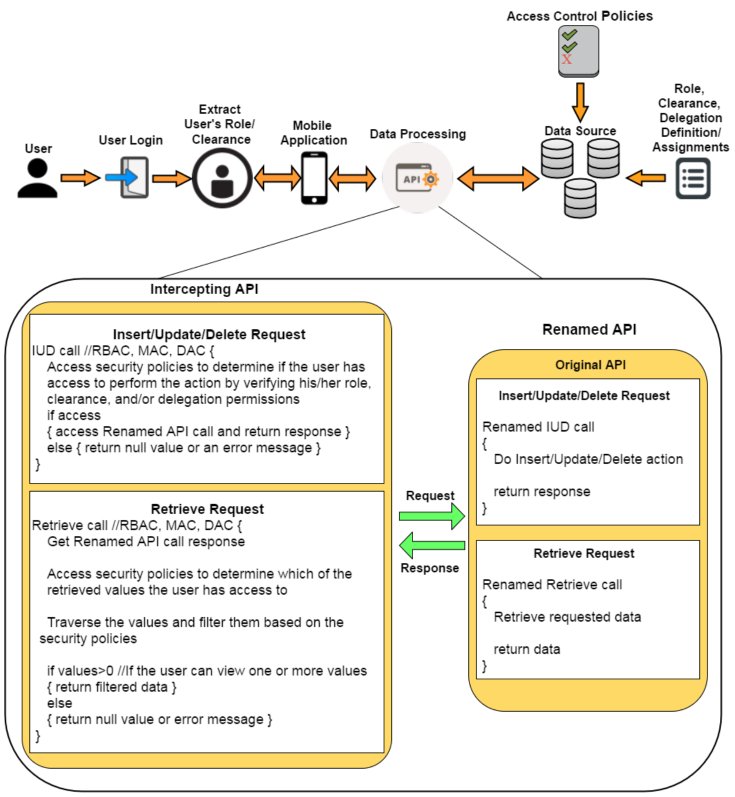


Figure 5.3. RBAC, MAC, and DAC Permissions for API Services.

5.4. Interactions and Infrastructure

This section discusses the interactions and infrastructure of the Intercepting API Calls option. To begin, Figure 5.4 depicts the detailed interactions of the *Intercepting API Calls* option within the configurable framework (see Figure 1.3 of Chapter 1 again). The steps from the user's perspective from left to right are: login to his/her mobile app account; for successful login, extract the user's role/clearance that is part of the login credentials; store the extracted user role/clearance in a secure access token in order to use it in future API calls; utilize the mobile app which results in multiple mobile app API calls and are intercepted (data processing in top of Figure 5.4); and, the intercepted API call interacts with the access control permissions and policies to enforce the defined security before invoking the original mobile app API call. There are two possible requests that can occur as an end result of the interactions: *insert/update/delete requests* where the data that the user is trying to insert/update/delete is not allowed if the user/role/clearance combination does not have permission to do so; and, *retrieve requests* where the data that the user/role/clearance combination is trying to retrieve is filtered according to his/her role/clearance.

In the insert/update/delete request (via an intercepting mobile app API call in the upper portion of the Access Control API oval in Figure 5.4), the request is intercepted to perform the access control checks, and depending on the response, the action is either done (the original mobile app API call is allowed) or not. In the retrieve request the user is trying to retrieve data (via an intercepting mobile app API call in the lower portion of the Access Control API oval in Figure 5.4), the data source performs this action but the mobile app API is intercepted to allow access control checks to be performed. This allows the intercepted API call to determine if the user has access to all/some/none parts of the data with the resulting original API call returning data (all/some case) or null/error message (none case).



```
Insert/Update/Delete Request
IUD call //RBAC, MAC, DAC {
    Access security policies to determine if the user has
    access to perform the action by verifying his/her role,
    clearance, and/or delegation permissions
    if access
    { access Renamed API call and return response }
    else { return null value or an error message }
}
```

```
Retrieve Request
Retrieve call //RBAC, MAC, DAC {
    Get Renamed API call response

    Access security policies to determine which of the
    retrieved values the user has access to

    Traverse the values and filter them based on the
  security policies

    if values>0 //If the user can view one or more values
    { return filtered data }
    else
    { return null value or error message }
}
```

```
Insert/Update/Delete Request
Renamed IUD call
{
   Do Insert/Update/Delete action

   return response
}
```

```
Retrieve Request
Renamed Retrieve call
{
   Retrieve requested data

   return data
}
```

Figure 5.4. Interactions for Intercepting API Calls.

To manage which resources a specific role and/or clearance can access, we store the access control policy in a database, a subset of the main entity relationship diagram shown on Figure 3.3 of Chapter 3, represented in Figure 5.5 as an entity-relationship diagram to support our service-based RBAC, MAC, and DAC approach. Once the user's role and/or clearance has been verified, we can access the specific permission we want to evaluate through the means of an API service as stated in the previous paragraph. The database would hold the roles and/or clearances for each user of each mobile app along with the permissions for each role and/or clearance to each HIT system supported with the HAPI FHIR server. Specifically, to track which services of which FHIR RESTful APIs for each HIT are authorized by role and/or clearance to a user of a particular mHealth app. Moreover, the `secure_unsecure_services` and the `labeled_unlabeled_services` entities provides details of whether a user has access to the resource he/she requested or not by his/her role or clearance, respectively. In addition, the security policy tables store information about the available CRUD services, resources, roles (RBAC), clearances (MAC), classifications (MAC), read and write constraints (MAC), and delegations (DAC). Note that this ER diagram also applies to the Server Interceptor API option of the configurable framework, which is discussed in Chapter 6.

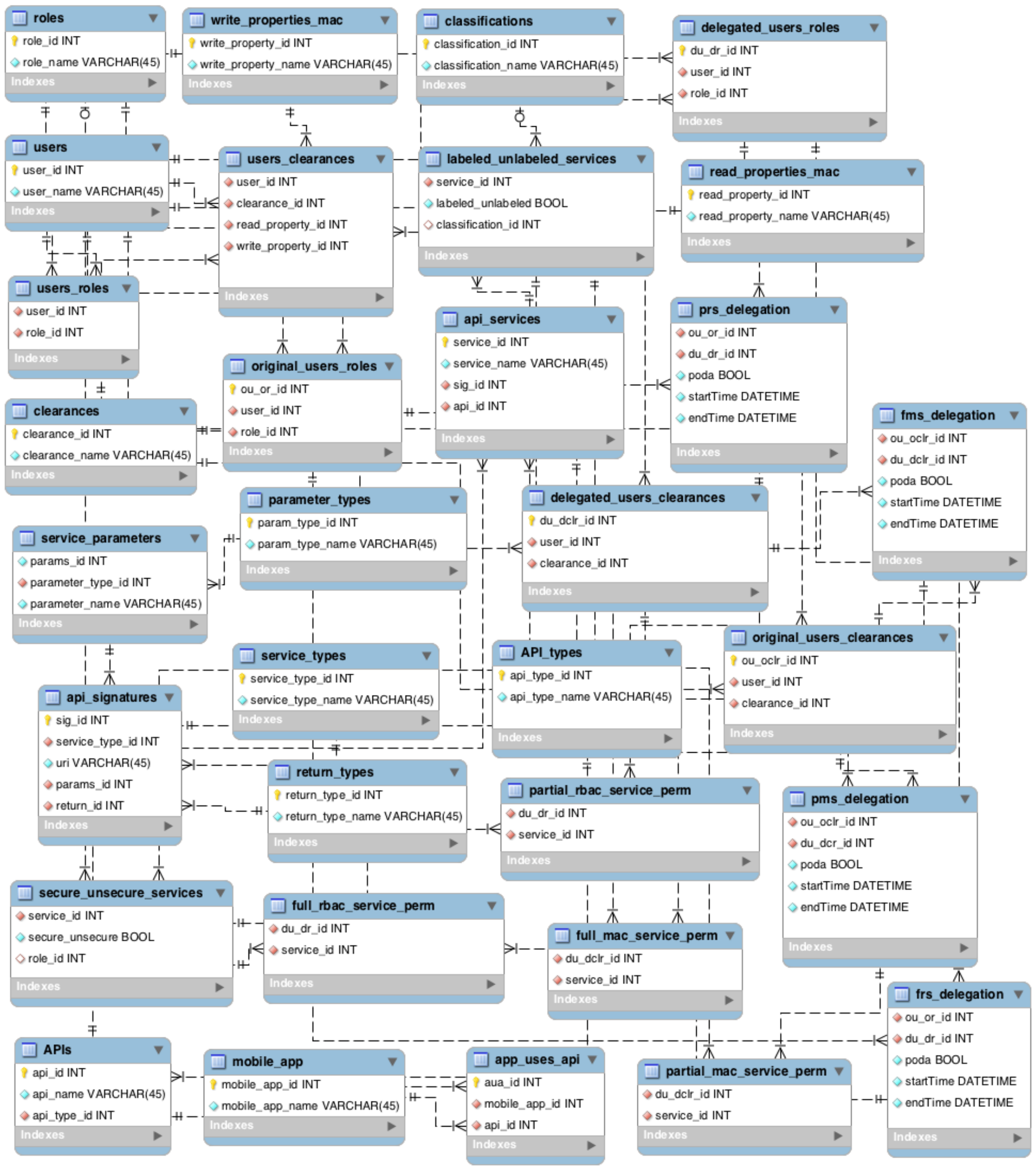


Figure 5.5. A Subset of the ER Diagram from Figure 3.3 for Supporting the Unified Security Model for the Server Interceptor API option.

5.5. Algorithm Generation of Intercepting API Calls Option

This section reviews the algorithm, that is able to automatically generate the intercepting API in support of the Intercepting API Calls option. The primary changes to support the Intercepting API Calls option are made in the backend of the mobile app (server-side – bottom portion of Figure 5.1) and include the addition of RBAC, MAC, and DAC security policies in a permission database to create the mapping from the original mobile app API calls to the corresponding new intercept API calls as shown in Figure 5.5. Each new intercepting API call has the same signature (same address and parameter) as its original mobile counterpart, so that the intercepting API call can substitute for the original API call of the mobile app to allow the aforementioned security checks for retrieve and insert/update/delete requests. As a result, the intercepting API calls effectively wrap the original mobile app calls. The mobile app now seamlessly invokes the intercepting API calls. These intercepting API calls contain the appropriate RBAC, MAC, and DAC security checks, adding a layer of security to enforce the policies. The renamed mobile app API calls is invoked based on the outcomes of the security checks. The end result is that the mobile app appears differently based on the user/role combination, to limit information that is delivered (retrieve request) or that impacts the data that is stored (insert/update/delete requests).

The intercepting API call option utilizes an algorithm to automatically generate the intercepting code in support of Contribution D: Access Control Security Enforcement Code Generation and Interceptors. Pseudo-code for the algorithm is shown in Figure 5.6. In order to automatically generate the code for the Intercepting API Calls option, we need to create a file that contains the same API calls as the original mobile app API via the generate function `Access_Control_API_Generator` which has a parameter that contains an array of all the API calls available in the mobile application (line 1 of Figure 5.6). For each of the API calls in the array, we obtain the parameters (if any), which are stored in a database and store these (line 5 of Figure 5.6). Once we obtain the parameters of the API call that is being evaluated, we can generate the heading of the intercepting API call function by using the current API call as well as its parameters (if any) (line 6 of Figure 5.6). After generating the heading for the intercepting API call function, we then generate the body of the API call, which contains the security policies for that specific call and invokes the original mobile app API call if the user has access to it (line 9 of Figure 5.6). The resulting heading and body of the current API call is stored in an array (line 11 of Figure 5.6). Once all of the intercepting calls have been created, we traverse the array in which they are stored in order to generate the intercepting file (line 13 of Figure 5.6).

```
1    Access_Control_API_Generator(API_Calls)
2    {
3        foreach(API_Calls as currentAPICall)
4       {
5          params = getParams(currentAPICall);
6          API_Call_Heading = generateHeading(currentAPICall, params);
7
8          /*API_Call_Body - Contains security policies and a call to the original API
             service. */
9          API_Call_Body = generateBody(currentAPICall);
10
11         API_Calls_Array = insert(generateAPICall(API_Call_Heading, API_Call_Body));
12      }
13      GenerateFile(API_Calls_Array);
14   }
```

Figure 5.6. Pseudo Code Algorithm for Generating Code of the Intercepting API Calls Option.

To demonstrate the algorithm in Figure 5.6, Figure 5.7 contains the actual PHP code that we implemented in order to generate our approach to the API of the CT^2 mobile app. The function presented in Figure 5.7 is utilized to generate the services in the intercepting API. In order to create a renamed API call for each of the original mobile app API calls, we need the name of the service we are going to generate, if the service needs to be secured by adding permissions and, the name of the file in which we add the generated service (line 1 of Figure 5.7). Note that the permissions we add in each of the intercepting services (if needed) are a layer of security that is not part of the original API services (lines

3-13 of Figure 5.7). Basically, there are three different types of security permissions we can enforce: permissions based on a user’s role, permissions based on a user’s clearance and MAC properties and, permissions based on delegations. To verify if the user has access to the requested service, we access the security policy stored in the database which contains entities (`secure_unsecure_services`, `labeled_unlabeled_services`, `frs_delegation`, `prs_delegation`, `fms_delegation`, `pms_delegation` entities shown in Figure 5.5 of Section 5.4) that specify the requested service’s role/classification (lines 3-7 and lines 11-13 of Figure 5.7). If the role/clearance that is been verified does have permission to perform the requested action, then the service proceeds to access the service in the renamed API file (lines 8-10 of Figure 5.7); otherwise, the intercepting API service returns a null value (line 12 of figure 5.7). Nonetheless, if the renamed service does not need to verify a user’s role in order to be executed then the intercepting service calls it directly, in other words, the intercepting API service does not add security permissions in this case (lines 8-10 of Figure 5.7). Finally, the generated API service gets written in the file that serves as the intercepting API (line 15 of Figure 5.7). Appendix B contains the complete code utilized to generate the intercepting API file and for generating a renamed version of the original API file, which were generated for the CT^2 mobile app.

```
1 function echoInterceptBody($serviceName, $need_permission, $write_file){
2   $wrapper_string = "public function ".$serviceName."{";
3   if($need_permission){
4           $wrapper_string = $wrapper_string."
5           \$permission = \$this->verifyAPIPermissions(__FUNCTION__);
6           if(\$permission == 1){";
7   }
8   $wrapper_string = $wrapper_string."
9   \$renamedConcussionUConn = new renamedConcussionUConn();
10  return \$renamedConcussionUConn->RENAMED".$serviceName.";";
11  if($need_permission){
12                  $wrapper_string = $wrapper_string.”} else{ return NULL;}";
13          }
14        $wrapper_string = $wrapper_string."}";
```

```
 fwrite($write_file, $wrapper_string);
}
```

Figure 5.7. Code for Generating the Body of the Services in the CT^2 API.

The code given in Figure 5.7 generates, for each original services of the CT^2 API, a REST API for generating an intercepting API file in support of the intercepting API option. This is shown in Figure 5.8 for the original CT^2 API service *updateStudent* while Figure 5.9 shows the renamed CT^2 API of the aforementioned service.

```
public function updateStudent($studentObject, $studentId){
   $permission = $this->verifyAPIPermissions(__FUNCTION__);
   if($permission == 1){
      $renamedConcussionUConn = new renamedConcussionUConn();
      return $renamedConcussionUConn->
      RENAMEDupdateStudent($studentObject, $studentId);
   }else{return NULL;}}
```

Figure 5.8. Portion of Generated Code for the Intercepting API.

```
public function RENAMEDupdateStudent($studentObject,$studentId){
    $sqlGeneralStudent = "UPDATE students SET first_name =
       '" . $studentObject->firstName . "',
       middle_name = '" . $studentObject->middleName . "',
       last_name = '" . $studentObject->lastName . "',
       suffix = '" . $studentObject->suffix . "',
       email = '" . $studentObject->email . "',
       student_number = '" . $studentObject->studentNumber . "',
       school_id = '" . $studentObject->schoolId . "'
       WHERE student_id = " . $studentId;
    $recordId = $this->updateRecord($sqlGeneralStudent);

       if($recordId){
                 $sqlStudentDemo = "UPDATE student_demographics date_of_birth = '" .
                 $studentObject->dateOfBirth . "',
           gender = '" . $studentObject->gender . "'
           WHERE student_id = " . $studentId;

           if($this->updateRecord($sqlStudentDemo)) return 1;
           else return 2;
       }
       else{return 0;}
}
```

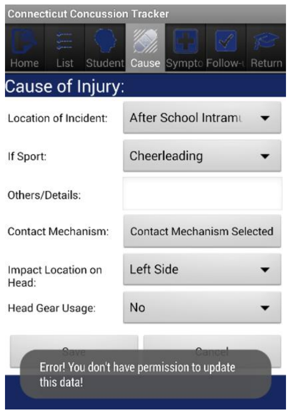
Figure 5.9. Portion of Generated Code for the Renamed API.

5.6. Example of the Intercepting API Calls Option

To evaluate the Intercepting API Calls option, the Connecticut Concussion Tracker (CT^2) mobile application, database, and its server are utilized as an example. As currently designed, the CT^2 app supports RBAC, MAC, and DAC that allows for the different screens and the content of different screens to be available by role, clearance, and delegations. There are four roles: the *Nurse* role, which has access to all tabs for a school nurse to manage a student's concussion incident from its occurrence to its resolution; the *Athletic Trainer* (*AT*) role which has access to home, list, student, cause, and symptoms tabs to do a limited preliminary assessment if a concussion incident occurs at the event; the *Coach* role, which has access to home, list, student and cause tabs to report a concussion incident at an athletic event with very limited information on the student; and, the *Parent* role, which has access to home, list, student, cause, and symptoms tabs to both report a concussion incident on his/her child while attending the athletic event or to track the current status of his/her children that have ongoing concussions. In addition, each of the users of the mobile app have a clearance assigned and delegation permissions.

Programmatically, we have source code for the Android version of the CT^2 app and a REST API that accesses the concussion MySQL database. The source code of the mobile app is organized by tabs that are loaded for a given user/role combination, and each tab is augmented with if/else conditions that either display the data on a tab if it was available in the database or display an error message stating that the contents couldn't be retrieved. The realization of the Intercepting API Calls option is achieved without any modification to the mobile app UI and is intended to allow fine-grained access control on the information that is displayable and/or storable of the authorized tabs for each user/role/clearance combination. There is a very clear mapping from the process described in this section and the accompanying figures to its realization in CT^2. The database is augmented with a table that contains a list of all the API calls available along with a service_id, and tables that contain the security policies that determine which calls the available roles/clearances have access to (`secure_unsecure_services` and `labeled_unlabeled_services` entities in Figure 5.5 of Section 5.4). Given these database changes, we then take the original CT^2 REST API calls and rename as shown in Figure 5.2. Then a set of new CT^2 intercepting REST API calls are defined that perform a series of RBAC, MAC, and DAC checks and if successful, invoke the corresponding renamed original CT^2 REST API calls.

From a process perspective, the steps follow the top portion of Figure 5.4. The user logs on to the CT^2 mobile app and his/her the role/clearance is stored in a global variable in order to support the class that manages the API calls. Figure 5.10 illustrates the impact of the Intercepting API Calls and associated process for a user with the role of *Coach* and a clearance of Confidential which has access to only the home, list, student, and cause tabs. This role-clearance combination can add basic information on the ‘Student’ tab and can add information in the ‘Cause’ tab and, after adding the information, can view but not edit. The original mobile app CT^2 API calls support the insert of information in the database and the intercepting API call in this case allows that first save to occur. At a later point in time, if the user attempts to edit and perform another save, the intercepting API call in this case, performs the access control check that does not allow the edit. As a result, when a user with the role of Coach that is using the ‘Cause’ tab attempts to save, the intercepting API call alerts that he/she does not have permission to perform that action. The other tabs of CT^2, ‘Symptoms’, ‘Follow-Up’ and ‘Return’, are still visible within the app. However, when a user with the Coach role and Confidential clearance attempts to access one of these tabs, the application tries to obtain the pertinent data via the former original CT^2 API call that has been replaced by a new CT^2 intercepting API call that checks for permissions and returns that the specified role-clearance combination does not have permission to retrieve the data for those screens.



Cause of Injury:

Location of Incident:	After School Intram
If Sport:	Cheerleading
Others/Details:	
Contact Mechanism:	Contact Mechanism Selected
Impact Location on Head:	Left Side
Head Gear Usage:	No

Error! You don't have permission to update this data!

Figure 5.10. ‘Cause’ screen for the role of *Coach* in CT^2.

5.7. Related Work

There are many efforts that propose access control mechanisms to secure mobile applications by limiting the permissions and resources a mobile app can access in different areas of the mobile device/app. In this section, we discuss several existing proposed approaches that attempt to apply access control mechanisms on different locations on a mobile device and, we explain the way our approach compares and contrasts. The first area of related work involves sensor management on smartphones that is commonly addressed by applying access control mechanisms to the sensors of a mobile device so that mobile apps obtain fine-grained permissions. This facilitates the managing of sensor data in mobile apps (e.g., user’s location, use of Bluetooth) (Cappos et al., 2014; Xu and Zhu, 2015). BlurSense (Cappos et al., 2014) and SemaDroid (Xu and Zhu, 2015) allow users to define and add privacy filters to sensor data, through the means of a user interface, that is being used on their mobile applications. In contrast to these efforts, our work presented in this chapter focuses on API access control management for the API services that are utilized within a mobile app to populate data in the app and to add/edit data and store it in a data source. In other words, instead of focusing on modifying the operating system to filter sensor data we modify the backend of a specific mobile app and filter the data that a user can have access to according to his/her role, which can include sensor data as well if there was an API service included in the intercepted API that managed this. The second area of related work involves permission control in Android in which access control can be applied on the mobile device itself. There are many existing approaches (Beresford et al., 2011; Benats et al., 2011; Wang et al., 2014; Jin et al., 2015; Hao et al., 2013; Backes et al., 2014) that focus on applying fine-grained access control policies to mobile devices that contain Android as their operating system. This is due to the fact that Android contains a coarse-grained access control mechanism when it comes to allowing permissions in mobile applications. In other words, in order for a user to install a mobile app he/she needs to accept all of the permissions that the app requires. This may disregard the fact that some permissions may not be necessary for the app to function and that some of the permissions may not make sense for app that is being downloaded and could result in using the allowed component for malicious purposes (e.g., a flashlight app tells user it needs permission to get the user's location).

Adding fine-grained access control to the APIs that Android uses for the device and apps to function properly has been addressed by: mocking the values that an app receives in order to function (Beresford et al., 2011) (e.g., mocking latitude and longitude coordinates); extending the security policies of the mobile device (Benats et al., 2011;

Wang et al., 2014; Jin et al., 2015); by rewriting the bytecode of the mobile device (Hao et al., 2013); and by adding security modules to the mobile device (Backes et al., 2014). In contrast to this effort, our work presented in this chapter focuses on applying access control mechanisms to the APIs that are not part of the mobile system itself. In addition, most of these works are specific for Android OS/API while ours can be implemented for any type of application (even though we focus on the mobile setting) since our access control approach is enforced server-side. The third area of related work involves role-based access control and extensions that expand RBAC with context-aware techniques in order to provide finer-grained access control security policies to those systems that contain highly sensitive data. One effort does this by proposing an RBAC model with a spatiotemporal extension for web applications (Aich et al., 2009) and another effort proposes a similar approach but for mobile applications (Abdunabi et al., 2013). The proposed access control system made for web applications (Aich et al., 2009) can be applied to an existing system as a dll component. Another approach proposes a dynamic RBAC approach for Android devices (Rohrer et al., 2013). That approach focuses on modifying the Android framework to provide a uniform security policy to mitigate security risks in mobile devices that are utilized by users who are part of an enterprise. Finally, an effort (Fadhel et al., 2016) proposed a model that extends RBAC to generate RBAC conceptual policies. Nevertheless, the aforementioned effort does not provide details of which specific application domain(s) the approach could support. Our framework could easily be extended to support other types of access control, can be applied to mobile web applications and, it is not domain-specific; this contrasts to the discussed related work.

Chapter 6
Server Interceptor API

Health Information Exchange (HIE) provides a more complete health record with the aim to improve patient care with relevant data gathered from multiple health information technology (HIT) systems that provide APIs for interactions. In support of HIE, the Health Level Seven (HL7) (HL7, 2013) XML standard was developed to manage, exchange, integrate, and retrieve electronic health information. In 2011, the Fast Healthcare Interoperable Resources (FHIR) (FHIR DSTU2, 2015) standard, based on HL7, was proposed to facilitate the development of mobile health (mHealth) apps with HIT data sharing via a common modeling format. FHIR utilizes RESTful APIs enabled with a FHIR server for information usage and exchange in the cloud. FHIR has a security specification, but does not define actual security mechanisms for secure data exchange via service invocations. In support of the interaction of the mobile app API services with multiple HIT systems (data sources) operating on the server side, this chapter presents the Server Interceptor API option of our configurable framework for controlling access. In Chapter 5, we intercepted the API service invocations to the mobile app. In this section, we intercept the API service invocations between the mobile app API and services of the APIs of the HIT data sources. This second security layer between the two different APIs (mobile app and server-side) is accomplished through the creation of a server interceptor API associated with a cloud computing infrastructure to intercept invocations for RBAC, MAC, and DAC checks, to be presented in this chapter. The resulting RBAC, MAC, and DAC service-based model as presented in Sections 3.1 to 3.5 for the unified security model is incorporated into the FHIR standard to control access of who can invoke which services of FHIR RESTful

APIs that manage sensitive healthcare data; work is demonstrated via a mHealth application that interacts with the OpenEMR HIT system via the HAPI FHIR server. The work in this chapter on the interceptor supports Contribution D: Access Control Security Enforcement Code Generation and Interceptors.

This chapter provides details about the Server Interceptor API option that controls the service invocations from the mobile app API services to the server-side APIs of the data sources in five sections. Section 6.1 motivates the way that the Intercepting API Calls option as given in Chapter 5 is adapted and evolved to the Server Interceptor API option. Section 6.2 reviews the HAPI FHIR reference implementation capabilities with a focus on the intercepting process. Section 6.3 presents a set of modifications that incorporate RBAC, MAC, and DAC into the FHIR specification and its realization within the HAPI FHIR reference implementation. To demonstrate the inclusion and realization of RBAC, MAC, and DAC for a mHealth app within HAPI FHIR, Section 6.4 provides an implementation of the service-based access control approach by using the Connecticut Concussion Tracker (CT^2) mHealth app and the OpenEMR HIT system (OpenEMR, 2002). Finally, Section 6.5 reviews related works and compares and contrasts these to the work of this chapter.

6.1. Motivating the Server Interceptor API Option

As discussed in Chapter 5, the Intercepting API Calls option performs security checks to determine whether the API service call can occur based on the role and/or clearance of a user with optional delegation capabilities and perhaps also limit what is returned to the user. The approach as described in Sections 5.2 to 5.5 of Chapter 5 essentially creates a replica of the API that the mobile app uses so that the mobile app calls our intercepting API which can perform security checks and then pass the call through to the original mobile app API if the security permissions are met. Originally and as discussed throughout Chapter 5, this was accomplished by renaming the calls to the services of the original mobile app API. There are two possible issues with this approach. First, we may not have access to the mobile app API. Second, even if we do, then the renaming would require changes to the service names of the original mobile app API. As a result, we believe that it is possible to realize a solution to eliminate these two issues by proposing a Server Interceptor API option that does not modify the original mobile app API file but contains the original service calls that the mobile app has access to and then forwards each call from the intercepting API call of the same name to the original API call; this option operates between the mobile app API and the APIs of the server-side data sources.

Figure 6.1 illustrates an alternate approach for the Intercepting API Calls option that establishes an intercepting server that has an API that mimics what the mobile app is expecting but is our intercepting API (second box from the left in Figure 6.1). The intercepting API server must also be able to mimic multiple APIs since the mobile app may interact with multiple APIs (depicted in third box from the left in Figure 6.1). The original and intercepting servers would need to be run on different ports if on the same host with the intercepting server being accessible publicly while the original might only be accessible locally, then the intercepting server(s) could forward any allowed calls to the original server(s) and filter the results as needed before returning to the client. This is facilitated by utilizing the login credentials (user/role/clearance combination) in order to determine the security level in each of the API Calls. Currently, we manage to pass on the user’s id, role, clearance, and delegation permissions between calls by storing these server-side in a JSON Web Token (JWT) (JWT, 2015). This is done to secure the user’s data and to verify that the user has access to the action he/she requested by utilizing his/her role, clearance, and/or delegation permissions. However, the need to operate multiple servers on different ports in order to fool the mobile app into calling the mimicked services is not necessarily easy to accomplish.

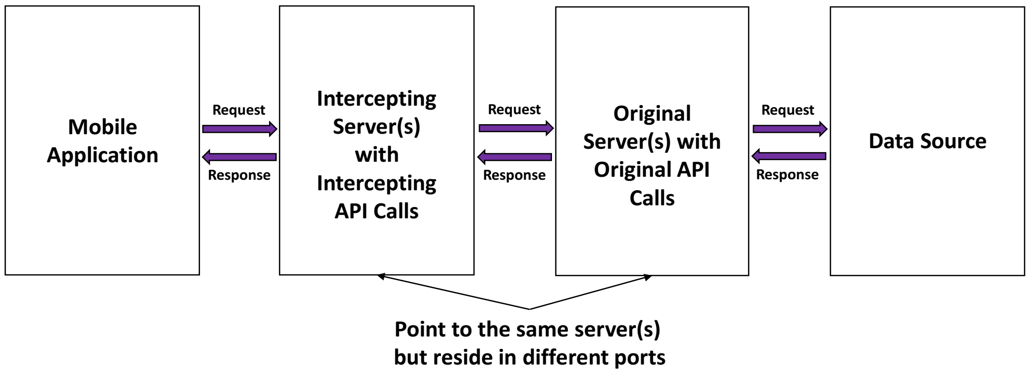

Mobile Application
Request
Response
Intercepting Server(s) with Intercepting API Calls
Request
Response
Original Server(s) with Original API Calls
Request
Response
Data Source
Point to the same server(s) but reside in different ports

Figure 6.1. Intercepting Server Original Idea.

In support of HIE and multiple HIT systems having their APIs called by the mobile app API services, we want to provide a way to intercept the calls to the APIs of data sources without having to modify the original API services as we did for the approach presented in Chapter 5 nor adding a server that contains the same signatures as the original server that hold the original mobile app API calls (as shown in Figure 6.1). Instead, we focus on intercepting services on the server APIs without making modifications to the original server API services through the means of Java servlets (Java, 2013) and HAPI FHIR (HAPI FHIR, 2014). Figure 6.2 depicts a general idea of the intended process to be explained in the remainder of this chapter.

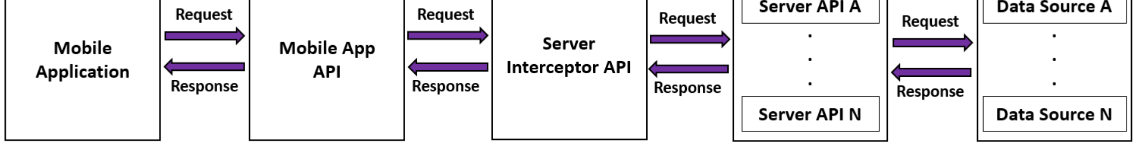


Figure 6.2. Server Interceptor API Option General Idea.

6.2. Access Control in FHIR

This section reviews the HAPI FHIR reference implementation capabilities with a focus on the intercepting process that can support our service-based access control approach. The FHIR security specification does not yet include explicit security capabilities, but does define different exchange protocols and models that could be utilized with security policies created by an organization (FHIR, 2016). To handle security, FHIR suggests several points including: securing data exchange with TLS/SSL (e.g., HTTPs); authenticating users with an authentication method (e.g., OAuth); and utilizing digital signatures. In addition, FHIR defines security labels to support access control management based on (HL7 v3, 2013 ; HL7, 2013). The security labels were done with the purpose of restricting access on resources based on security policies established by security administrators of the data that is being exchanged. Nevertheless, there is very little documentation on the way that role-based, mandatory, or discretionary access control policies could be both defined and enforced through the means of these labels.

In addition to suggesting security labels, the FHIR security specification provides a notion of where security needs to be placed in the logical layout of a system in order to handle users, user authentication, and user authorization. The FHIR security specification also assumes that a security system exists and that is positioned before or after the FHIR API (FHIR, 2016). Figure 6.3 illustrates three possible scenarios on the locations where a security system can be deployed on the logical layout of a system, where the layout is composed of clients, of a client (mHealth) app, of data sources (EHRs and other HIT systems), and of FHIR layers. The first scenario applies security mechanisms between two layers of FHIR where the first layer would be a security system. The second scenario places security on the client application. Lastly, the third scenario places security server-side (FHIR, 2016). For the purposes of this chapter we focus on applying RBAC, MAC, and DAC by using a similar idea to the one exposed in the first scenario. In the remainder of the chapter, we make two assumptions: a mobile app needs to obtain information from one or more data sources (e.g., EHRs or HITs) which are connected by a cloud server; and, the cloud server that connects the mobile app with the data sources is a FHIR sharing infrastructure.

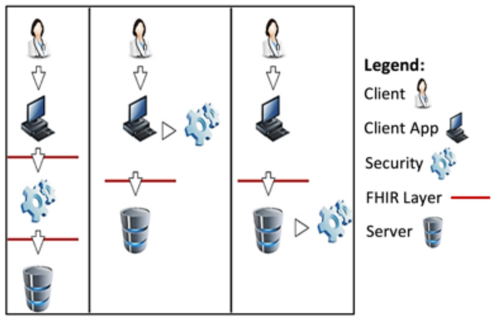


Figure 6.3. Security system placement in deployment architecture (FHIR, 2016).

In addition, HAPI FHIR developers are in the process of implementing authentication and authorization interceptors (HAPI FHIR Server Security, 2016). The authorization interceptor attempts to apply security to FHIR by creating a set of rules via rule-based access control within the function and utilize if/else statements in order to whitelist/blacklist requests. In our case, instead of utilizing rule-based access control, our service-based approach involves the use of RBAC, MAC, and DAC. Further, instead of modifying the code of the intercepting function every time we want to add/modify a permission to/in the policy (e.g., change a role or a permission on a role), we implement an interceptor function once that consults RBAC, MAC, and/or DAC permissions and privilege definitions by role/clearance/classification in a database. When changes to the security policies stored in a repository are made, the RBAC, MAC, and/or DAC interceptor functions continue to work to check intercepted RESTful API calls.

6.3. Access Control Server Interceptor

This section introduces the *Access Control Server Interceptor* that supports the Server Interceptor API option as a combination of the RBAC, MAC, and DAC interceptor functions that were introduced in Section 6.2, along with an associated architecture in Figure 6.4 that is used to evaluate which FHIR resources users are allowed to access by role, clearance/classification, and/or delegation permissions via the corresponding FHIR RESTful API. In Figure 6.4, when the user makes a request using the mobile application, this request is handled by a Mobile App HAPI FHIR Server. The Mobile App HAPI FHIR Server is implemented with the HAPI FHIR reference implementation library with programmatic access to the server interceptor class, which is explained in more detail in the discussion of Appendix C-2. This class intercepts the user's request and retrieves access control policies from the Access Control Security Policy through the means of the Server Interceptor API as shown in Figure 6.4. After retrieving the pertinent access control policies, the class performs access control checks to ensure that the user has access to the requested resource, i.e., can invoke the requested service. Currently, we focus on enforcing RBAC, MAC, and DAC security policies on the requested services; in a future, we plan to enforce security policies of other access control mechanisms such as ABAC. In order for the interceptor class to enforce RBAC, MAC, and DAC in the services, the class retrieves security policies from a data source through the means of API services we implemented for this purpose. More details on the way that the security policy would operate for the access control server interceptor are discussed later on and are depicted in Figure 6.5.

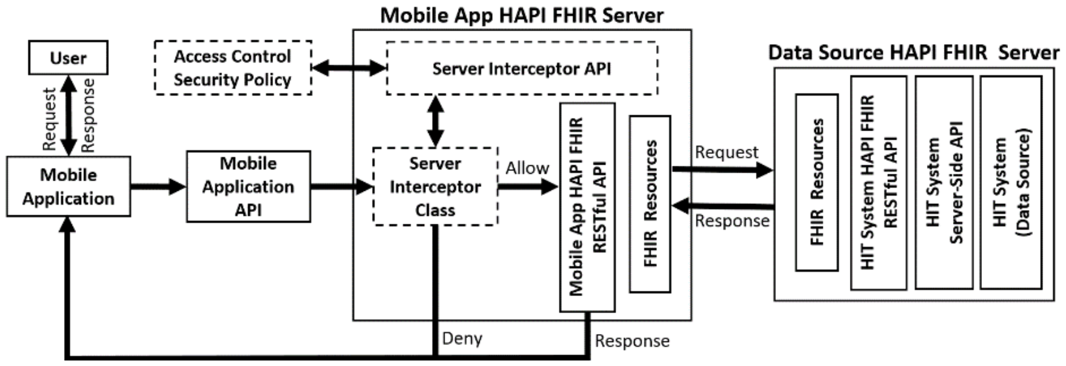


Figure 6.4. Server Interceptor API Option Architecture.

The Server Interceptor API shown in Figure 6.4 contains the API services to retrieve the security policies from the Access Control Security Policy DB and then send these to the Server Interceptor Class which calls a service of the Mobile App HAPI FHIR RESTful App. The Server Interceptor Class shown in Figure 6.4 can be explained utilizing pseudocode of our service-based access control approach as shown in Appendix C-1 as implemented within the `incomingRequestPostProcessed` function, which is within the Server Interceptor class available in HAPI FHIR. Appendix C rewrites HAPI's `incomingRequestPostProcessed` interceptor function to obtain details of the request (e.g., HTTP method used) and the resource that is being evaluated before an object can be obtained. Next, we explain and review the processing of the code presenting and explaining excerpts from Appendix C-1. The Mobile App HAPI FHIR Server interacts with the Mobile App HAPI FHIR RESTful API via the FHIR Resources with the Data Source HAPI FHIR Server which is FHIR RESTful API that maps to/from the HIT Server-side API of the Data Source. The HIT System HAPI FHIR RESTful API of the Data Source is able to map/to from the HIT System Server-Side API of the Data Source. When the `incomingRequestPostProcessed` interceptor function determines that a service of the Mobile App API is allowed to call a Data Source HAPI FHIR Server service, the call proceeds via the common FHIR Resource mapping to the HIT System HAPI FHIR RESTFul API of the Data Source HAPI FHIR Server.

The remainder of this section explores the `incomingRequestPostProcessed` interceptor function in detail. Note that we repeat the line numbers in the code from Appendix C-1 so the reader can easily determine where the excerpt has come from. In addition, the pseudo code of Appendix C-1 is applicable to other cloud frameworks in addition to FHIR. The first segment of code from Appendix C-1 retrieves the authorization header passed with the request which contains an access token that is used to verify the user's identity by calling an API service; obtains the HTTP method of the request, the name of the requested resource the id of the requested service; and, sets the permission to access the requested resource to false in the following code:

```
1  //Serves as Access Control Interceptor function
2  public boolean incomingRequestPostProcessed(requestDetails, request, response){
3    authToken = requestDetails.getHeader("Authorization");
4    //Retrieves the user's id, clearance and, read and write MAC properties
5    [userId,userRole,userClearance,readP,writeP] = verifyUser(authToken);
6    httpMethod = request.getMethod();
7    resourceName = requestDetails.getResourceName();
8    serviceId = getServiceId(httpMethod, resourceName);
9    acPermission = false;
```

If the user passes the verification, the API service returns the user id as proof that the user was successfully verified as well as the user's role id, clearance id, and MAC read and write properties assigned to the user. If the user id is not a valid one, an error message is returned to the user stating that he/she could not be verified. Otherwise, if the user id is valid (condition on line 10 is true) and, if the requested resource is secured and/or labeled, we proceed to enforce security checks. If a resource does not have any RBAC and/or MAC security involved, true is returned meaning that the service can be called without security checks. Otherwise, the requested resource is secured or labeled.

```
    10  if(userId > 0){
    11        //Check if requested resource is secured/labeled
    12        [secured,labeled] = getResourceSecurity(httpMethod,resourceName);
    13        if(!(secured || labeled)){
    14           return true; //Continue with request processing
    15        }
    16        //Analyze MAC policies (if any)
    17        if(userClearance > 0 && labeled){
    18           acPermission = checkAndEnforceMAC(userId, userClearance, serviceId, readP, 
readW);
    19        }
    20        //Analyze RBAC policies (if any)
    21        if((roleId > 0 && secured) && (acPermission || !labeled)){
    22           acPermission = checkAndEnforceRBAC(userId, userRole, serviceId);
    23        }
    24     }
    25     else{//Error Message: User could not be verified}
```

If there are MAC policies defined (condition in line 17 is true), we call the `checkAndEnforceMAC` function in line 18 invoking function in line 50. In the function, we first check if DAC delegations are available and if the user has delegated MAC permissions:

```
    50  private boolean checkAndEnforceMAC(userId, userClearance, serviceId, readP, readW
){
    51    acPermission = false;
    52    //MAC services delegation
    53    if(dacPermission() && checkIfDacMac(userId)) {
    54       delClr = delClrDAC(userId, serviceId);
    55       if(delClr>0) { userClearance=delClr; } //Delegated clearance_id
    56    }
```

If the user has delegated permissions (condition in line 53 is true), we determine if he/she has a delegated permission for the requested service through the means of the `delClrDAC` function:

```
    33  private int delClrDAC(userId, serviceId) {
    34      //Check if delegated user has a delegated clearance for the requested service
    35      if(currentTime>getStartTimeMAC() && currentTime<getEndTimeMAC()) {
    36         if(serviceId in service_permissions_mac(userId))
    37         {return delegatedClearance;}
    38      }
    39      return 0;
    40  }
```

The `delClrDAC` function returns 0 if the user does not have any delegated permissions for the requested service and the user's clearance remains the same. Otherwise, if the user have has delegated permissions for the requested service, then the function returns the delegated clearance and we replace the original user's clearance with this one. Afterwards, we proceed to obtain the classification and the http method of the requested service through the means of an API service that has the service id of the service the user is trying to access as a parameter (line 36). This is done to verify if the user's assigned clearance has access to the requested resource per Definitions 8v2 (clearance assignment), 18 (Labeled services), and 20 (labeled API permissions). If the requested service consists of retrieving data from a data source, we evaluate using the MAC read properties simple security and strict * shown below:

```
59    //Retrieve MAC read or write property for pertinent user
60    if(httpMethod == "GET"){
61       //Simple security property
62       if(readP == simpleSecurityProperty){
63          if(userClearance >= serviceClassification){
64             acPermission = true;
65          }
66       }
67       //Strict * property
68       elseif(readP == strictStarProperty){
69          if(userClearance == serviceClassification){
70             acPermission = true;
71          }
72       }
```

In the case where the requested service consists of modifying data (e.g., creating, updating, or deleting) from a data source, we evaluate utilizing the MAC write properties simple integrity, strict *, and liberal * given in:

```
74    else{
75          //Simple integrity property
76          if(writeP == simpleIntegrityProperty){
77             if(userClearance >= serviceClassification){
78                acPermission = true;
79             }
80          }
81          //Strict * property
82          elseif(writeP == strictStarProperty){
83             if(userClearance == serviceClassification){
84                acPermission = true;
85             }
86          }
87          //Liberal * property
88          elseif(writeP == liberalStarProperty){
89             if(userClearance <= serviceClassification){
90                acPermission = true;
91             }
92          }
93    }
```

Finally, in the case where the role is a valid one and the request contains RBAC checks and, if the `checkAndEnforceMAC` function returns that the user has access to the requested resource (`acPermission` = *true*) or MAC policies were not established for the requested resource, we move on to analyze the requested resource against the available RBAC policies in the `checkAndEnforceRBAC` function (invocation on line 22 and function on line 97). First, the function checks if DAC delegations are available and if the user has delegated RBAC permissions. If the user has delegated permissions, we determine if he/she has a delegated permission for the requested service through the means of the `delRoleDAC` function given in:

```
42  private int delRoleDAC(userId, serviceId) {
43     //Check if delegated user has a delegated role for the requested service
44     if(currentTime>getStartTimeRBAC() && currentTime<getEndTimeRBAC() {
45        if(serviceId in service_permissions_rbac(userId))
46        {return delegatedRole;}
47     }
48     return 0;
49  }
```

The `delRoleDAC` function returns 0 if the user does not have any delegated permissions for the requested service and the user's role remains the same. Otherwise, if the user has delegated permissions for the requested service then the function returns the delegated role and we replace the original user's role with this one. If this is the case, then we retrieve the role of the requested service and verify if the user's role is one of the roles that has access to the requested service (lines 106-108 of Appendix C) per Definitions 11 (role assignment), 17 (Secured services) and 19 (secure API permissions). The function returns whether the user passed the RBAC security checks (`acPermission` = *true*) or not (`acPermission` = *false*). Once we evaluate the requested resource against the available permissions, the function returns true if the user does have access to the requested resource and we move on to perform the request (`acPermission` = *true*):

```
97  private boolean checkAndEnforceRBAC(userId, userRole, serviceId){
98     acPermission = false;
99     //RBAC services delegation
100    if(dacPermission() && checkIfDacRbac(userId)) {
101             delRole = delRoleDAC(userId, serviceId);
102             if(delRole>0) { userRole=delRole; } //Delegated role_id
103       }
104    //Get service set of roles
105    serviceRoles = getRoleSet(serviceId);
106    if(roleId in serviceRoles){
107       acPermission = true;
108    }
109    return acPermission;
110 }
```

Upon return to `incomingRequestPostProcessed`, line 2 above, if the user does not have access to the requested resource (`acPermission` == *false*), an error message to the user states that he/she does have access to the request:

```
26  if(acPermission == false){
27     //Error message: User does not have permission to access the
28     //requested resource
29  }
30  return acPermission;
```

To manage which resources a specific role and/or clearance can access, we store the access control policy in a database, a subset of the main entity relationship diagram shown on Figure 3.3 of Chapter 3, represented in Figure 6.5 as an entity-relationship diagram to support our service-based RBAC and MAC approach, which is a subset of the main entity relationship diagram shown on Figure 3.3 of Chapter 3. Once the user's role and/or clearance has been verified, we can access the specific permission we want to evaluate through the means of an API service as stated in the previous paragraph. The database would hold the roles and/or clearances for each user of each mHealth mobile app along with the permissions for each role and/or clearance to each HIT system supported with the HAPI FHIR server. Specifically, to track which services of which FHIR RESTful APIs for each HIT are authorized by role and/or clearance to a user of a particular mHealth app. Moreover, the `secure_unsecure_services` and the `labeled_unlabeled_services` entities provides details of whether a user has access to the resource he/she requested or not by his/her role or clearance, respectively. In addition, the security policy tables store information about the available CRUD services, resources, roles (RBAC), clearances (MAC), classifications (MAC) and, about read and write constraints (MAC). Sample access control policies will be provided on Table 6.1 and Table 6.2 of Section 6.4.

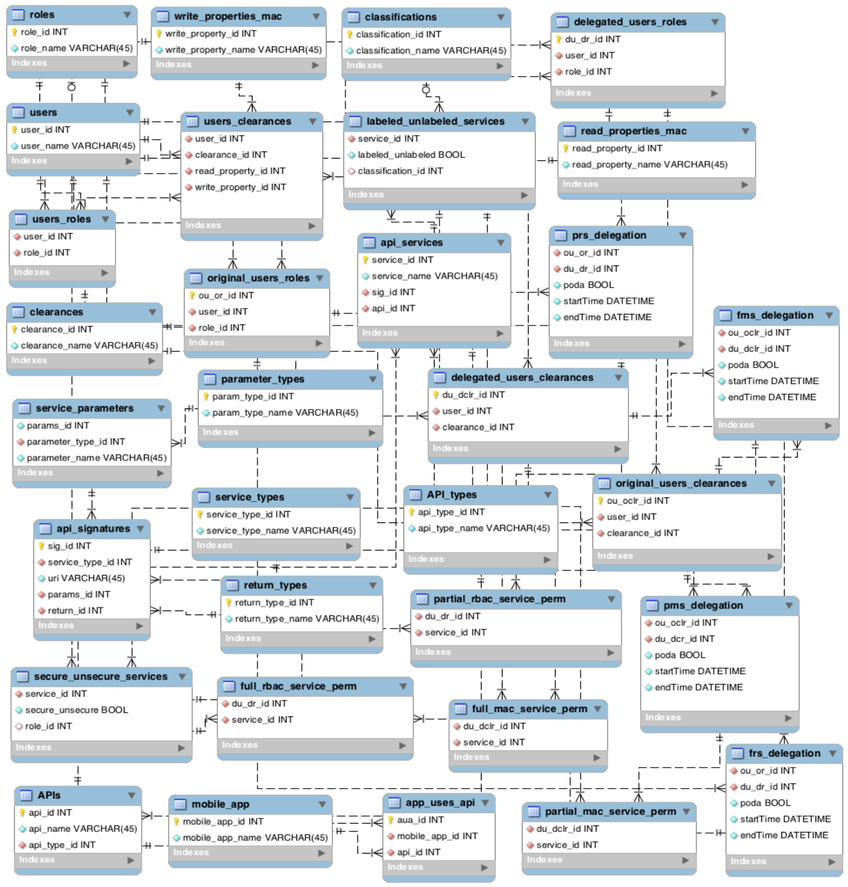


Figure 6.5. A Subset of the ER Diagram from Figure 3.3 for Supporting the Unified Security Model for the Server Interceptor API option.

6.4. Implementation

To evaluate the incorporation of RBAC, MAC, and DAC into FHIR, which is the realization of the Server Interceptor API option, the Connecticut Concussion Tracker (CT^2) mHealth app, database, and server are connected to an instance of the OpenEMR

(OpenEMR, 2002) electronic health record via HAPI FHIR as reviewed in Section 2.7 of Chapter 2. The CT^2 app is for Android and iOS devices and uses a FHIR server to manage its data, which is stored in an instance of OpenEMR (OpenEMR, 2002) (accessed through a FHIR server as well and located in a cloud server). The CT^2 mHealth app utilizes four FHIR resources in order to Create (POST), Read (GET), and Update (PUT) subsets of the data (Delete is not allowed in this app): *Patient* to track demographics and other basic information of patients (students that suffer concussions); *Condition* to track a medical condition, in our case a concussion; *Observation* to track symptoms of patients (students); and *CarePlan* to track the planned treatment for a condition (concussion). The CRU services defined on these four FHIR resources are the ones that are authorized by role (Secure and Unsecure services from Defn. 17) and controlled by sensitivity (Labeled and Unlabeled services from Defn. 18). In our case, the services that need to be controlled are secured and labeled since all of them manage highly-sensitive data.

To begin, Figure 6.6 shows the mapping of resources and the location of the *access control server interceptor* of Section 6.3 and Figure 6.4 within the FHIR instance of the CT^2 mHealth app. Note concepts in Figure 6.4 can be mapped to Figure 6.6 as follows: Mobile App maps to CT^2 mHealth App; Mobile App API maps to CT^2 RESTful API; Mobile App HAPI FHIR Server maps to the CT^2 HAPI FHIR Server; Mobile App HAPI FHIR RESTful API maps to CT^2 HAPI FHIR RESTful API; Data Source HAPI FHIR Server maps to the OpenEMR FHIR Server. HIT System Server-Side API maps to OpenEMR RESTful API; and, HIT System (Data Source) maps to OpenEMR. Each of the FHIR Resources (Patient, Condition, etc.) for both CT^2 and OpenEMR have a FHIR RESTful API with CRU services (no delete), CT^2 HAPI FHIR RESTful API and

OpenEMR HAPI FHIR RESTful API, respectively. We incorporate the access control server interceptor as a security layer (Server Interceptor API and Access Control Server Interceptor class shown in Figure 6.6) before the requested resource attempts to retrieve/insert/update data from the OpenEMR system, similar to the first option in Figure 6.2, where the first FHIR layer consisted of a security layer. If the requested resource successfully passes the security checks in the server interceptor function, then the CT^2 HAPI FHIR RESTful API receives the request which is then sent to the appropriate service of OpenEMR HAPI FHIR RESTful API class along with any parameters by using another instance of FHIR. In this case, both of CT^2 HAPI FHIR RESTful API and OpenEMR HAPI FHIR RESTful instances were implemented using the HAPI FHIR library. Specifically, FHIR requires that information in the CT^2 database is mapped to/from FHIR resources and information in the OpenEMR repository is mapped to/from FHIR resources and, exchanged through shared resources via the CRUD FHIR APIs defined for each resource for both CT^2 HAPI FHIR RESTful API and OpenEMR HAPI FHIR RESTful APIs. We have reported extensively on this mapping process in (Baihan, et al., 2017).

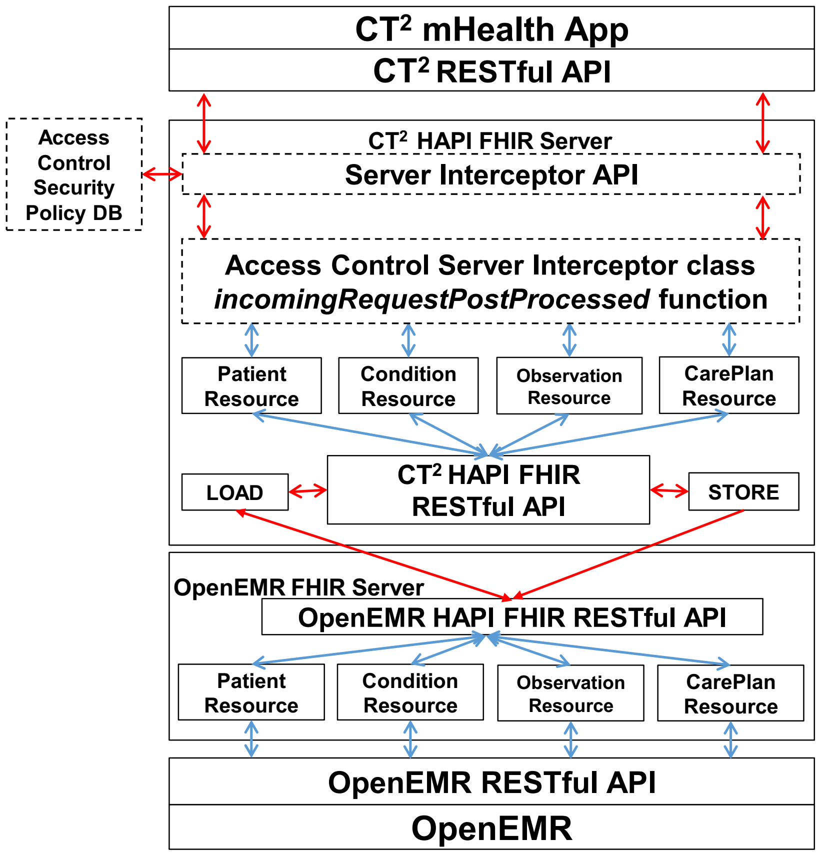


Figure 6.6. CT2-OpenEMR FHIR Mapping.

To determine the type of action a user of CT^2 can perform on these FHIR resources, we defined four roles: *Nurse* which has access to all of the services of the resources (CRU for the CT^2 HAPI FHIR RESTful API) to manage a student's concussion incident from its occurrence to its resolution; *Athletic Trainer* (*AT*) which is allowed to do a limited preliminary assessment if a concussion incident occurs at the event and hence some U services are not allowed; *Coach* which can report a concussion incident at an athletic event with very limited information on the student with no access to U services; and, *Parent* which can report a concussion incident on his/her child while attending the athletic event, update his/her concussed child's demographics, or track the current status of his/her children that have ongoing concussions but with no U services for the Condition, Observation, and CarePlan resources. In addition, we assigned clearances and MAC properties to users and classifications to services in support of MAC. Table 6.1 defines the permissions for CRU (POST, GET, PUT) for all of the resources for all four roles in support of RBAC as well as their assigned classification in support of MAC (see Defns. 8v2 and 10). Note that the services listed in Table 1 are both Secure and Labeled services of CT^2. These policies are stored in a security policy database (as shown in Figure 6.5 and Figure 6.6) and is accessed through the use of the incomingRequestPostProcessed function which is part of the server interceptor class the HAPI FHIR library offers. Notice that in Table 6.1, a user with a Nurse role and/or a user with clearance TS has access to all CRU services of all resources; at the other extreme, a Coach role and/or a user with clearance C is limited to reporting and reading the basic information on the concussion by having only access to CR for the Patient and Condition resources. The creation of the RBAC and MAC permissions to FHIR services by role and clearance in Table I allows fine-grained access control of the FHIR RESTful APIs for the services of the four Resources utilized by the CT^2 mHealth app. Table 6.2 depicts a subset of the users of the CT^2 mHealth app as well as their assigned role, clearance, and read and write MAC properties. Note that CT^2 mHealth app has many unsecure and unlabeled services for populating drop downs and select options of the UI, that have been omitted from the discussion in order to focus on services that need control.

The access control server interceptor pseudo code as given in of Appendix C-1 is transitioned to the access control server interceptor given in Appendix C-2. There are, five functions in the access control server interceptor. The main function (`incomingRequestPostProcessed` function), verifies the user's identity:

```
1   public boolean incomingRequestPostProcessed(RequestDetails theRequestDetails,
    HttpServletRequest theRequest, HttpServletResponse theResponse) {
2       String jwt = theRequest.getHeader("Authorization");
3       Boolean acPermission = false; //Initially, the user does not have access to the
resource
4       String identifiers = "";
5       JSONObject object = null;
6       HttpClient httpClient = new DefaultHttpClient();
7       HttpContext localContext = new BasicHttpContext();
8       // Verify if the user is a valid one
9       HttpGet httpGet= new HttpGet(serviceLink+"/verifyUser/"+jwt);
10      try {
11         HttpResponse response = httpClient.execute(httpGet, localContext);
12         HttpEntity entity = response.getEntity();
13         identifiers = EntityUtils.toString(entity);
14          //Returns user_id, role_id, clearance_id, write_property, and read property
15         object = new JSONObject(identifiers); //Convert String to JSON Object
16      } catch (Exception e) {/*throw new UnprocessableEntityException();*/}
17      //If the user's identity could be properly validated then it returns the user's
role,
18      //clearance, and an indicator that the request was successful
```

If the user's identity is valid, the function proceeds to verify if the requested resource is secure and/or labeled (lines 28-38), if not, the request continues to be processed (no further security checks are required):

```
19 try {
20      int user_id = Integer.parseInt(object.getString("user_id"));
21      int mac_read = Integer.parseInt(object.getString("mac_read"));
22      int mac_write = Integer.parseInt(object.getString("mac_write"));
23   if(user_id>0) {
24      JSONObject securedResource = null;
25      String httpMethod = theRequest.getMethod();
26      String resourceName = theRequestDetails.getResourceName();
27      //Check if requested resource is secured/labeled
28      httpGet = new
HttpGet(serviceLink+"/resourceSecurity/"+httpMethod+"/"+resourceName);
29      try {
30           HttpResponse response = httpClient.execute(httpGet, localContext);
31           HttpEntity entity = response.getEntity();
32           identifiers = getASCIIContentFromEntity(entity);
33           securedResource = new JSONObject(identifiers);
34      } catch (Exception e) {/*throw new UnprocessableEntityException();*/}
35      boolean secured = securedResource.getBoolean("secured");
36      boolean labeled = securedResource.getBoolean("labeled");
37      if(!(secured||labeled)){
38       return true; //Continue with request processing (resource can be accessed by
anyone)
39      }
```

If the requested resource is labeled, we do checks for enforcement for MAC by calling the `checkAndEnforceMAC` function (lines 90-153 in Appendix C-2). In the function, we first verify if the user has delegated clearance permissions for the requested resource by calling the `delClrDAC` function:

```
187 private int delClrDAC(int user_id, int service_id) {
188   Integer delclr_id = 0;
189   HttpClient httpClient = new DefaultHttpClient();
190   HttpContext localContext = new BasicHttpContext();
191   //MAC Service Delegation
192   //Check if delegated user has a delegated clearance for the requested service
193   JSONObject userDelegation = null;
194   HttpGet httpGet = new
HttpGet(serviceLink+"/userClrDelegation/"+user_id+service_id);
195   try {
196      HttpResponse response = httpClient.execute(httpGet, localContext);
197      HttpEntity entity = response.getEntity();
198      String identifiers = getASCIIContentFromEntity(entity);
199      userDelegation = new JSONObject(identifiers);
200      delclr_id = userDelegation.getInt("du_dclr_id");
201      return delclr_id;
202   } catch (Exception e){/*throw new UnprocessableEntityException();*/}
203   return 0;
204 }
```

For GET and read operations, we verify the user's clearance and the classification of the service against MAC read properties within the `checkAndEnforceMAC` function:

```
116  if(httpMethod=="GET") {
117      macProperty = mac_read;
118      //Simple security property
119      if (macProperty == 1) {
120          if (clearance_id > class_id) {
121              acPermission = true;
122          }
123      }
124      //Strict * property
125      else if (macProperty == 2) {
126          if (clearance_id == class_id) {
127              acPermission = true;
128          }
129      }
130  }
```

For all PUT, POST and write operations, where the request is to modify the data shown in the app, we evaluate the user's clearance and the service's classification against the corresponding MAC write property within the `checkAndEnforceMAC` function (lines 90-153 in Appendix C-2):

```
131  else{
132      macProperty = mac_write;
133      //Simple integrity property
134      if (macProperty == 3) {
135          if (clearance_id >= class_id) {
136              acPermission = true;
137          }
138      }
139      //Strict * property
140      else if (macProperty == 4) {
141          if (clearance_id == class_id) {
142              acPermission = true;
143          }
144      }
145      //Liberal * property
146      else if (macProperty == 5) {
147          if (clearance_id <= class_id) {
148              acPermission = true;
149          }
150      }
151  }
```

If the requested resource is secured, we do checks for enforcement for RBAC, by calling `checkAndEnforceRBAC` function (lines 155-185 of Appendix C-2). In the function, we first verify if the user has delegated role permissions for the requested resource by calling the `delRoleDAC` function:

```
206 private int delRoleDAC(int user_id, int service_id) {
207   Integer delrole_id = 0;
208   HttpClient httpClient = new DefaultHttpClient();
209   HttpContext localContext = new BasicHttpContext();
210   //RBAC Service Delegation
211   //Check if delegated user has a delegated role for the requested service
212   JSONObject userDelegation = null;
213   HttpGet httpGet = new
HttpGet(serviceLink+"/userRoleDelegation/"+user_id+service_id);
214   try {
215       HttpResponse response = httpClient.execute(httpGet, localContext);
216       HttpEntity entity = response.getEntity();
217       String identifiers = getASCIIContentFromEntity(entity);
218       userDelegation = new JSONObject(identifiers);
219       delrole_id = userDelegation.getInt("du_drole_id");
220       return delrole_id;
221   } catch (Exception e){/*throw new UnprocessableEntityException();*/}
222   return 0;
223 }
```

Then, we verify if the user's role has access to the service request within the `checkAndEnforceRBAC` function (lines 155-185 of Appendix C-2):

```
167   //Get service set of roles
168   HttpGet httpGet = new HttpGet(serviceLink+"/roleSet/"+service_id);
169   try {
170       HttpResponse response = httpClient.execute(httpGet, localContext);
171       HttpEntity entity = response.getEntity();
172       String identifiers = getASCIIContentFromEntity(entity);
173       JSONObject rs = new JSONObject(identifiers);
174       role_set = rs.getJSONArray("");
175       Integer role = 0;
176       for (int i = 0; i < role_set.length(); i++) {
177           role = role_set.getInt(i);
178           if(role_id == role) {
179               acPermission = true;
180               break;
181           }
182       }
183   } catch (Exception e) {//throw new UnprocessableEntityException();}
```

The main function outputs an error message if the user could either not be verified or, if the user does not have access to the requested resource (if either the MAC or RBAC functions return false):

```
58   if(!acPermission) {
59      try {
60         theResponse.setContentType("application/json+fhir");
61         PrintWriter out = theResponse.getWriter();
62         out.println("{");
63         out.println("\"status\": \"403\",");
64         out.println("\"errorMessage\": \"User does not have permission to access 
the 
           requested resource.\"");
65         out.println("}");
66         out.close();
67      } catch (IOException e) {e.printStackTrace();}
68      return false;
69   }
70   return true;
71 }
72 else {
73    try {
74       theResponse.setContentType("application/json+fhir");
75       PrintWriter out = theResponse.getWriter();
76       out.println("{");
77       out.println("\"status\": \"400\",");
78       out.println("\"errorMessage\": \"User Verification failed. Please try to do 
                    the request again...\"");
79               out.println("}");
80               out.close();
81           } catch (IOException e) {
82               e.printStackTrace();
83           }
84           return false;
85       }
```

The `incomingRequestPostProcessed` function (lines 1-88 of Appendix C-2) is included within a class (shown in Appendix C-3) that extends the `InterceptorAdapter` class (line 15 of Appendix C-3), which is part of the server interceptor feature HAPI FHIR offers. In addition, in order for the server to recognize the existence of the intercepting class, the access control server interceptor is registered in the `RestfulServer` instance of the FHIR server, namely, on the CT^2 FHIR server, by making reference to the class by calling `registerInterceptor` (line 16 of Appendix C-3).

Service	Classification	Roles
GET /Patient/	C	{AT, Coach, Nurse, Parent, Sub_Nurse}
POST /Patient/	C	{AT, Coach, Nurse, Parent}
PUT /Patient/:pid	TS	{Nurse, Parent}
GET /Condition/	C	{AT, Coach, Nurse, Parent, Sub_Nurse }
POST /Condition/	C	{AT, Coach, Nurse, Parent}
PUT /Condition/:cid	TS	{Nurse}
GET /Observation/	C	{AT, Coach, Nurse, Parent, Sub_Nurse }
POST /Observation/	S	{AT, Nurse}
PUT /Observation/:oid	TS	{Nurse}
GET /CarePlan/	C	{AT, Coach, Nurse, Parent, Sub_Nurse }
POST /CarePlan/	TS	{Nurse}
PUT /CarePlan/:cpid	TS	{Nurse}

Table 6.1. Service permissions of CT^2 FHIR resources.

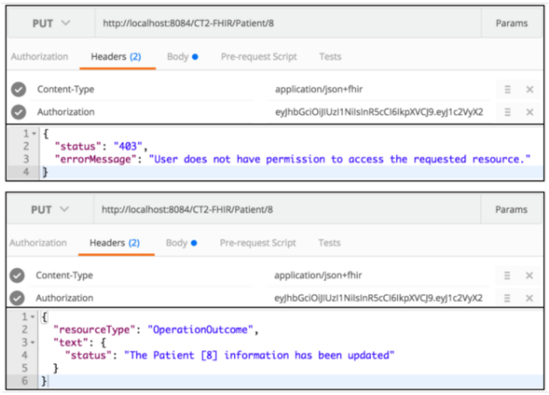
User	Role	Clearance	MAC Read	MAC Write
Peter	AT	S	SS	SI
Joe	Coach	C	SS	SI
Karen	Nurse	TS	SS	SI
Carmen	Parent	C	SS	L*
Lois	Sub_Nurse	C	S*-r	S*-w

Table 6.2. Permission assignment of CT^2 users.

The CT^2 mHealth app was tested with two accounts: user *Karen* (third row of Table 6.2) and user *Joe* (second row of Table 6.2). In the test, both Karen and Joe attempt to utilize the CT^2 mHealth app to update general information of a student (the UPDATE/PUT operation). As we can see in Table 6.1, the classification of the requested service both users are attempting to access is *TS* and the only role who has access to this particular service is the *Nurse* role (shown in second row and third column of Table 6.1). Taking these details into consideration, we move on to evaluate each of the users' assigned clearance and role. Karen has a clearance of *TS* and the *Nurse* role assigned therefore, she has access to perform the aforementioned request since $u_{CLR} \geq u_{CLS}$ (satisfies the Simple Integrity MAC write property, see Defn. 10) and, her role is within the roles that the requested service combination allows (see Defn. 11). On the other hand, Joe has a clearance of *C* and C<TS (does not satisfy the Simple Integrity MAC write property, see Defn. 10), therefore, the first security check fails meaning that we do not proceed to verify the user's (Joe's) role as both security checks (in the case of there being RBAC and MAC checks) need to return true. Therefore, he obtains an error message informing him that he does not have

permission to access the requested resource (lines 58-69 of Appendix C-2). In addition, Nurse Karen has the ability to delegate her RBAC and MAC permissions. Nurse Karen can delegate a subset of her allowed Secure API services to the substitute Nurse, *Lois* (see Defn. 30 on Section 3.5 of Chapter 3) by assigning Lois the *Sub_Nurse* role. In this case, we are not considering Labeled API services, so we only evaluate Lois' request with the `checkAndEnforceRBAC` function (lines 155-185 of Appendix C-2). If Lois attempts to modify the data for a student, she'd be prevented since she only has permission to read data from the app (see substitute Nurse Lois' permissions on Tables 6.1 and 6.2). As a result, the RBAC security check fails and Lois receives an error message that informs her that she does not have permission to perform such an action (lines 58-69 of Appendix C-2).

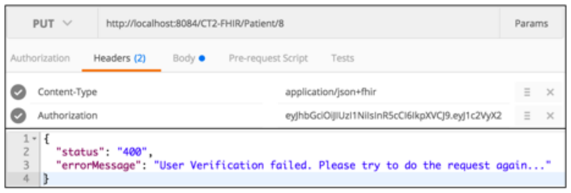
Programmatically, in the case of Coach Joe and substitute Nurse Lois, the incomingRequestPostProcessed function located within the CT^2 FHIR server java code aborts the execution of the request and returns an error message to inform the user that he does not have permission to the requested resource as shown in the top image of Figure 6.7. In the case of Nurse Karen, the request is performed successfully as shown in the bottom image of Figure 6.7. Another situation that is also supported in the access control server interceptor is if the user attempts to tamper with his/her role and/or clearance in order to obtain more privileges. This action is identified when the access token sent on the header is verified and blocks any further execution of the request and send an error message to the user telling him/her that the user verification failed as shown in Figure 6.8. Note that the access control server interceptor can return error messages as either JSON or XML format, which are two of the formats that are available in FHIR to return the client a response. We utilized JSON since this is the format that the CT^2 mHealth app has for handling the user requests and responses. In addition, the requests shown on Figures 6.7 and 6.8 were made with Postman (Postman, 2013) instead of doing these directly with the CT^2 mHealth app in order to present a clear view of the response the requests can have in different scenarios.



```
PUT http://localhost:8084/CT2-FHIR/Patient/8
Content-Type     application/json+fhir
Authorization    eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX2
{
  "status": "403",
  "errorMessage": "User does not have permission to access the requested resource."
}
```

```
PUT http://localhost:8084/CT2-FHIR/Patient/8
Content-Type     application/json+fhir
Authorization    eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX2
{
  "resourceType": "OperationOutcome",
  "text": {
    "status": "The Patient [8] information has been updated"
  }
}
```

Figure 6.7. JSON Response Messages from Interceptor.



```
PUT http://localhost:8084/CT2-FHIR/Patient/8
Content-Type     application/json+fhir
Authorization    eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VyX2
{
  "status": "400",
  "errorMessage": "User Verification failed. Please try to do the request again..."
}
```

Figure 6.8. JSON Response – Disallowed Request.

6.5. Related Work

There are many efforts that propose access control mechanisms to secure mobile applications by limiting the permissions and resources a mobile app can access in different areas of the mobile device/app. In this section, we discuss related work in three areas: permission control in Android; extending RBAC with context-aware techniques; and, proposed security mechanisms for FHIR. As part of the discussion, we compare and contrast these efforts to the work presented herein.

The first area of related work involves permission control in Android where many approaches (Beresford et al., 2011; Benats, Bandara, Yu, Colin, & Nuseibeh, 2011; Wang et al., 2014; Jin et al., 2015; Hao et al., 2013) focus on applying fine-grained access control policies to mobile devices that contain Android as their operating system. This is due to the fact that Android contains a coarse-grained access control mechanism for allowing permissions in mobile applications. Specifically, in order for a user to install a mobile app, he/she needs to accept all of the permissions that the app requires. This may disregard the fact that some permissions may not be necessary for the app to function and that some of the permissions may not make sense for the app that is being downloaded and could result in using the allowed component for malicious purposes (e.g., a flashlight app tells user it needs permission to get the user's location). Adding fine-grained access control to the APIs that Android utilizes for the device and apps to function properly has been addressed by: mocking the values that an app receives in order to function (Beresford, et al., 2011) (e.g., mocking latitude and longitude coordinates); extending the security policies of the mobile device (Benats, Bandara, Yu, Colin, & Nuseibeh, 2011; Wang, Hariharan, Zhao, Liu, & Du, 2014; Jin, Wang, Luo, & Du, 2015); rewriting the bytecode of the mobile device (Hao, et al., 2013); and adding security modules to the mobile device (Backes, et al., 2014). In contrast to these efforts, our work presented in this chapter focuses on applying access control mechanisms to the APIs that are not part of the mobile system itself (we enforce security in FHIR API services), with the intent to control access to the data utilized by the mobile app via its services. In addition, while most of these works are specific for Android OS/API, our service-based RBAC and MAC approach can be applied to any type of application (even though we focus on the mobile setting) by controlling access to the services. This was demonstrated via FHIR and the HAPI FHIR library that is enforced within the HAPI FHIR server, but is adaptable to other cloud frameworks.

The second area of related work involves role-based access control and extensions that expand RBAC with context-aware techniques in order to provide finer-grained access control security policies to those systems that contain highly sensitive data. One effort proposes an RBAC model with a spatiotemporal extension for web applications (Aich, et al., 2009) that can be applied to an existing system as a dll component. Apart from web applications, our server-based RBAC and MAC approach could work for any type of system that utilizes a JAVA-based implementation of the server that handles the users' API requests and responses. Another approach proposes a dynamic RBAC approach for Android devices (Rohrer, et al., 2013). That approach focuses on modifying the Android framework to provide a uniform security policy to mitigate security risks in mobile devices that are utilized by users who are part of an enterprise. Unlike the aforementioned approach, our work is not device-specific and is enforced on the server side of a mobile application (and can be applied to other systems as well). Finally, an effort (Fadhel et al., 2016) proposed a model that extends RBAC to generate RBAC conceptual policies, but does not provide details of which specific application domain(s) the approach could support. This is contrary to our approach which is intended to work for any type of application that can be connected with a JAVA-based server. In addition, our access control server interceptor as discussed in Section 4.3 operates on the server-side and can be easily extended to support other types of access control as given in (Aich et al., 2009; Rohrer et al., 2013; Fadhel et al. , 2016), expanding the interceptor logic (Appendix C-2). Moreover, even though we are focusing on utilizing the HAPI FHIR server to enforce our access control model, we intend to generalize this approach so that it works with a wider variety of cloud frameworks that use RESTful, cloud, or web APIs.

The third area of related work involves security mechanisms which can be applied to the FHIR specification. The FHIR standard is an evolving standard (FHIR Release 3), for which there are a number of efforts that address the way that security in FHIR could be realized. The first effort (Lamprinakos, et al., 2014) applied FHIR to the AidIT mHealth app for patients, doctors, and pharmacists in order to demonstrate the way that a mHealth app can be integrated with the FHIR server side. The AidIT app can be utilized by users who have different privileges, uses QR codes to secure the data, and includes an RBAC mechanism to apply security policies to CRUD actions. The access control model is enforced within the user interface of the mobile app by only showing the components (buttons and tabs) that can be accessed according to a user's role. Unlike our work, this approach does not enforce access control via FHIR and the RESTful API services, but focuses on programmatic changes to the AidIT mHealth app making reuse of the approach problematic for other mobile apps. Our approach to control services by role and by clearance/classification is applicable to any mHealth app accessing RESTful API FHIR services to be controlled by RBAC and MAC, and is reusable without needing to modify app code. The second effort (Franz, et al., 2015) applies FHIR to a health monitoring system of vital signs (e.g., blood pressure, blood sugar, etc.) for patients in cardiac rehab and for elderly with chronic diseases. The effort collected information using the Observation FHIR resource for 68 patients and nearly 2000 data points. Such an effort would need services used by patients to store their individual data and for researchers to be able to access data from multiple participants. Clearly, there is a need to have the ability to control who can enter/view data for different stakeholders (roles), making our approach suitable. However, despite the work acknowledging the need to support security, this was a subject for future work. The third approach, SecFHIR (Altamimi, 2016), presented a security specification model on FHIR resources represented using XML or JSON schemas. The work on SecFHIR defined permissions on schemas, which implicitly specify the permissions on the corresponding XML instances. As previously mentioned, ONC (Health and Human Services Department, 2015) is promoting a national effort to have RESTful APIs for healthcare data availability and security at the data level using XML in SecFHIR while relevant, is not targeting the way that mHealth applications access healthcare data via services. Our focus on applying RBAC and MAC at the role/clearance-classification/RESTful API level means that our policies are applicable regardless of the actual data format. Our work also does not need to add permission tags that would impact the FHIR resource standard schemas for XML and JSON.

Chapter 7
Conclusion

This dissertation presented and explained a comprehensive configurable framework for RBAC (Ferraiolo & Kuhn, 1992), MAC (Sandhu & Samarati, 1994) and DAC (Department of Defense, 1985) for mobile applications that is capable of supporting access control in different security layers. Security was controlled: for the user interface in terms of which screens and/or their components are accessible to a user under RBAC with optional delegation via DAC; for interactions from the mobile app to the mobile application's API services in order to define the API services that can be invoked by a particular user based on RBAC and/or MAC permissions with optional delegation via DAC; and, for interactions between the mobile app API services that seek to invoke the API of server-side data sources also based on RBAC and/or MAC permissions with optional delegation via DAC. The main objectives have been four-fold: define a Software Architecture for a Configurable Access Control Framework for Mobile Applications; create a Unified Mobile Computing and Security Model with Access Control to capture characteristics of mobile applications and RBAC, MAC, and DAC that are defined on both the user interface and various levels of API services; describe the ability to support the Dynamic Combination of Access Control Models and Configuration Options; and, define and implement Access Control Security Enforcement Code Generation and Interceptors that operates for both the look-and-feel by RBAC of the mobile app as well as the services that can be intercepted for RBAC, MAC, and DAC delegation checks on services. These objectives were illustrated using a mHealth application linked to OpenEMR via the HAPI FHIR server.

The remainder of this conclusion is organized as follows. Section 7.1 summarizes the dissertation, highlighting the attainment of the four aforementioned main objectives in detail. Using this as a basis, Section 7.2 discusses the research contributions of this dissertation, primarily in the areas of: Software Architecture for a Configurable Access Control Framework for Mobile Applications; Unified Mobile Computing and Security Model with Access Control to capture characteristics of mobile applications and RBAC, MAC, and DAC; Dynamic Combination of Access Control Models and Configuration Options; and, Access Control Security Enforcement Code Generation and Interceptors. Then, on Section 7.3, we detail the ongoing and future research directions that include, but are not limited to: supporting other access control models, supporting another cloud framework for the interceptor, finer-grained DAC for UI, controlling services by instances, adding time-based permissions, and testing the configurable framework with other mobile apps and data sources.

7.1. Summary

The work presented in this dissertation attempts to incorporate RBAC, MAC, and DAC access control mechanisms in different parts of the architecture of mobile applications in order to protect highly-sensitive data managed by mobile applications. The main focus of the dissertation was to create a software architecture for our configurable framework approach in which we could define different permissions on the UI, API, and data sources of the mobile application that could be combined and enforced either on the source code of a mobile application, on the mobile application's APIs, or on the server-side APIs of a mobile application. In support of our focus, the discussion was presented throughout six chapters.

Chapter 1 introduced the main areas for our research and a high-level view of the proposed configurable access control framework. Section 1.1 discussed the motivation of adding access control to mobile applications as our main area of interest. Section 1.2 explore the motivation of the work in the healthcare domain as an appropriate context to present the work of the dissertation due to its need for strict control of PHI and the emergence and usage of mobile health (mHealth) applications for patients and medical providers. As noted in Section 3.1, mHealth applications have critical requirements regarding information usage and exchange with different stakeholders needing different types of access. With the motivation in hand, Section 1.4 presented and explained the configurable framework for RBAC, MAC and DAC for mobile applications of this dissertation. Section 1.5 provided a list of the research objectives and expected contributions for the dissertation. Section 1.6 discussed the work that has been published by us in order to support the work presented in the dissertation. Finally, Section 1.7 presented an outline of the dissertation.

Chapter 2 provided background information on the main concepts and topics that support the discussion and explanation of the dissertation. Section 2.1 presented the logical architecture of a mobile application on different layers and their interaction. Section 2.2, 2.3, and 2.4 reviewed, respectively, Role-Based Access Control (RBAC) (Ferraiolo & Kuhn, 1992), Mandatory Access Control (MAC) (Bell & La Padula, 1976), and Discretionary Access Control (DAC) (Department of Defense, 1985). Section 2.5 reviewed the application programming interface (API) concept which is instrumental in support of permissions based on which user is authorized to which API service. Section 2.6 explained the Fast Healthcare Interoperable Resources (FHIR) specification (FHIR DSTU2, 2015) and the HAPI FHIR reference implementation (HAPI FHIR, 2014), which support the proof-of-concept discussion in Chapter 6. Section 2.7 introduced and reviewed the Connecticut Concussion Tracker (CT^2) mobile application, a collaboration between the Departments of Physiology and Neurobiology, and Computer Science & Engineering at the University of Connecticut and Schools of Nursing and Medicine in support of a new law passed to track concussions of children from kindergarten through high school in public schools (CT Law HB6722) (Connecticut General Assembly, 2015).

Chapter 3 contained a detailed discussion of a unified model of access control for mobile applications. Section 3.1 introduced definitions for the generalized structure of a mobile application. Section 3.2 reviewed definitions for RBAC and MAC including: roles, sets of roles, users, sets of users' clearances and classification for MAC. Section 3.3 presented definitions for RBAC permissions on the user interface. Section 3.4 presented definitions for RBAC and MAC permissions on the mobile application API that is partitioned into secure/unsecure services (RBAC) and labeled/unlabeled services (MAC); and discussed service permission assignment to roles and users. Section 3.5 detailed definitions for DAC that included the delegation of permissions from one user/group to another user/group for RBAC permissions on the UI of a mobile application and RBAC and/or MAC permissions on the services of the mobile application API. Collectively, the model presented in Sections 3.1 to 3.5, allowed for the ability to model RBAC, MAC, and/or DAC on the mobile application and its API and supports contribution B: Unified Mobile Computing and Security Model with Access Control from Section 1.5 of Chapter 1. Section 3.6 discussed the ability to take the model concepts as given in Sections 3.1 to 3.5 and pick-and-choose in order to define and design a unique set of security capabilities for each mobile application; this supports contribution C: Dynamic Combination of Access Control Models and Configuration Options. Section 3.7 contained an entity relationship diagram to store information programmatically from the Unified Security Model in Sections 3.1 to 3.5. Finally, Section 3.8 presented related work on access control in mobile computing.

Chapter 4 presented the security policy definition and generation process for the screens, components, and interactions of the user interface for the *Direct UI Modifications* option (see Section 1.4 and Figure 1.3 again) that changes the look-and-feel of the UI according to RBAC and/or DAC permissions. Section 4.1 reviewed a subset of the model and permissions from Chapter 3 for the mobile app UI that define which screens and components can be viewed/edited/viewed once/enabled/hidden in order to customize the look-and-feel of the UI by role. Section 4.2 reviewed a subset of the ER diagram for the unified security model in Figure 3.3 of Section 3.7, focusing on UI, screens, components, screen interactions, users, roles, permissions, and optional delegation with examples using the CT^2 mobile app. Section 4.3 explained the programmatic changes that must be made to the mobile application itself to allow for the screens and their components to be customized. Section 4.4 provided a guide that stated which programmatic changes need to be done in a mobile app in order to apply the Direct UI Modifications option. Finally, Section 4.5 presented related work on the customization of user interfaces via adaptive UIs and the usage of RBAC.

Chapter 5 presented the *Intercepting API calls* option on the interactions between the UI and the mobile applications' API services to control by RBAC, MAC, and/or DAC permissions which services are allowed to be invoked for on a user-by-user basis through the generation of an intercepting API that mirrors the original mobile applications API. Section 5.1 motivated the Intercepting API Calls option by explaining the important role of the API in accessing information, especially PPI and PHI. Section 5.2 presented the high-level processing of the Intercepting API Calls option using the classic architecture of the User Layer, Presentation Layer, Business Layer, and, Data Layer. Section 5.3 explored the underlying processing of the Intercepting API Calls option by examining the way that API services are categorized. Section 5.4 examined the Interactions and Infrastructure for the Intercepting API Calls option. Section 5.5 explored the algorithm generation process for the Intercepting API Calls option. Section 5.6 illustrated the Intercepting API Calls option via the CT^2 mHealth application. Finally, Section 5.7 discussed related work in security and access control mechanisms for mobile applications.

Chapter 6 presented the *Server Interceptor API* option on the interactions between the mobile application's API services and their invocations to server-side APIs of data sources, with a server interceptor API defined using the HAPI FHIR reference implementation. Section 6.1 motivated the way that the Intercepting API Calls option as given in Chapter 5 is adapted and evolved to the Server Interceptor API option. Section 6.2 reviewed the HAPI FHIR reference implementation capabilities with a focus on the intercepting process. Section 6.3 presented a set of modifications that incorporate RBAC, MAC, and DAC into the FHIR specification and its realization within the HAPI FHIR reference implementation. To demonstrate the inclusion and realization of RBAC, MAC, and DAC for a mHealth app within HAPI FHIR, Section 6.4 provided an implementation of the service-based access control approach by using the Connecticut Concussion Tracker

(CT^2) mHealth app and the OpenEMR HIT system (OpenEMR, 2002). Finally, Section 6.5 reviewed related works and compares and contrasts these to the work of this chapter.

7.2. Research Contributions

This section revisits the expected research contributions given in Section 1.5 of Chapter 1 and provides insight of their attainment across the chapters of the dissertation. The Configurable Framework for RBAC, MAC, and DAC for Mobile Applications has the following contributions:

A. **Software Architecture for a Configurable Access Control Framework for Mobile Applications**: The contribution involved the specification, design, and description of a software architecture for the configurable access control framework as given in Figure 1.4 of Chapter 1. This facilitates the ability to insert role-based, mandatory, and discretionary access controls at alternate and multiple locations throughout the architecture. In support of this contribution, Chapter 4, 5, and 6 provided details of the architecture of the different options that are part of the configurable framework (shown in Figure 1.3 of Chapter 1): Direct UI Modifications option (Figure 4.4 of Chapter 4), Intercepting API Calls option (Figure 5.1 of Chapter 5), and Server Interceptor API option (Figure 6.4 of Chapter 6).

B. **Unified Mobile Computing and Security Model with Access Control**: The first four components in Figure 1.4 of Chapter 1 (i.e., Mobile Application, Mobile Application Clients, Access Control Models, and Permissions and Impact on Mobile App) all influenced the creation of a unified mobile computing and security model which contains: a generalized structure of a mobile application as a user interface of screens, components (text fields, drop down, buttons, etc.), and interactions among screens (Defns. 1-5 in Section 3.1 of Chapter 3); roles, sets of roles, users, and sets of users (Defns. 6-12 in Section 3.2 of Chapter 3); permission assignments of users and roles on screens, components, and interactions (Defns. 13-16 in Section 3.3 of Chapter 3); mobile application API that is partitioned into secure/unsecure services (RBAC) (Defn. 17 in Section 3.4 of Chapter 3) and labeled/unlabeled services (MAC) (Defns. 18 in Section 3.4 of Chapter 3); service permission assignment to roles and users (Defns. 19-21 in Section 3.4 of Chapter 3); and, delegation permissions assignment (Defns. 22-33 in Section 3.5 of Chapter 3). This allowed the ability to model role-based, mandatory, and discretionary access controls on the mobile application and the mobile app's API and server-side APIs of data sources.

C. **Dynamic Combination of Access Control Models and Configuration Options:** The third and fourth components in Figure 1.4 of Chapter 1 (i.e., Access Control Models, Permissions and Impact on Mobile App) are combinable on different ways. The contribution provides the ability to combine different aspects of access control models (RBAC, MAC, and DAC), of the mobile application (UI, API, and APIs of Data Source), and of the configuration options (Direct UI Modifications, Intercepting API Calls, and Server Interceptor API) into custom access control solutions for a mobile application as given in Table 3.11 in Section 3.6 of Chapter 3. Each allowable combinations in Table 3.11 when selected results in the Generation of Security Policies (fifth component in Figure 1.4) which in turn supports the specific Enforcement of Security Policies (sixth component in Figure 1.4).

D. **Access Control Security Enforcement Code Generation and Interceptors**: The fifth and sixth components in Figure 1.4 of Chapter 1 (i.e., Generation of Security Policies and Enforcement of Security Policies) are the programmatic changes or generation of interceptor code for the configuration framework (research contribution A) and the chosen combination (Contribution C) under the unified model (Contribution B). For the Direct UI Modifications option, a process for modifying mobile app code was described in Sections 4.3 and 4.4 of Chapter 4 with associated source code given in Appendix A. Processes for the Direct UI Modifications are often human assisted and involve the need to actually modify limited portions of the mobile application code, API, and/or server database. For the Intercepting API Calls and Server Interceptor API options, algorithms were generated for the different configuration options for the framework that support the interceptors for the Intercepting API Calls (pseudocode and source code explained on Section 5.5 of Chapter 5 and fully shown on Appendix B), and Server Interceptor API options (pseudocode and source code explained on Section 6.3 and 6.4, and fully shown on Appendix C). Algorithms for the Intercepting API Calls, and Server Interceptor API options were defined for those cases where actual code is generated.

7.3. Ongoing and Future Work

The work presented in this dissertation can serve as a foundation for further enhancements and extensions. A list of ongoing and future topics includes: incorporating additional access control models in support of Direct UI Modifications, Intercepting API Calls, and Server Interceptor API options; supporting alternate cloud frameworks for the interceptor; Partial RBAC UI Delegation to have fine-grained control on delegating a subset of screens and/or component permissions; controlling access to services by instances to limit which data can be modified; time-based permissions for the UI and API which have the ability to expire; and, inclusion of additional mobile apps and data sources to demonstrate feasibility of the work.

Incorporating Additional Access Control Models: Currently, our configurable framework utilizes RBAC, MAC, and DAC to enforce security mechanisms on different parts of the mobile application architecture in support of Direct UI Modifications, Intercepting API Calls, and Server Interceptor API options. As part of future work, we are considering additional access control models. For example, attribute-based access control (ABAC) and identity-based access control (IBAC) can be useful to generate finer-grained access control and to contemplate other healthcare scenarios that could benefit from these models (e.g., granting access to a user if he/she is on an specific location for perhaps a physician that moves among locations). This future work will require additions to both the unified security model and interceptors.

Supporting Alternate Cloud Frameworks for the Interceptor: Presently, our Server Interceptor API option of the configurable framework relies on the HAPI FHIR library in order to implement the server interceptors that enforce security on the server-side APIs. As part of ongoing and future work, we are contemplating to support other cloud frameworks such as openstack (openstack, 2012) and cloud stack (Apache Cloud Stack, 2016). In addition, we are exploring the generalization of our service-based RBAC, MAC, and DAC approach with HAPI FHIR in order to obtain a solution that can be utilized in other apps that implement FHIR, and more generally, to other cloud computing frameworks.

Partial RBAC UI Delegation: Recall that the Direct UI Modifications option of the configurable framework can enforce RBAC and DAC permissions on the screens, screen components, and screen interactions of a mobile application. Specifically, DAC permissions allow an original user (see Defn. 22 on Section 3.5 of Chapter 3 again) or a delegated user (see Defn. 24 on Section 3.5 of Chapter 3 again) with pass-on-delegation authority (see Defn. 27 on Section 3.5 of Chapter 3 again) to delegate his/her full UI permissions (see Defn. 28 on Section 3.5 of Chapter 3 again) to a delegated user. The future work consider DAC delegations that could be more fine-grained by allowing users to delegate a subset of their UI permissions meaning that they would have the choice to delegate a subset of the screens, screen components, and screen interactions they have access to instead of all of these permissions as it currently does.

Controlling Access to Services by Instances: As previously mentioned, our configurable framework supports the assignment of a role and/or clearance to users of a mobile application to control access to the mobile app services and server-side services. However, we want to control which values the user can send/retrieve from the data source instead of allowing them to send/retrieve all of the values that the service contains. In other words, we want to be able to control which parts of the data a user can add/update/delete from the mobile application and, which data a user can retrieve when he/she requests it through the means of an API service. For example, in the CT^2 mHealth app, we would want to limit parents to only have access to the data of their own children, or in a school district that has multiple schools, limit a nurse to only seeing students within his/her school. This work might also be achieved by constraining the parameters and/or return types values on a service by service basis.

Time-Based Permissions for UI and APIs: Currently, a user has access to the UI screens and screen components of a mobile app, and to the API services depending on their role/clearance/MAC properties/delegations. This future work would augment these permissions with time constraints that are definable on users, roles, UI permissions, service permissions, etc., so a particular user/role or user/clearance only has access to a specific UI component/API service at certain periods of time. For instance, suppose a user contains a role that should only be available from 9am to 5pm on weekdays. With time-based permissions we can restrict the user to the specified role between that period of time and could assign another role that has less privileges on the weekends, for instance.

Inclusion of Additional Mobile Apps and Data Sources: In examples throughout the dissertation, we relied on the the CT^2 mHealth app and OpenEMR data source to test the Direct UI Modifications, Intercepting API Calls and Server Interceptor API options; nonetheless, the app only connects to one data source. Therefore, we plan to continue to test our approach with other mHealth apps that obtain data from multiple data sources and determine if this affects our configurable framework. For example, ShareMyHealth is an mHealth app developed over the last year, by a team of undergraduate students at the University of Connecticut, for Android and iOS devices. ShareMyHealth provides patients with a means to manage and share their fitness data across multiple systems. Patients can gather data from multiple sources (e.g., MyGoogle, OpenEMR, etc.) that can then be made available to medical providers. Applying our configurable framework to ShareMyHealth can led us to consider if we need to make modifications to the framework in terms of contemplating other locations to enforce access control mechanisms and also determine how we could add such mechanisms to the APIs if they have existing access control permissions.

References

Abdunabi, R., Sun, W., & Ray, I. (2014). Enforcing spatio-temporal access control in mobile applications. *Computing*, *96*(4), 313-353.

Aich, S. Mondal, S., Sural, S., & Majumdar, A. K. (2009). "Role Based Access Control with Spatiotemporal Context for Mobile Applications," *Transactions on Computational Science IV: Special Issue on Security in Computing*.

Aitken, M. (2013, October). *Patient Apps for Improved Healthcare: From Novelty to Mainstream*. Retrieved from http://obroncology.com/imshealth/content/IIHI%20Apps%20report%20231013F_interactive.pdf.

Alhaqbani, B., & Fidge, C. (2008). Access Control Requirements for Processing Electronic Health Records, Business Process Management Workshops. (A. ter Hofstede, B. Benatallah, & H. Paik, Eds.) *LNCS, 4928*, 371-382.

Altamimi, A. (2016). "SecFHIR: A security specification model for Fast Healthcare Interoperability Resources". *International Journal of Advanced Computer Science and Applications (ijacsa)*, *7*(6).

Apache CloudStack. (2016, June 15). *Apache CloudStack: Open Source Cloud Computing*. Retrieved from https://cloudstack.apache.org/.

Apigee. (2015). *Apigee: API Management*. Retrieved from http://apigee.com/about/.

Apigee Docs. (2015). *Managing organization users*. Retrieved from http://docs.apigee.com/api-services/content/managing- organization-users.

Apple. (2015). *ResearchKit and CareKit*. Retrieved from http://www.apple.com/researchkit/.

Backes, M., Bugiel, S., Gerling, S., & von Styp-Rekowsky, P. (2014). "Android Security Framework: Extensible multi-layered access control on Android," in *30th Annual Computer Security Applications Conference*.

Baihan, M., Rivera Sánchez, Y. K., Shao, X., Gilman, C., Demurjian, S., & Agresta, T. "A Blueprint for Designing and Developing an mHealth Application for Diverse Stakeholders Utilizing Fast Healthcare Interoperability Resources," in *Contemporary Applications of Mobile Computing in Healthcare Settings*, R. Rajkumar, Ed., IGI Global.

Baracaldo, N. & Joshi, J. (2013). An Adaptive Risk Management and Access Control Framework to Mitigate Insider Threats. *Computers & Security* (39), 237-254.

Bell, D. E., & La Padula, L. J. (1976). Secure Computer System: Unified Exposition and Multics Interpretation. *MITRE Corp*.

Benats, G., Bandara, A., Yu, Y., Colin, J., & Nuseibeh, B. (2011). "PrimAndroid: privacy policy modelling and analysis for android applications," in *Symposium on Policies for Distributed Systems and Networks (POLICY '11)*.

Beresford, A., Rice, A., Skehin, N., & Sohan, R. (2011). "MockDroid: trading privacy for application functionality on smartphones". In *12th Workshop on Mobile Computing Systems and Applications*, Phoenix, Arizona.

Biba, K. J. (1977, April). Integrity considerations for secure computer systems. *Technical report, MITRE Corp*. Bedford, MA, USA.

Bleigh, M. (2010, April 29). *REST isn't what you think it is, and that's OK*. Retrieved from https://www.mobomo.com/2010/04/rest-isnt-what-you-think-it-is/.

Bugiel, S., Heuser, S., & Sadeghi, A. (2013). Flexible and Fine-grained Mandatory Access Control on Android for Diverse Security and Privacy Policies. *22nd USENIX Security Symposium*, 131-146.

Caine, K., & Hanania, R. (2013). Patients want granular privacy control over health information in electronic medical records. *Journal of the American Medical Informatics Association*, *20*(1), 7-15.

Capzule. (2012). *Capzule PHR*. Retrieved from http://www.capzule.com/.

Care360. (2014). *MyQuest*. Retrieved from https://myquest.questdiagnostics.com/web/home.

Cisco. (2014). Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2014-2019. Retrieved from http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11-520862.pdf.

Cobb, M. (2014, March 11). *API security: How to ensure secure API use in the enterprise*. Retrieved from http://searchsecurity.techtarget.com/tip/API- security-How- to-ensure-secure-API-use-in-the-enterprise.

Cohen, J. (2015, January 7). *11 Health and Fitness Apps That Achieve Top Results*. Retrieved from http://www.forbes.com/sites/jennifercohen/2015/01/07/the-11-top-health-fitness-apps-that-achieve-the-best-results/#11f5c21a1aca.

Collet, S., 2015. *API security leaves apps vulnerable: 5 ways to plug the leaks.* Retrieved from http://www.csoonline.com/article/2956367/mobilesecurity/api-security-leaves-apps-vulnerable-5-waysto-plug-the-leaks.html.

Connecticut General Assembly. (2015). *Substitute for Raised H.B. No. 6722*. Retrieved from https://www.cga.ct.gov/asp/CGABillStatus/CGAbillstatus.asp?which_year=2015&selBillType=Bill&bill_num=HB6722.

CVS Pharmacy. (2015). *myCVS On the Go*. Retrieved from http://www.cvs.com/mobile-cvs.

Dellinger, A.J. (2015, November 11). *This Instagram app may have stolen over 500,000 usernames and passwords*. Retrieved from http://www.dailydot.com/debug/instaagent-instagram-app-malware-ios-android/.

Department of Defense. (1985, December 26). *Department of Defense Trusted Computer System Evaluation Criteria*. Retrieved from http://csrc.nist.gov/publications/history/dod85.pdf.

DICOM. (2012). *Digital imaging and communications in medicine.* Retrieved from http://dicom.nema.org/.

Drugs.com. (2008). *Drugs.com App*. Retrieved from http://www.drugs.com/apps/.

Duffy, J. (2016, December 28). *The 25 Best Fitness Apps of 2017*. Retrieved from http://www.pcmag.com/article2/0,2817,2485287,00.asp.

eMarketer. (2015, January 8). *Tablet Users to Surpass 1 Billion Worldwide in 2015*. Retrieved from https://www.emarketer.com/Article/Tablet-Users-Surpass-1-Billion-Worldwide-2015/1011806.

Fadhel, A., Bianculli, D., Briand, L., & Hourte, B. (2016). A Model-driven Approach to Representing and Checking RBAC Contextual Policies. *CODASPY 2016*, 243–253. ACM.

Fernández-Alemán, J. L., Señor, I., Lozoya, P., & Toval, A. (2013). Security and privacy in electronic health records: A systematic literature review. *J Biomed. Inform, 46*(3), 541–562.

Ferraiolo, D., & Kuhn, R. (1992). Role-Based Access Control. In *Proceedings of the NIST-NSA National (USA) Computer Security Conference*, 554-563.

Ferraiolo, D., Sandhu, R., Gavrila, S., Kuhn, D., & Chandramouli, R. (2001). Proposed NIST standard for role-based access control. *ACM Transactions on Information and System Security (TISSEC)*, 4, 224-274.

FHIR. (2016, September). *FHIR security*. Retrieved from https://www.hl7.org/fhir/security.html.

FHIR DSTU2. (2015). *Guide to resources*. Retrieved from https://www.hl7.org/fhir/resourceguide.html.

FHIR Resources. (2015). *Resource Index*. Retrieved from https://www.hl7.org/fhir/resourcelist.html.

Franz, B., Schuler, A., & Krauss, O. (2015, January). "Applying FHIR in an integrated health monitoring system". *EJBI 2015*, *11*(2), 51-56.

Gajanayake, R., Iannella, R., & Sahama, T. (2014). Privacy Oriented Access Control for Electronic Health Records. (e15, Ed.) *Special Issue on e-Health Informatics and Security, electronic Journal for Health Informatics, 8*(2).

Gartner Newsroom. (2015). *Gartner Says Global Devices Shipments to Grow 2.8 Percent in 2015*. Retrieved from http://www.gartner.com/newsroom/id/3010017.

Google Play. (2013). *Fitness Tracker*.
Retrieved from https://play.google.com/store/apps/details?id=com.realitinc.fitnesstracker.

Hafner, M., Memon, M., & Alam, M. (2007). Modeling and Enforcing Advanced Access Control Policies in Healthcare Systems with Sectet. *MoDELS Workshops*, (pp. 132-144).

Hansen, F. & Oleshchuk, V. (2003). SRBAC: A Spatial Role-Based Access Control Model for Mobile Systems. In *Proceedings of the 7th Nordic Workshop on Secure IT Systems (NORDSEC'03)*. Gj`vik, Norway, 129-141.

Hao, H., Singh, V., & Du, W. (2013). "On the effectiveness of API-level access control using bytecode rewriting in Android", in *8th ACM SIGSAC symposium on Information, computer and communications security*, Hangzhou, China.

HAPI FHIR. (2014). *HAPI*. Retrieved from http://hapifhir.io/.

HAPI FHIR Server Interceptors. (2016). *Server Interceptors*. Retrieved from http://hapifhir.io/doc_rest_server_interceptor.html.

HAPI FHIR Server Security. (2016). *Server Security*. Retrieved from http://hapifhir.io/doc_rest_server_security.html.

Hazelwood, L. (2012, January). *The New RBAC: Resource-Based Access Control*. Retrieved from https://stormpath.com/blog/new-rbac- resource-based-access-control/.

Health and Human Services Department. (2015, March). *2015 Edition Health Information Technology Certification Criteria, 2015 Edition Base Electronic Health Record Definition, and ONC Health IT Certification Program Modifications.* Retrieved from https://www.gpo.gov/fdsys/pkg/FR-2015-03-30/pdf/2015-06612.pdf.

Health Level Seven International. (2011). Retrieved from http://www.hl7.org/implement/standards/product_brief.cfm?product_id=185.

Heisey-Grove, D., & Patel, V. (2015, September). *Any, certified, and basic: Quantifying Physician EHR adoption through 2014*. Retrieved from https://www.healthit.gov/sites/default/files/briefs/oncdatabrief28_certified_vs_basic.pdf

HIT Consultant. (2014, June 23). *The Evolving Landscape of Medical Apps in Healthcare*. Retrieved from http://hitconsultant.net/2014/06/23/the-evolving-landscape-of-medical-apps-in-healthcare/.

HHS.gov. (2013). *Health Information Privacy*. Retrieved from http://www.hhs.gov/hipaa/index.html.
Himss. (2016). *Meaningful use stage 3 final rule*. Retrieved from: http://www.himss.org/ResourceLibrary/genResourceDetailPDF.aspx?ItemNumber=44987.

HL7. (2013). *HL7 Confidentiality Definitions*. Retrieved from http://www.hl7.org/documentcenter/public_temp_F7525D5D-1C23-BA17-0C9A9B2F4EEFA395/standards/vocabulary/vocabulary_tables/infrastructure/vocabulary/Confidentiality.html

HL7 v3. (2013). *HL7 Version 3 - Value sets using code system: Confidentiality*. Retrieved from http://www.hl7.org/documentcenter/public_temp_5969D197-1C23-BA17-0C1ADD88E2E4CEBD/standards/vocabulary/vocabulary_tables/infrastructure/vocabulary/vs_Confidentiality.html

iOS 9 Health. (2014). *Health An innovative new way to use your health and fitness information*. Retrieved from https://www.apple.com/ios/health/.

Instagram. (2010). *Instagram*. Retrieved from https://www.instagram.com/.

Java. (2013). *Java Servlet Technology*. Retrieved from http://docs.oracle.com/javaee/6/tutorial/doc/bnafd.html.

Jin, X., Wang, L., Luo, T., & Du, W. (2015). "Fine-Grained Access Control for HTML5-Based Mobile Applications in Android," in *16th Information Security Conference (ISC)*.

JWT. (2015, April 23). *JSON Web Tokens*. Retrieved from https://jwt.io/.

Kelly, S. M. (2014, June 27). *In Google Fit vs. Apple HealthKit, Fitness Apps Stay Neutral*. Retrieved from http://mashable.com/2014/06/27/healthkit-google-fit-apps/#nf_r0U7flmqC.

Khan, M. F., & Sakamura, K. (2012). Toward a synergy among discretionary, role-based and context-aware access control models in healthcare information technology. *2012 World Congress on Internet Security (WorldCIS)*, (pp. 66-70).

Lamprinakos, G. C, Mousas, A. S., Kapsalis, A. P., Kaklamani, D. I., Venieris, I. S., Boufis, A. D., & et al. (2014). "Using FHIR to develop a healthcare mobile application", 132-135.

Larson, S. (2015, November 18). *Instagram restricts API following password breach, will review all apps going forward*. Retrieved from http://www.dailydot.com/debug/instagram-api-restrictions/.

Lella, A., Lipsman, A., & Martin, B. (2015, September 22). *The 2015 Mobile App Report*. Retrieved from https://www.comscore.com/Insights/Presentations-and-Whitepapers/2015/The-2015-US-Mobile-App-Report.

Liebrand, M., Ellis, H., Phillips, C., Demurjian, S., Ting, T.C., and Ellis, J. (2003, July). Role Delegation for a Resource-Based Security Model. In *Research Directions in Data and Applications Security*. E. Gudes and S. Shenoi (eds.), Vol. IFIP 128, Springer, pp. 37-48.

Mashery. (2015). *API Security*. Retrieved from http://www.mashery.com/api/security.

MedWatcher. (2012). *MedWatcher*. Retrieved from https://medwatcher.org/.

Microsoft Corporation. (2008). *Mobile Application Architecture Guide*. Retrieved from http://robtiffany.com/wp- content/uploads/2012/08/Mobile_Architecture_Guide_v1.1.pdf.

Morris, J. (2013, July 11). *Overview of Linux Kernel Security Features*. Retrieved from https://www.linux.com/learn/overview-linux-kernel-security-features.

My Imaging Records App. (2013). *My Imaging Records App*. Retrieved from http://myimagingrecords.com/index.html.

MTBC PHR. (2011). *MTBC PHR*. Retrieved from https://phr.mtbc.com/.

NIST Computer Security Resource Center. (2015). *Role Based Access Control - Frequently Asked Questions*.
Retrieved from http://csrc.nist.gov/groups/SNS/rbac/ faq.html.

openstack. (2012, July 21). *openstack Software*. Retrieved from https://www.openstack.org/software/.

Osborn, S., Sandhu, R., & Munawer, Q. (2000). Configuring role-based access control to enforce mandatory and discretionary access control policies. *ACM Transactions on Information and System Security*, *3*(2).

Peleg, M., Beimel, D., Dori, D., & Denekamp, Y. (2008). Situation-Based Access Control: Privacy management via modeling of patient data access scenarios. *Journal of Biomedical Informatics, 41*(6), 1028-1040.

PEW Research Center. (2015). *Mobile Technology Fact Sheet*. Retrieved from http://www.pewinternet.org/fact-sheets/mobile-technology-fact-sheet/.

Pollock, J. (2013, May 6). *New Feature: Access Control on APIs*. Retrieved from http://support.mashery.com/blog/read/New_Feature_Access_Control_on_APIs.

Ponemon Institute. (2009, April 6). *Ponemon Institute*. Retrieved from http://www.ponemon.org/.

Ponemon Institute. (2015, May). *2015 Cost of Data Breach Study: Global Analysis*. Retrieved from http://public.dhe.ibm.com/common/ssi/ecm/se/en/sew03053wwen/SEW03053WWEN.PDF?.

Postman. (2013). *Postman*. Retrieved from http://www.getpostman.com/.

Radicati, S. (2014). *Mobile Statistics Report 2014-2018*. Retrieved from http://www.radicati.com/wp/wp-content/uploads/2014/01/Mobile-Statistics-Report-2014-2018-Executive-Summary.pdf.

Rindfleisch, T.C. Privacy, Information Technology, and Health Care. (1997). *Communications of the ACM*, *40*(8), 93-100.

Rivera Sánchez, Y.K., Demurjian, S.A. (2016). Chapter 6: User Authentication Requirements for Mobile Computing. *Handbook of Research on Innovations in Access Control and Management*. IGI Global.

Rivera Sánchez, Y. K., Demurjian, S. A., Conover, J., Agresta, T., Shao, X., and Diamond, M. (2016). Chapter 6: An Approach for Role-Based Access Control in Mobile Applications. *Handbook of Mobile Application Development, Usability, and Security*. S. Mukherja (ed.). IGI Global.

Rivera Sánchez, Y. K., Demurjian, S.A., and Gnirke, L. (2017). An Intercepting API-based Access Control Approach for Mobile Applications. In proceedings of *The 13th International Conference on Web Information Systems and Technologies (WEBIST 2017)*.

Rivera Sánchez, Y. K., Demurjian, S.A., and Baihan, M. (2017a). Achieving RBAC on RESTful APIs for Mobile Apps using FHIR. In proceedings of *The 5th IEEE International Conference on Mobile Cloud Computing, Services, and Engineering (IEEE Mobile Cloud 2017)*.

Rivera Sánchez, Y.K., Demurjian, S. A., and Baihan, M. S. (2017b). A Service-Based RBAC & MAC Approach Incorporated into the Fast Healthcare Interoperable Resources (FHIR) standard, submitted to special issue on 2017IEEE Mobile Cloud Conference submissions, Elsevier journal of Digital Communications and Networks, https://www.journals.elsevier.com/digital-communications-and-networks/call-for-papers/special-issue-on-the-security-privacy-and-digital-forensics

Rohrer, F., Zhang, Y., Chitkushev, L., & Zlateva, T. (2013). "DR BACA: dynamic role based access control for Android," in *29th Annual Computer Security Applications Conference*, New Orleans, Louisiana, USA.

Rouse, M. (2014, December 2). *REST (representational state transfer)*. Retrieved from http://searchsoa.techtarget.com/definition/REST

Rouse, M. (2012, April 15). *Role-Based Access Control (RBAC)*. Retrieved from http://searchsecurity.techtarget.com/definition/role-based- access-control-RBAC.

Sandhu, R., Ferraiolo, D. F., & Kuhn, R. (2000). The NIST Model for Role Based Access Control: Toward a Unified Standard. In *Proceedings of the Fifth ACM Workshop on Role-based Access Control (RBAC '00)*, 47-63. Berlin, Germany.

Sandhu, R.S., & Samarati, P. (1994). Access Control: Principles and Practice. *Communications Magazine, IEEE*, *32*(9), 40–48.

Santos-Pereira, C., Augusto, A.B., Correia, M.E., Ferreira, A., & Cruz-Correia, R. (2012). A Mobile Based Authorization Mechanism for Patient Managed Role Based Access Control, *ITBAM 2012, LNCS*, *7451*, *54-68*. Springer, Heidelberg.

Schefer-Wenzl, S., & Strembeck, M. (2013). Modelling Context-Aware RBAC Models for Mobile Business Processes. *International Journal of Wireless and Mobile Computing (IJWMC)*, *6*(5), 448.

Shebaro, B., Oluwatimi, O., & Bertino, E. (2015). Context-Based Access Control Systems for Mobile Devices. *Fellow, IEEE*.

SlideShare. (2012, December 5). *Constrained RBAC diagram*. Retrieved from http://image.slidesharecdn.com/rbac6576-121205031439-phpapp01/95/rbac-18-638.jpg?cb=1354677352

Smith, A. (2015). *U.S. Smartphone Use in 2015*. Retrieved from http://www.pewinternet.org/2015/04/01/us-smartphone-use-in-2015/.

Snapchat. (2013, December 27). *Finding Friends with Phone Numbers*.
Retrieved from http://blog.snapchat.com/post/71353347590/finding- friends-with-phone-numbers

Snapchat. (2011). *Snapchat*. Retrieved from https://www.snapchat.com/

Stormpath. (2015). *Stormpath User Identity API*. Retrieved from https://stormpath.com/

Sujansky, W. V., Faus, S. A., Stone, E., Brennan, P. F. (2010). A method to implement fine-grained access control for personal health records through standard relational database queries. *Journal of Biomedical Informatics, 43*(5), 46-50.

Symantec. (2014, October 7). *Securing Mobile App Data - Comparing Containers and App Wrappers*. Retrieved from https://www.symantec.com/content/en/us/enterprise/white_papers/b-securing-mobile-app-data-comparing-containers-wp-21333969.pdf

Ventola C. L. (2014). Mobile Devices and Apps for Health Care Professionals: Uses and Benefits. *P&T*, *39*, 356-364.

W3C. (2007). *Latest SOAP versions*. Retrieved from https://www.w3.org/TR/soap/.

Walker, J., Pan, E., Johnston, D., Adler-Milstein, J., Bates, D. W., & Middleton, B. (2005). The Value of Health Care Information Exchange and Interoperability. *24*(2), 10-18. *Health Affairs*, 24(2), pp. 10-18.

Wang, Y., Hariharan, S., Zhao, C., Liu, J., & Du, W. (2014). "Compac: enforce component-level access control in android," in *4th ACM conference on Data and application security and privacy*, San Antonio, Texas, USA.

WebMD. (2016). *WebMD App*. Retrieved from http://www.webmd.com/mobile.

West, D. & Miller, E.A. (2009). Digital Medicine: Health Care in the Internet Era. *Brooking Institution Press*, 4.

WhatIsREST.com. (2012, October 26). *REST Constraints*. Retrieved from http://whatisrest.com/rest_constraints/index

Wiech, D. (2013, April 17). *Role-Based Access Control for Healthcare Data Security*. Retrieved from http://healthcare-executive-insight.advanceweb.com/Features/Articles/Role-based-Access-Control-for-Healthcare-Data-Security.aspx

Zeman, E. (2015, April 7). *Snapchat Lays Down The Law On Third-Party Apps*. Retrieved from http://www.programmableweb.com/news/snapchat-lays- down-law-third-party-apps/2015/04/07

Appendix

Appendix A – Sample Programmatic Changes for CT^2 (Android version)

RBAC API – getScreenAccessJSON method:

```
public static int getScreenAccessJSON(int roleId, int screenId) {
    String action = "/screenaccesses";
    String params = "/" + roleId + "/" + screenId;

    try {
        JSONArray json = ServerConnectionHelper.toJSONArray
         (ServerConnectionHelper.serverAsyncRequestGet
         (action + params, LOCALSERVER_URL_API4430));

        //Get screen permission based on the user’s role
        screenAccess = new Screens(json.getJSONObject(0));
    }

    catch (Throwable t) {
        t.printStackTrace();
    }

    return screenAccess.getAccess();
}
```

Screen Permissions:

```
public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.tabs_activity);
      setTitle(R.string.app_head_name);

      // resource object to get drawables
      Resources res = getResources();
      // the activity TabHost
      TabHost tabHost = getTabHost();
      // reusable TabSpec for each tab
      TabHost.TabSpec spec;
      // reusable intent for each tab
      Intent intent;

      Bundle bundle = getIntent().getExtras();
      int value = bundle.getInt("RoleID");
      userID=bundle.getInt("UserID");
      language=bundle.getInt("language");

      // retrieve extras from intent
      Bundle extras = getIntent().getExtras();
      stateID = extras.getString("StateID");
      cityID=extras.getString("CityID");
      districtID=extras.getString("DistrictID");
      schoolID=extras.getString("SchoolID");

      this.school_id=schoolID;

      // Search Tab
      intent = new Intent().setClass(this, SearchStudentsActivity.class);
      intent.putExtra("StateID", stateID);
      intent.putExtra("CityID", cityID);
      intent.putExtra("DistrictID", districtID);
      intent.putExtra("SchoolID", schoolID);
      intent.putExtra("UserID", userID);
      intent.putExtra("RoleID", value);
      intent.putExtra("language", language);
      spec =  tabHost.newTabSpec("home").setIndicator
               (getString(R.string.home),res.getDrawable
               (R.drawable.ic_tab_home)).setContent(intent);
      tabHost.addTab(spec);

      // List of Students Tab
      intent = new Intent().setClass(this, ListOfStudentsActivity.class);
      intent.putExtra("StateID", stateID);
      intent.putExtra("CityID", cityID);
      intent.putExtra("DistrictID", districtID);
      intent.putExtra("SchoolID", schoolID);
      intent.putExtra("UserID", userID);
      intent.putExtra("RoleID", value);
      intent.putExtra("language", language);

      spec = tabHost.newTabSpec("list").setIndicator
              (getString(R.string.list),res.getDrawable
              (R.drawable.ic_tab_list)).setContent(intent);
      tabHost.addTab(spec);

      // New Student Tab move
      intent = new Intent().setClass(this, HomeAddNewStudentActivity.class);
      intent.putExtra("StateID", stateID);
      intent.putExtra("CityID", cityID);
      intent.putExtra("DistrictID", districtID);
      intent.putExtra("SchoolID", schoolID);
      intent.putExtra("UserID", userID);
      intent.putExtra("RoleID", value);
      intent.putExtra("language", language);
      spec = tabHost.newTabSpec("addNewStudent").setIndicator
              (getString(R.string.student),res.getDrawable
              (R.drawable.ic_tab_student)).setContent(intent);
      tabHost.addTab(spec);

      // Cause of Injury Tab
      intent = new Intent().setClass(this, CauseOfInjuryActivity.class);
      intent.putExtra("StateID", stateID);
      intent.putExtra("CityID", cityID);
      intent.putExtra("DistrictID", districtID);
      intent.putExtra("SchoolID", schoolID);
      intent.putExtra("UserID", userID);
      intent.putExtra("RoleID", value);
      intent.putExtra("language", language);
      spec = tabHost.newTabSpec("cause").setIndicator
              (getString(R.string.cause), res.getDrawable
              (R.drawable.ic_tab_cause)).setContent(intent);
      tabHost.addTab(spec);

      // Immediate Symptoms Tab
      intent = new Intent().setClass(this, ImmediateSymptomsActivity.class);
      intent.putExtra("StateID", stateID);
      intent.putExtra("CityID", cityID);
      intent.putExtra("DistrictID", districtID);
      intent.putExtra("SchoolID", schoolID);
      intent.putExtra("UserID", userID);
      intent.putExtra("RoleID", value);
```

```
        intent.putExtra("language", language);
        spec = tabHost.newTabSpec("symptoms").setIndicator
                (getString(R.string.symptoms),res.getDrawable
                (R.drawable.ic_tab_symptoms)).setContent(intent);
        tabHost.addTab(spec);

        // Injury Follow-up Tab
        intent = new Intent().setClass(this, InjuryFollowUpActivity.class);
        intent.putExtra("StateID", stateID);
        intent.putExtra("CityID", cityID);
        intent.putExtra("DistrictID", districtID);
        intent.putExtra("SchoolID", schoolID);
        intent.putExtra("UserID", userID);
        intent.putExtra("RoleID", value);
        intent.putExtra("language", language);

         spec = tabHost.newTabSpec("follow_up").setIndicator
                (getString(R.string.injury_follow_up), res.getDrawable
                (R.drawable.ic_tab_follow_up)).setContent(intent);
         tabHost.addTab(spec);

          // Return to Learn Tab
          intent = new Intent().setClass(this,ReturnToLearnActivity.class);
          intent.putExtra("StateID", stateID);
          intent.putExtra("CityID", cityID);
          intent.putExtra("DistrictID", districtID);
          intent.putExtra("SchoolID", schoolID);
          intent.putExtra("UserID", userID);
           intent.putExtra("RoleID", value);
          intent.putExtra("language", language);
          spec = tabHost.newTabSpec("return").setIndicator
                  (getString(R.string.return_to_learn), res.getDrawable
                  (R.drawable.ic_tab_return)).setContent(intent);
          tabHost.addTab(spec);

          // set the current tab (default Home)
          tabHost.setCurrentTab(0);

          for(int i = 1; i <= 7; i++) {
              if (ServerConnection.getScreenAccessJSON(value, i) == 0)
              {
                tabHost.getTabWidget().getChildAt(i).setVisibility(View.GONE);
              }
          }
}
```

Appendix B – Intercepting API

Generate Code for the Intercepting API

```
<?php
function echoInterceptingClassStart($write_file){
$start_string = "<?php
require_once \"renamedConcussionUConn.php\";
require_once \"JWT/APIJWT.php\";\n\n\n

class ConcussionUConn {
       public function __construct(){
           session_start();

           \$this->dbServerName = \"localhost\";
           \$this->dbUser = \"root\";
           \$this->dbPassword = \"--------\";
           \$this->dbName = \"concussion_uconn\";
       }";

fwrite($write_file, $start_string);
}

function echoAPIPermissionCheckFunctions($write_file){
       $permission_check_string = "
       private function verifyAPIPermissions(\$function_name) {
              \$jwt = \$_SESSION['jwt'];
              \$JWT = new API_JWT();

              \$user_role = \$JWT->getRole(\$jwt);
              if(\$user_role == NULL){
                     return 0;
              }

              \$action_id = \$this->getActionID(\$function_name);
              \$role_permissions_query = 'SELECT * FROM json_calls_map_access WHERE
              action_id='.\$action_id;
              \$role_permissions = \$this->performQuery
              (\$role_permissions_query);

              foreach(\$role_permissions as \$array){
                     if(\$array['role_id'] == \$user_role){
                            return \$array['enable_disable'];
                     }
              }
       }

       private function getParamNames(\$functionName){
              \$paramString = \"\";
              \$numCommas = 0;
              \$reflectionMeth = new ReflectionMethod('ConcussionUConn',
              \$functionName);
              \$numParam = \$reflectionMeth->getNumberOfParameters();

              foreach(\$reflectionMeth->getParameters() as \$parameter){
                     \$paramString = \$paramString.'\$'.\$parameter->getName();
                     if(\$parameter->isOptional()){
                            \$defaultValue = \$parameter->getDefaultValue();
                            if(\$defaultValue == ''){
                                   \$defaultValue = \"''\";
                            }
                            \$paramString = \$paramString.\" = \".\$defaultValue;
```

```
                    }
                    if(\$numCommas < (\$numParam - 1)){
                          \$paramString = \$paramString.\", \";
                          \$numCommas++;
                    }
              }

              return \$paramString;
       }

       private function getActionID(\$function_name){
              \$action_id = NULL;
              \$function_name = \$function_name.\"(\".\$this->
              getParamNames(\$function_name).\")\";

              \$query = \"SELECT * FROM json_calls WHERE action='\".\$function_name.\"'
              ORDER BY action_id\";
              \$result = \$this->performQuery(\$query);
              foreach(\$result as \$array){
                    if(\$array['action'] == \$function_name){
                          \$action_id = \$array['action_id'];
                    }
              }
              return \$action_id;
       }";
       fwrite($write_file, $permission_check_string);
}

function echoPrivateHelperFunctions($write_file){
       $helper_string = "
       private function performQuery(\$query){
                  \$result = NULL;
                  \$mysqlConnection = \$this->createMySqlConnection();
                  \$result = \$mysqlConnection->query(\$query);

                  if (!\$result) {
                    throw new Exception(\"Database Error [{\$this->database->
                      errno}] {\$this->database->error}\");
                  } else {
                    \$array = array();
                    while(\$row = \$result->fetch_assoc()) \$array[] = \$row;
                  }

                  \$mysqlConnection->close();
                  return \$array;
         }

       private function createMySqlConnection(){
              \$conn = new mysqli(\$this->dbServerName, \$this->dbUser,
              \$this->dbPassword, \$this->dbName);
                  if (\$conn->connect_error) {
                    die(\"Connection failed: \" . \$conn->connect_error);
                  }
                  else{
                    return \$conn;
                  }
      }

       private function setJWT(\$username, \$hashed_password){
              \$query = \"SELECT * FROM user_accounts WHERE username = \" . \$username
              . \" AND hashed_password = \" . \$hashed_password;
              \$result = \$this->performQuery(\$query);
```

```
            foreach(\$result as \$array){
                  \$role_id = \$array['role_id'];
                  \$user_id = \$array['user_id'];
            }

            \$header = array('typ' => 'JWT', 'alg' => 'HS256');
            \$payload = array('user_id' => \$user_id, 'role_id' => \$role_id);

            \$JWT = new API_JWT();
            \$jwt = \$JWT->create(\$header, \$payload);

            return \$jwt;
      }";
      fwrite($write_file, $helper_string);
}

function echoInterceptingClassEnd($write_file){
      $end_string = "}?>";
      fwrite($write_file, $end_string);
}

function echoInterceptBody($functionName, $need_permission, $setJWT, $write_file){
      $wrapper_string = "public function ".$functionName."{";
      if($setJWT){
            $wrapper_string = $wrapper_string."\n\t\t\$_SESSION['jwt'] = \$this-
            >setJWT(\$username, \$hashed_password);";
      }
      if($need_permission){
            $wrapper_string = $wrapper_string."
            \$permission = \$this->verifyAPIPermissions(__FUNCTION__);
            if(\$permission == 1){";
      }

      $wrapper_string = $wrapper_string."
                  \$renamedConcussionUConn = new renamedConcussionUConn();
                  return \$renamedConcussionUConn->
                  RENAMED".$functionName.";";
      if($need_permission){
            $wrapper_string = $wrapper_string."
            }else{
                  return NULL;
            }";
      }
      $wrapper_string = $wrapper_string."
      }
      ";
      fwrite($write_file, $wrapper_string);
}

function getFunctions(){
      $functionList = array();
      $query = "SELECT * FROM json_calls";
      $mysqlConnection = initMySqlConnection();
      $result = getResultsFromQuery($query);

      foreach($result as $array){
            $functionName = $array['action'];
            $need_permission = $array['need_permission'];
            $setJWT = $array['setJWT'];
            $functionList[] = $need_permission.$setJWT.$functionName;
      }

      $mysqlConnection->close();
```

```
        return $functionList;
}

function getResultsFromQuery($query){
         $result = NULL;
         $mysqlConnection = initMySqlConnection();
         $result = $mysqlConnection->query($query);

         if (!$result) {
              throw new Exception("Database Error [{$this->database->errno}] {$this-
              >database->error}");
         } else {
           $array = array();
           while($row = $result->fetch_assoc()) $array[] = $row;
         }

         $mysqlConnection->close();
         return $array;
}

function initMySqlConnection(){
         $dbServerName = "localhost";
         $dbUser = "root";
         $dbPassword = "--------";
         $dbName = "concussion_uconn";

         $conn = new mysqli($dbServerName, $dbUser, $dbPassword, $dbName);

         if ($conn->connect_error) {
           die("Connection failed: " . $conn->connect_error);
         }
         else
         {return $conn;}
}

function createInterceptingAPI(){
       $write_file = fopen("../vTest/concussionUConn.php", "w");

       $functionList = getFunctions();
       $numFunctions = count($functionList);

       echoInterceptingClassStart($write_file);
       echoAPIPermissionCheckFunctions($write_file);

       for($i = 0; $i < $numFunctions; $i++){
             $functionName = substr($functionList[$i], 2);
             $need_permission = $functionList[$i][0];
             $setJWT = $functionList[$i][1];
             echoInterceptBody($functionName, $need_permission, $setJWT, $write_file);
       }

       echoPrivateHelperFunctions($write_file);
       echoInterceptingClassEnd($write_file);

       fclose($write_file);
}
createInterceptingAPI();?>
```

Generate Code for the Intercepting API – Output

```
<?php
require_once "renamedConcussionUConn.php";
require_once "JWT/APIJWT.php";
```

```
class ConcussionUConn {
      public function __construct(){
          session_start();
          $this->dbServerName = "localhost";
          $this->dbUser = "root";
          $this->dbPassword = "--------";
          $this->dbName = "concussion_uconn";
      }
      private function verifyAPIPermissions($function_name) {
             $jwt = $_SESSION['jwt'];
             $JWT = new API_JWT();
             $user_role = $JWT->getRole($jwt);
             if($user_role == NULL){
                    return 0;
             }
             $action_id = $this->getActionID($function_name);
             $role_permissions_query = 'SELECT * FROM json_calls_map_access WHERE
             action_id='.$action_id;
             $role_permissions = $this->performQuery($role_permissions_query);
             foreach($role_permissions as $array){
                    if($array['role_id'] == $user_role {
                           return $array['enable_disable'];
                    }
             }
      }
      private function getParamNames($functionName){
             $paramString = "";
             $numCommas = 0;
             $reflectionMeth = new ReflectionMethod('ConcussionUConn', $functionName);
             $numParam = $reflectionMeth->getNumberOfParameters();

             foreach($reflectionMeth->getParameters() as $parameter){
                    $paramString = $paramString.'$'.$parameter->getName();
                    if($parameter->isOptional()){
                           $defaultValue = $parameter->getDefaultValue();
                           if($defaultValue == ''){
                                  $defaultValue = "''";
                           }
                           $paramString = $paramString." = ".$defaultValue;
                    }
                    if($numCommas < ($numParam - 1)){
                           $paramString = $paramString.", ";
                           $numCommas++;
                    }
             }
             return $paramString;
      }
      private function getActionID($function_name){
             $action_id = NULL;
             $function_name = $function_name."(".$this->getParamNames
             ($function_name).")";
             $query = "SELECT * FROM json_calls WHERE action=
             '".$function_name."' ORDER BY action_id";
             $result = $this->performQuery($query);
             foreach($result as $array){
                    if($array['action'] == $function_name){
                           $action_id = $array['action_id'];
                    }
             }
             return $action_id;
      }

      public function getListOfScreenObjects(){
```

```
		$permission = $this->verifyAPIPermissions(__FUNCTION__);
		if($permission == 1){
			$renamedConcussionUConn = new renamedConcussionUConn();
			return $renamedConcussionUConn->
			RENAMEDgetListOfScreenObjects();
		}else{
			return NULL;
		}
	}

	public function getListOfScreens(){
		$permission = $this->verifyAPIPermissions(__FUNCTION__);
		if($permission == 1){
			$renamedConcussionUConn = new renamedConcussionUConn();
			return $renamedConcussionUConn->RENAMEDgetListOfScreens();
		}else{
			return NULL;
		}
	}

	public function getListofScreenSequences(){
		$permission = $this->verifyAPIPermissions(__FUNCTION__);
		if($permission == 1){
			$renamedConcussionUConn = new renamedConcussionUConn();
			return $renamedConcussionUConn->
			RENAMEDgetListofScreenSequences();
		}else{
			return NULL;
		}
	}

	public function getScreenObjectByScreenID($screen_id){
		$permission = $this->verifyAPIPermissions(__FUNCTION__);
		if($permission == 1){
			$renamedConcussionUConn = new renamedConcussionUConn();
			return $renamedConcussionUConn->
			RENAMEDgetScreenObjectByScreenID($screen_id);
		}else{
			return NULL;
		}
	}

	public function getScreenObjectByName($object_name){
		$permission = $this->verifyAPIPermissions(__FUNCTION__);
		if($permission == 1){
			$renamedConcussionUConn = new renamedConcussionUConn();
			return $renamedConcussionUConn->
			RENAMEDgetScreenObjectByName($object_name);
		}else{
			return NULL;
		}
	}

	public function getScreenObjectByObjectID($object_id){
		$permission = $this->verifyAPIPermissions(__FUNCTION__);
		if($permission == 1){
			$renamedConcussionUConn = new renamedConcussionUConn();
			return $renamedConcussionUConn->
			RENAMEDgetScreenObjectByObjectID($object_id);
		}else{
			return NULL;
		}
	}
```

```
public function getObjectsLabelsByScreenID($screen_id){
       $permission = $this->verifyAPIPermissions(__FUNCTION__);
       if($permission == 1){
             $renamedConcussionUConn = new renamedConcussionUConn();
             return $renamedConcussionUConn->
             RENAMEDgetObjectsLabelsByScreenID($screen_id);
       }else{
             return NULL;
       }
}

public function getScreenByID($screen_id){
       $permission = $this->verifyAPIPermissions(__FUNCTION__);
       if($permission == 1){
             $renamedConcussionUConn = new renamedConcussionUConn();
             return $renamedConcussionUConn->
             RENAMEDgetScreenByID($screen_id);
       }else{
             return NULL;
       }
}

public function getScreenByName($screen_name){
       $permission = $this->verifyAPIPermissions(__FUNCTION__);
       if($permission == 1){
             $renamedConcussionUConn = new renamedConcussionUConn();
             return $renamedConcussionUConn->
             RENAMEDgetScreenByName($screen_name);
       }else{
             return NULL;
       }
}

public function getScreenSequenceByRoleID($role_id){
       $permission = $this->verifyAPIPermissions(__FUNCTION__);
       if($permission == 1){
             $renamedConcussionUConn = new renamedConcussionUConn();
             return $renamedConcussionUConn->
             RENAMEDgetScreenSequenceByRoleID($role_id);
       }else{
             return NULL;
       }
}

public function getScreenSequenceByObjectID($object_id){
       $permission = $this->verifyAPIPermissions(__FUNCTION__);
       if($permission == 1){
             $renamedConcussionUConn = new renamedConcussionUConn();
             return $renamedConcussionUConn->
             RENAMEDgetScreenSequenceByObjectID($object_id);
       }else{
             return NULL;
       }
}

public function getScreenSequenceByScreenID($screen_id){
       $permission = $this->verifyAPIPermissions(__FUNCTION__);
       if($permission == 1){
             $renamedConcussionUConn = new renamedConcussionUConn();
             return $renamedConcussionUConn->
             RENAMEDgetScreenSequenceByScreenID($screen_id);
       }else{
```

```
                return NULL;
        }
}

public function getScreenAccessByRole($role_id, $screen_id){
        $permission = $this->verifyAPIPermissions(__FUNCTION__);
        if($permission == 1){
                $renamedConcussionUConn = new renamedConcussionUConn();
                return $renamedConcussionUConn->
                RENAMEDgetScreenAccessByRole($role_id, $screen_id);
        }else{
                return NULL;
        }
}

public function returnAllowableComponents($role_id,$screen_id){
        $permission = $this->verifyAPIPermissions(__FUNCTION__);
        if($permission == 1){
                $renamedConcussionUConn = new renamedConcussionUConn();
                return $renamedConcussionUConn->
                RENAMEDreturnAllowableComponents($role_id,$screen_id);
        }else{
                return NULL;
        }
}

public function getNamesOfScreen($screen_id){
        $permission = $this->verifyAPIPermissions(__FUNCTION__);
        if($permission == 1){
                $renamedConcussionUConn = new renamedConcussionUConn();
                return $renamedConcussionUConn->
                RENAMEDgetNamesOfScreen($screen_id);
        }else{
                return NULL;
        }
}

public function getLabelsOfScreen($screen_id,$lang){
        $permission = $this->verifyAPIPermissions(__FUNCTION__);
        if($permission == 1){
                $renamedConcussionUConn = new renamedConcussionUConn();
                return $renamedConcussionUConn->
                RENAMEDgetLabelsOfScreen($screen_id,$lang);
        }else{
                return NULL;
        }
}

public function getUserAccountByLogin($username, $hashed_password){
        $_SESSION['jwt'] = $this->setJWT($username, $hashed_password);
        $permission = $this->verifyAPIPermissions(__FUNCTION__);
        if($permission == 1){
                $renamedConcussionUConn = new renamedConcussionUConn();
                return $renamedConcussionUConn->
                RENAMEDgetUserAccountByLogin($username, $hashed_password);
        }else{
                return NULL;
        }
}

public function getListOfStates(){
                $renamedConcussionUConn = new renamedConcussionUConn();
                return $renamedConcussionUConn->RENAMEDgetListOfStates();
```

```
}

public function getListOfRegions($stateId){
       $permission = $this->verifyAPIPermissions(__FUNCTION__);
       if($permission == 1){
              $renamedConcussionUConn = new renamedConcussionUConn();
              return $renamedConcussionUConn->
              RENAMEDgetListOfRegions($stateId);
       }else{
              return NULL;
       }
}

public function getListOfDistricts($regionId){
       $permission = $this->verifyAPIPermissions(__FUNCTION__);
       if($permission == 1){
              $renamedConcussionUConn = new renamedConcussionUConn();
              return $renamedConcussionUConn->
              RENAMEDgetListOfDistricts($regionId);
       }else{
              return NULL;
       }
}

public function getListOfSchools($districtId){
       $permission = $this->verifyAPIPermissions(__FUNCTION__);
       if($permission == 1){
              $renamedConcussionUConn = new renamedConcussionUConn();
              return $renamedConcussionUConn->
              RENAMEDgetListOfSchools($districtId);
       }else{
              return NULL;
       }
}

public function getSchoolDetails($schoolId){
       $permission = $this->verifyAPIPermissions(__FUNCTION__);
       if($permission == 1){
              $renamedConcussionUConn = new renamedConcussionUConn();
              return $renamedConcussionUConn->
              RENAMEDgetSchoolDetails($schoolId);
       }else{
              return NULL;
       }
}

public function getDistrictAndRegionBySchool($schoolId){
       $permission = $this->verifyAPIPermissions(__FUNCTION__);
       if($permission == 1){
              $renamedConcussionUConn = new renamedConcussionUConn();
              return $renamedConcussionUConn->
              RENAMEDgetDistrictAndRegionBySchool($schoolId);
       }else{
              return NULL;
       }
}

public function getStateByRegion($regionId){
       $permission = $this->verifyAPIPermissions(__FUNCTION__);
       if($permission == 1){
              $renamedConcussionUConn = new renamedConcussionUConn();
              return $renamedConcussionUConn->
              RENAMEDgetStateByRegion($regionId);
```

```
        }else{
            return NULL;
        }
    }

    public function addEmployee($employeeObject){
        $permission = $this->verifyAPIPermissions(__FUNCTION__);
        if($permission == 1){
            $renamedConcussionUConn = new renamedConcussionUConn();
            return $renamedConcussionUConn->
            RENAMEDaddEmployee($employeeObject);
        }else{
            return NULL;
        }
    }

    public function getEmployee($userId){
        $permission = $this->verifyAPIPermissions(__FUNCTION__);
        if($permission == 1){
            $renamedConcussionUConn = new renamedConcussionUConn();
            return $renamedConcussionUConn->
            RENAMEDgetEmployee($userId);
        }else{
            return NULL;
        }
    }

    public function getEmployees(){
        $permission = $this->verifyAPIPermissions(__FUNCTION__);
        if($permission == 1){
            $renamedConcussionUConn = new renamedConcussionUConn();
            return $renamedConcussionUConn->RENAMEDgetEmployees();
        }else{
            return NULL;
        }
    }

    public function getUserRoleSchoolId($userId){
        $permission = $this->verifyAPIPermissions(__FUNCTION__);
        if($permission == 1){
            $renamedConcussionUConn = new renamedConcussionUConn();
            return $renamedConcussionUConn->
            RENAMEDgetUserRoleSchoolId($userId);
        }else{
            return NULL;
        }
    }

    public function getUsersRoleSchoolSport(){
        $permission = $this->verifyAPIPermissions(__FUNCTION__);
        if($permission == 1){
            $renamedConcussionUConn = new renamedConcussionUConn();
            return $renamedConcussionUConn->
            RENAMEDgetUsersRoleSchoolSport();
        }else{
            return NULL;
        }
    }

    public function addUserRoleSchool($userRoleSchool){
        $permission = $this->verifyAPIPermissions(__FUNCTION__);
        if($permission == 1){
            $renamedConcussionUConn = new renamedConcussionUConn();
            return $renamedConcussionUConn->
            RENAMEDaddUserRoleSchool($userRoleSchool);
        }else{
            return NULL;
        }
    }

    public function addUserAccount($userAccountObject){
        $permission = $this->verifyAPIPermissions(__FUNCTION__);
        if($permission == 1){
            $renamedConcussionUConn = new renamedConcussionUConn();
            return $renamedConcussionUConn->
            RENAMEDaddUserAccount($userAccountObject);
        }else{
            return NULL;
        }
    }

    public function getUserAccount($userId){
        $permission = $this->verifyAPIPermissions(__FUNCTION__);
        if($permission == 1){
            $renamedConcussionUConn = new renamedConcussionUConn();
            return $renamedConcussionUConn->
            RENAMEDgetUserAccount($userId);
        }else{
            return NULL;
        }
    }

    public function getUserAccounts(){
        $permission = $this->verifyAPIPermissions(__FUNCTION__);
        if($permission == 1){
            $renamedConcussionUConn = new renamedConcussionUConn();
            return $renamedConcussionUConn->RENAMEDgetUserAccounts();
        }else{
            return NULL;
        }
    }

    public function getUsers(){
            $renamedConcussionUConn = new renamedConcussionUConn();
            return $renamedConcussionUConn->RENAMEDgetUsers();
    }

    public function getUserById($userId){
        $permission = $this->verifyAPIPermissions(__FUNCTION__);
        if($permission == 1){
            $renamedConcussionUConn = new renamedConcussionUConn();
            return $renamedConcussionUConn->
            RENAMEDgetUserById($userId);
        }else{
            return NULL;
        }
    }

    public function getSchoolStudents($schoolId){
        $permission = $this->verifyAPIPermissions(__FUNCTION__);
        if($permission == 1){
            $renamedConcussionUConn = new renamedConcussionUConn();
            return $renamedConcussionUConn->
            RENAMEDgetSchoolStudents($schoolId);
        }else{
            return NULL;
```

```
		}
	}

	public function getStudentByID($studentId){
		$permission = $this->verifyAPIPermissions(__FUNCTION__);
		if($permission == 1){
			$renamedConcussionUConn = new renamedConcussionUConn();
			return $renamedConcussionUConn->
			RENAMEDgetStudentByID($studentId);
		}else{
			return NULL;
		}
	}

	public function searchForStudents($firstName = '', $lastName = ''){
		$permission = $this->verifyAPIPermissions(__FUNCTION__);
		if($permission == 1){
			$renamedConcussionUConn = new renamedConcussionUConn();
			return $renamedConcussionUConn->
			RENAMEDsearchForStudents($firstName = '', $lastName = '');
		}else{
			return NULL;
		}
	}

	public function getStudentGuardians($studentId){
		$permission = $this->verifyAPIPermissions(__FUNCTION__);
		if($permission == 1){
			$renamedConcussionUConn = new renamedConcussionUConn();
			return $renamedConcussionUConn->
			RENAMEDgetStudentGuardians($studentId);
		}else{
			return NULL;
		}
	}

	public function addStudent($studentObject){
		$permission = $this->verifyAPIPermissions(__FUNCTION__);
		if($permission == 1){
			$renamedConcussionUConn = new renamedConcussionUConn();
			return $renamedConcussionUConn->
			RENAMEDaddStudent($studentObject);
		}else{
			return NULL;
		}
	}

	public function updateStudent($studentObject, $studentId){
		$permission = $this->verifyAPIPermissions(__FUNCTION__);
		if($permission == 1){
			$renamedConcussionUConn = new renamedConcussionUConn();
			return $renamedConcussionUConn->
			RENAMEDupdateStudent($studentObject, $studentId);
		}else{
			return NULL;
		}
	}

	public function addStudentGuardian($studentGuardian){
		$permission = $this->verifyAPIPermissions(__FUNCTION__);
		if($permission == 1){
			$renamedConcussionUConn = new renamedConcussionUConn();
			return $renamedConcussionUConn->
			RENAMEDaddStudentGuardian($studentGuardian);
		}else{
			return NULL;
		}
	}

	public function updateStudentGuardian($studentGuardian, $guardianId){
		$permission = $this->verifyAPIPermissions(__FUNCTION__);
		if($permission == 1){
			$renamedConcussionUConn = new renamedConcussionUConn();
			return $renamedConcussionUConn->
			RENAMEDupdateStudentGuardian($studentGuardian, $guardianId);
		}else{
			return NULL;
		}
	}

	public function getConcussion($concussionId){
		$permission = $this->verifyAPIPermissions(__FUNCTION__);
		if($permission == 1){
			$renamedConcussionUConn = new renamedConcussionUConn();
			return $renamedConcussionUConn->
			RENAMEDgetConcussion($concussionId);
		}else{
			return NULL;
		}
	}

	public function getConcussionsByUserID($concussionId){
		$permission = $this->verifyAPIPermissions(__FUNCTION__);
		if($permission == 1){
			$renamedConcussionUConn = new renamedConcussionUConn();
			return $renamedConcussionUConn->
			RENAMEDgetConcussionsByUserID($concussionId);
		}else{
			return NULL;
		}
	}

	public function getConcussionFollowups($concussionId){
		$permission = $this->verifyAPIPermissions(__FUNCTION__);
		if($permission == 1){
			$renamedConcussionUConn = new renamedConcussionUConn();
			return $renamedConcussionUConn->
			RENAMEDgetConcussionFollowups($concussionId);
		}else{
			return NULL;
		}
	}

	public function getSymptomsWithRecord($recordId){
		$permission = $this->verifyAPIPermissions(__FUNCTION__);
		if($permission == 1){
			$renamedConcussionUConn = new renamedConcussionUConn();
			return $renamedConcussionUConn->
			RENAMEDgetSymptomsWithRecord($recordId);
		}else{
			return NULL;
		}
	}

	public function getSchoolConcussions($schoolId){
		$permission = $this->verifyAPIPermissions(__FUNCTION__);
```

```
        if($permission == 1){
            $renamedConcussionUConn = new renamedConcussionUConn();
            return $renamedConcussionUConn->
            RENAMEDgetSchoolConcussions($schoolId);
        }else{
            return NULL;
        }
    }

    public function getStudentConcussions($studentId){
        $permission = $this->verifyAPIPermissions(__FUNCTION__);
        if($permission == 1){
            $renamedConcussionUConn = new renamedConcussionUConn();
            return $renamedConcussionUConn->
            RENAMEDgetStudentConcussions($studentId);
        }else{
            return NULL;
        }
    }

    public function getUserConcussionsByID($userId){
        $permission = $this->verifyAPIPermissions(__FUNCTION__);
        if($permission == 1){
            $renamedConcussionUConn = new renamedConcussionUConn();
            return $renamedConcussionUConn->
            RENAMEDgetUserConcussionsByID($userId);
        }else{
            return NULL;
        }
    }

    public function getIncidentOperationHistory($incidentId){
        $permission = $this->verifyAPIPermissions(__FUNCTION__);
        if($permission == 1){
            $renamedConcussionUConn = new renamedConcussionUConn();
            return $renamedConcussionUConn->
            RENAMEDgetIncidentOperationHistory($incidentId);
        }else{
            return NULL;
        }
    }

    public function getFollowUpOperationHistory($followupId){
        $permission = $this->verifyAPIPermissions(__FUNCTION__);
        if($permission == 1){
            $renamedConcussionUConn = new renamedConcussionUConn();
            return $renamedConcussionUConn->
            RENAMEDgetFollowUpOperationHistory($followupId);
        }else{
            return NULL;
        }
    }

    public function changeIncidentStatus($incidentId, $status){
        $permission = $this->verifyAPIPermissions(__FUNCTION__);
        if($permission == 1){
            $renamedConcussionUConn = new renamedConcussionUConn();
            return $renamedConcussionUConn->
            RENAMEDchangeIncidentStatus($incidentId, $status);
        }else{
            return NULL;
        }
    }
```

```
public function addConcussionEvent($concussionEvent){
       $permission = $this->verifyAPIPermissions(__FUNCTION__);
       if($permission == 1){
             $renamedConcussionUConn = new renamedConcussionUConn();
             return $renamedConcussionUConn->
             RENAMEDaddConcussionEvent($concussionEvent);
       }else{
             return NULL;
       }
}

public function updateConcussionEvent($concussionEvent, $incidentId){
       $permission = $this->verifyAPIPermissions(__FUNCTION__);
       if($permission == 1){
             $renamedConcussionUConn = new renamedConcussionUConn();
             return $renamedConcussionUConn->
             RENAMEDupdateConcussionEvent($concussionEvent, $incidentId);
       }else{
             return NULL;
       }
}

public function addConcussionEventFollowup($concussionEventUpdate,
$incidentId){
       $permission = $this->verifyAPIPermissions(__FUNCTION__);
       if($permission == 1){
             $renamedConcussionUConn = new renamedConcussionUConn();
             return $renamedConcussionUConn->
             RENAMEDaddConcussionEventFollowup($concussionEventUpdate,
             $incidentId);
       }else{
             return NULL;
       }
}

public function updateConcussionEventFollowup($concussionEventUpdate,
$followUpId, $lingeringSymptomsRecordId){
       $permission = $this->verifyAPIPermissions(__FUNCTION__);
       if($permission == 1){
             $renamedConcussionUConn = new renamedConcussionUConn();
             return $renamedConcussionUConn->
             RENAMEDupdateConcussionEventFollowup($concussionEventUpdate,
             $followUpId, $lingeringSymptomsRecordId);
       }else{
             return NULL;
       }
}

public function getEventSymptoms($referenceId){
       $permission = $this->verifyAPIPermissions(__FUNCTION__);
       if($permission == 1){
             $renamedConcussionUConn = new renamedConcussionUConn();
             return $renamedConcussionUConn->
             RENAMEDgetEventSymptoms($referenceId);
       }else{
             return NULL;
       }
}

public function getAssessmentTools(){
       $renamedConcussionUConn = new renamedConcussionUConn();
       return $renamedConcussionUConn->
```

```
		RENAMEDgetAssessmentTools();
	}

	public function getEventLocations(){
		$renamedConcussionUConn = new renamedConcussionUConn();
		return $renamedConcussionUConn->RENAMEDgetEventLocations();
	}

	public function getContactMechanisms(){
		$renamedConcussionUConn = new renamedConcussionUConn();
		return $renamedConcussionUConn->
		RENAMEDgetContactMechanisms();
	}

	public function getImpactHeadLocations(){
		$renamedConcussionUConn = new renamedConcussionUConn();
		return $renamedConcussionUConn->
		RENAMEDgetImpactHeadLocations();
	}

	public function getSports(){
		$renamedConcussionUConn = new renamedConcussionUConn();
		return $renamedConcussionUConn->RENAMEDgetSports();
	}

	public function getSymptoms($type = 2){
		$renamedConcussionUConn = new renamedConcussionUConn();
		return $renamedConcussionUConn->RENAMEDgetSymptoms($type = 2);
	}

	public function getRoles(){
		$permission = $this->verifyAPIPermissions(__FUNCTION__);
		if($permission == 1){
			$renamedConcussionUConn = new renamedConcussionUConn();
			return $renamedConcussionUConn->RENAMEDgetRoles();
		}else{
			return NULL;
		}
	}

	public function getMedicalImaging(){
		$renamedConcussionUConn = new renamedConcussionUConn();
		return $renamedConcussionUConn->RENAMEDgetMedicalImaging();
	}

	public function getDiagnosingRoles(){
			$renamedConcussionUConn = new renamedConcussionUConn();
			return $renamedConcussionUConn->
			RENAMEDgetDiagnosingRoles();
	}

	public function importSchoolDistrict($districtName, $ctrName){
		$permission = $this->verifyAPIPermissions(__FUNCTION__);
		if($permission == 1){
			$renamedConcussionUConn = new renamedConcussionUConn();
			return $renamedConcussionUConn->
			RENAMEDimportSchoolDistrict($districtName, $ctrName);
		}else{
			return NULL;
		}
	}

	public function importSchool($districtName, $schoolName){
			$permission = $this->verifyAPIPermissions(__FUNCTION__);
			if($permission == 1){
				$renamedConcussionUConn = new renamedConcussionUConn();
				return $renamedConcussionUConn->
				RENAMEDimportSchool($districtName, $schoolName);
			}else{
				return NULL;
			}
		}

		private function performQuery($query){
				$result = NULL;
				$mysqlConnection = $this->createMySqlConnection();
				$result = $mysqlConnection->query($query);

				if (!$result) {
				  throw new Exception("Database Error [{$this->database->
				    errno}] {$this->database->error}");
				} else {
				  $array = array();
				  while($row = $result->fetch_assoc()) $array[] = $row;
				}

				$mysqlConnection->close();
				return $array;
		  }

		private function createMySqlConnection(){
				  $conn = new mysqli($this->dbServerName, $this->dbUser, $this-
				  >dbPassword, $this->dbName);
				if ($conn->connect_error) {
				  die("Connection failed: " . $conn->connect_error);
				}
				else{
				  return $conn;
				}
		  }

		private function setJWT($username, $hashed_password){
			$query = "SELECT * FROM user_accounts WHERE username = " . $username . "
			AND hashed_password = " . $hashed_password;
			$result = $this->performQuery($query);

			foreach($result as $array){
				$role_id = $array['role_id'];
				$user_id = $array['user_id'];
			}

			$header = array('typ' => 'JWT', 'alg' => 'HS256');
			$payload = array('user_id' => $user_id, 'role_id' => $role_id);
			$JWT = new API_JWT();
			$jwt = $JWT->create($header, $payload);

			return $jwt;
		}
}
?>
```

Renaming API Code

```
<?php
function renameAPI(){
		$concussionUConnFile = fopen("../concussionUConn.php", "r");
```

```
        $write_file = fopen("../vTest/RENAMEDConcussionUConn.php", "w");
        $functionList = array();
        $renameFunction = 0;
        $renameClass = 0;

        while(!feof($concussionUConnFile)){
                $line = fgets($concussionUConnFile);
                $lineArray = explode(' ', $line);
                $write_string = "";

                for($i = 0; $i < count($lineArray); $i++){
                        if($renameClass == 1){
                                $write_string = $write_string."RENAMED".$lineArray[$i];
                                $renameClass = -1;
                        }else if($renameFunction == 1){
                                $write_string = $write_string."RENAMED".$lineArray[$i];
                                $renameFunction = 0;
                        }else{
                                $write_string = $write_string.$lineArray[$i]." ";
                        }
                        if($lineArray[$i] == "function" && $i < count($lineArray)-1 && $i
> 0){
                                if(substr($lineArray[$i+1], 0, 2)!="__" &&
                                $lineArray[$i-1] == "public"){
                                        $renameFunction = 1;
                                }
                        }
                        if($lineArray[$i] == "class" && $renameClass == 0){
                                $renameClass = 1;
                        }
                }
                fwrite($write_file, $write_string);
        }
        fclose($concussionUConnFile);
        fclose($write_file);
}
renameAPI();?>
```

Renaming API Code - Output

```
<?php
class RENAMEDConcussionUConn
{
   public function __construct(){
       $this->dbServerName = "localhost";
       $this->dbUser = "root";
       $this->dbPassword = "--------";
       $this->dbName = "concussion_uconn";
   }

   public function RENAMEDgetListOfScreenObjects(){
       $sql = "SELECT * FROM screen_objects ORDER BY object_id";
       return $this->getResultsFromQuery($sql);
   }

   public function RENAMEDgetListOfScreens(){
       $sql = "SELECT * FROM screens ORDER BY screen_id";
       return $this->getResultsFromQuery($sql);
   }

   public function RENAMEDgetListofScreenSequences(){
       $sql = "SELECT * FROM screen_objects ORDER BY screen_id";
       return $this->getResultsFromQuery($sql);
```

```
}

public function RENAMEDgetScreenObjectByScreenID($screen_id){
    $sql = "SELECT * FROM screen_objects WHERE screen_id = " . $screen_id . " ORDER
    BY object_id";
    return $this->getResultsFromQuery($sql);
}

public function RENAMEDgetScreenObjectByName($object_name){
    $sql = "SELECT * FROM screen_objects WHERE object_name = " . $object_name . "
    ORDER BY object_id";
    return $this->getResultsFromQuery($sql);
}

public function RENAMEDgetScreenObjectByObjectID($object_id){
    $sql = "SELECT * FROM screen_objects WHERE object_id = " . $object_id . "
    ORDER BY object_name";
    return $this->getResultsFromQuery($sql);
}

public function RENAMEDgetObjectsLabelsByScreenID($screen_id){
    $sql = "SELECT * FROM screen_objects_labels WHERE screen_id = " . $screen_id .
    " ORDER BY object_screen_id";
     return $this->getResultsFromQuery($sql);
}

public function RENAMEDgetScreenByID($screen_id){
    $sql = "SELECT * FROM screens WHERE screen_id = " . $screen_id . " ORDER BY
    screen_name";
     return $this->getResultsFromQuery($sql);
}

public function RENAMEDgetScreenByName($screen_name){
    $sql = "SELECT * FROM screens WHERE screen_name = " . $screen_name . " ORDER
    BY screen_id";
    return $this->getResultsFromQuery($sql);
}

public function RENAMEDgetScreenSequenceByRoleID($role_id){
    $sql = "SELECT * FROM screen_sequence WHERE role_id = " . $role_id . " ORDER
    BY screen_id";
    return $this->getResultsFromQuery($sql);
}

public function RENAMEDgetScreenSequenceByObjectID($object_id){
    $sql = "SELECT * FROM screen_sequence WHERE object_id = " . $object_id . "
    ORDER BY screen_id";
    return $this->getResultsFromQuery($sql);
}

public function RENAMEDgetScreenSequenceByScreenID($screen_id){
    $sql = "SELECT * FROM screen_sequence WHERE screen_id = " . $screen_id . "
    ORDER BY role_id";
    return $this->getResultsFromQuery($sql);
}

public function RENAMEDgetScreenAccessByRole($role_id,$screen_id){
    $sql = "SELECT role_id,screen_id,access FROM screen_access WHERE screen_id = "
    . $screen_id . " AND role_id = " . $role_id;
    return $this->getResultsFromQuery($sql);
}

public function RENAMEDreturnAllowableComponents($role_id,$screen_id){
    $sql = "SELECT role_id,screen_id,access,visibility FROM object_access WHERE
    role_id=".$role_id." AND screen_id=".$screen_id;
    return $this->getResultsFromQuery($sql);
}

public function RENAMEDgetNamesOfScreen($screen_id){
    $sql = "SELECT * FROM screens WHERE screen_id = ".$screen_id;
    return $this->getResultsFromQuery($sql);
}

public function RENAMEDgetLabelsOfScreen($screen_id,$lang){
    if($lang == 1){
          $sql = "SELECT objects_screen_id, object_name_e from
    screen_objects_labels WHERE screen_id = ".$screen_id;
    }
    else{
          $sql = "SELECT objects_screen_id, object_name_s from
    screen_objects_labels WHERE screen_id = ".$screen_id;
    }
    return $this->getResultsFromQuery($sql);
}

public function RENAMEDgetUserAccountByLogin($username,$hashed_password){
    $sql = "SELECT * FROM user_accounts WHERE username = " . $username . " AND
    hashed_password = " . $hashed_password;
    return $this->getResultsFromQuery($sql);
}

public function RENAMEDgetListOfStates(){
    $sql = "SELECT * FROM state_territory ORDER BY state_name";
    return $this->getResultsFromQuery($sql);
}

public function RENAMEDgetListOfRegions($stateId){
    $sql = "SELECT * FROM city_town_region WHERE state_id = " . $stateId . " ORDER
    BY ctr_Name";
     return $this->getResultsFromQuery($sql);
}

public function RENAMEDgetListOfDistricts($regionId){
    $sql = "SELECT * FROM districts WHERE ctr_id = " . $regionId . " ORDER BY
    district_name";
    return $this->getResultsFromQuery($sql);
}

public function RENAMEDgetListOfSchools($districtId){
    if($districtId != "all"){
          $sql = "SELECT * FROM schools WHERE district_id = " . $districtId . "
          ORDER BY school_name";
    } else {
          $sql = "SELECT * FROM schools ORDER BY school_name";
    }
    return $this->getResultsFromQuery($sql);
}

public function RENAMEDgetSchoolDetails($schoolId){
    $sql = "SELECT * FROM schools, school_details WHERE schools.school_id =
    school_details.school_id AND schools.school_id = " . $schoolId;
    return $this->getResultsFromQuery($sql);
}

public function RENAMEDgetDistrictAndRegionBySchool($schoolId){
    $sql = "SELECT district_id, ctr_id FROM schools WHERE school_id=" . $schoolId;
```

```
        return $this->getResultsFromQuery($sql);
    }

    public function RENAMEDgetStateByRegion($regionId){
        $sql = "SELECT state_id FROM city_town_region WHERE ctr_id = ".$regionId;
        return $this->getResultsFromQuery($sql);
    }

    public function RENAMEDaddEmployee($employeeObject){
        $sqlEmployee = "INSERT INTO employees (user_id, first_name, middle_name,
         last_name, suffix, email, title, employee_id, phone)
         VALUES ("  . $employeeObject->userId . ",
         '" . $employeeObject->firstName . "','" . $employeeObject->middleName . "',
         '" . $employeeObject->lastName . "','" . $employeeObject->suffix . "',
        '" . $employeeObject->email . "','" . $employeeObject->title . "',
        " . $employeeObject->employeeId . ",'" . $employeeObject->phone . "')";

         $recordId = $this->addRecord($sqlEmployee);

         if($recordId){return 1;}
         else
         {return 0;}
    }

    public function RENAMEDgetEmployee($userId){
        $sql = "SELECT * FROM employees WHERE user_id = ". $userId;
        return $this->getResultsFromQuery($sql);
    }

    public function RENAMEDgetEmployees(){
        $sql = "SELECT * FROM employees";
        return $this->getResultsFromQuery($sql);
    }

    public function RENAMEDgetUserRoleSchoolId($userId){
        $sql = "SELECT * FROM school_users_roles WHERE user_id = ". $userId;
        return $this->getResultsFromQuery($sql);
    }

    public function RENAMEDgetUsersRoleSchoolSport(){
        $sql = "SELECT * FROM school_users_roles";
        return $this->getResultsFromQuery($sql);
    }

    public function RENAMEDaddUserRoleSchool($userRoleSchool){
        $sqlUser = "INSERT INTO school_users_roles (user_id,
        role_id,school_id,student_id) VALUES (" . $userRoleSchool->userId . ",
         " . $userRoleSchool->roleId . ",
         " . $userRoleSchool->schoolId .",". $userRoleSchool->studentId.")";
        $recordId = $this->addRecord($sqlUser);
        if($recordId)
        {return $recordId;}
         else
         {return 0;}
    }

    public function RENAMEDaddUserAccount($userAccountObject){
        $sqlUserAccount = "INSERT INTO user_accounts (
         email,username,hashed_password,enabled,role_id)
         VALUES ('" . $userAccountObject->email . "',
         '" . $userAccountObject->username . "',
         '" . $userAccountObject->hashedPassword . "',
         " . $userAccountObject->enabled . ",
    " . $userAccountObject->roleId . ")";

    $userId = $this->addNewRecord($sqlUserAccount);

    if($userId)
    {return $userId;}
    else
    {return 0;}
}

public function RENAMEDgetUserAccount($userId){
    $sql = "SELECT user_id, email, username, hashed_password, enabled, role_id
    FROM user_accounts
    WHERE user_id = ". $userId;
    return $this->getResultsFromQuery($sql);
}

public function RENAMEDgetUserAccounts(){
    $sql = "SELECT user_id, email, username, hashed_password, enabled, role_id
    FROM user_accounts";
    return $this->getResultsFromQuery($sql);
}

public function RENAMEDgetUsers(){
    $sql = "SELECT * FROM users";
    return $this->getResultsFromQuery($sql);
}

public function RENAMEDgetUserById($userId){
    $sql = "SELECT * FROM users WHERE user_id = ".$userId;
    return $this->getResultsFromQuery($sql);
}

public function RENAMEDgetSchoolStudents($schoolId){
    $sql = "SELECT students.student_id, first_name, middle_name, last_name, suffix,
    email, student_number, school_id, date_of_birth, gender FROM students,
    student_demographics
    WHERE students.student_id = student_demographics.student_id
    AND school_id = " . $schoolId . "
    ORDER BY last_name";
    return $this->getResultsFromQuery($sql);
}

public function RENAMEDgetStudentByID($studentId){
    $sql = "SELECT * FROM students, student_demographics
    WHERE students.student_id = " . $studentId . "AND
    student_demographics.student_id = " . $studentId;
    return $this->getResultsFromQuery($sql);
}

public function RENAMEDsearchForStudents($firstName= '', $lastName = ''){
    $sql = "SELECT * FROM (SELECT students.student_id, first_name, middle_name,
    last_name, suffix, email, student_number, school_id, date_of_birth, gender FROM
    students, student_demographics WHERE students.student_id =
    student_demographics.student_id) AS A
    WHERE A.first_name LIKE  '%" . $firstName . "%' OR A.last_name LIKE  '%" .
    $lastName . "%' ORDER BY A.last_name";
    return $this->getResultsFromQuery($sql);
}

public function RENAMEDgetStudentGuardians($studentId){
    $sql = "SELECT * FROM parents_or_guardians WHERE student_id = " . $studentId;
    return $this->getResultsFromQuery($sql);
```

```
}

public function RENAMEDaddStudent($studentObject){
    $sqlGeneralStudent = "INSERT INTO students (first_name,middle_name,last_name,
    suffix,email,student_number,school_id)
    VALUES ('" . $studentObject->firstName . "',
    '" . $studentObject->middleName . "','" . $studentObject->lastName . "',
    '" . $studentObject->suffix . "','" . $studentObject->email . "',
    '" . $studentObject->studentNumber . "'," . $studentObject->schoolId . ")";

    $recordId = $this->addNewRecord($sqlGeneralStudent);

    if($recordId){
      $sqlStudentDemo = "INSERT INTO student_demographics (student_id,
        date_of_birth,
        gender)
        VALUES ('" . $recordId . "',
        '" . $studentObject->dateOfBirth . "',
        '" . $studentObject->gender . "')";

        if($this->addRecord($sqlStudentDemo)) return $recordId;
        else return $recordId;
    }
    else{return 0;}
}

public function RENAMEDupdateStudent($studentObject,$studentId){
    $sqlGeneralStudent = "UPDATE students SET first_name =
    '" . $studentObject->firstName . "',
    middle_name = '" . $studentObject->middleName . "',
    last_name = '" . $studentObject->lastName . "',
    suffix = '" . $studentObject->suffix . "',
    email = '" . $studentObject->email . "',
    student_number = '" . $studentObject->studentNumber . "',
    school_id = '" . $studentObject->schoolId . "'
    WHERE student_id = " . $studentId;
    $recordId = $this->updateRecord($sqlGeneralStudent);

    if($recordId){
          $sqlStudentDemo = "UPDATE student_demographics date_of_birth = '" .
          $studentObject->dateOfBirth . "',
          gender = '" . $studentObject->gender . "'
          WHERE student_id = " . $studentId;

          if($this->updateRecord($sqlStudentDemo)) return 1;
          else return 2;
    }
    else{return 0;}
}

public function RENAMEDaddStudentGuardian($studentGuardian){
    $sqlGeneralStudent = "INSERT INTO parents_or_guardians (student_id,
    parent_name,parent_email,parent_phone,parent_address,`primary`)
    VALUES (" . $studentGuardian->studentId . ",
    '" . $studentGuardian->name . "','" . $studentGuardian->email . "',
    '" . $studentGuardian->phone . "','" . $studentGuardian->address . "',
    " . $studentGuardian->isPrimary . ")";
    return $this->addNewRecord($sqlGeneralStudent);
}

public function RENAMEDupdateStudentGuardian($studentGuardian,$guardianId){
    $sqlGeneralStudent = "UPDATE parents_or_guardians SET student_id = " .
    $studentGuardian->studentId . ",parent_name = '" . $studentGuardian->name . "',
    parent_email = '" . $studentGuardian->email . "',
    parent_phone = '" . $studentGuardian->phone . "',
    parent_address = '" . $studentGuardian->address . "',
    primary` = " . $studentGuardian->isPrimary . "
    WHERE guardian_id = " . $guardianId;
    return $this->updateRecord($sqlGeneralStudent);
}

public function RENAMEDgetConcussion($concussionId){
    $sql = "SELECT * FROM incidents WHERE incident_id = " . $concussionId;
    return $this->getResultsFromQuery($sql);
}

public function getConcussionsByUserID($concussionId){
    $sql = "SELECT * FROM incidents WHERE reporting_user_id = ". $concussionId;
       return $this->getResultsFromQuery($sql);
}

public function RENAMEDgetConcussionFollowups($concussionId){
    $sql = "SELECT * FROM incident_updates WHERE incident_id = " .
    $concussionId;
    return $this->getResultsFromQuery($sql);
}

public function RENAMEDgetSymptomsWithRecord($recordId){
    $sql = "SELECT * FROM incident_lingering_symptoms WHERE record_id = " .
    $recordId;
    return $this->getResultsFromQuery($sql);
}

public function RENAMEDgetSchoolConcussions($schoolId){
    $sql = "SELECT incidents.incident_id, incident_reference_id, student_id,
    school_id, reporting_user_id, incident_location_id,
    incident_location_details, school_location_id, sport_id,
    contact_mechanism_id,impact_location_id,protection_present,
    loss_conciousness, parents_notified, protocol, removed, removed_by_user_id,
    tool_id, symptom_comments, date, closed, updated AS status_updated
    FROM incidents, incident_status
    WHERE incidents.incident_id = incident_status.incident_id
    AND incidents.school_id = " . $schoolId;
    return $this->getResultsFromQuery($sql);
}

 public function RENAMEDgetStudentConcussions($studentId){
    $sql = "SELECT incidents.incident_id, incident_reference_id, student_id,
    school_id, reporting_user_id, incident_location_id,
    incident_location_details, school_location_id, sport_id,
    contact_mechanism_id, impact_location_id, protection_present,
    head_gear_usage, loss_conciousness, parents_notified, protocol, removed,
    removed_by_user_id, tool_id, symptom_comments, date, closed, updated AS
    status_updated FROM incidents, incident_status
    WHERE incidents.incident_id = incident_status.incident_id
    AND incidents.student_id = " . $studentId. " ORDER BY incidents.date DESC";
    return $this->getResultsFromQuery($sql);
}

public function getUserConcussionsByID($userId){
    $sql = "SELECT * FROM school_users_roles WHERE user_id = ". $userId;
    return $this->getResultsFromQuery($sql);
}


public function RENAMEDgetIncidentOperationHistory($incidentId){
    $sql = "SELECT * FROM incident_records
    WHERE incident_id = " . $incidentId;
    return $this->getResultsFromQuery($sql);
}

public function RENAMEDgetFollowUpOperationHistory($followupId){
    $sql = "SELECT * FROM incident_records WHERE follow_up_id = ". $followupId;
    return $this->getResultsFromQuery($sql);
}

public function RENAMEDchangeIncidentStatus($incidentId,$status){
    $sql = "UPDATE incident_status SET status = " . $status . "
    WHERE incident_id = " . $incidentId;
     return $this->updateRecord($sql);
}

public function RENAMEDaddConcussionEvent($concussionEvent){
    $referenceId = crypt($concussionEvent->studentId .
    $concussionEvent->schoolId . date('Y-m-d H:i:s'));

    $sqlConcussionEvent = "INSERT INTO incidents (incident_reference_id,
    student_id, school_id, reporting_user_id, incident_location_id,
    incident_location_details, school_location_id, sport_id,
    contact_mechanism_id, impact_location_id, protection_present,
    head_gear_usage, loss_conciousness, parents_notified, protocol,
    removed, removed_by_user_id, tool_id, symptom_comments, date)
    VALUES ('" . $referenceId . "'," . $concussionEvent->studentId . ",
    " . $concussionEvent->schoolId . "," . $concussionEvent->reportingUserId . ",
    " . $concussionEvent->incidentLocationId . ",
    '" . $concussionEvent->incidentLocationDetails . "',
    " . $concussionEvent->schoolLocationId . ",
    " . $concussionEvent->sportId . "," . $concussionEvent->contactMechanismId . ",
    " . $concussionEvent->impactLocationId . ",
    " . $concussionEvent->wasProtectionPresent . ",
    " . $concussionEvent->headGearUsage . ",
    " . $concussionEvent->wasLossOfConciousness . ",
    " . $concussionEvent->parentsNotified . "," . $concussionEvent->protocol . ",
    " . $concussionEvent->isRemoved . "," . $concussionEvent->removedByUserId . ",
    " . $concussionEvent->assessmentToolId . ",
    '" . $concussionEvent->symptomComments . "'," . $concussionEvent->date . ")";

    $incidentId = $this->addNewRecord($sqlConcussionEvent);
    $arrayOfSymptomsIds = $concussionEvent->symptomsArray;

    $sqlConcussionEventSymptoms = "INSERT INTO incident_lingering_symptoms
    (record_id, symptom_id) VALUES ";
       $symptomValues = "";

    foreach ($arrayOfSymptomsIds as $key => $value){
          $symptomValues .= "('" .$referenceId. "', ". $value->symptomId . "), ";
    }

    $sqlConcussionEventSymptoms = substr($sqlConcussionEventSymptoms .
    $symptomValues, 0, -2);
    $this->addRecordNoReturn($sqlConcussionEventSymptoms);
    $sqlIncidentStatus = "INSERT INTO incident_status (incident_id) VALUES (" .
    $incidentId . ")";
    $this->addRecordNoReturn($sqlIncidentStatus);

    $sqlIncidentRecordWrite = "INSERT INTO incident_records (incident_id,
    operation_type_id) VALUES (" . $incidentId . ", 2)";
    $this->addRecordNoReturn($sqlIncidentRecordWrite);
    return $incidentId;
```

```
}


public function RENAMEDupdateConcussionEvent($concussionEvent,$incidentId){
    $sqlConcussionEvent = "UPDATE incidents SET student_id = " .
    $concussionEvent->studentId . ",school_id = " . $concussionEvent->schoolId . ",
     reporting_user_id = " . $concussionEvent->reportingUserId . ",
     incident_location_id = " . $concussionEvent->incidentLocationId . ",
     incident_location_details='".$concussionEvent->incidentLocationDetails. "',
     school_location_id = " . $concussionEvent->schoolLocationId . ",
     sport_id = " . $concussionEvent->sportId . ",
     contact_mechanism_id = " . $concussionEvent->contactMechanismId . ",
     impact_location_id = " . $concussionEvent->impactLocationId . ",
     protection_present = " . $concussionEvent->wasProtectionPresent . ",
     head_gear_usage = " . $concussionEvent->headGearUsage . ",
     loss_conciousness = " . $concussionEvent->wasLossOfConciousness . ",
     parents_notified = " . $concussionEvent->parentsNotified . ",
     protocol = " . $concussionEvent->protocol . ",
     removed = " . $concussionEvent->isRemoved . ",
     removed_by_user_id = " . $concussionEvent->removedByUserId . ",
     tool_id = " . $concussionEvent->assessmentToolId . ",
     symptom_comments = '" . $concussionEvent->symptomComments . "'
     WHERE incident_id = " . $incidentId;

     if ($this->updateRecord($sqlConcussionEvent)) {
           $sql = "SELECT * FROM incidents WHERE incident_id = " . $incidentId;
           $result = $this->getResultsFromQuery($sql);

           if (sizeof($result) > 1) {return 0;}
           else {
              $sql = "DELETE FROM incident_lingering_symptoms WHERE record_id =
              '" . $result[0]["incident_reference_id"] ."'";
              $this->addRecordNoReturn($sql);
              $arrayOfSymptomsIds = $concussionEvent->symptomsArray;
              $sqlConcussionEventSymptoms = "INSERT INTO
              incident_lingering_symptoms (record_id, symptom_id) VALUES ";
              $symptomValues = "";

              foreach ($arrayOfSymptomsIds as $key => $value){
                 $symptomValues .= "('" . $result[0]["incident_reference_id"] . "',
                 " . $value->symptomId . "), ";
              }

               $sqlConcussionEventSymptoms = substr($sqlConcussionEventSymptoms .
               $symptomValues, 0, -2);
               $this->addRecordNoReturn($sqlConcussionEventSymptoms);

               $sqlIncidentRecordWrite = "INSERT INTO incident_records
               (incident_id, operation_type_id) VALUES (" . $incidentId . ", 1)";
               $this->addRecordNoReturn($sqlIncidentRecordWrite);
               return 1;
             }
        }
      return 0;
}

public function RENAMEDaddConcussionEventFollowup($concussionEventUpdate,
$incidentId)
{
    $lingeringSymptomsRecordId = crypt($concussionEventUpdate->incidentId .
    $incidentId . date('Y-m-d H:i:s'));
    $arrayOfSymptomsIds = $concussionEventUpdate->lingeringSymptomsArray;
    $sqlConcussionEventFollowupSymptoms = "INSERT INTO incident_lingering_symptoms
    (record_id, symptom_id) VALUES ";
    $symptomValues = "";

    foreach ($arrayOfSymptomsIds as $key => $value){
           $symptomValues .= "('" . $lingeringSymptomsRecordId . "', " .
           $value->symptomId . "), ";
    }

    $sqlConcussionEventFollowupSymptoms =
    substr($sqlConcussionEventFollowupSymptoms .
    $symptomValues, 0, -2);
    $this->addRecordNoReturn($sqlConcussionEventFollowupSymptoms);
    $sqlConcussionEventFollowup = "INSERT INTO incident_updates (incident_id,
    reporting_user_id, lingering_symptoms_record_id, lingering_description,
    time_resolved, diagnosed_by, pcs_diagnosis, imaging, follow_up_comments,
    days_absent, scheduled_modified, plan_504, rtl_date, rtp_date, date)
    VALUES (" . $concussionEventUpdate->incidentId . ",
    " . $concussionEventUpdate->reportingUserId . ",
    '" . $lingeringSymptomsRecordId . "',
    '" . $concussionEventUpdate->lingeringDescription . "',
    " . $concussionEventUpdate->timeResolved . ",
    '" . $concussionEventUpdate->diagnosedBy . "',
    " . $concussionEventUpdate->pcsDiagnosis . ",
    " . $concussionEventUpdate->imaging . ",
    '" . $concussionEventUpdate->followUpComments . "',
    " . $concussionEventUpdate->daysAbsent . ",
    " . $concussionEventUpdate->scheduledModified . ",
    " . $concussionEventUpdate->plan504 . ",
    '" . $concussionEventUpdate->rtlDate . "',
    '" . $concussionEventUpdate->rtpDate . "',
    " . $concussionEventUpdate->date . ")";
    $followUpId = $this->addNewRecord($sqlConcussionEventFollowup);
    $sqlIncidentRecordWrite = "INSERT INTO incident_records (follow_up_id,
    operation_type_id) VALUES (" . $followUpId . ", 2)";
    $this->addRecordNoReturn($sqlIncidentRecordWrite);
    return $followUpId;
}

public function RENAMEDupdateConcussionEventFollowup($concussionEventUpdate,
$followUpId, $lingeringSymptomsRecordId){
    $sqlConcussionEvent = "UPDATE incident_updates SET reporting_user_id =  " .
    $concussionEventUpdate->reportingUserId . ",
    lingering_description = '" . $concussionEventUpdate->lingeringDescription . "',
    time_resolved = " . $concussionEventUpdate->timeResolved . ",
    diagnosed_by = '" . $concussionEventUpdate->diagnosedBy . "',
    pcs_diagnosis = " . $concussionEventUpdate->pcsDiagnosis . ",
    imaging = " . $concussionEventUpdate->imaging . ",
    follow_up_comments = '" . $concussionEventUpdate->followUpComments . "',
    days_absent = " . $concussionEventUpdate->daysAbsent . ",
    scheduled_modified = " . $concussionEventUpdate->scheduledModified . ",
    plan_504 = " . $concussionEventUpdate->plan504 . ",
    rtl_date = '" . $concussionEventUpdate->rtlDate . "',
    rtp_date = '" . $concussionEventUpdate->rtpDate . "'
    WHERE follow_up_id = " . $followUpId;

    if($this->updateRecord($sqlConcussionEvent)) {
       $sql = "SELECT * FROM incident_updates WHERE follow_up_id = " . $followUpId;
       $result = $this->getResultsFromQuery($sql);

       if(sizeof($result) > 1) {return 0;}
       else {
          $sql = "DELETE FROM incident_lingering_symptoms WHERE record_id = '" .
          $lingeringSymptomsRecordId ."'";
```

```
            $this->addRecordNoReturn($sql);
            $arrayOfSymptomsIds = $concussionEventUpdate->lingeringSymptomsArray;
            $sqlConcussionEventFollowupSymptoms = "INSERT INTO
            incident_lingering_symptoms
            (record_id, symptom_id) VALUES ";
            $symptomValues = "";
            foreach ($arrayOfSymptomsIds as $key => $value)
            {
                    $symptomValues .= "('" . $lingeringSymptomsRecordId . "', " .
                    $value->symptomId . "), ";
            }
            $sqlConcussionEventFollowupSymptoms =
            substr($sqlConcussionEventFollowupSymptoms . $symptomValues, 0, -2);
            $this->addRecordNoReturn($sqlConcussionEventFollowupSymptoms);
            $sqlIncidentRecordWrite = "INSERT INTO incident_records (follow_up_id,
            operation_type_id) VALUES (" . $followUpId . ", 1)";
            $this->addRecordNoReturn($sqlIncidentRecordWrite);
            return 1;
        }
    }
}

public function RENAMEDgetEventSymptoms($referenceId){
    $sql = "SELECT * FROM incident_lingering_symptoms WHERE record_id = '" .
    $referenceId ."'";
    return $this->getResultsFromQuery($sql);
}

public function RENAMEDgetAssessmentTools(){
    $sql = "SELECT * FROM concussion_assessment_tools";
    return $this->getResultsFromQuery($sql);
}

public function RENAMEDgetEventLocations(){
    $sql = "SELECT * FROM incident_locations";
    return $this->getResultsFromQuery($sql);
}

public function RENAMEDgetContactMechanisms(){
    $sql = "SELECT * FROM contact_mechanisms";
    return $this->getResultsFromQuery($sql);
}

public function RENAMEDgetImpactHeadLocations(){
    $sql = "SELECT * FROM impact_head_location";
    return $this->getResultsFromQuery($sql);
}

public function RENAMEDgetSports(){
    $sql = "SELECT * FROM sports";
    return $this->getResultsFromQuery($sql);
}
public function RENAMEDgetSymptoms($type= 2){
    $sql = "SELECT * FROM symptoms";
    if($type == 0){
        $sql = "SELECT * FROM symptoms WHERE isFollowUpType = 0";
    } else if($type == 1){
        $sql = "SELECT * FROM symptoms WHERE isFollowUpType = 1";
    }
    return $this->getResultsFromQuery($sql);
}

public function RENAMEDgetRoles(){
        $sql = "SELECT * FROM roles";
        return $this->getResultsFromQuery($sql);
    }

    public function RENAMEDgetMedicalImaging(){
        $sql = "SELECT * FROM medical_imaging";
        return $this->getResultsFromQuery($sql);
    }

    public function RENAMEDgetDiagnosingRoles(){
        $sql = "SELECT * FROM diagnosing_roles";

        return $this->getResultsFromQuery($sql);
    }

    public function RENAMEDimportSchoolDistrict($districtName,$ctrName){
        $sql = "SELECT * FROM city_town_region WHERE ctr_Name = '" . $ctrName . "'";
        $result = $this->getResultsFromQuery($sql);

        if (sizeof($result) > 0) {
                $sqlInsert = "INSERT INTO districts (district_name,ctr_id)
                VALUES ('" . $districtName . "', " . $result[0]["ctr_id"] . ")";
                $this->addRecordNoReturn($sqlInsert);
        }
    }

    public function RENAMEDimportSchool($districtName,$schoolName){
        $sql = "SELECT * FROM districts WHERE district_name = '" . $districtName . "'";
        $district = $this->getResultsFromQuery($sql);

        if (count($district) > 0) {
                $sql = "SELECT * FROM city_town_region WHERE ctr_id = " .
                $district[0]["ctr_id"];
                $ctr = $this->getResultsFromQuery($sql);

                if(count($ctr) > 0) {
                        $sqlSchoolInsert = "INSERT INTO schools (school_name, district_id,
                        ctr_id) VALUES ('" . preg_replace('/[^a-zA-Z0-9\s]/', '',
                        strip_tags(html_entity_decode($schoolName))) . "',
                         " . $district[0]["district_id"] . ",”. $ctr[0]["ctr_id"] . ")";
                        $schoolId = $this->addNewRecord($sqlSchoolInsert);

                        $sqlSchoolDetailsInsert = "INSERT INTO school_details (school_id,
                        address,phone_number) VALUES (" . $schoolId . ", '--', '--')";
                        $this->addRecordNoReturn($sqlSchoolDetailsInsert);
                }
        }
    }

    private function getResultsFromQuery($query){
        $result = NULL;
        $mysqlConnection = $this->initMySqlConnection();
        $result = $mysqlConnection->query($query);
        if (!$result) {
                throw new Exception("Database Error [{$this->database->errno}] {$this
                ->database->error}");
        } else {
                $array = array();
                while($row = $result->fetch_assoc()) $array[] = $row;
        }
        $mysqlConnection->close();
        return $array;
    }
```

```
   private function addNewRecord($query){
       $result = NULL;
       $mysqlConnection = $this->initMySqlConnection();
       $result = $mysqlConnection->query($query);
       $recordId = $mysqlConnection->insert_id;
       $mysqlConnection->close();
       return $recordId;
   }

   private function addRecord($query){
       $result = NULL;
       $mysqlConnection = $this->initMySqlConnection();
       $result = $mysqlConnection->query($query);
       $mysqlConnection->close();
       return $result;
   }

   private function addRecordNoReturn($query){
       $mysqlConnection = $this->initMySqlConnection();
       $mysqlConnection->query($query);
       $mysqlConnection->close();
   }

   private function updateRecord($query){
       $result = NULL;
       $mysqlConnection = $this->initMySqlConnection();
       $result = $mysqlConnection->query($query);
       $mysqlConnection->close();
       return $result;
   }

   private function initMySqlConnection(){
       $conn = new mysqli($this->dbServerName, $this->dbUser, $this->dbPassword,
       $this->dbName);
       if ($conn->connect_error) {
             die("Connection failed: " . $conn->connect_error);
       }
       else{
             return $conn;
       }
   }
}

class Employee{
      public $userId;
      public $firstName;
      public $middleName;
      public $lastName;
      public $suffix;
      public $email;
      public $title;
      public $employeeId;
      public $phone;
      public function __construct($userId = 0, $firstName = 'Y',$middleName =
      'K',$lastName = 'R',$suffix = '',$email = '',$title = '',$employeeId = 1,
      $phone = ''){
             $this->userId = $userId;
             $this->firstName = $firstName;
             $this->middleName = $middleName;
             $this->lastName = $lastName;
             $this->suffix = $suffix;
             $this->email = $email;
```

```
            $this->title = $title;
            $this->employeeId = $employeeId;
            $this->phone = $phone;
      }
}

class UserAccount{
      public $email;
      public $username;
      public $hashedPassword;
      public $enabled;
      public $roleId;
      public function __construct($email = '',$username = 'Jane Doe',
      $hashedPassword = '',$enabled = 0, $roleId = 1){
            $this->email = $email;
            $this->username = $username;
            $this->hashedPassword = $hashedPassword;
            $this->enabled = $enabled;
            $this->roleId = $roleId;
      }
}

class UserRoleSchool{
      public $userId;
      public $roleId;
      public $schoolId;
      public $studentId;
      public function __construct($userId = 0, $roleId = 0,$schoolId =0,
      $studentId = 0){
            $this->userId = $userId;
            $this->roleId = $roleId;
            $this->schoolId = $schoolId;
            $this->studentId = $studentId;
      }
}

class Screen {
      public $screenId;
      public $screenName;
      public function __construct($screenId = 0, $screenName = '') {
            $this->screenId = $screenId;
            $this->screenName = $screenName;
      }
}

class ScreenObject {
      public $objectId;
      public $objectName;
      public $screenId;
      public function __construct($objectId = 0, $objectName = '', $screenId = 0) {
            $this->objectId = $objectId;
            $this->objectName = $objectName;
            $this->screenId = $screenId;
      }
}

class ScreenAccess {
      public $roleId;
      public $screenId;
      public $access;
      public function __construct($roleId = 0, $screenId = 0, $access = 1) {
            $this->roleId = $roleId;
            $this->screenId = $screenId;
```

```
		$this->access = $access;
	}
}

class ObjectAccess {
	public $screenId;
	public $objectId;
	public $roleId;
	public $readWriteClickable;
	public $isButton;
	public $showHide;
	public function __construct($screenId = 0, $objectId = 0, $roleId = 0,
	$readWriteClickable = 1,$isButton = 1,$showHide = 1) {
		$this->screenId = $screenId;
		$this->objectId = $objectId;
		$this->roleId = $roleId;
		$this->readWriteClickable = $readWriteClickable;
		$this->isButton = $isButton;
		$this->showHide = $showHide;
	}
}

class ScreenSequence {
	public $roleId;
	public $objectId;
	public $screenId;
	public function __construct($roleId = 0, $objectId = 0, $screenId = 0) {
		$this->roleId = $roleId;
		$this->objectId = $objectId;
		$this->screenId = $screenId;
	}
}

class Student {
	public $firstName;
	public $middleName;
	public $lastName;
	public $suffix;
	public $email;
	public $studentNumber;
	public $schoolId;
	public $dateOfBirth;
	public $gender;

	public function __construct($firstName = 'John', $middleName = '', $lastName =
	'Doe', $suffix = '', $email = '', $studentNumber = '', $schoolId = 0,
	$dateOfBirth = '01/01/2000', $gender = ''){
		$this->firstName = $firstName;
		$this->middleName = $middleName;
		$this->lastName = $lastName;
		$this->suffix = $suffix;
		$this->email = $email;
		$this->studentNumber = $studentNumber;
		$this->schoolId = $schoolId;
		$this->dateOfBirth = $dateOfBirth;
		$this->gender = $gender;
	}
}

class StudentGuardian {
	public $studentId;
	public $name;
	public $email;
	public $phone;
	public $address;
	public $isPrimary;
	public function __construct($studentId = 0, $name = 'Jane Doe', $email = '',
	$phone = '', $address = '', $isPrimary = 0){
		$this->studentId = $studentId;
		$this->name = $name;
		$this->email = $email;
		$this->phone = $phone;
		$this->address = $address;
		$this->isPrimary = $isPrimary;
	}
}

class ConcussionEvent {
	public $studentId;
	public $schoolId;
	public $reportingUserId;
	public $incidentLocationId;
	public $incidentLocationDetails;
	public $schoolLocationId;
	public $sportId;
	public $contactMechanismId;
	public $impactLocationId;
	public $wasProtectionPresent;
	public $headGearUsage;
	public $wasLossOfConciousness;
	public $parentsNotified;
	public $protocol;
	public $isRemoved;
	public $removedByUserId;
	public $assessmentToolId;
	public $symptomComments;
	public $date;
	public $symptomsArray;

	public function __construct($studentId = 0,$schoolId = 0,$reportingUserId = 0,
	$incidentLocationId = 0,$incidentLocationDetails = '',$schoolLocationId = 0,
	$sportId = 0,$contactMechanismId = 0,$impactLocationId = 0,
	$wasProtectionPresent = 0,$headGearUsage = 0,$wasLossOfConciousness = 0,
	$parentsNotified = 0,$protocol = 0,$isRemoved = 0,$removedByUserId = 0,
	$assessmentToolId = 0,$symptomComments = '',$date = 0,$symptomsArray =
	array()){
		$this->studentId = $studentId;
		$this->schoolId = $schoolId;
		$this->reportingUserId = $reportingUserId;
		$this->incidentLocationId = $incidentLocationId;
		$this->incidentLocationDetails = $incidentLocationDetails;
		$this->schoolLocationId = $schoolLocationId;
		$this->sportId = $sportId;
		$this->contactMechanismId = $contactMechanismId;
		$this->impactLocationId = $impactLocationId;
		$this->wasProtectionPresent = $wasProtectionPresent;
		$this->headGearUsage = $headGearUsage;
		$this->wasLossOfConciousness = $wasLossOfConciousness;
		$this->parentsNotified = $parentsNotified;
		$this->protocol = $protocol;
		$this->isRemoved = $isRemoved;
		$this->removedByUserId = $removedByUserId;
		$this->assessmentToolId = $assessmentToolId;
		$this->symptomComments = $symptomComments;
		$this->date = $date;
		$this->symptomsArray = $symptomsArray;
	}
}

class ConcussionFollowup{
	public $incidentId;
	public $reportingUserId;
	public $lingeringSymptomsArray;
	public $lingeringDescription;
	public $timeResolved;
	public $diagnosedBy;
	public $pcsDiagnosis;
	public $imaging;
	public $followUpComments;
	public $daysAbsent;
	public $scheduledModified;
	public $plan504;
	public $rtlDate;
	public $rtpDate;
	public $date;
	public function __construct($incidentId = 0,$reportingUserId = 0,
	$lingeringSymptomsArray = array(),$lingeringDescription = '',$timeResolved = 0,
	$diagnosedBy = 0,$pcsDiagnosis = 0,$imaging = 0,$followUpComments = '',
	$daysAbsent = 0,$scheduledModified = 0,$plan504 = 0,$rtlDate = '',
	$rtpDate = '',$date = 0){
		$this->incidentId = $incidentId;
		$this->reportingUserId = $reportingUserId;
		$this->lingeringSymptomsArray = $lingeringSymptomsArray;
		$this->lingeringDescription = $lingeringDescription;
		$this->timeResolved = $timeResolved;
		$this->diagnosedBy = $diagnosedBy;
		$this->pcsDiagnosis = $pcsDiagnosis;
		$this->imaging = $imaging;
		$this->followUpComments = $followUpComments;
		$this->daysAbsent = $daysAbsent;
		$this->scheduledModified = $scheduledModified;
		$this->plan504 = $plan504;
		$this->rtlDate = $rtlDate;
		$this->rtpDate = $rtpDate;
		$this->date = $date;
	}
}
```

Appendix C – Server Interceptor API

1) Pseudocode for Access Control Interceptor function

```
//Serves as Access Control Interceptor function
public boolean incomingRequestPostProcessed(requestDetails, request, response){
   authToken = requestDetails.getHeader("Authorization");
   //Retrieves the user's id, clearance and, read and write MAC properties
   [userId,userRole,userClearance,readP,writeP] = verifyUser(authToken);
   httpMethod = request.getMethod();
   resourceName = requestDetails.getResourceName();
   serviceId = getServiceId(httpMethod, resourceName);
   acPermission = false;
   if(userId > 0){
      //Check if requested resource is secured/labeled
      [secured,labeled] = getResourceSecurity(httpMethod,resourceName);
      if(!(secured || labeled)){
         return true; //Continue with request processing
      }
      //Analyze MAC policies (if any)
      if(userClearance > 0 && labeled){
         acPermission = checkAndEnforceMAC(userClearance, serviceId, readP, readW);
      }
      //Analyze RBAC policies (if any)
      if((roleId > 0 && secured) && (acPermission || !labeled)){
         acPermission = checkAndEnforceRBAC(userRole, serviceId);
      }
   }
   else{//Error Message: User could not be verified}
   if(acPermission == false){
      //Error message: User does not have permission to access the
      //requested resource
   }
   return acPermission;
 }

 private int delClrDAC(userId, serviceId) {
    //Check if delegated user has a delegated clearance for the requested service
    if(currentTime>getStartTimeMAC() && currentTime<getEndTimeMAC()) {
       if(serviceId in service_permissions_mac(userId))
       {return delegatedClearance;}
    }
    return 0;
 }

  private int delRoleDAC(userId, serviceId) {
    //Check if delegated user has a delegated role for the requested service
    if(currentTime>getStartTimeRBAC() && currentTime<getEndTimeRBAC() {
       if(serviceId in service_permissions_rbac(userId))
       {return delegatedRole;}
    }
    return 0;
 }
 private boolean checkAndEnforceMAC(userId, serviceId){
      acPermission = false;
      //MAC services delegation
      if(dacPermission() && checkIfDacMac(userId)) {
         delClr = delClrDAC(userId, serviceId);
         if(delClr>0) { userClearance=delClr; } //Delegated clearance_id
     }
   //Get service classification and http method
   serviceClassification = getServiceClass(serviceId);
   //Retrieve MAC read or write property for pertinent user
   if(httpMethod == "GET"){
      //Simple security property
      if(readP == simpleSecurityProperty){
         if(userClearance >= serviceClassification){
            acPermission = true;
         }
      }
      //Strict * property
      elseif(readP == strictStarProperty){
         if(userClearance == serviceClassification){
            acPermission = true;
         }
      }
   }
   else{
      //Simple integrity property
      if(writeP == simpleIntegrityProperty){
         if(userClearance >= serviceClassification){
            acPermission = true;
         }
      }
      //Strict * property
      elseif(writeP == strictStarProperty){
         if(userClearance == serviceClassification){
            acPermission = true;
         }
      }
      //Liberal * property
      elseif(writeP == liberalStarProperty){
         if(userClearance <= serviceClassification){
            acPermission = true;
         }
      }
   }
   return acPermission;
 }

private boolean checkAndEnforceRBAC(userRole, serviceId){
   acPermission = false;
   //RBAC services delegation
   if(dacPermission() && checkIfDacRbac(userId)) {
            delRole = delRoleDAC(userId, serviceId);
            if(delRole>0) { userRole=delRole; } //Delegated role_id
     }
   //Get service set of roles
   serviceRoles = getRoleSet(serviceId);
   if(roleId in serviceRoles){
      acPermission = true;
   }
   return acPermission;
}
```

2) Source Code for incomingRequestPostProcessed function

```java
public boolean incomingRequestPostProcessed(RequestDetails theRequestDetails,
               HttpServletRequest theRequest, HttpServletResponse theResponse) {
   String jwt = theRequest.getHeader("Authorization");
   Boolean acPermission = false; //Initially, the user does not have permission to
                                   access the resource
   String identifiers = "";
   JSONObject object = null;
   HttpClient httpClient = new DefaultHttpClient();
   HttpContext localContext = new BasicHttpContext();
   // Verify if the user is a valid one
   HttpGet httpGet= new HttpGet(serviceLink+"/verifyUser/"+jwt);
   try {
      HttpResponse response = httpClient.execute(httpGet, localContext);
      HttpEntity entity = response.getEntity();
      identifiers = EntityUtils.toString(entity);
      //Returns user_id, role_id, clearance_id, write_property, and read property
      object = new JSONObject(identifiers); //Convert String to JSON Object
   } catch (Exception e) {/*throw new UnprocessableEntityException();*/}
   //If the user's identity could be properly validated then it returns the user's
     role, clearance,
   //and an indicator that the request was successful
   try {
       int user_id = Integer.parseInt(object.getString("user_id"));
       int mac_read = Integer.parseInt(object.getString("mac_read"));
       int mac_write = Integer.parseInt(object.getString("mac_write"));
       if(user_id>0) {
           JSONObject securedResource = null;
           String httpMethod = theRequest.getMethod();
           String resourceName = theRequestDetails.getResourceName();
           //Check if requested resource is secured/labeled
           httpGet = new
           HttpGet(serviceLink+"/resourceSecurity/"+httpMethod+"/"+resourceName);
           try {
               HttpResponse response = httpClient.execute(httpGet, localContext);
               HttpEntity entity = response.getEntity();
               identifiers = getASCIIContentFromEntity(entity);
               securedResource = new JSONObject(identifiers);
           } catch (Exception e) {/*throw new UnprocessableEntityException();*/}
           boolean secured = securedResource.getBoolean("secured");
           boolean labeled = securedResource.getBoolean("labeled");
           if(!(secured||labeled)){
               return true; //Continue with request processing (resource can be
              accessed by anyone)
           }
           //Obtain the id of the requested service
           JSONObject sid = null;
           httpGet = new
           HttpGet(serviceLink+"/serviceId/"+httpMethod+"/"+resourceName);
           try {
               HttpResponse response = httpClient.execute(httpGet, localContext);
               HttpEntity entity = response.getEntity();
               identifiers = getASCIIContentFromEntity(entity);
               sid = new JSONObject(identifiers);
               int service_id = sid.getInt("service_id");
               Integer clearance_id =
               Integer.parseInt(object.getString("clearance_id"));
               if(clearance_id>0 && labeled) {//Analyze MAC policies (if any)
                   acPermission = checkAndEnforceMAC(user_id, clearance_id,
                                  service_id, mac_read, mac_write);
               }
               Integer role_id = Integer.parseInt(object.getString("role_id"));
               if((role_id>0 && secured) && (acPermission || !labeled)) {//Analyze RBAC policies (if any)
                   acPermission = checkAndEnforceRBAC(user_id, role_id, service_id);
               }
           } catch (Exception e) {/*throw new UnprocessableEntityException();*/}
           if(!acPermission) {
               try {
                   theResponse.setContentType("application/json+fhir");
                   PrintWriter out = theResponse.getWriter();
                   out.println("{");
                   out.println("\"status\": \"403\",");
                   out.println("\"errorMessage\": \"User does not have permission to access the requested resource.\"");
                   out.println("}");
                   out.close();
               } catch (IOException e) {e.printStackTrace();}
               return false;
           }
           return true;
       }
       else {
           try {
               theResponse.setContentType("application/json+fhir");
               PrintWriter out = theResponse.getWriter();
               out.println("{");
               out.println("\"status\": \"400\",");
               out.println("\"errorMessage\": \"User Verification failed. Please try to do the request again...\"");
               out.println("}");
               out.close();
           } catch (IOException e) {
               e.printStackTrace();
           }
           return false;
       }
   } catch (JSONException e) {e.printStackTrace();}
   return true;
 }

 private boolean checkAndEnforceMAC(int user_id, int clearance_id, int service_id,
                                    int mac_read, int mac_write) {
   boolean acPermission = false;
   JSONObject serviceClassification = null;
   Integer delclr_id = 0;
   Integer class_id = 0;
   Integer macProperty = 0;
   String httpMethod = "";
   HttpClient httpClient = new DefaultHttpClient();
   HttpContext localContext = new BasicHttpContext();

   //MAC Service Delegation
   delclr_id = delClrDAC(user_id, service_id);
   if(delclr_id>0)
   {clearance_id = delclr_id;}

   //Get service classification and http method
   HttpGet httpGet = new HttpGet(serviceLink+"/serviceClass/"+service_id);
   try {
       HttpResponse response = httpClient.execute(httpGet, localContext);
       HttpEntity entity = response.getEntity();
       String identifiers = getASCIIContentFromEntity(entity);
       serviceClassification = new JSONObject(identifiers);
```

```
        class_id = serviceClassification.getInt("clearance_id");
        httpMethod = serviceClassification.getString("http_method");
    } catch (Exception e){/*throw new UnprocessableEntityException();*/}
    //Retrieve MAC read or write property for pertinent user
    if(httpMethod=="GET") {
        macProperty = mac_read;
        //Simple security property
        if (macProperty == 1) {
            if (clearance_id > class_id) {
                acPermission = true;
            }
        }
        //Strict * property
        else if (macProperty == 2) {
            if (clearance_id == class_id) {
                acPermission = true;
            }
        }
    }
    else{
        macProperty = mac_write;
        //Simple integrity property
        if (macProperty == 3) {
            if (clearance_id >= class_id) {
                acPermission = true;
            }
        }
        //Strict * property
        else if (macProperty == 4) {
            if (clearance_id == class_id) {
                acPermission = true;
            }
        }
        //Liberal * property
        else if (macProperty == 5) {
            if (clearance_id <= class_id) {
                acPermission = true;
            }
        }
    }
    return acPermission;
}

private boolean checkAndEnforceRBAC(int user_id, int role_id, int service_id) {
    boolean acPermission = false;
    Integer delrole_id = 0;
    JSONArray role_set = null;
    HttpClient httpClient = new DefaultHttpClient();
    HttpContext localContext = new BasicHttpContext();

    //MAC Service Delegation
    delrole_id = delClrDAC(user_id, service_id);
    if(delrole_id>0)
    {role_id = delrole_id;}

    //Get service set of roles
    HttpGet httpGet = new HttpGet(serviceLink+"/roleSet/"+service_id);
    try {
        HttpResponse response = httpClient.execute(httpGet, localContext);
        HttpEntity entity = response.getEntity();
        String identifiers = getASCIIContentFromEntity(entity);
        JSONObject rs = new JSONObject(identifiers);
        role_set = rs.getJSONArray("");
```

```java
        Integer role = 0;
        for (int i = 0; i < role_set.length(); i++) {
            role = role_set.getInt(i);
            if(role_id == role) {
                acPermission = true;
                break;
            }
        }
    } catch (Exception e) {//throw new UnprocessableEntityException();}
    return acPermission;
}

private int delClrDAC(int user_id, int service_id) {
    Integer delclr_id = 0;
    HttpClient httpClient = new DefaultHttpClient();
    HttpContext localContext = new BasicHttpContext();
    //MAC Service Delegation
    //Check if delegated user has a delegated clearance for the requested service
    JSONObject userDelegation = null;
    HttpGet httpGet = new
            HttpGet(serviceLink+"/userClearanceDelegation/"+user_id+service_id);
    try {
        HttpResponse response = httpClient.execute(httpGet, localContext);
        HttpEntity entity = response.getEntity();
        String identifiers = getASCIIContentFromEntity(entity);
        userDelegation = new JSONObject(identifiers);
        delclr_id = userDelegation.getInt("du_dclr_id");
        return delclr_id;
    } catch (Exception e){/*throw new UnprocessableEntityException();*/}
    return 0;
}

private int delRoleDAC(int user_id, int service_id) {
    Integer delrole_id = 0;
    HttpClient httpClient = new DefaultHttpClient();
    HttpContext localContext = new BasicHttpContext();
    //RBAC Service Delegation
    //Check if delegated user has a delegated role for the requested service
    JSONObject userDelegation = null;
    HttpGet httpGet = new
            HttpGet(serviceLink+"/userRoleDelegation/"+user_id+service_id);
    try {
        HttpResponse response = httpClient.execute(httpGet, localContext);
        HttpEntity entity = response.getEntity();
        String identifiers = getASCIIContentFromEntity(entity);
        userDelegation = new JSONObject(identifiers);
        delrole_id = userDelegation.getInt("du_drole_id");
        return delrole_id;
    } catch (Exception e){/*throw new UnprocessableEntityException();*/}
    return 0;
}
```

3) Source Code for registering the server interceptor in HAPI FHIR

```java
public class FHIR_RestfulServer extends RestfulServer {
  private static final long serialVersionUID = 1L;
  FhirVersionEnum fhirVersion = FhirVersionEnum.DSTU2;

  @Override
  protected void initialize() throws ServletException {
     // Set the resource providers used by this server
```

```
    super.setFhirContext(new FhirContext(fhirVersion));
    List<IResourceProvider> providerList=new ArrayList<IResourceProvider>();
    providerList.add(new PatientResourceProvider());
    providerList.add(new ConditionResourceProvider());
    providerList.add(new ObservationResourceProvider());
    providerList.add(new CarePlanResourceProvider());
    setResourceProviders(providerList);
    InterceptorAdapter addInterceptor = new AuthInterceptor();
    registerInterceptor(addInterceptor);
  }
}
```