

**CMPE 277 – Smartphone Application Development**

Project Report

On

**“CERTIFICATE PINNING”**

**Submitted to:**

**Prof. Chandrasekar Vuppalapati**

Submitted By:

Edward Reyes (004955640)

Sahil Arora (008730033)

Suantak Niangneihoi (008719893)

**Spring 2014**

***Abstract* – Security in mobile platforms has become an issue of interest and caution in the world of technology. As the level of advancement in mobile technologies keeps growing by the day, the requirement for robust, secure mobile environments has become a necessity more than a feature. This project describes how Certificate Pinning has been implemented in mobile applications for the Android platform. It includes an Android app that pins a certain website certificate. We also further extended the pinning project to a web service, added user authentication features for a small messaging app and also included some database access capabilities. A RESTful service has been implemented with certificate pinning and user authentication. Extra functionalities were added to the application to demonstrate different possible mobile security mechanisms at the application level.**

1. INTRODUCTION

The concept of network attacks is not new and as old a method as the Man-In-The-Middle attack may seem, it is still a persistent and common technique used by network hackers, even on secure networks “protected" by SSL. The prominence of SSL MITM attacks is what prompts techniques like Certificate Pinning to be a topic of in-depth research for network security analyzers. Authentication of users and the pinning of website certificates is one of the more secure techniques to maintain mobile security. However, it is not entirely foolproof. There are possible ways to bypass it using certain debugging tools in different platforms. Certificate pinning basically involves strong programming knickknacks that are embedded in source code that authenticates network entities be it clients or servers via web browsers or mobile services and apps. In our project, we pin the certificate for a certain website so that access to it is secured and limited only to a legitimate or permitted user or set of users. The idea is to have a secure mobile environment that keeps track of every activity in connection to the app and the services it connects to. This report describes every aspect of the implementation of the project – the client and server side designs at the front and back ends, the workflow, interfaces and programming mechanisms used, and the architecture. All of these parameters are discussed at every level of relevance.

1. **PROJECT DESCRIPTION**

The project comprises of the main certificate pinning activity that tests different URLs for certificate pinning:

1. The website we created for testing certificate pinning: <https://sadp-web.azurewebsites.net/>
2. The Gmail site
3. A custom URL

The URL whose certificate is pinned is the one that was created for that purpose. The other URLs would just show a toast that mentions the absence of a peer certificate.

In case of user authentication, every device that a user uses for registration is entitled to one authentication session. On registering to a particular device, user credentials are generated along with tokens to register and authenticate that user associated with that device on which the app has been installed. In this way, every device is registered on the server-side database as a member entry, authenticated with its distinct credentials that cannot be reused. Thus, user identities allocated at the time of registration can be used to make sure that no one can spoof anyone’s identity and thus, effectively prevent any attempt to perform an MITM attack, since each user can be registered once and only once on a single device. This mechanism protects user identities and strengthens authentication.

Furthermore, user credentials come with distinct member IDs and names which allow entries for each user only once. Thus, users have the capability to edit other credentials apart from the distinguishing ones i.e. the member ID and the name.

The basic functionality of the project is the implementation of the certificate pinning to avoid a possible MITM attack and secure the mobile app environment. The other functionalities in the project include the implementation of a RESTful web service that enhances the certificate pinning ability as well as the user authentication methodology. This has been integrated with the Azure cloud service that also maintains the Azure SQL Database which is responsible for storing and keeping track of user credentials.

The technique used for certificate pinning is the basic procedure of obtaining the certificate for the website created which can be done by exporting it via the web browser. Certificate import was done and stored in the keystore created using the *keytool* command. This was implemented using the BouncyCastle provider. Certificates obtained were in the Base64 encoded X.509 format. The database used to store user credentials was created with an Azure account as an SQL database. It helps keep track of the entries for membership registration.

1. **PROJECT REQUIREMENTS**

A variety of tools are required for the process of building this project, ranging from simple software and IDEs to complex cloud and mobile technologies coupled with security mechanisms.

1. ***Debug and execution requirements:*** The project requires a few other intricate details for it to function well. For instance, while executing it, a few points must be kept in mind:

* The project targets Android 4.4.2 (KitKat) which is API Level 19, the latest. However, the minimum SDK version is API Level 18. This must be remembered before debugging on an emulator or a device.
* The manifest file (AndroidManifest.xml) includes Android Manifest permissions that are required for the app to function:

*android.permission.INTERNET*

*android.permission.ACCESS\_COARSE\_LOCATION*

1. ***Eclipse:*** Since the mobile app environment is Android, we use the common Android mobile application development tool, the Eclipse IDE which works with Java. The programming part involves coding and playing with Java methods and classes in collaboration with Android activities and fragments that comprise the major part of the UI and the project as a whole. This is the front end or client side design.
2. ***Visual Studio Professional 2012:*** The back end server-side design is done in Visual Studio which takes care of the database and the language used here is obviously C#. The .NET Framework is utilized as the underlying infrastructure of the project and ASP.NET technologies have been put to use. The IDE requires a lot of disk space and works only for Windows operating systems.
3. ***Microsoft Windows Azure:*** The accessibility and connectivity to the cloud are done with the help of Microsoft’s Azure cloud services, which also provides the SQL Database that is highly secured and accessible only to the owner of the mobile app, who controls the membership tokens and entries from the server side using Visual Studio. The Azure cloud enhances the implementation of RESTful web services with respect to the mobile application. A Microsoft Azure account is required to be able to create and access the cloud SQL database. However, this is not much trouble as cloud services work on the Pay-As-You-Go principle. The process of creating and accessing the Azure database and mobile services, as well as the RESTful web services is not complex. [14]
4. ***Android Device/Emulator:*** A couple of Android devices were used during the testing of the app – a Google Nexus 7 and an emulator via the Eclipse IDE. Once a device was used to register a user, it was exclusively only for that user since unique identities were assigned to that user. This supports the concept of being able to register on a single device once and only once, enhancing the security mechanism that the app aims to demonstrate. The emulator on Eclipse is created keeping the target API Levels in mind, using the Android Virtual Device Manager. The app can also be run using an Android device by installing the .apk file on the device generated at runtime. This requires the app to be run at least once to produce the .apk file which gets stored in the /bin directory of the workspace folder.
5. ***Indigo Studio:*** The user interface of the app was created with storyboards and wireframes using the Indigo Studio Infragistics tool. This tool is helpful in ascertaining the layout of the app and designing what the UI would be like. Indigo Studio can be downloaded either as a 30-day trial version or purchased. It works on both Windows and well as Mac OS.
6. **MOBILE UI DESIGN PRINCIPLES**
7. ***Storyboard:*** The storyboard has been designed using the Indigo Studio storyboarding and wireframe tool by Infragistics. The user interface is a simple one, consisting of a single main activity with activity fragments. The storyboard can be illustrated with the following screenshot taken from Indigo Studio:

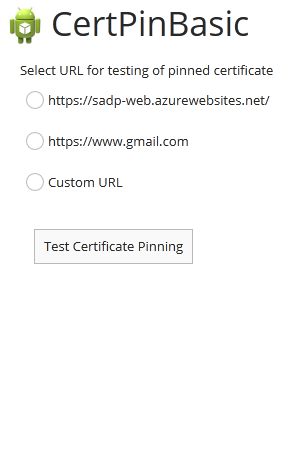


**Fig.1. Storyboard of the Certificate Pinning Android app**

The main activity of the application is basically the screen where the certificate pinning is tested for various URLs including the one whose certificate is pinned to the app. There are basically three options laid out as activity fragments:

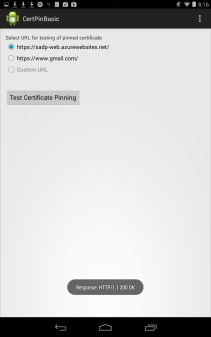
* Certificate Pin Test
* Register
* Update

1. ***Wireframes:*** Indigo Studio is used again for designing the wireframes of the app i.e. the difference UI screen designs. It is not too fancy but serves the purpose of demonstrating the main concept of the app. The various wireframe screens are illustrated as follows:
2. ***Test Cert Pin:*** This screen basically tests the given URLs to see if their certificates have been pinned to the app or not.



**Fig.2. Test Certificate Pinning Screen**

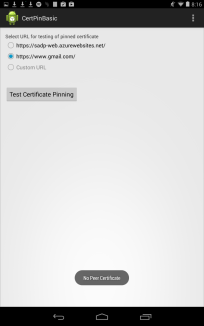
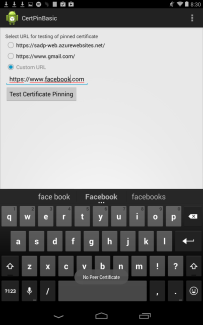
The different functionalities of this screen UI can be shown using the following screenshots.



**Fig.3. Testing the website whose certificate has been pinned.**

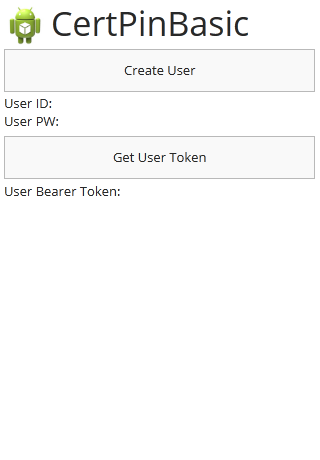
In the above screenshot, the given website that we created for the sole purpose of demonstrating certificate pinning, <https://sadp-web.azurewebsites.net/> is itself tested to make sure that its certificate is pinned to the app. The result of the test is returned via toast at the bottom of the screen.

The following screenshots demonstrate failure in testing the Gmail and custom websites for certificate pinning since their URL certificates have not been pinned to the app. As an example, the Facebook site is used as a custom URL.

**Fig.4. Testing the Gmail and Facebook URLs for certificate pinning.**

1. ***Register:*** In this screen, the details for every distinct user are registered and a unique member ID and name are assigned to every new user. The UI screen can be shown as a wireframe:



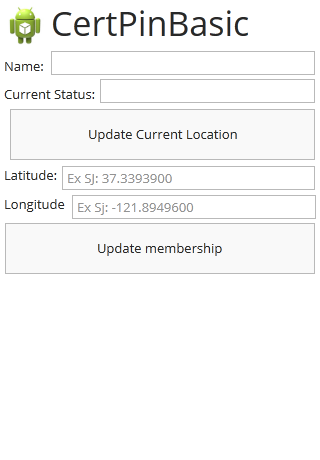
**Fig.5. Creating a new user, assigning user IDs and passwords, generating user tokens.**

The screenshot for the same UI screen is as follows:



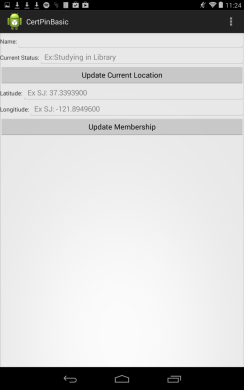
**Fig.6. Screenshot of Register Fragment**

1. ***Update:*** This part of the UI allows a registered user to change and update membership credentials like current status, current location etc. However, the name may not be updated since every name is associated with a unique member ID and can be assigned only once. This is the part of the security framework in ensuring integrity within the app. The wireframe for this screen can be shown as follows:

****

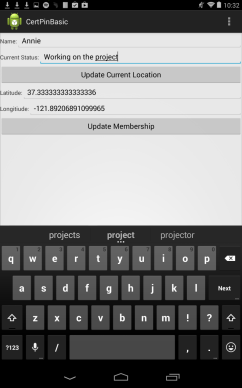
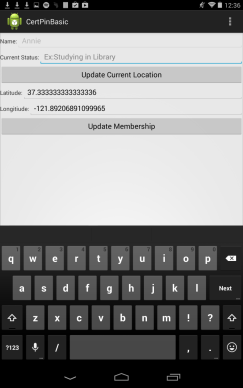
**Fig.7. Wireframe for Updating User Credentials**

The actual screenshots are shown below:



**Fig.8. Screenshot for Updating User Credentials (Before Updating)**

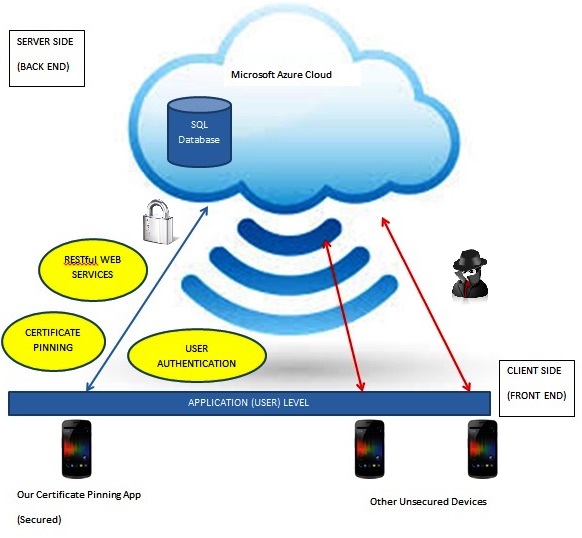
The screenshots below give an idea of how names can be entered into the app database just once:

**Fig.9. Screenshot for Updating User Credentials (During & After Updating)**

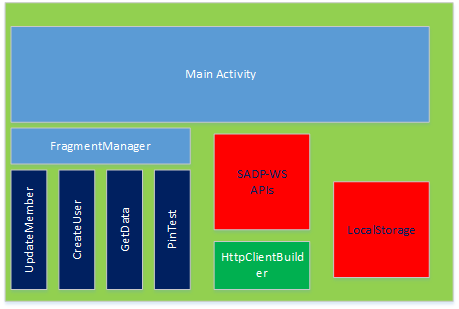
1. **HIGH-LEVEL ARCHITECTURE DESIGN**

The underlying architectural framework behind the project, its concepts and execution mechanisms can be summed up in the following diagram. It shows the interaction between the server side and the client side design. The services offered by the Azure cloud including the SQL database, the RESTful web services and user authentication functionality are all part of the secure mobile app environment established in the app that mainly uses certificate pinning. The figure shows the distinctness in the security of the app in comparison to other apps on other devices for other users who don’t implement it, and their vulnerability to MITM attacks over the network.

****

**Fig.10. High Level Architecture Design**

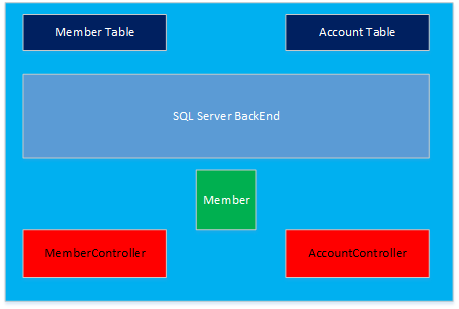
1. **COMPONENT LEVEL DESIGN**
2. ***Android Component Level:***

****

**Fig.11. Android Component Level Diagram**

* ***MainActivity:*** The activity that manages the fragments that call the read, create, and update functions using the SADP-WS APIs.
* ***HttpClientBuilder:*** Used for every web request to build an HttpClient that uses a pinned certificate.
* ***SADP-WS APIs:*** Component that implements the business logic for using the SADP-WS in Android.
* ***LocalStorage:*** Responsible for storing user ID, user PW, and authentication token.

1. ***Server Component Level:***

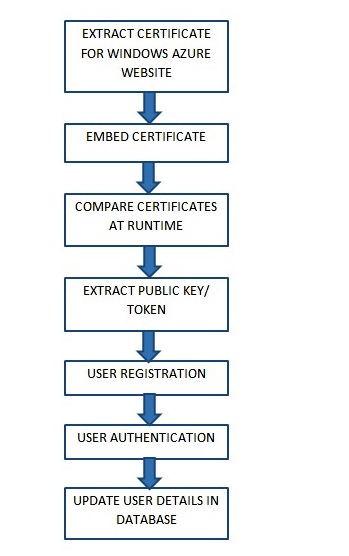


**Fig.12. Server Component Diagram**

* ***AccountController:*** The AccountController is the auto-generated class that provides account creation and login services for the SADP-WS. When a new account is created its added to the Account Table which is a part of the SQL server backend.
* ***Account Table:*** The table holds a record of all the user that have registered with the SADP-WS. For the project each user is actually represented by an ID/PW pair that is generated by the Android device’s unique ID.
* ***Member:*** The class defines the data model for the Member Table. Read, update, and create functions all handle objects of type Member.
* ***MemberController:*** The member controller is the implementation of the read, update, and create methods of the SADP-WS. Each method of the MemberController implements a database interaction with the Member Table that is stored in the SQL Server Backend.
* ***Member Table:*** This table contains the Member data that is created by updates from the SADP-App. The main service that the SADP-WS provides is the interaction with this SQL table.

1. **SEQUENCE OR WORKFLOW**

The sequence of execution steps followed in the project to demonstrate the concept of certificate pinning and securing the app with a mobile security infrastructure can be shown in a flow chart that illustrates the flow of control of events. The process of obtaining the certificate chain can be shown with the SSL flow diagram with web service authentication and the authentication bearer token generation.



**Fig.13. SSL Flow Diagram with Web Service Authentication**

The actual process of obtaining the certificate chain requires a few steps. First of all, the order of the SSL Certificate Chain should be as follows:

<your certificate>

<your cert signer>

<signer for your cert signer>

<etc>

The certificate for the Azure website is obtained using a web browser in the X.509 format. This certificate is embedded when building the app.

A keystore needs to be created to be used in the app. This is done using the *keytool* command line tool and the Bouncy Castle Provider to import the certificate to the keystore.

keytool -importcert -v -trustcacerts -file "azurecert.crt" -alias ca -keystore "sadp-keystore.bks" -provider org.bouncycastle.jce.provider.BouncyCastleProvider -providerpath "bcprov-jdk16-145.jar" -storetype BKS -storepass sadpsecret

For the verification of correct import of the certificate to the keystore, the following command is used:

keytool -list -keystore "sadp-keystore.bks" -provider org.bouncycastle.jce.provider.BouncyCastleProvider -providerpath "bcprov-jdk16-145.jar" -storetype BKS -storepass sadpsecret [3][4]

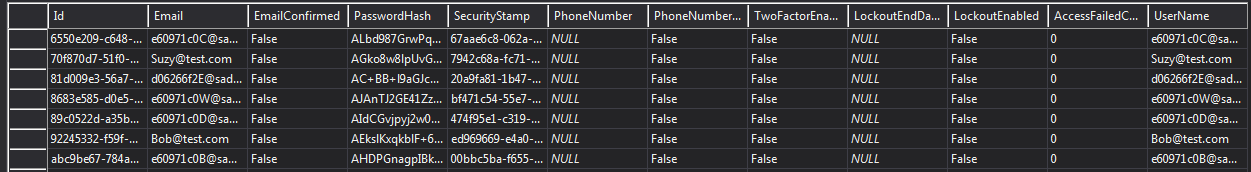
***Setting up website secure database access:*** This is done by setting up the development environment using the Windows Azure SDK for .NET (Visual Studio). This helps set up the Azure environment for setting up the Azure website for pinning the certificate to, and the Azure SQL Website. The back end was built using an ASP.NET MVC5 application that controls the database and the website created. [6]

Next, creating the user and user tokens for authentication are also done using the ASP.NET MVC5 driven backend as the Web API. Authentication is done using the bearer token and transmitted with SSL. The working of it is done by sending HTTP Requests to the API. All these mechanisms enable the registration and authentication of users as valid members. Bearer tokens are issued once authentication has been performed and confirmed. Authorized requests can also be sent as part of the update option in the app. [8]

1. **MOBILE & CLOUD TECHNOLOGIES USED & DESCRIPTIONS**
2. ***Android – HTTP Web Client:*** The Android platform provides the DefaultHttpClient which includes two HTTP clients: HttpURLConnection and the Apache HTTP Client. Both these clients support HTTPS along with streaming data, timeouts that can be configured and pooling of connections, including IPv6. The Apache HTTP Client is a better option for Android v2.2 (Froyo) and earlier, considering its nearly bug-free characteristic, while the HttpURLConnection serves a better purpose for the Android versions starting Android 2.3(Gingerbread) and higher. Network resource usage and battery power consumption are highly optimized, coupled with greater speeds. The HTTP Web Client for Android provides an excellent API. [26]
3. ***Azure Website:*** The Microsoft Windows Azure service is the cloud service that implements the RESTful web services to the mobile app while also providing the essential components required for the project including the Azure Website whose certificate was extracted and pinned to the app, and also the SQL database server that is used to store the database entries for the purpose of demonstrating user registration and authentication.
4. ***ASP.NET:*** This is the basic component of the server-side design that provides the underlying the framework via .NET at the back end. ASP.NET provides the MVC5 application for building the Azure website. The database created in the Azure cloud and the website for the certificate pinning demonstration, are both managed using the ASP.NET MVC5 application, which administers and controls events to the database including the user registration, bearer token generation and authentication processes. The access to the Azure website is authorized using ASP.NET operated from the back end of the project. All security and administration parameters are all maintained using ASP.NET. [6]
5. ***BouncyCastle:*** This is the tool used for creating the keystore and importing the certificate of the Azure website obtained, to the created keystore that is secured with a password. The BouncyCastle provider is basically used for encryption and includes a bunch of APIs that provide algorithms for cryptography. It also includes the Java Cryptography Extension (JCE) that is mostly responsible for dealing with the keys and the keystore entries, the certificates imported and stored in the keystore. [27]
6. **INTERFACES – RESTFUL AND SERVER SIDE DESIGN**

The web service for the project was implemented using Microsoft Azure. There are many tools available from Azure that allow developers to quickly create a web service and use it on mobile platforms. The SADP-WS could have been implemented using Azure’s cloud services, but the approach used in the project provided a more hands on learning experience.

1. ***Server Side Design:*** The server running the SADP-WS is hosted on Azure’s web-site services which provide the SADP project with many benefits including: free hosting, SQL server, and user authentication. The development of the server design was accelerated by the fantastic tools provided by Microsoft Visual Studio 2013.
2. ***SQL Database:*** The SQL database (SQL-DB)contains several tables and the main data storage table is discussed more in Section XIII: Design Patterns. The SQL-DB provides several features for the SADP-WS that add value to the web service.
3. ***User Authentication:*** The SADP-WS provides user authentication by allow app users to register with the service. The SQL-DB stores user credentials in a table called dbo.aspNetUsers.



**Fig.14. Table View of dbo.aspNetUsers from SQL Server Object Explorer**

The creation of new users is handled by the AccountController.cs class which was defined by Visual Studio 2013 during project creation. The automated creation of the AccountController.cs allowed for rapid development by the SADP team without having to risk mistakes, and security flaws, by developing their own account management methods. Several restrictions were placed upon account creation for the purpose of simplifying the implementation of the server side code. These restrictions and their reasoning are listed in the following table.

|  |  |
| --- | --- |
| **Restriction** | **Reasoning** |
| Each app instance generates its own user credentials based on the unique device ID | Eliminate coding required to validate user input based on security policies enforced by the SADP-WS |
| Once the credentials are created, they can’t be changed by the user | Reinstallation of the app on the same device will use already created user credentials. Eliminates the need for server side methods to support altering credentials |
| Users can’t login with same credentials across multiple devices | Simplifies the management of users by treating each device as a user |

**Table 1. Authentication Restrictions and Reasoning**

Microsoft Azure, Visual Studio, and ASP.net provide support for authentication using third party identy providers like Google. [17] Using these capabilities the SADP-WS could support login across multiple devices easily but the ability was not required to meet the intitial goals of the project. Further discussions of improvements to the SADP-WS are discussed in Section XIV: Future Work.

1. ***Token Requirement:***  Access to SADP-WS services can be controlled by using access attributes like [Authorize] and [AllowAnonymous]. By preceding method calls with these attributes, the access to the methods can, respectively, be restricted to only authorized users or first time users of the service who haven’t registered yet. Further discussion of these attributes takes place in Section XIII. A client is determined to be authorized by providing user credentials to the SADP-WS and receiving an authorization token in return. Once these tokens are created, the tokens can be used by any client, not just the client that provided user authentication credentials[ACCT\_API]. To avoid these tokens from being hijacked the transmission channel needs to be protected.
2. ***HTTPS Request Only:*** In order to protect the transmission channel, the SADP-WS is forced to communicate only using HTTPS.  The HTTPS protocol may be forced by adding HTTPS to the filters found in the FilterConfig.cs file in the SADP-WS asp.net project. The following shows what that looks like:

public static void RegisterGlobalFilters(GlobalFilterCollection filters)

        {

            filters.Add(new RequireHttpsAttribute());

…………………………………………....

By adding this filter to the project, all communications to the SADP-WS will be protected by the HTTPS protocol. If a request is made to the service using HTTP, the request will be ignore by the service. Now the transmission of the authentication token, user credentials, and member data is encrypted between the SADP-App and the SADP-WS.

1. ***RESTful Service Design:*** The SADP-WS is designed as a RESTful service. The design of a RESTful service uses standard HTTP protocol to create an easy to user web service interface that uses “uses a handful of HTTP’s verb methods (GET, POST, and so forth) plus a few simple concepts such as URIs and headers.” [18] HTTP is supported across all major computing platforms and allows mobile devices to take advantage of these easy to use web services. The design of the SADP-WS and its RESTful interface was assisted by Visual Studio 2013.
2. ***URIs and Headers:*** The implementation of RESTful methods relies on HTTP’s verb methods such as POST, GET, and PUT. With these three methods the SADP-WS is fully implemented and is able to create user credentials, retrieve user authentication tokens, read and manipulate the SQL-DB. In order to achieve these functions requires the usage of URIs and headers. The URIs are HTTP addresses that correspond to application components running on the SADP-WS’s server. The headers in the HTTP request describe how the SADP-WS should treat the incoming HTTP request. An example header is “Content-type” which can be set to “application/json” which tells the SADP-WS to treat the entity body of the HTTP request as JSON formatted data.
3. ***Http Entity Body:*** The HTTP entity body carries the data between the mobile app and the web service. The data entity is determined by the headers defined in the request and the type of request being made. Data may be returned from the SADP-WS to communicate important data from the SQL-DB.
4. ***Table of Methods:*** The table below shows the format of the RESTful web requests that were used to implement the web service calls in Android.

|  |  |
| --- | --- |
| **Action** | **Create User** |
| HTTP Method | POST |
| URI | https://sadp-web.azurewebsites.net/api/Account/Register |
| Headers | Content-Type: application/json  charset=utf-8 |
| Request Entity Body | { Email : [userId], Password: [userPw], ConfirmPassword : [userPw] } |
| Return Code Success | HTTP/1.1 200 OK  Response Entity Body =empty |
| Return Code Failure | HTTP/1.1 400 Bad Request  Response Entity Body= JSON Formatted Error Messagein ModelState object |

**Table 2. Create User Method**

Several of these API methods return a Bad Request code when there is an internal error. Reading the contents of the ModelState object in the JSONObj the cause of the error can be determine. It can be caused by a password or username not meeting the account creation policies or the account already existing. IN the case of the account already existing the application must attempt to use the user credentials that it created for itself from its device ID. It is likely that when a user already exists in the system it was created by a previous installation of the mobile application.

The Get Token function gets a bearer authentication token from the web service. The type of token returned is a bearer token which grants access to the restricted resources on the SADP-WS. All of the remaining API calls in the SADP-WS API have restricted access and require this token in order for the SADP-App to use the service properly. A token is created for users that authenticate using the user credentials that have been created by the mobile application previously; however, the token may be used by any client once it has been created. [8] A bearer token needs to be handled with care and not transmitted unencrypted.

|  |  |
| --- | --- |
| **Action** | **Get Token** |
| HTTP Method | POST |
| URI | https://sadp-web.azurewebsites.net/Token |
| Headers | Content-Type: application/x-www-form-urlencoded  charset=utf-8 |
| Request Entity Body | grant\_type=password&username=[userId]&password=[userPw] |
| Return Code Success | HTTP/1.1 200 OK  Response Entity Body = JSON access\_token |
| Return Code Failure | HTTP/1.1 400 Bad Request  Response Entity Body= JSON Formatted Error Messagein ModelState object |

**Table 3. Get Token**

The final two methods are Update and Add member functions which show the beautiful simplicity of a RESTful API. Both access the same base URI (<https://sadp-web.azurewebsites.net/odata/Members> ) but the Update method distinguishes itself with the addition of the MemberId value at the end of the URI and inside the Entity Body. The two methods are distinguished by their HTTP method verbs which describe their function, POST adds an item to the database, and PUT updates data already in the database.

|  |  |
| --- | --- |
| **Action** | **Add Member** |
| HTTP Method | POST |
| URI | https://sadp-web.azurewebsites.net/odata/Members |
| Headers | Authorization: Bearer [Token]  Content-Type: application/json  charset=utf-8 |
| Request Entity Body | JSON Format {Name:[name], LocLongCoord:[userLong], LocLatCoord:[userLat],Status[userStatus]} |
| Return Code Success | HTTP/1.1 201 Created  Response Entity Body = JSONObj of Request Entity Body with MemberId set. MUST BE REMEMBERED |

**Table 4. Add Member Method**

|  |  |
| --- | --- |
| **Action** | **Update Member** |
| HTTP Method | PUT |
| URI | <https://sadp-web.azurewebsites.net/odata/Members(MemberID)>  Must be a valid MemberId retrieved from the Add Member request |
| Headers | Authorization: Bearer [Token]  Content-Type: application/json  charset=utf-8 |
| Request Entity Body | JSON Format { MemberID : [(Must match URI)], ...any fields that need to be updated } |
| Return Code Success | HTTP/1.1 204 No Content  Response Entity Body = Empty |

**Table 5. Update Member Method**

1. **CLIENT SIDE DESIGN**

The client side in the SADP project is an Android mobile application. The primary goal of the SADP-App is to implement certificate pinning which was the first component implemented by the SADP team.

1. ***HTTP Client with Pinned Certificate:*** The workhorse of network security relies upon private/public key pairs. Through the magic of cryptography a client may encrypt messages using a server’s public key that only the server will be able to decrypt since the server alone holds the private key. However, there remains a problem of trust. There is no easy way for a client to be sure that a particular public key belongs to who she thinks it belongs too. Certificate authorities (CA) came to the rescue by signing keys and giving a guarantee that these keys were to be trusted. But problems still remain with Man in the Middle (MITM) attacks where an attack presents a fake certificate to a client and is able to insert himself between a conversation between two parties. [19]

A solution to the problem is using certificate pinning. The process involves embedded the certificate of a server into the application thus the application always knows what key to trust. [19] The downside is the application needs to be update every time the certificate changes for the server that the application connects too. [19]

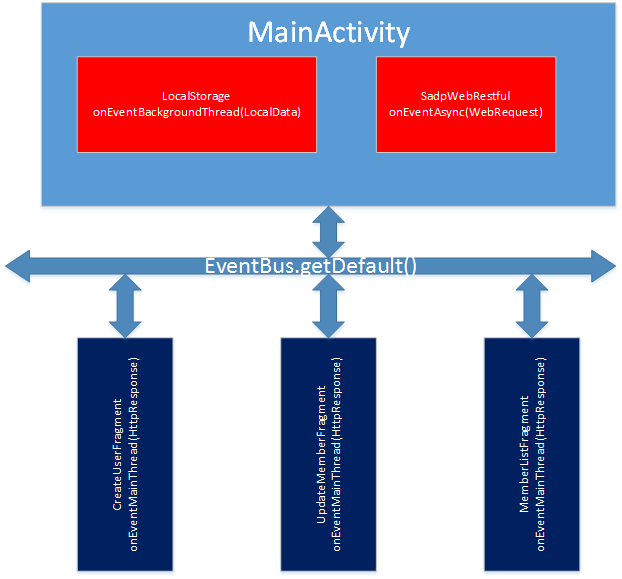
The SADP team implemented certificate pinning by using an open source HttpBuilder which embeds the pinned certificate in every instance it creates of the DefaultHttpClient. The certificate for the SDAP-WS website was retrieved and embedded into the SADP-App using Bouncy Castle API’s and its .bks format [INFIN]. The beauty of using the HttpBuilder class is that everywhere that the SADP team needed to call a web service, certificate pinning was already implemented. It was a simple step that increased the security of the SADP-App against MITM attacks.

1. ***EventBus and Avoiding Intents:*** An inherent danger in Android development is the power of using Intents. A careless developer can easily get into trouble since Intents are used all over Android to launch activities, start services, and communicate between activities. Implicit intents especially are dangerous if they are used without consideration that any app on the device can respond to the Intent if its defined poorly. Intents can often be used safely if they are made explicit, but sometimes they feel more like a nuisance than a feature.The SADP team decided to use the EventBus library to simplify the design of the SADP-App. Intents are still a necessary part of Android for accessing services, and apps outside of an application, but the EventBus library is too convenient to use anything else for application control. The EventBus library implements an event messaging system for Android applications. Developers can easily publish and subscribe to the EventBus which can be used to easily command events on Android threads. EventBus handlers are defined simply by the convention onEvent\*(Object obj). The table below describes the types of event handlers.

|  |  |
| --- | --- |
| **Handler Thread Type\*** | **Description** |
| PostThread | Event runs on the same thread as the calling thread |
| MainThread | Event runs on the UI thread |
| BackGroundThread | If posting thread isn’t main thread, event runs in posting thread else event runs in a backgroundthread |
| Async | Events are run in a new created thread, similar to AsyncTask |

**Table 6. EventBus Handler Types**

By using the EventBus the SADP-App could be implemented with only fragments and one activity. Larger apps may require many more Activities but the tedious task of passing data between activities may be handled with the EventBus instead of Intents. The following diagram shows the general flow of the EventBus in the SADP-App project.



**Fig.15. EventBus Flow Diagram**

Each fragment registers and unregisters from the EventBus in their onResume() and onStop() methods respectively. Each fragment has a handler for an HttpResponse, but by following the steps mentioned previously, only the current/viewable fragment will have its handler registered with the EventBus. The EventBus has a default bus which is used exclusively in the SADP-App with the following examples:

EventBus.getDefault().register(context)

EventBus.getDefault().unregister(context)

EventBus.getDefault().post(obj)

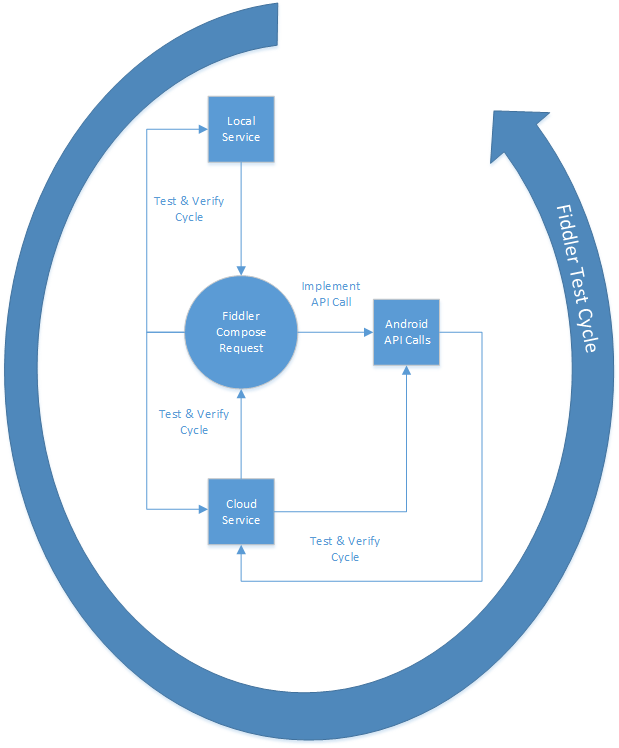
Each fragment posts events that are handled in the LocalStorage, and SadpWebRestful classes. Since these are businesss logic classes, in order to have their event handlers available for the lifetime of the application they are instantiated in the onCreate method of the MainActivity. The constructors of these classes register with the EventBus and their event handlers are available for posting. The EventBus eliminated a lot of boilerplate code that would have been dedicated to handling Intents and allowed the SADP team to focus on the interesting aspects of coding.

1. **TESTING (UI OR STRESS TEST)**

Testing of the SADP project involved two phases. Testing the SADP-WS to ensure the API behaved as expected and the implementation of the SADP-WS calls in the SADP-App.

1. ***RESTful Service Testing with Fiddler:*** The SADP-WS was the first part of the project to be completed and it involved the most tedious testing phase. The process of testing the web service involved cycles of experimentation and verification. The use of the Fiddler 2 program was an invaluable asset, and was essential for the SADP team’s understanding of RESTful web services. Fiddler 2 is a web debugging tool which allows testers to compose custom HTTP requests and view the HTTP responses [ACCT\_HTTP]. The use of Fiddler 2 led to the development of the tables found in Section IX. The process required a lot of trial and error by composing HttpRequest and testing the HttpResponse. Once a HttpRequest was working with a Local Service, it was tested against the Cloud Service, and then as a API call from an Android application. Each HttpRequest was developed in this way and then integrated as a API call inside an Android application.

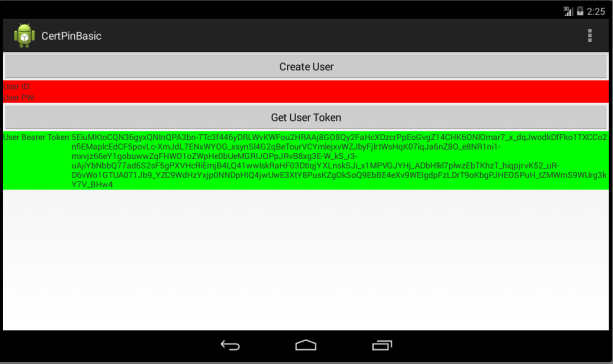
The following figure illustrates the process of testing with Fiddler.



**Fig.16. Fiddler 2 Testing Process**

1. ***UI Testing:*** UI Testing occured as the SADP team played with the SADP-App. Through testing problems arose and exposed incorrect assumptions made during the development process.

***SADP-App Reinstallation:*** A decision was made that users of the SADP-App would not be able to change their login credentials with the SADP-WS. Since the credentials are created by a devices unique ID, any attempt to create an ID that already exists would create a Bad Request response from the SADP-WS. However, the assumption was that this Bad Request response invalidated the generated UserID and UserPW. But if the SADP-App is reinstalled on a device it should be able to authenticate properly with the SADP-WS. The difference is show in the screen captures shown below.



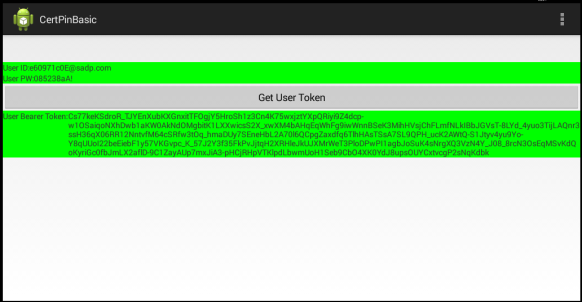
**Fig.17. Screen Captures of Error**

The logic behind the creation of a new user now temporarily lists the UserID/PW when a bad request is returned to the client. The next screen captures shows this state.



**Fig.18. Screen Capture Unconfirmed User ID/PW**

Attempting to get the user token will determine if the UserID/PW are valid user credentials. If the Token is returned successfully then that means that the app has been reinstalled on the device, and the user credentials already exist in the web service from the previous installation. The screen capture below shows the final state of the application.



**Fig.19. Screen Capture User Token Confirms Previous User Credentials**

1. **AUTOMATION**

The automated testing was used for two cases: update, and user creation. Test cases for the retrieving the database entries was left out due to time constraints. However, the simplicity of the initial project design means that robust test cases for the user interface isn’t necessary yet. When future version of the project have more features, more test cases will be required.

The automated tests are implemented using Robotium which simulates simple user input by analyzing the current context of the application under test. For example referencing an EditText object is as simple as using the order it appears in the current context’s layout to reference the object. The first EditText object is 0, the next one is 1, and so on in the order they appear in the layout. The following code is a sample from the update test case.

// Ensure the Activityhas time to load

assertTrue(solo.waitForActivity

(cmpe277.sadp.certpinbasic.CertPinMainActivity.class, 10000));

// Ensure that the test is on the expected Activity

solo.assertCurrentActivity("Wrong Activity", CertPinMainActivity.class);

// Navigate to Update Page

solo.clickOnMenuItem(solo.getString( cmpe277.sadp.certpinbasic.R.string.action\_update));

// Click the Update Location Button

solo.clickOnButton(solo.getString(cmpe277.sadp.certpinbasic.R.string.Up\_Frag\_Current\_Loc\_txt));

The first segment of code navigates from the initial page of the application to the update page by navigating through the options menu. After the test arrives on the update page, the test is directed to click on the location update button.

// Get EditText for name

EditText nameBox = solo.getEditText(0);

// Check if its enabled, if so, enter a name

if(nameBox.isEnabled()) {

nameBox.setText("Robotium");

}

// Update status, currentTime should ensure that each status is diferent when tests are run

solo.enterText(1,"Robot Hell" + System.currentTimeMillis());

This segment of code checks to see if the name has already been assigned in a previous update request. If the EditText box for the name is disabled the test overlooks the name field and updates the status field.

solo.clickOnButton(solo.getString(cmpe277.sadp.certpinbasic.R.string.Up\_Frag\_Update\_Member\_txt));

Finally the test is directed to update the member in the cloud. Verification of the change is noted in visual studio.

https://lh5.googleusercontent.com/dtJuMJdBB3BD7WsN6qRY6wfUrxmtmE8EvAqg8XKIHc53OVCAurdy7oYdUH03CCOmUEPTVEu9GlINVOMezJ86D13DWlBCLksd9HDq7WzC10RzNJM4fCL3kZLFcNX4f7N0tQ

**Fig.20. Visual Studio SQL Object View of test result**

1. **DESIGN PATTERNS USED**

The main application of design patterns in the SADP project were in the design of the SADP-WS. Microsoft’s Visual Studio 2013 provides many helpful tools for getting started quickly on a project. The two most useful patterns used in the project are discussed in this section.

1. ***ASP.NET MVC5:*** MVC stands for Model View Controller and the ASP.net MVC5 is a web application framework that makes creating web applications easy. In general the Model in MVC stands for the data represented at its lowest level in an application. The View is the presentation of the data Model and any UI controls for the user to interact with. The Controller is the glue that manages the control flow of an application and the interactions between the View and a underlying data Model. The project created by VIsual Studio 2013 supported user authentication and account creation right after the first initialization by creating an AccountController class. There are many interesting faces of the AccountController to focus on but most intriguing are the attributes declared before the AccountrController class declaration:

[Authorize]

   [RoutePrefix("api/Account")]

   public class AccountController : ApiController

   {..............................................................

The RoutePrefix attribute sets the URI path that can be used to access the AccountController. As it is shown in Section IX in the Create User table the URI used for the api request has “api/Account/Register” as the path at the end of the URI. The AccountController is accesible from the SADP-WS or the browser and it can handle the requests from either by responding in a platform agnostic format. Additionally the [Authorize] attribute was generated by VS13. If this were the only access attribute declared in AccountController.cs no new user could be created without a user who already had access. The AccountControlller allows new user registration by applying the [AllowAnonymous] attribute to the Register method. [25] The other web service methods all use the [Authorize] attribute and will only allow users who’ve registered and received an access token to interact with the web service.

The MVC 5 framework is a powerful tool for developers and it is made even better by leveraging the Entity Framework which is discussed in the next section.

1. ***Entity Framework:*** The Entity Framework is a powerful Object Relational Mapping (ORM) that is part of the .NET framework that makes complex data access and services simple from the perspective of a developer. [21] The Entity Framework (EF) is what enables the SADP-WS to handle translation from a web request to storage in a backend  SQL-DB and also the code first development approach taken in the SADP project. The Code first development approach allows the developer to create a data model class and use it to generate a database. [22]

For the SADP project the Members.cs class was coded first and it defines the format of the data in the SADP-WS. Through VS13, the MVC 5 scaffolding feature was used to automatically generate code to perform (CRUD) actions: create, read, update, and delete[23]. The code generated by this method is in MembersController.cs which has attributes defined that make the URI[ <https://sadp-web.azurewebsites.net/odata/Members>] a valid URI for building the requests for the CRUD operations shown in the tables in Section IX.

1. **FUTURE WORK AND COMPARISON TO AZURE**

The SADP project has succesfully implemented user authentication and certificate pinning into the SADP-WS. However, there is still future work and improvements that can be made. It is also worth comparing the SADP-WS to Azures Web Service platform.

1. ***Client Authentication:*** The SADP currently has one major flaw that can be fixed with more research and a time investment by the SADP group. SADP-WS can be accessed by anyone who is able to generate the required API calls. It would be benefitial to only allow access to the web service to trusted clients. Facebook implements this feature for developers by using an App Secret which is used in the process to make app access tokens. [24] However, this app secret must be protected, and must be obscured in the application code. Methods of code obfuscation and construction a secret at runtime would need to be researched. Both methods would slow down efforts of attackers trying to extract the secret from the application.
2. ***Database Roles:*** The current implementation of the SADP-WS uses a SQL-DB that allows any of the registered and authenticated user to edit the database. For this projects application it isn’t a concern but larger projects with expanded features would need a way to grant control and or limit control to registered users. The way to implement this feature is to user Roles defined by the SADP-WS. Using the [Authorize(Roles = “canEdit”)] would only allow authenticated users with registered under that role to use the underlying method. [25]
3. ***Third Party Identity Provider:*** The SADP project would benefit from using an identity provider( Google, Yahoo, Twitter) by allowing user registration to be separated from a physical device. This would allow a user to login using the same credentials across multiple platforms.
4. **CONCLUSION**

The major purpose of this project is to eliminate MITM attacks. No matter how safe an environment the device is in, there is still a chance of it being vulnerable. All major business organization use smartphones, it is important that private transactions take place in a secure environment. And not everyone can afford to have security at the transport layer, there needs to be some sort of security at the application level. That purpose is served by this app. Not only does it show a way to guarantee the client’s data integrity, but also the user authentication for the server. Using all existing protocols and infrastructure in a hardened manner, the app ensures complete safety from an MITM attack.

**REFERENCES**

[1] Kerns, S. (2013, April 1). Certificate Pinning in a Mobile Application. Retrieved April 30, 2014, from www.netspi.com: https://www.netspi.com/blog/entryid/204/certificate-pinning-in-a-mobile-application

[2] Marlinspike, M. (2011, December 5). Your app shouldn't suffer SSL's problems. Retrieved April 30, 2014, from www.thoughtcrime.org: http://www.thoughtcrime.org/blog/authenticity-is-broken-in-ssl-but-your-app-ha/

[3] Edgecloud, Inc. (2012, Marxh 8). SSL Certificate Chain order matters. Retrieved April 30, 2014, from blog.edgecloud.com: http://blog.edgecloud.com/post/19519955133/ssl-certificate-chain-order-matters

[4] Hauck, A. (2010, October 22). Android: Trusting SSL Certificates. Retrieved April 25, 2014, from blog.antoine.li: http://blog.antoine.li/2010/10/22/android-trusting-ssl-certificates/

[5] www.github.com. (2014, February). Retrieved April 30, 2014, from Example of certificate pinning on Android: https://github.com/ikust/hello-pinnedcerts

[6] Microsoft, Inc. (n.d.). Deploying a Secure ASP.NET MVC 5 app with Membership, OAuth, and SQL database to an Azure Web Site. Retrieved April 29, 2014, from azure.microsoft.com: http://azure.microsoft.com/en-us/documentation/articles/web-sites-dotnet-deploy-aspnet-mvc-app-membership-oauth-sql-database/

[7] Anderson, R., & Reitan, E. (2014, April 3). Code! MVC 5 App with Facebook, Twitter, LinkedIn, Google OAuth Sign-on (C#). Retrieved April 29, 2014, from www.asp.net: http://www.asp.net/mvc/tutorials/mvc-5/create-an-aspnet-mvc-5-app-with-facebook-and-google-oauth2-and-openid-sign-on

[8] Wasson, M. (2013, October 15). Individual Accounts in ASP.NET Web API. Retrieved April 30, 2014, from www.asp.net: http://www.asp.net/web-api/overview/security/individual-accounts-in-web-api

[9] McMurray, R. (2013, June 26). External Authentication Services (C#). Retrieved April 29, 2014, from www.asp.net: http://www.asp.net/web-api/overview/security/external-authentication-services

[10] How to add Web API to an existing ASP.NET MVC 4 Web Application project? (2012, August 17). Retrieved April 30, 2014, from www.stackoverflow.com: http://stackoverflow.com/questions/11990036/how-to-add-web-api-to-an-existing-asp-net-mvc-4-web-application-project

[11] System.Web.Http.Owin WebApi2 installation woes. (2014, February 20). Retrieved April 30, 2014, from www.stackoverflow.com: http://stackoverflow.com/questions/21906866/system-web-http-owin-webapi2-installation-woes

[12] CodePainters. (2009, December 11). Android: IMEI number and the emulator. Retrieved April 29, 2014, from codepainters.wordpress.com: http://codepainters.wordpress.com/2009/12/11/android-imei-number-and-the-emulator/

[13] Vogel, L. (2014, January 7). Usings lists in Android (ListView) - Tutorial. Retrieved May 1, 2014, from www.vogella.com: http://www.vogella.com/tutorials/AndroidListView/article.html

[14] Helmig, J. (2014, January 3). Integrate a REST API into Android application in less than 15 minutes. Retrieved April 29, 2014, from blog.strikeiron.com: http://blog.strikeiron.com/bid/73189/Integrate-a-REST-API-into-Android-Application-in-less-than-15-minutes

[15] Ensure web service only accessed by authorized applications. (2013, September 19). Retrieved April 30, 2014, from security.stackexchange.com: http://security.stackexchange.com/questions/42586/ensure-web-service-only-accessed-by-authorized-applications

[16] Vogel, L. (2013, December 6). Android user interface testing with Robotium - Tutorial. Retrieved May 1, 2014, from www.vogella.com: http://www.vogella.com/tutorials/Robotium/article.html

[17] Microsoft, Inc. (2011, April 7). How to: Configure Google as an Identity Provider. Retrieved April 30, 2014, from msdn.microsoft.com: http://msdn.microsoft.com/en-us/library/gg185976.aspx

[18] Microsoft, Inc. (2013, February 18). Build RESTful API's with ASP.NET Web API. Retrieved May 1, 2014, from www.asp.net: http://www.asp.net/web-api/tutorials/hands-on-labs/build-restful-apis-with-aspnet-web-api

[19] Walton, J., Steven, J., Manico, J., & Kevin, W. (n.d.). Certificate and Public Key Pinning. Retrieved April 27, 2014, from www.owasp.org: https://www.owasp.org/index.php/Certificate\_and\_Public\_Key\_Pinning

[20] Microsoft, Inc. (n.d.). ProfilePropertySettings.AllowAnonymous Property. Retrieved April 30, 2014, from msdn.microsoft.com: http://msdn.microsoft.com/en-us/library/system.web.configuration.profilepropertysettings.allowanonymous(v=vs.110).aspx

[21] Dykstra, T. (2014, April 23). Getting Started with Entity Framework 6 Code First using MVC 5. Retrieved April 30, 2014, from www.asp.net: http://www.asp.net/mvc/tutorials/getting-started-with-ef-using-mvc/creating-an-entity-framework-data-model-for-an-asp-net-mvc-application

[22] Microsoft, Inc. (n.d.). ASP.NET Data Access Options. Retrieved April 30, 2014, from msdn.microsoft.com: http://msdn.microsoft.com/en-us/library/ms178359.aspx#dbfmfcf

[23] Anderson, R., & Tom, D. (2014, March). REST service using ASP.NET Web API and SQL Database. Retrieved April 29, 2014, from azure.microsoft.com: <http://azure.microsoft.com/en-us/documentation/articles/web-sites-dotnet-rest-service-aspnet-api-sql-database/>

[24] Facebook. (n.d.). Login Security. Retrieved April 30, 2014, from developers.facebook.com: https://developers.facebook.com/docs/facebook-login/security/

[25] Anderson, R., & Dykstra, T. (2014, April 2). Deploy a Secure ASP.NET MVC 5 app with Membership, OAuth, and SQL database to an Azure Web Site. Retrieved April 30, 2014, from azure.microsoft.com: <http://azure.microsoft.com/en-us/documentation/articles/web-sites-dotnet-deploy-aspnet-mvc-app-membership-oauth-sql-database/>

[26] DefaultHttpClient | Android Developers. (n.d.). Retrieved April 25, 2014, from developers.android.com: <http://developer.android.com/reference/org/apache/http/impl/client/DefaultHttpClient.html>

[27] The Legion of the Bouncy Castle. (2013). Retrieved April 29, 2014, from www.bouncycastle.org: https://www.bouncycastle.org/java.html