A Holistic Approach for Securing In-app Purchase (IAP) Vulnerability in Mobile Applications

Yeh-chi Lai
California State Polytechnic University, Pomona
3801 West Temple Avenue
Pomona, California 91768
yehchilai@cpp.edu

Mohammad Husain
California State Polytechnic University, Pomona
3801 West Temple Avenue
Pomona, California 91768
mihusain@cpp.edu

ABSTRACT

Because of the successful mobile app society, there is a new business model that has appeared, the In-App Purchase (IAP). IAP allows users to purchase products, such as exclusive items, digital goods, and additional contents directly from a mobile app. From a developer's point of view, this is a lucrative opportunity, so there are more and more independent developers who have begun to participate in using this new business model. In the developer community, there has also been a huge change because of a popular game engine called Unity 3D Engine. On Unity, the developers need to write the code only once, and then can deploy their app on Android, iOS, and Windows mobile platform seamlessly. Because of this convenience, there are many new apps published every day and developers are quick to integrate IAP functionality. However, many app developers are not very familiar with security issues and lack the background knowledge or resources to protect their apps. In this paper, we first provide a detailed analysis of security implications of IAP vulnerabilities in mobile apps such as compromising an app, man-in-the-middle attack, and reverse engineering. To address the problem, we then developed a security plug-in for IAP functionality in app development engines, which can be easily integrated with existing development platforms for independent developers. We also provide a detailed demonstration of how the plug-in can prevent IAP attacks by providing confidentiality and integrity guarantee, securing network traffic as well as applying code obfuscation. The plug-in is open-source and available for the community.

CCS Concepts

•Security and privacy → Vulnerability management; Software security engineering; Web application security; Software reverse engineering;

Keywords

in-app purchase; IAP

ACSAC '16 December 5?9, 2016, Los Angeles, California, USA

DOI: 10.1145/1235

1. INTRODUCTION

In-App Purchase (IAP) is a purchasing process used in mobile apps. It provides different marketing strategies to developers, micro independent software vendors (MicroISV). Because it is difficult to have a high number of downloads at the beginning of app publication, developers usually publish free mobile applications with IAP purchasing process first. By using this strategy, they may obtain better downloads for their mobile application. On the other hand, users can download the mobile app for free, and use most of its functionality. If users want to get more or complete contents, users can use IAP to purchase extra content, such as exclusive items, new characters, virtual goods and so on. This purchasing process has benefits for both developers and users. However, there are more and more IAP attackers trying to exploit this purchasing process. The following will discuss two categories of how attackers exploit IAP. One is exploitation the operating system, and another is exploitation the mobile application itself.

First, the simplest way to attack IAP is to exploit the operating system, for example, root an Android [9] or jailbreak an iOS [4] device. The idea of this method is to obtain the root permission and compromise IAP purchasing process. It is difficult to prevent this exploitation because the authentication process of both operating systems needs to be redesigned.

The other method is exploiting IAP without getting the root permission [11]. There are two branches of this method based on where the data is saved on local devices or servers. If the IAP data is saved on the server, attackers can use network packet analyzers such as Paros, Charles, and Wireshark [15]. Attackers can identify IAP authentication packets to duplicate or modify the packets and send them to local devices. The local device will take these packets as an authenticated message, and then attackers can obtain new contents for free. In this case, the packets are not encrypted, so it is easy to be exploited. Another branch is using file management tools such as iTools, DiskAid, and iPhone Explorer to modify local IAP data. Attackers can use these tools to search a particular file which contains the data of purchased records and edit several code lines of this file. They also can replace the file by a compromised file. This IAP exploitation method is popular to attackers and users who are not willing to pay. When attackers exploit an application, other users can use the package which attackers provided to escape IAP process and get new content for free.

The last method is reverse engineering. Attackers can reverse developer's application by using some tools, such as

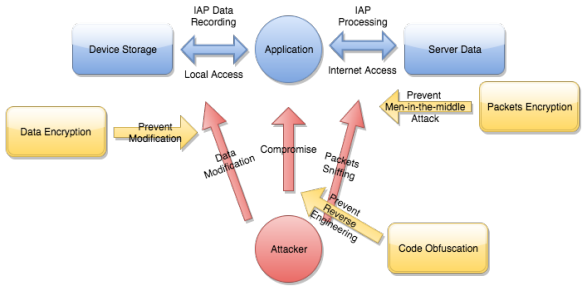


Figure 1: Potential solutions of attack scenarios

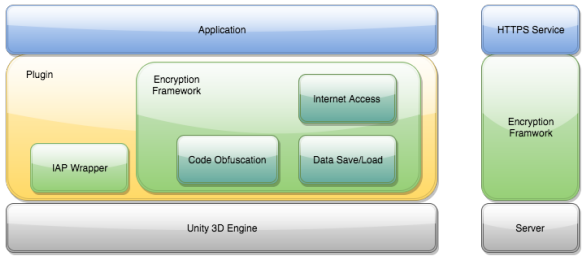


Figure 2: Overview of the plug-in

dex2jar [13] and JDI-GUI [3]. After attackers decompress the application, they analyze the source code and find out the part of the IAP process. The IAP process will be modified based on attackers' requirements. Then, attackers rebuild the application and can obtain the products for free. To prevent vulnerabilities appearing in each development stage, this paper will show several scenarios including device storage access, internet access, and application compromise and finally how to solve these vulnerabilities.

Figure 1 shows the overall method used to prevent vulnerabilities in different scenarios. The blue diagrams show normal mobile app working mechanism of IAP processing. The app can access data from local devices, and can communicate with a web service through the Internet. The red diagrams indicate the vulnerabilities that attackers will try to compromise. The yellow diagrams represent the method that we will utilize to prevent these attacks.

Finally, this paper will provide a general solution plug-in for developers to secure IAP processing in mobile application. In this plug-in, three parts will be included to help developers prevent attacks on their applications. This plug-in will provide secure data access on local devices, use a secure web server for transferring data, and implement code obfuscation.

2. BACKGROUND: UNITY 3D ENGINE

From the developer point of view, there are three tiers to demonstrate the process of developing an application by using Unity 3D Engine. The first one is the development tool provided by Unity 3D Engine. The second is third party plug-ins. Plug-ins have many functionalities, such as enhanced shader, animation controller, AI, and so on. The last tier is an application. The application can be a simple game or a chat app. We will focus on the second one and implement our plug-in for developers. Figure 3 illustrates the basic structure of the Unity 3D Engine. We will skip the

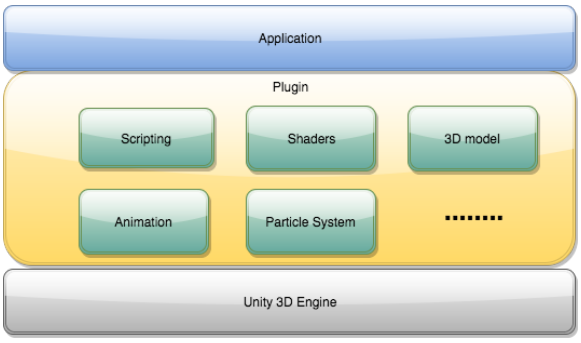


Figure 3: Unity 3D engine infrastructure (developer's point of view).

first and the third tier and only integrate a new mechanism into the second tier. After the integration is completed, this plug-in can be published, and independent developers can use it.

3. OVERVIEW OF THE PROPOSED PLUG-IN

Based on the Unity 3D Engine infrastructure, a plug-in is implemented in the second tier. A new mechanism will contain two parts, an original IAP wrapper, and an encryption framework. The IAP wrapper includes original development APIs from both Android and iOS systems. It will be responsible for non-consumable and subscription products. The encryption framework will be responsible for consumable products. This framework contains three mechanisms, Data Save/Load, Internet Access, and Code Obfuscation illustrated in Figure 2.

3.0.1 IAP wrapper module

The first part of the proposed IAP plug-in is an original IAP wrapper. The original IAP means the application platform interface (API) that is provided by the manufacturer, such as Google Inc., Apple Inc., and Microsoft Inc. The first thing we have to understand is what an API is and how we make a wrapper connect with the API.

The API is software that connects application software and hardware devices. Developers do not need to consider the difference between various device models. Manufacturers provide many functionalities, such as Internet connection, camera controllers, user interface builders, and so on. The IAP functionality is one part of the whole API so that developers can integrate or conveniently use this functionality.

Different manufacturers provide their API with various programming languages, and developers must learn these languages to build their application on different software platforms. Fortunately, the Unity provides functionality that can call compiled scripts. This means that developers can write a separate script for different platforms and compile the script to integrate into the Unity. Figure 4 illustrates how the Unity API communicates with different platforms.

According to Figure 4, the IAP wrapper is able to call original platform APIs. We implement different IAP function with a particular API. For example, the Google Store IAP process can be compiled as a jar file. Then, the Unity

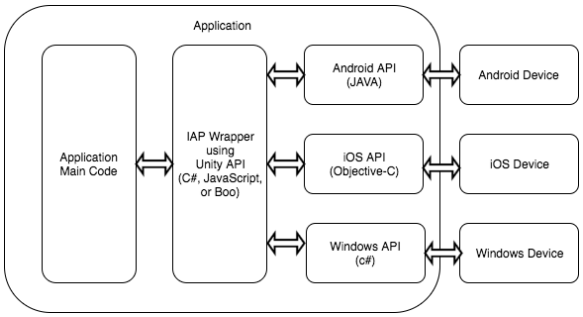


Figure 4: IAP Wrapper.

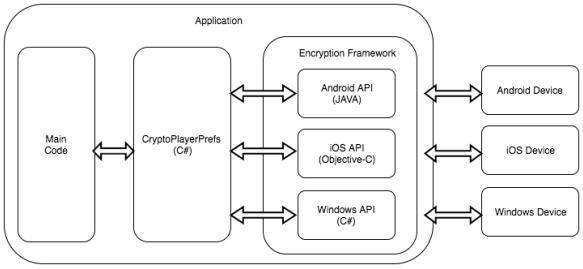


Figure 5: Encryption Module.

API can call the compiled jar file to run the IAP process on Android devices. Similar to the Android device, an IAP process can be compiled on iOS and Windows devices. The compiled file will be a dynamic link library (DLL) file. The Unity API can import this DLL file and run the IAP process script. That is how the Unity IAP wrapper communicates with variant manufacture devices.

3.1 Encryption module

The encryption module contains original save and load functionality with an encryption layer. Unity provides data saving and loading API for developers. The API name is PlayerPrefs. For this paper, a new class called CryptoPlayerPrefs will be created. Moreover, the CryptoPlayerPrefs provides developers with the same logic, the similar method names, and an additional encryption layer. For developers, they do not have a learning curve to use the plug-in. Figure 5 represents how the CryptoPlayerPrefs access local data by using original APIs.

To access local data securely, CryptoPlayerPrefs call an encryption framework, and the framework saves or loads a particular data from local devices. For example, when a target device is Android and users would like to update game data, the application will run the CryptoPlayerPrefs first. This class utilizes the Unity API to call compiled scripts. In this case, the script will be a jar file. In the jar file, there is an encryption framework and a native data access function. This framework can perform encryption and decryption with various encryption algorithms. Developers can select an algorithm or implement their unique algorithm. Then, the framework decrypts file information and updates data by using the native data access function. Finally, the encryption framework will encrypt the updated file.

3.2 Internet access module

There are two ways to make Internet access more secure.

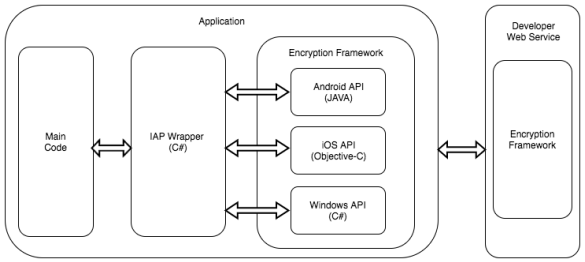


Figure 6: Internet access with the encryption framework.

The HTTPS server can transfer the data through a secure channel, and the encryption framework used in the local data access can provide one more layer security. Figure 6 illustrates the Internet access between the application and the developer's web service.

The HTTPS server is a standard security requirement for the modern web application. Because of this, instruction for how to set up an HTTPS server with a sample code will be provided. It can help developers who lack security experience to build their web service. Moreover, it shows how to connect to the database and send the JavaScript Object Notation (JSON) format to users.

In the encryption framework, the plug-in not only provides encryption and decryption methods on the application side but also on the server side. This plug-in contains sample code for the server side and helps developers establishing their web service with an extra encryption layer. The client can encrypt the message and send it to the server. The server can decrypt the message, perform a particular function, and send a message back to clients. This design may help developers reduce their concern of security and developers can focus on the main functionality of the application.

3.3 Code obfuscation module

The last module is code obfuscation. This mechanism complicates the reverse engineering attack and diverts the attacker. It will make reverse engineering harder and rebuild the developer app. The idea of the code obfuscation is replacing all class and method names with random characters. This process makes the source code unreadable, so attackers spend more time to find their interest codes. Usually, attackers will search keywords first when they extract the application source code. Then, they modify a piece of code to get products for free. If the system uses replaced names for every class and method, it would be difficult to understand the logic of the application.

In this plug-in, we provide an example that will use ProGuard [12] with Android. ProGuard is a software tool and could be a file shrinker, obfuscator, or optimizer. It converts the code when developers build the jar file, and it also generates a mapping file that indicates the relationship between the original name and modified name. When the plug-in is made, this mapping file should be followed to call the native API.

3.4 Server side module

The server here means the developer server, and is not an in-app purchase server. Some developer would like to record or track user data such as consumable products. Because of

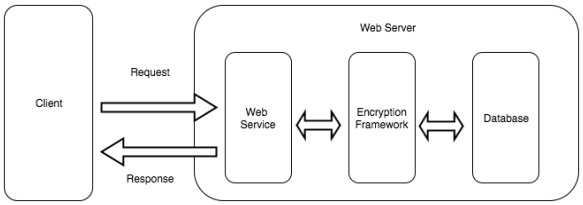


Figure 7: Server side module

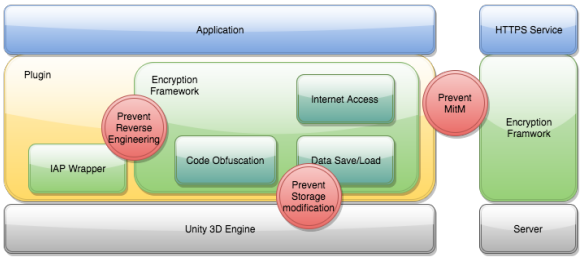


Figure 8: Proposed plug-in addressing IAP vulnerabilities

this, the HTTPS server, the encryption framework, and the database access functions on the server side are provided. The HTTPS server can receive the client request and send a response to clients. The encryption framework will be based on the client side encryption algorithm to match the correct encryption method.

Figure 7 shows how the web server works. When the server receives a client request, the web service checks the message and sends it to the encryption framework. The encryption framework verifies the information by decrypting the message. If the message is authenticated, it will be allowed to pull data from the database. According to the client request, the web service will receive the encrypted message from the encryption framework and send it back to the client.

To summarize, this plug-in will provide a design template to avoid common vulnerabilities when developers implement their IAP processing. The plug-in allows developers to choose or implement their encryption method. In this plug-in, one general encryption, the Advanced Encryption Standard (AES) algorithm, will be provided. The purpose of this plug-in is to reduce the development time, potential vulnerabilities and workload. Figure 8 shows how the plug-in resolves the potential vulnerabilities. The red indicates the potential vulnerabilities. For example, the IAP wrapper and the code obfuscation can protect reverse engineering, the encryption Framework can prevent storage modification, and the HTTPS server can prevent the men-in-the-middle attack.

4. IMPLEMENTATION OF THE PLUG-IN

4.1 Implementation of the encryption module

In this module, there are three parts, data save/load, Internet access, and code obfuscation in the encryption and verification framework. The first one, data save/load, is to prevent attackers compromising local data. The second module, Internet access, is protecting data transferring through the Internet. The last is code obfuscation. It can prevent reverse engineering attacks.

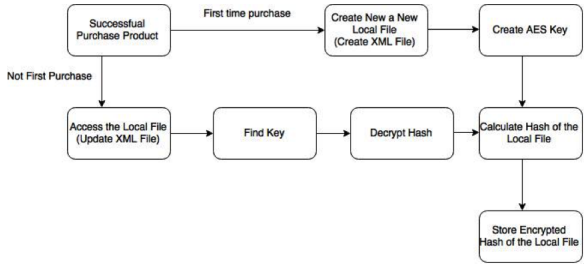


Figure 9: The mechanism of purchasing products

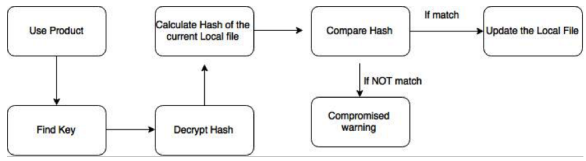


Figure 10: The mechanism of using products

4.1.1 *Secure local data accessibility implementation*

The first one is saving and loading data on local devices. There are two mechanism designs. The first one is purchasing products, and the second one is using products. After users purchase products, this module will try to build or update local files and encryption keys based on a particular encryption algorithm. The algorithm can be implemented by developers or be chosen by a default one. When the system has a local file and an encryption key, it can calculate the hash of the local file and encrypt the hash with the key. Then, the encrypted hash will be stored in a separate place. Figure 9 shows the mechanism of purchasing products.

After purchasing products, users should be able to use or consume products. When users try to use products, this module looks for the encryption key and decrypts the encrypted hash. In the meantime, the system compares the decrypted hash with the current hash which is calculated by the current local file. If these hashes are not the same, it means that the local file has been compromised. The system will alert users. On the other hand, if these hashes are the same, the product will be consumed and updated. Figure 10 shows the mechanism of using products.

Similar to the Unity API, the `CryptoPlayerPrefs` class is provided to load data from local storages. The software design of the `CryptoPlayerPrefs` is the same with the original Unity class, `PlayerPrefs`. The difference between them is that the `CryptoPlayerPrefs` has an authentication mechanism to verify whether the software is compromised or not. Figure 11 shows the mapping of these classes. Figure 12 illustrates the authentication progress.

The original data access class provided by the Unity is `PlayerPrefs`. This class has static methods that allow users to use it in any place. The class can set and get values. The set function is a dictionary structure. Developers use a string as a key and an integer as a value. The data type of the key can only be the string, and the value can be an integer, a float number, or a string. In our new data access class, we only provide getting value function because we only allow setting value when users complete purchasing process successfully. If the new class let developers modify the value directly, it could cause potential vulnerabilities. According

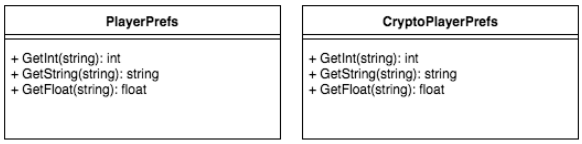


Figure 11: The relationship between `CryptoPlayerPrefs` and `PlayerPrefs`

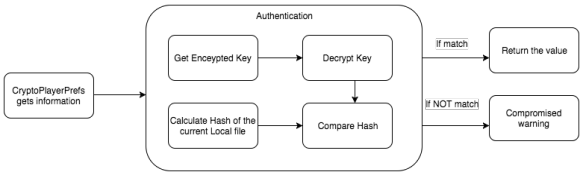


Figure 12: The authentication process

to the original data access class, `CryptoPlayerPrefs` is made to contain the same static methods. Developers can use the new one to replace the original one. Figure 11 illustrates the method name in these classes and shows the return value types.

Comparing both classes, the `CryptoPlayerPrefs` has an authentication layer to protect application from data modifying attacks shown in Figure 12. The purpose of the authentication is that software itself has to verify local data. When an app tries to get local data, it has to check if the data is an authenticated one. The `CryptoPlayerPrefs` class gets an encrypted key from a separate place and decrypts the key to getting the original hash. Then, it calculates the current hash and compares it with the original one. If both are matched, the class will return request information. Otherwise, the class will pop up a warning message and does not allow the application to get the information. Ultimately, the `CryptoPlayerPrefs` can prevent file-compromising attacks, and it is convenient to developers because the new data access function has the same software design with the Unity one.

4.1.2 Secure Internet accessibility implementation

For connecting to web services safely, this plug-in uses the HTTPS to communicate with a server. HTTPS provides an authentication website, a web server and bidirectional encryption communication. There is another feature in the plug-in. It provides an asymmetric encryption to encrypt messages before the message sends to a web server. Because the Unity releases as IAP plug-in, this plug-in will follow the Unity software design. The web server setting process will be discussed in the next section.

From the client side, developers give their web service address to the plug-in. The plug-in will run a communication process. Figure 13 presents the communication between a client and a server. When the client purchases or consumes a product, a connection starts. First, the system generates hash data from a local storage file. The hash data represents the file status. If the hash data is different, the system will know the local file is modified or compromised. Next, the system encrypts the hash data and sends it to the web server through the HTTPS. The web server receives the encrypted data and decrypts it. It also checks if the hash data matches the information from the database. If the hash data

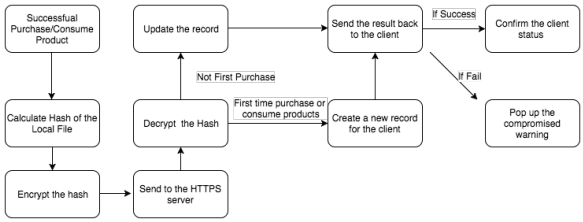


Figure 13: Securing the network access

is correct, the web server sends a confirmation message to the client, and clients can use their products. If not, the web server sends a fail message, and the client will see the warning message.

According to the instruction of the Unity IAP functionality, this paper will follow its design to let developers feel more conveniently. Classes and interfaces are implemented with the prefix, `Crypto`. It means that if a Unity interface is called `IStoreController`, our plug-in will be `ICryptoStoreController`. Table 1 lists classes and interfaces comparing with the Unity one. Methods in the new classes are the same with the original one. Developers can pick up and understand this plug-in quickly.

The progress of using this tool is the same with the Unity. Developers implement the `ICryptoStoreListener` that contains the same methods, so they do not need to update the previous code and just replace class names. The next step is using `CryptoConfigurationBuilder` to initialize an IAP function. Then, the following step is that developers implement a purchase function and a finished call back function. Developers can use the previous implementation because the method name is the same, so they only need to verify the purchasing process. If developers need to use their web server, they have to provide the URL and set the `useWebVerification` property to true in the `CryptoConfigurationBuilder`. Also, they will need to implement a `msgReceiver` method to receive messages from their web server. Overall, the objective of the plug-in design is reducing the development time and preventing common attacks.

4.1.3 Code obfuscation implementation

The code obfuscation is one way to prevent reverse engineering attacks. It modifies class, property and method names when developers release their application. Because of the random name of the code, it is hard to understand the logic behind programs. Therefore, the program can reduce compromised possibilities. An example will be provided for Android and use ProGuard as a code obfuscation tool. ProGuard is a JAVA based tool. It can create a compact code for developers and make programs and libraries more difficult to reverse engineering. This tool will be used to build a part of this plug-in.

Android Studio [6] will be used as the integrate development environment (IDE). Android Studio is the official IDE for Android development provided by Google Inc. The ProGuard is a standard tool in this IDE. How to use the ProGuard is quiet simple. Developers open the build.gradle file and modify the minifyEnabled to true. This step can turn on the ProGuard function. When Android Studio builds the application, it will apply the ProGuard. For more setting, developers can manipulate the proguard-project.txt file. This file can customize the ProGuard process. For example, if de-

Table 1: Classes and Interfaces mapping

Attack Type	iOS	Android
Class	ConfigurationBuilder	CryptoConfigurationBuilder
	GoogleStoreController	CryptoGoogleStoreController
	InitializationFailureReason	CryptoInitializationFailureReason
	Product	CryptoProduct
	ProductCollection	CryptoProductCollection
	ProductDefinition	CryptoProductDefinition
	ProductMetadata	CryptoProductMetadata
	ProductType	CryptoProductType
	PurchaseEventArgs	CryptoPurchaseEventArgs
	PurchaseFailureReason	CryptoPurchaseFailureReason
	PurchaseProcessingResult	CryptoPurchaseProcessingResult
Interface	IGoogleStoreConfiguration	ICryptoGoogleStoreConfiguration
	IStoreConfiguration	ICryptoStoreConfiguration
	IStoreController	ICryptoStoreController
	IStoreListener	ICryptoStoreListener

```
android {
    compileSdkVersion 21
    buildToolsVersion "23.0.3"

    defaultConfig {
        minSdkVersion 9
        targetSdkVersion 19
    }

    buildTypes {
        release {
            minifyEnabled true
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-project.txt'
        }
    }
}
```

Figure 14: ProGuard setting

```
com.iamhomebody.iap.IABBinder -> com.iamhomebody.iap.IABBinder:
    android.app.Activity mActivity -> d
    com.iamhomebody.iap.util.IabHelper mIabHelper -> e
    java.lang.String mEventHandler -> f
    int mAmount -> g
    boolean ifRsaPurchase -> h
    boolean ifRsaConsumeLocal -> a
    java.lang.String mSKU -> i
    int mRequestCode -> j
    java.lang.String mPayload -> k
    java.lang.String mOldRsa -> l
    java.lang.String mVerificationUrl -> m
    android.content.SharedPreferences mSharedPreferences -> b
    com.iamhomebody.iap.util.IabHelper$OnIabPurchaseFinishedListener mPurchaseFinishedListener -> c
    boolean consumeLocalProduct(java.lang.String,int) -> a
    boolean checkFile() -> b
    boolean putString(java.lang.String,java.lang.String) -> a
    void toastMsg(java.lang.String) -> a
    java.lang.String rsa() -> a
    void showNetworkRSA(com.iamhomebody.iap.ResultData) -> a
    android.app.Activity access$800(com.iamhomebody.iap.IABBinder) -> a
com.iamhomebody.iap.IABBinder$7 -> com.iamhomebody.iap.c:
    java.lang.String val$str -> a
    com.iamhomebody.iap.IABBinder this$0 -> b
    void <init>(com.iamhomebody.iap.IABBinder,java.lang.String) -> <init>
```

Figure 15: The ProGuard mapping file

velopers want to keep a particular class that is not allowed to change the name, they can add the exception in this file. Figure 14 represents how developers enable the ProGuard tool.

After developers build their application, ProGuard will generate a mapping file. This file indicates information between the original name and the modified name. Figure 15 shows the mapping file. For instance, the property, `mActivity`, in the `IABBinder` Class, is modified to d. The method `checkFile()` becomes b. In this paper, an application will not be built. Only a library will be built which is the jar file for Unity plug-in.

4.2 Web service implementation

A web service will be implemented by using Python and toolkits, such as Flask and SQLAlchemy. On the other hand, an encryption framework is applied to match the client encryption algorithm. The Flask is a micro web server framework to host the HTTPS server and provides web services. The SQLAlchemy is an object relational mapper to communicate with databases. The system consists of the Linux server and MySQL database system. The database contains two tables as a default setting to record clients' information. The encryption algorithm for this demonstration is RSA, an asymmetric key cryptosystem. Developers can use it directly or integrate their encryption algorithm. Eventually, web service functionalities will be implemented by using Python on the web server. Figure 16 illustrates the server implementation.

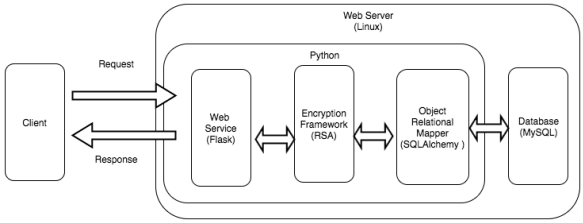


Figure 16: Web service implementation

The Flask, a micro Python framework, can handle web request and response function. It can build an interactive website, but only be used as a web service that only receives, processes and returns the message. When the web service receives client messages, it passes the message to the encryption framework. Since RSA is used here, the system will use the private key to decrypt messages. The encryption framework system receives decrypted messages. If the message is an authenticate one, it will be passed to the next step. Otherwise, the system gives a fail message to the Flask and clients will receive a warning message. After the system verifies the message, it accesses the database to get, set or update the user's data. Then, the system will give the result information to the Flask, and the Flask returns the message to the client.

On the database design, two tables are provided. The first one is a user table. It records user information, such as usernames, passwords, and data create time. The information in this table can authenticate incoming requests. Based on different clients, the web service has different responses. For

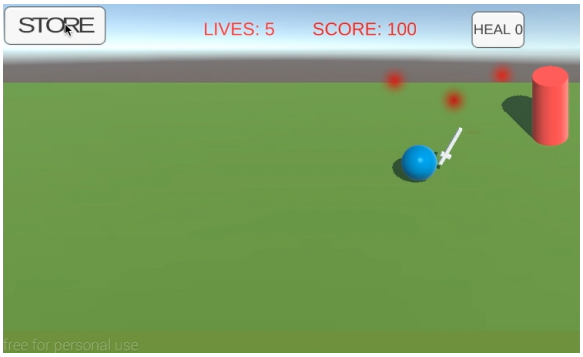


Figure 17: The demo game play

example, if a client tries to connect the server with a correct username and a wrong password, the web service will reject this client's request and respond a fail message. If the client provides the correct username and password, the system can authenticate this request and authorizes this request to connect to the database. The second table is a SHA_HASH table. It contains hash information with a particular user and the last update time. When the system authorizes the request to check database information, it compares the information between the request hash and the database hash. If they are the same, the system will respond with the verification message. Finally, the client can be allowed to consume or purchase products.

5. EVALUATION OF THE SECURE IAP PLUG-IN

The evaluation method will integrate the plug-in in a demo game made by Unity 3D engine. The demo game is an action tower defense game and builds on an Android device. It has a basic IAP function to purchase consumable products. Several different levels of attacks will be provided to measure how difficult existing attacks can compromise developers' application. The attacking scenarios are data compromise, men-in-the-middle, and reverse engineering.

5.1 Addressing the data compromise attack

The following will evaluate whether the plug-in can prevent data compromise attacks. Figure 17 illustrates the basic game play of the demo game. Users represent a blue ball and red cylinders are enemies. Users should avoid enemies passing the left line of the screen. If an enemy passes through the line, players will reduce one life. The game is over when the life becomes zero. However, players can use the heal button to increase their lives. According to this game play design, the product in this demo game is HEALS. Users can purchase five HEALS, fifteen HEALS, or a hundred HEALS. The first evaluation will show the comparison between the demo game without the plug-in and with the plug-in.

First of all, the demo game is built without the plug-in to demonstrate data compromising attacks. The demo game shows a purchasing product process through the Unity IAP package. Users can click which product they want to buy, and the Google Play Store will confirm this purchase order. Figure 18(a) and 18(b) show these two steps. After the purchase process finishes, users can obtain the HEALS in this demo game. Figure 18(c) represents this process. Attackers can compromise the data file by modifying the data files that contain purchase information from any product types. They can compromise the data files by using the ADB (Android Debug Bridge) command [1]. Figure 18(d) demonstrates the modification of the local data file, and Figure 18(e) reveals that the product, HEAL, increase to nine hundred and ninety-nine.

Next, the demo game is built again and integrated the secure IAP plug-in that we developed. The steps of data compromising processes are the same with the previous one. After attackers modify the data file, the plug-in will detect the difference between the original file and the modified one. Therefore, the program will know that the data file is not an authenticate file. Figure 19 illustrates the warning in the game. When the plug-in detects the application is compromised, it will not allow clients to use the product.

5.2 Addressing the man-in-the-middle attack

The scenario is that a user tries to consume the HEAL product to obtain the LIVES. The WireShark [15] is used to monitor the Internet traffic. If plaintext information can be received, a proxy server will be set up by using Charles [2] and replay the attack.

First, the demo game is built without the plug-in and uses a regular HTTP server. When clients consume products, the WireShark can get the Internet information. Figure 20(a) shows internet traffic. The web server IP is 192.168.58.1, and the client IP is 192.168.58.101. The WireShark can collect packets between them and reconstruct the information. Figure 20(b) represents the rebuild information. The information can be got as plain texts. Attackers can analyze the request and response information. In this case, Attackers can know the server responds the true status and clients are authorized to consume products. It means that attackers can try to modify the information by using the proxy server. Figure 20(c) illustrates the Charles proxy server trying to modify the web server response. The Charles proxy can set up breakpoints to suspend Internet connection and modify packets. In this demo, attackers can always modify the response status as true. That means that even though the server does not authorize users, users still can consume the product. Therefore, if the demo game is built without our plug-in, the game will suffer the men-in-the-middle attack. Next, the demo game integrated our plug-in and uses the HTTPS server. The man-in-the-middle attack is repeated again. The WireShark monitors the Internet and tracks packets between the server and the client. Figure 21(a) shows the packets that the WireShark caught. Figure 21(b) illustrates the reconstructed information. Since the HTTPS server is used, the reconstructed packet is encrypted information. It is hard to analyze packets and complete the man-in-the-middle attack. Eventually, the only way to consume products is obtaining the web service's authorization.

5.3 Addressing the reverse engineering attack

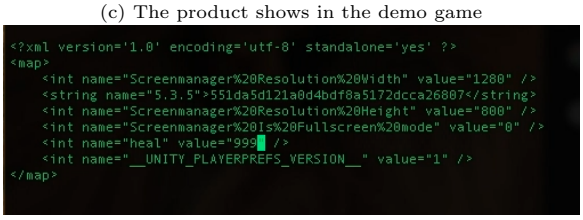
The purpose of preventing the reverse engineering is to make attackers hard to understand program function. The Android Application Package (APK) is a compressed file. Unzip command can unpack the APK. Then, the reverse engineering attack can be duplicated.

The demo game is first built without using the code obfuscation. When attackers get an APK, they can unzip the APK file and get a classes.dex file. The dex2jar tool [13]

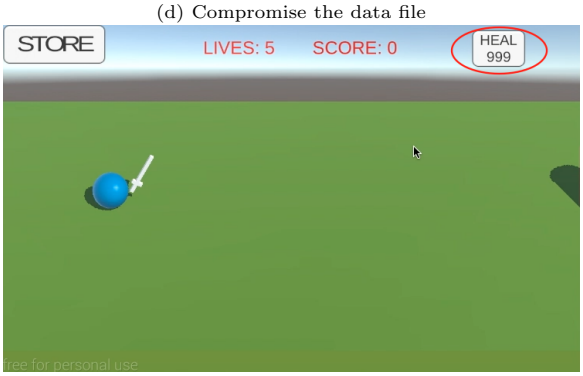
(a) The product menu in the demo game

(b) The confirmation from the Google Play Store

(c) The product shows in the demo game



(d) Compromise the data file



(e) Successful to compromise the demo game to obtain free products

Figure 18: Data compromise attack in the demo game without the plug-in

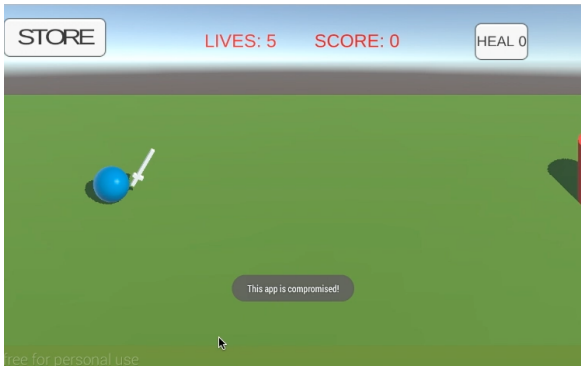


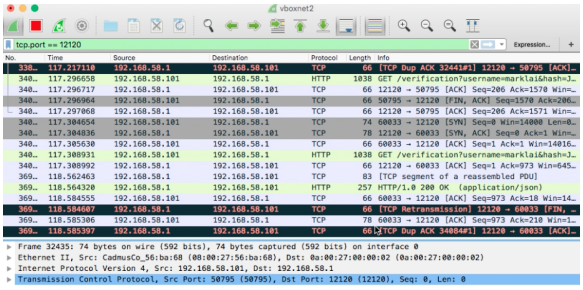
Figure 19: The demo game with secure IAP plug-in and the warning message

can convert the classes.dex file to a classes-dex2jar.jar file. When attackers obtain the jar file, they can use the JDI-GUI tool [3] to read the source code. After attackers realize the program, they can modify a part of the code and re-build the modified one. Ultimately, the attacker can avoid all authentication process to get products for free. Figure 22 illustrates the source code from the reverse engineering attack.

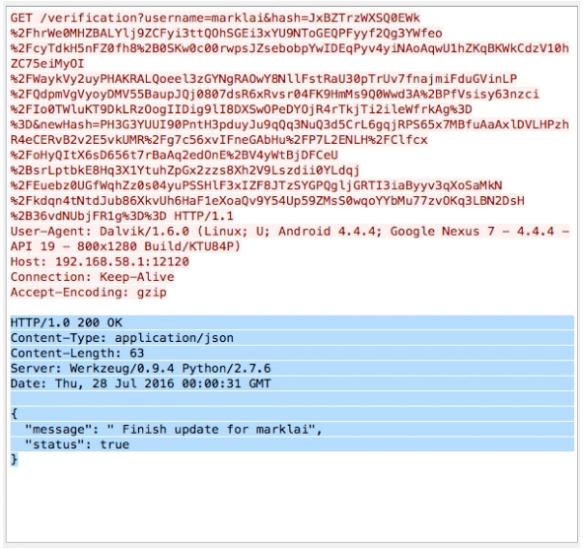
Next, the application is built with code obfuscation, and duplicates the reverse engineering attack. According to the source code from the revers engineering, the part with most content is replaced to simple characters. For example, the member variable, the `mActicity`, is replaced to d. The `IabHelper` type is replaced to d. The source code becomes hard to understand. Because of the difficulty of understanding, attackers have to spend a significant amount of time to analyze the source code. Developers' application will be much safer to prevent the reverse engineering attack.

6. RELATED WORK

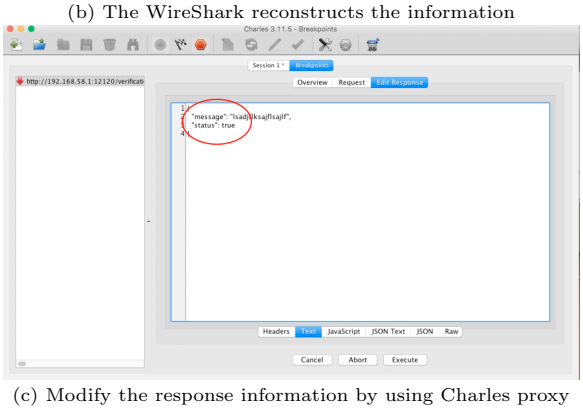
Since the growth of the Android marketplace, both malicious attackers and security researchers have dedicated their lives to compromise and protect products of in-app purchase. Attackers first access sensitive data through application leaks of Android permissions, but researchers provide an Android framework, which can monitor an Android application to provide a proper permission setting[5]. Because directly compromising an application is not efficient, attackers focus on the market, Google Play. They made a compromised Google Play app to bypass the in-app purchase process [14]. After Google realizes this attack, a guideline is provided to prevent this attack by using the signature authentication process [7]. However, attackers develop an automatic attack tool to modify the signature at runtime by using a dynamic Dalvik instrumentation approach [10]. On the other hand, researchers proposed a test tool of the signature verification process to avoid the attack [8]. The previous security researches focused on applications and Google Play process and only proposed solutions on the Android operation system. However, this paper not only considers those issues but also deal with other problems such as device data access and data transmission between clients and servers on different platforms.



(a) The WireShark monitors the Internet packets

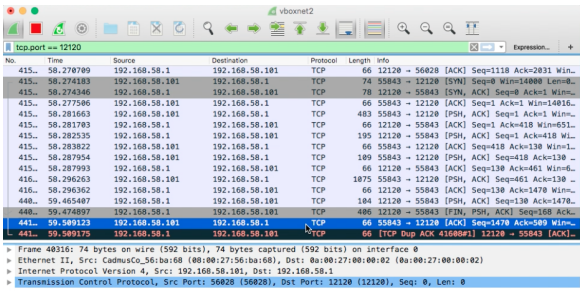


(b) The WireShark reconstructs the information



(c) Modify the response information by using Charles proxy

Figure 20: The man-in-the-middle attack when using a regular HTTP server



(a) The WireShark monitors the Internet packets

(b) The WireShark reconstructs the information

Figure 21: The man-in-the-middle attack when using HTTPS server

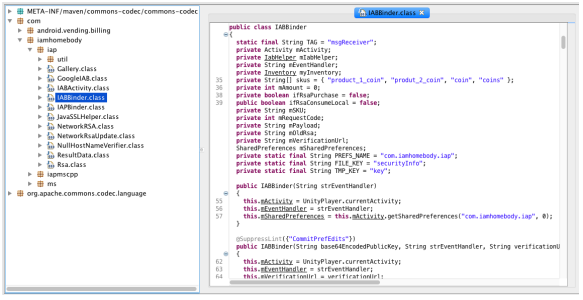


Figure 22: Reverse engineering attacks without code obfuscation

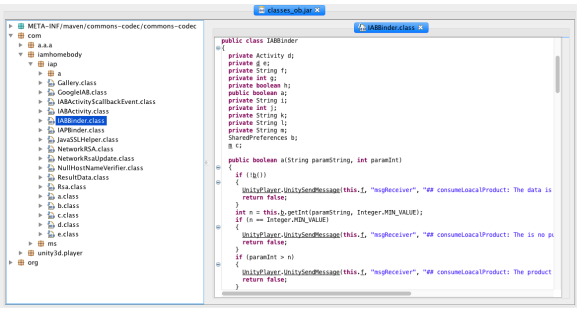


Figure 23: Reverse engineering attacks with code obfuscation

7. CONCLUSION AND FUTURE WORK

Nowadays, many attackers are trying to compromise mobile applications and take products from developers without permission. They use different technologies to break the IAP process, which is the new business model of directly purchasing products in the application. This new design allows users to buy extra content or virtual goods. However, most independent developers and small-scale companies do not have enough resources to resolve the security issues. In this paper, a potential solution to prevent some common attacks, such as data compromise attack, men-in-the-middle attack, and reverse engineering is provided.

To demonstrate the solution, a demonstration app (demo) was built. The Unity Engine was used to create the demo since the population of the Unity community is increasing and is close to fifty percent of the mobile app industry. The demo is an action tower defense game, a typical case in the industry. It has essential functions, such as gameplay, a product menu, and in-app purchases. This demo was used to show the potential vulnerabilities of in-app purchases and to duplicate attacks. The paper is a plug-in for Unity, which was later integrated into the demo to show the result of our research.

There are three methods proposed to avoid these attacks. The encryption framework is designed to prevent data compromise attacks. This framework checks the local data file by using the hash function and encryption. Before users purchase or consume products, the system first verifies the data. If the data is not verified, the system will not allow users to purchase or consume products and vice versa. The next solution for man-in-the-middle attack is using the HTTPS server. Today, the HTTPS server is a standard secure Internet communication protocol. A simple web service is provided by the Python micro-framework, Flask. This web service can give independent developers the concept to set up and design their web server. It also shows how to verify the incoming web request and how to connect the database. Moreover, it can be used directly if it matches developers' requirements. The last well-known attack attempted to be solved is reverse engineering. The solution provided is code obfuscation. Reverse engineering of the software is the process of decomposing the application product, manipulating the application content and rebuilding the product based on the original information. This means that attackers can bypass the regular in-app purchasing process to get content or virtual goods. The code obfuscation modifies the source code by replacing readable words with unreadable characters. In the program, developers usually use meaningful names to declare the variables or functions, but the code obfuscation will replace all names with random combination characters. Even though attackers extract the source code, they will encounter the obfuscated program. Because of this, using reverse engineering makes it hard to attack this application.

8. REFERENCES

[1] K. Boss. "android - adb backup/restore- adb extractor (abe) - extracting adbs with openssl", 2014.

[2] Charles. "charles 3.11.5", 2016. Accessed July 29, 2016.

[3] J. Contributors. "java decompiler", 2015. Accessed July 29, 2016.

[4] Cydia. "how to install and use localiapstore ios 8.1.1 – ios 8.1.2 to get free in app purchases.", 2015. Accessed January 5, 2015.

[5] B. Davis, B. Sanders, A. Khodaverdian, and H. Chen. I-arm-droid: A rewriting framework for in-app reference monitors for android applications. In *Workshop on Mobile Security Technologies (MoST)*, 2012.

[6] Google. "android studio 2.1", June 2016. Accessed July 29, 2016.

[7] Google. In-app billing security and design, 2016.

[8] H. Kim and S. won Kim. Securing android in-app billing service against automated attacks. In *International Journal of Security and Its Applications*, 2016.

[9] Millz. "free in app purchases android.", 2013. Accessed January 5, 2015.

[10] C. Mulliner, W. Robertson, and E. Kirda. Virtualswindle: An automated attack against in-app billing on android. In *Proceedings of the 9th ACM symposium on Information, computer and communications security*, 2014.

[11] K. Pitakronachai. "ios games money hack compilation tutorial.!!! non jailbreak devices.", 2013. Accessed January 5, 2015.

[12] ProGuard. "proguard version 5.2", 2015. Accessed July 29, 2016.

[13] pxb1988. "dex2jar", 2016. Accessed July 29, 2016.

[14] D. Reynaud, E. C. R. Shin, T. R. Magrino, E. X. Wu, and D. Song. Freemarket: Shopping for free in android applications. In *ISOC Network and Distributed System Security Symposium (NDSS)*, 2012.

[15] WireShark. "wireshark 2.0.5", July 2016. Accessed July 29, 2016.