Journal of Basic and Applied Engineering Research
p-ISSN: 2350-0077; e-ISSN: 2350-0255; Volume 4, Issue 6; July-September, 2017, pp. 502-505

http://www.krishisanskriti.org/Publication.html

Secure Coding for Mobile Payment Application

Aquil Ahmad Khan[1] and Mayank Jain[2]

[1,2]ICERT, New Delhi
E-mail: [1]akhan2786@gmail.com, [2]engineermayankjain@gmail.com

Abstract—*With the Government's 'Digital India' initiative, the share of online transactions and consequently the share of payments made via mobile phones has been increasing. Mobile devices provide a high level of accessibility; users have access to everything, all of the time. However there is an inverse relationship between security & accessibility and finding the right balance between the two when building mobile payment applications is not only tricky but can also determine its success or failure. This paper is presented as a list of considerations for people managing and building mobile payment services. Moreover, following the guidelines mentioned in this document does not guarantee security of the application as the mobile threat landscape is ever-changing.*

Keywords*: Secure Coding, Mobile Payment; Encryption; Security, OWASP.*

1. INTRODUCTION

Mobile computing is one of the greatest recent areas of growth in the field of information technology. Our lives are impacted by our mobile devices much more than the computers that we leave have at our homes and workplaces. Unlike those devices, our mobile phones are always on and we carry them around with us. This makes them highly valuable targets for malicious actors.

From the second quarter of 2013, smartphones or phones capable of performing many of the functions of a computer have been outselling feature phones. This explains the amount of applications being developed for mobile phones, including payments. Online transactions and consequently the share of payments made via mobile phones has been increasing day by day. Customers tend to abandon a payment service if, a security flaw is discovered and the service does not provide them with enough assurance of its security. This paper are to be considered as such and do not replace any mandatory requirements of the reader's organisation. Moreover, following the recommendations mentioned in this document does not guarantee security of the application as the mobile threat landscape is ever-changing.

2. THREAT LANDSCAPE

There are differences between security considerations for personal computers and mobile devices. The important ones are listed here:

- Mobile phones are easier to steal and/or lose. Research has shown that 9 million mobile devices are either lost or stolen globally every year, which is equivalent to 1 device every 3.5 seconds;
- Mobile phones have limited input capabilities, so the usage of long or complicated passwords has a huge impact on the user experience and severely increases the number of failed logins;
- mobile phones are designed as portable devices, so two factor authentication mechanisms requiring the user to use additional hardware (in addition to the mobile phone itself) such as one time password generators are likely to significantly hamper the service accessibility;
- Mobile phones interact with several hosts outside of the user's control. Generally no or weak mutual authentication controls exists between the phone and a third party. Third party applications might have access to unprotected sensitive information stored and processed;
- The owner of a mobile phone (handset hardware) has virtually no control on the device security configuration. A mobile phone is generally chosen for its functionality not security;
- Malware on mobile phones is rising fast and is apparently causing a drop in the creation and use of new, computer-related malware for the first time;
- Users are either unaware or tend not to worry about malware on mobile phones, and have limited options to deal effectively with it.

3. SECURE DESIGN

Developers need to build applications in a secure way. However, it is not possible to spend all the time focusing on security. The answer is to use good design principles, tools, and mindsets that make security an implicit result - it's secure by design. Then secure-by-design becomes a guiding principle in how the software is built, from code to architecture. Understanding principles, designs and patterns that promote security is a fundamental requirement in building a secure application.

The Open Web Application Security Project (OWASP) lists the following secure design principles:

- **Minimize attack surface area:** Every feature that is added to an application adds a certain amount of risk to the overall application. The aim for secure development is to reduce the overall risk by reducing the attack surface area.
- **Establish secure defaults:** There are many ways to deliver an "out of the box" experience for users. However, by default, the experience should be secure, and it should be up to the user to reduce their security, if they are allowed.
- **Principle of Least privilege:** The principle of least privilege recommends that accounts have the least amount of privilege required to perform their business processes. This encompasses user rights, resource permissions such as CPU limits, memory, network, and file system permissions.
- **Principle of Defense in depth:** The principle of defense in depth suggests that where one control would be reasonable, more controls that approach risks in different fashions are better. Controls, when used in depth, can make severe vulnerabilities extraordinarily difficult to exploit and thus unlikely to occur. With secure coding, this may take the form of tier-based validation, centralized auditing controls and requiring users to be logged on all pages.
- **Fail securely:** Applications regularly fail to process transactions for many reasons. How they fail can determine if an application is secure or not.
- **Don't trust services:** Many organizations utilize the processing capabilities of third party partners, who more than likely have differing security policies and posture than you. It is unlikely that you can influence or control any external third party, whether they are home users or major suppliers or partners. Therefore, implicit trust of externally run systems is not warranted. All external systems should be treated in a similar fashion.
- **Separation of duties:** A key fraud control is separation of duties. For example, someone who requests a computer cannot also sign for it, nor should they directly receive the computer. This prevents the user from requesting many computers, and claiming they never arrived. Certain roles have different levels of trust than normal users. In particular, administrators are different to normal users. In general, administrators should not be users of the application.
- **Avoid security by obscurity:** Security through obscurity is a weak security control, and nearly always fails when it is the only control. This is not to say that keeping secrets is a bad idea, it simply means that the security of key systems should not be reliant upon keeping details hidden.
- **Keep security simple:** Attack surface area and simplicity go hand in hand. Certain software engineering fads prefer overly complex approaches to what would otherwise be relatively straightforward and simple code. Developers should avoid the use of double negatives and complex architectures when a simpler approach would be faster and simpler.
- **Fix security issues correctly:** Once a security issue has been identified, it is important to develop a test for it, and to understand the root cause of the issue. When design patterns are used, it is likely that the security issue is widespread amongst all code bases, so developing the right fix without introducing regressions is essential.

4. SECURE CODING

Once the application has been constructed with the fundamental principles of secure design in mind, it is important to make sure that the code is written by developers with experience in secure coding. An insecure code can open the door to a variety of attacks such as:

- Improper Platform Usage
- Insecure Data Storage
- Insecure Communication
- Insecure Authentication
- Insufficient Cryptography
- Insecure Authorization
- Client Code Quality
- Code Tampering
- Reverse Engineering
- Extraneous Functionality

All of the above mentioned attacks might either allow an attacker to execute code within the context of the mobile application, or run queries on the databases located on the mobile and (even worse) remote server. In general, the developers should follow the checklist at [OWASP Secure Coding Practices] while writing the code. They must also be aware and up-to-date with the common vulnerabilities associated with the platforms on which the application is deployed.

Additionally, some important recommendations specific to development of Android mobile applications are given below:

- **Lock-down application permissions:** It is necessary to follow the principle of least privilege when assigning permissions. Permissions should not be assigned unless they are required. The application should be granted only the minimum required permissions at the architecture

level. For instance, READWRITE permissions should not be granted when only READ permissions are required.

- **Handle broadcast messages carefully:** To handle event-driven tasks, an application can register a broadcast receiver that executes a function once it's passed an intent that matches specified criteria. Applications can send broadcast intents, which allow another application to receive it and process its data. Any object or data received from the broadcast should be checked for invalid data or exceptions before using it in an application.
- **Be prudent while using broadcast messages for Inter-Process Communication (IPC):** The nature of broadcast messages permits any application to receive a broadcasted Intent. If broadcast messages are used for IPC, then a malicious application may be able to gain access to another application's data. This is often a common occurrence that demonstrates the lack of understanding around broadcast messages. Use the local broadcast manager if the intent needs to be broadcast locally. The local broadcast manager allows intents to be broadcast locally within the application so no other application can gain unauthorized access to application data.
- **Do not use insecure storage:** When evaluating an application's storage usage, ensure that both online and offline functionality of the application is evaluated. Look specifically for code that allows storage of the data locally and ensure that no sensitive data is stored on the client side. At the architecture level, try to minimize the sensitive data that needs to be stored on the device. If any data needs to be stored, then encrypt it using a strong algorithm prior to storage. Data that is stored in /data/data/<package name of application> cannot be accessed by another application unless the application explicitly provides permissions or if the Android device is rooted. Data that is stored in /sdcard can be accessed by any application without the need for any special permission or rooting. Hence, it is common for malicious applications to access data in /sdcard.
- **Avoid insecure storage in process memory:** Data processed by the application may be stored within memory longer than necessary, which makes it more susceptible to attack. An attacker with access to the phone may be able to dump the memory of the process to gain access to sensitive information such as usernames, passwords, and other data. Analyse the classes that take username, password, and account number as input. Try to determine if the values are cleared in the memory after use. If not, the application may expose sensitive information if a memory dump can be obtained by an attacker. The Dalvik runtime allows garbage collection, but this does not allow a developer not to consider memory management. It is never advisable for any variable to hold sensitive information in it even when the user is logged in. This is especially true when the user logs off the application. At that point, all variables holding sensitive information should be cleared by initializing them to some junk values.
- **Protect pending intents:** The pending intents function allows the intent in your application to be invoked by another application. Just invoking the intent is not the issue. The issue is that the application that invokes the intent also executes at the same permission level as that of the application that had the pending intent to be invoked. The best and simplest approach is to find an alternative for the pending intent which is as good as eliminating risk. If a pending intent is an application requirement, make sure that only a trusted application receives it. This leaves no room for that intent to be used by an untrusted application.
- **Use WebView carefully and properly:** The WebView class is one of the most powerful classes, and it renders web pages inside a normal browser. It also allows applications to interact with WebView by adding a hook, monitoring changes being made, add JavaScript, and more. Even though this seems like a great feature, it brings in security loopholes if not used with caution. Since WebView can be customized, it creates the opportunity to break out of the sandbox and bypass the same origin policy.
- **Obfuscate the code:** The Dalvik byte code can be easily reversed to obtain Java code that is very close to the original Java code. This aids the attacker in understanding application logic and also gain deeper understanding of the application. dex2jar and JD-GUI are two free tools that can be used to reverse engineer Android applications. Code obfuscation is a method that involves mangling code during the build process. The generated code is difficult for humans to understand and increases the amount of work required for reverse Engineering.
- **Avoid Excessive logging:** Client-side data logging performed by Android applications has not garnered much attention from a security standpoint. However, during Android application review, we often see sensitive user data like user names, passwords, and account numbers written to application logs. This information can be easily retrieved by an attacker if he is able to gain access to the device. Perform proper exception management, and always perform logging only to the extent required. Sensitive data like account numbers and passwords should not be logged.
- **Perform data validation:** Data validation issues in Android are usually not considered as serious during penetration testing or while performing a code review. However, this is a mistake. WebView becomes vulnerable to all browser attacks because WebView itself is a browser instance and has all the capabilities of a browser. An Android application can be coded in Java or native

code, which is C++. When Java is used, many of the data validation issues like buffer overflow, format string issues, and others are eliminated, as the language itself is not vulnerable. When using native code, special care needs to be taken when data is read from an untrusted source because it is vulnerable to issues like buffer overflow, format string issues, and more. When performing data validation code review, it is necessary to identify the source and the sink. Source refers to the place where the data is received. Sink refers to the place where data is sent back to user. Once complete flow from the source to sink is understood, we can easily identify what kinds of issues there are, for example, XSS, SQL injection, buffer overflow, and many more data validation-related issues.

- **Properly verify server certificate on SSL/TLS:** Android apps that feature online payments must use SSL/TLS protocols for secure communication, and should properly verify server certificates. The developer has the freedom to customize their SSL implementation. The developer should properly use SSL as appropriate to the intent of the app and the environment the apps are used in. If the SSL is not correctly used, a user's sensitive data may leak via the vulnerable SSL communication channel. Insecure uses of SSL include:
  - Trusting all server certificate regardless of who signed it, what is the CN (Common Name) etc.
  - Allowing all hostnames, instead of verifying if the certificate is issued for the URL the client is connecting to.
  - Mixing secure and insecure connections in the same app or not using SSL at all.

5. CONCLUSION

Mobile application security is complicated, it is not just the code running on the devices, there are innumerable other factors like the device platform, web-services, cloud based 3rd party services etc., which play a very important role in mobile application security. This strategy should then translate to the creation of a custom Secure System Development Life Cycle(S-SDLC) for the development of organization mobile applications. By putting an S-SDLC in place, mobile application vulnerabilities can be identified and eliminated well in advance of deploying the application, thereby resulting in considerable saving on investment. Organizations should perform a detailed analysis of their risk posture against all possible known security threats to an application and use this to create a mobile security strategy.

REFERENCES

[1] OWASP Mobile Security Project https://www.owasp.org/index.php/OWASP_Mobile_Security_Project
[2] The protection of information in computer systems http://www.acsac.org/secshelf/papers/protection_information.pdf
[3] A short overview of Android banking malware http://www.net-security.org/malware_news.php?id=2595
[4] Master key Android vulnerability used to trojanize banking application http://blog.trendmicro.com/trendlabs-security-intelligence/masterkey-android-vulnerability-used-to-trojanize-banking-app/
[5] Master key Android vulnerability http://blog.trendmicro.com/trendlabs-security-intelligence/trendmicro-solution-for-vulnerability-affecting-nearly-all-android-devices/
[6] MWR attack on mobile ad networks https://labs.mwrinfosecurity.com/blog/2013/09/24/webviewaddjavascriptinterface-remote-code-execution/
[7] Mobile app top 10 risks http://www.veracode.com/directory/mobileapp-top-10.html
[8] Lessons learned from five years of building more secure software http://msdn.microsoft.com/en-gb/magazine/cc163310.aspx
[9] Bank apps riddled withsec urity holes http://finextra.com/news/fullstory.aspx?newsitemid=25601
[10] Android security overview http://source.android.com/devices/tech/security/
[11] iOS security http://images.apple.com/ipad/business/docs/iOS_Security_Feb14.pdf
[12] Windows Phone security http://www.windowsphone.com/en-GB/business/security