Vulnerability in Android Development

Aarushi Arya
B.Tech Student
Dept. of CSE,
HMRITM, INDIA

Rahul Malhotra
B.Tech student
Dept. of CSE
HMRITM, INDIA

Dayanand
Assistant Professor
Dept. of CSE
HMRITM, INDIA

***Abstract*— IT industry is focusing on malware and antivirus, we eliminated that risk using the security we had already. Most of the malware is removed. Main risk today, is leaky applications and vulnerable devices. Almost half of Android applications have at least one high security risk or privacy flaw. This happened as android made development easy for everyone, not just for developers, with the help of libraries and tools. Naive developers are using built in methods like loadUrl and WebView, without knowing their proper usage, making applications vulnerable to security risks. Resulting in millions of free applications being released on the Android market with serious flaws.**

Also, android support the development of third-party applications. These applications can use the phone's built in applications and provide a more personalised experience. Developers have the freedom to combine applications with web and use data on the android devices. These flexibilities have provided open platform for hackers to take advantage of the threats created. Security has become a major concern. In this paper, vulnerabilities in web applications are discovered using analysis tools and ways to mitigate these vulnerabilities are discussed.

Keywords: Vulnerability, Attacks, Security, Android application

I. INTRODUCTION

Everyone is using android devices, from mobile phones to television sets. Their applications are available in the Android market for free. Applications can also be downloaded from web directly. These applications introduce security threats in the device, putting user data on risk. Users are not aware of this risk and install applications from Android market or websites without the knowledge that an application can expose the device to multiple risks.

To be able to understand this, we must know how applications get installed and updated on the device. First, .apk file is checked by the Android to identify the developer using the valid digital signature before installing the application. If the .apk file is valid, permissions asked by the application are displayed to the user. User can agree and install the application or user can choose not to install the application. Then, each application is assigned a unique Linux user ID and group ID. A specific directory on the device is created to an application, where its data is stored and access is provided to permissions generated from the given ID.

While installing an application, permissions requested by the application are displayed. User can either accept and install or go back. This step is done, so that user is aware of the access application has on his device. It is crucial that the user takes a calculated risk of using free services provided by the application in exchange of the access allowed to the application on user's device and data.

If an application is installed directly from the web, the digital signature checking step is skipped. That means if a hacker publish an update to the application, data can be stolen. In this case, valuable information will be leaked and misused.

For applications which are downloaded, android provides the capabilities of removing application from Android Market and also from devices. This is done to prevent active circulation of vulnerable application, known as Remote Application Removal Feature (introduced in June 2010).

In this paper we made the following contributions-

1. Analyse applications using Baksmali tool that can detect different classes of vulnerabilities in applications and Wireshark to analyze the network traffic.
2. Identify vulnerable WebViews and methods.
3. Suggest solutions to mitigate these vulnerabilities.

II. BACKGROUND HISTORY

It is easier to develop Android application, using tools to develop the GUI. Functionality can be added using built in libraries or downloading plug-ins. To support these functionalities, applications use built in functions to collect and store user data. Hackers exploit these functions and find vulnerabilities, which later can be used to extract user data from multiple applications without the permission of the user or the developer. This issue is even more critical today, as smart phones contain personal and corporate data. From text messages to pictures and GPS location. Users feel that smart phones are secure, however free applications available on the Android market for free provide gateways to hackers to misuse the data of the users. To be able to prove this statement, we found security flaws in application available in the android market for free.

To do the analysis of applications, we use the open- source tools available and obtain flow graph of these applications. Further analyze the URL that can be reached starting from the initial URL in the application.

III. ANALYSIS

Tools used to analyze android applications are-

1. Dedexer
2. Baksmali
3. Soot
4. Wireshark

Dedexer

It is used to dissemble DEX files. It is able to read the DEX format and transform it into assembly-like format. This format contains Dalvik opcodes, we must understand these Dalvik opcodes in order to work with the tool. It has some limitations like the tool does not process the debug and annotation information in the DEX files[1].

Baksmali

Open source DEX dissembler. It comes with a Dalvik bytecode assembler(smali). The syntax of Baksmali[8] is loosely based on Jasmin's/dedexer's syntax, supporting full functionality of DEX format.

Soot

It is a java instrumentation and analysis framework that support Dalvik bytecode.

Dexpler is a modification of Soot, directly read Dalvik bytecode and perform analysis and maps Dalvik bytecode instructions to Jimple(Java sIMPLE) statements[2]. Soot generates call graph and control flow graph.

Wireshark

It is a free and open source packet analyzer. It is used for network troubleshooting and analysis. Wireshark is cross-platform, runs on Linux and Microsoft Windows.

Let us start by describing the security model for mobile web applications and several classes of vulnerabilities in mobile web applications.

Three relevant challenges considered in mobile web applications –

1. APPLICATION – It captures the attack capabilities of a malicious application running alongside a trusted application. This application may send intents to any application installed on the device.
2. NETWORK – It may receive , send or block messages on the network. As observed in the application providing free VPN service through Wireshark that 50mb of data containing the location information of the user is transferred without any knowledge.
3. NAVIGATION-RESTRICTED WEB – It is a variant of a typical web challenge. It may step up any number of malicious websites and place content on them.

Methods used to identify the vulnerabilities are following-

1. Finding initial reachable URL's from application code – String analysis is done to report values of parameters to navigation methods like loadUrl. Web applications access build URLs. String analysis is built using Soot.

   Android application is written in Java and transformed into Dalvik bytecode. This Dalvik bytecode can be converted to Java bytecode using Ded and then static analysis is done using Soot.

   We can dissemble Dex files and XML content is extracted, using Dedexer and Baksmali tools.

   Prefix such as http:// is also reported by string analysis, stating that application is loading content over insecure connection.
2. Looking into the navigation control implementations from application code – Application's implementation of methods like shouldOverrideUrlLoading and shouldInterceptRequest are extracted to create Java program. When URL is tested, reports are generated about how these methods would behave if it was executed in the application. It is more efficient to extract backward slice of method with respect to any return statements or calls to navigation methods using algorithms than

   using emulator to run the application and determine if a WebView is allowed to load a page. Statements that cannot be executed as standalone Java executables are removed. Here, we emphasize on the overall behaviour of the method. We check if the behaviour is similar in all paths and intents being called from the application. This pruning step enable us to execute slices[5] that branch based on application state for reasons other than controlling navigation. Slices are built using Soot.

3. Web crawl from initial URL is performed – Now we have the resources application under test can achieve by web crawling starting from initial URL. This is done, so that we can map out the URLs application uses during its execution. Then we note that how many of these are malicious URL or may lead to malicious ones.

We analyzed applications available in the android market and identified vulnerability classes which are as follows-

1. WEBVIEWS

Some of the major concerns are that the android supports rendering of web content obtained from the Internet or files on device in WebView. Methods such as loadUrl, loadData, postUrl and passing strings containing HTML content are used. These applications are designed to interact with specific web content. Developers can prevent unsupported web resources by implementing callback methods such as shouldOverrideUrlLoading(before loading new page) and shouldInterceptRequest(before making any web request). Application can prevent resources from loading. By overriding a URL load developer allow applications to correctly constrain the web content loaded in their application WebView[4]. This prevents WebView to act on its own when user clicks a link, which might harm user experience.

2. LOADING UNTRUSTED CONTENT

WebView allow Android developers to support multiple platforms, allowing HTML content be displayed within the application. It provides customization specifying how and what content is displayed. They pose threat to application security, by allowing web page to access data stored by the application. WebView may navigate to malicious page and access sensitive data. Also, if the user is using public network then man-in-the-middle could launch an attack by injecting malicious content into the page.

In android version 4.4 or below, WebView contains an unpatched Universal Cross-Site Scripting vulnerability. Also, mobile web applications lack a url bar, so users cannot know the sites they are visiting. Moreover, there is no guarantee that web content loaded in the WebView is confined to it.

1. LEAKY URL'S

Application can leak information through URL loads that are overridden in methods like loadUrl. When an application overrides a URL load and uses an Implicit Intent to load the resource, application can handle the URL load. If a leaked URL contains private information then that information is also leaked along with the URL. Developers might make the mistake of thinking that it is safe to use Implicit Intent to give URL to an application. In this case, URL might match a custom URL scheme but Android does not provide any protections on custom URL schemes. In relation to mobile OAuth implementation, Chen et al.[15] discussed a vulnerability. If an application registers a custom URL pattern to receive the final callback URL in an OAuth transaction and uses Implicit Intent to deliver URL then a malicious application can register the same URL pattern and steal the OAuth credentials.

IV. MITIGATION

The vulnerabilities discussed above, can be reduced by following suggestions.

1. App's Manifest should contain a whitelist of trusted domains.
2. Developers can safely constrain navigation by correctlyimplementingshouldOverrideUrlLoading and shouldInterceptRequest.
3. Application should not depend on correct navigation control to ensure untrusted content does not have access to the JavaScript Bridge.
4. Turn off JavaScript in WebView, if it is not needed.
5. Restrict navigability in WebView, limiting content loaded via links and not documents.
6. Register to only necessary interfaces, limiting exposure to API.

V. RESULT

Android devices can be both a target and a tool to carry out attacks by the hacker. Users must be aware of the risks and take measures to protect against misuse of their personal data. Developers also should pay attention to security concerns and take responsibility of protecting user data. Applications like appWatchdog use forensics techniques to determine whether username, passwords, credit card numbers are being insecurely stored. It provides a basic indication of whether a mobile app implements security by answering questions like does the application avoid Man-in-the-Middle attack, is application protected from session hijacking or how does the application handle web history and caching.

Users can secure their android devices by installing free security android application from Sophos or Semantic. Even if the device is using such software or data is encrypted, it can still be stolen and misused. So be aware of the risks while installing applications in the devices.

Developers can follow guidelines as they design, develop and test their applications.

1. Do not store username and password in plain text in the device.
2. Do not store credit or debit card details on the device.
3. Encrypt the sensitive data to make it difficult for hackers to retrieve information.
4. Do not store encryption key in the device.
5. Send encrypted data over the network.
6. Do not store authentication data after authorization.
7. Mask PAN when displayed.
8. Do not store the PIN(Personal Identification Number).

RELATED WORK

A number of studies have examined individual vulnerabilities related to mobile web applications such as Luo et al.[11], Chin et al.[12] and Georgiev et al.[13]. However, only few applications were tested focusing one of the vulnerability.

About 100,000 applications were tested and all the vulnerabilities were conclusively given by Patrick Mutchler,Adam Doup, John Mitchell,Chris Kruegel and Giovanni Vigna[13] concluding the following -

UNSAFE NAVIGATION-15% of applications we tested contain at least one fully computed URL for an Internet resource. Of these applications, 34% were able to reach untrusted web content by navigating from initial URL.

UNSAFE CONTENT RETRIEVAL-40% of the applications we tested had computable scheme for at least one URL. More than 50% of these applications contained a URL with an HTTP scheme.

VI. CONCLUSIONS

Android allow interaction between web and application code without taking necessary measures to ensure that these applications are safe. Several vulnerabilities in mobile web applications were identified using Soot and Wireshark. Finally, changes that must be done to ensure application is secure were listed.

REFERENCES

[1] BARTEL, A., KLEIN, J., LE TRAON, Y., AND MONPERRUS, M. Dexpler: Converting Android Dalvik Bytecode to Jimple for Static Analysis with Soot. In Proceedings of the ACM SIGPLAN International Workshop on State of the Art in Java Program Analysis (2012).

[2] VALL´EE-RAI, R., CO, P., GAGNON, E., HENDREN, L., LAM, P., AND SUNDARESAN, V. Soot: A Java Bytecode Optimization Framework. In Proceedings of the Conference of the Centre for Advanced Studies on Collaborative Research (1999).

[3] Apktool. code.google.com/p/android-apktool. Accessed: 2014-2-14.

[4] CHIN, E., AND WAGNER, D. Bifocals: Analyzing WebView Vulnerabilities in Android Applications. In Proceedings of the International Workshop on Information Security Applications (2013).

[5] WEISER, M. Program Slicing. In Proceedings of the 5th International Conference on Software engineering (1981).

[6] BALL, T., AND HORWITZ, S. Slicing Programs with Arbitrary ControlFlow. In Proceedings of the International Workshop on Automated and Algorithmic Debugging (1993).

[7] PALLER, G. Dedexer. http://dedexer.sourceforge.net/.

[8] Smali and baksmali. http://code.google.com/p/smali/.

[9] LUO, T., HAO, H., DU, W., WANG, Y., AND YIN, H. Attacks onWebView in the Android system. In Proc. of the 27th Annual Computer Security Applications Conference (2011).

[10] CHESS, B., AND MCGRAW, G. Static analysis for security. Security & Privacy, IEEE 2, 6 (2004), 76–79.

[11] LUO, T., HAO, H., DU, W., WANG, Y., AND YIN, H. Atacks on WebView in the Android System. In Proceedings of the 27th Computer Security Applications Conference (2011).

[12] CHEN, E. Y., PEI, Y., CHEN, S., TIAN, Y., KOTCHER, R., AND TAGUE, P. Oauth demystified for mobile application developers. In Proceedings of the 21st ACM Conference on Computer and Communications Security (2014).

[13] GEORGIEV, M., JANA, S., AND SHMATIKOV, V. Breaking and Fixing Origin-Based Acces Control in Hybrid Web/Mobile Application Frameworks. In Proceedings of the 21st Symposium on Network and Distributed System Security (2014).

[14] A Large-Scale Study of Mobile Web App Security Patrick Mutchler,Adam Doup, John Mitchell,Chris Kruegel and Giovanni Vigna