http://dx.doi.org/10.14236/ewic/bcsme2014.7

Two-factor Protection Scheme in Securing the Source Code of Android Applications

Daniel TSE
City University of Hong Kong
iswktse@cityu.edu.hk

Zihuan LI
City University of Hong Kong
zackli2-c@my.cityu.edu.hk

Yuhui TAO
City University of Hong Kong
yuhuitao2-c@my.cityu.edu.hk

Ka Fai WONG
City University of Hong Kong
kfwong48-c@my.cityu.edu.hk

Wai Hou CHOI
City University of Hong Kong
waihchoi4-c@my.cityu.edu.hk

Wei LIU
City University of Hong Kong
wliu222-c@my.cityu.edu.hk

While Android has become most popular OS in mobile phone market, more and more Android app developers are suffering from intellectual property infringement because it's easy to extract the assets stored in the Android apps and to decompile Android apps to Java source code. This issue also poses threats to users' privacy. In this article we reviewed the existing protection approaches for the protection of the source code and assets in Android apps, demonstrated the reason why Java Native Interface (JNI) approach can help improve the protection provided by existing approaches, developed 4 Android demo apps with 2 experiments conducted to evaluate the effectiveness of protection provided by the combination of Encryption and JNI approaches.

Android applications, decompilation, reverse-engineering, piracy, encryption, JNI, assets protection

1. INTRODUCTION

Android is always the hot point when public are talking about technology. Android operation system is also widely used and the users' number is increasing consistently. According to the latest research in the third quarter of 2013 about operation system, we found that android takes 75% in the global smartphone operating system market share, while Apple follow with 15.6% in second place. At the same time, as for the global smartphone operating system shipments, Android also takes the first place with 20.4 million, increasing by 75 million in just one year, and almost six times as many as that of Apple [1]. These two statistical data indicate the huge usage of Android.

Also the programming language trend indicates Android's fast growth, Java is a hot point when hunting a job, and Java programmers are in high demand when comparing with other languages, which illustrates that the high level of demand for the Java language comes from Android's speedy growth [2].

Along with the growth of Android operating system, the number of Android applications (apps) is also increasing dramatically [3]. There are more applications in Android application market than other operating systems, and an increasing number of apps developers prefer to choose Android version.

The number of Android applications grows twice as fast as that of Apple applications and nearly 64% of them are free. Such fast development of Android applications has attracted developers of other mobile platform join Android camp.

However, piracy of Android applications is increasingly threatening Android application developers and even the users of Android applications.

2. THE JNI APPROACH

As far as we researched, the existing tools for decompiling Android applications only support

decompiling applications written by Java, because the core decompilation mechanism of the tools is indeed a Java Class file decompilation tool. Therefore, if an Android application is written in a non-Java programming language and compiled in a different way, those decompilation tools might become ineffective.

JNI is a programming interface for Java. It allows developers to write C/C++ code in Java applications, and supports the interaction between the Java and C/C++ languages [13]. Android applications, which are developed by Java, also support JNI with the help of Android NDK provided by Google [14]. Although the main purpose of JNI is to support some features in some specific systems that are not accessible by Java, or to improve the runtime efficiency of the application [15], it could serve as a possible solution for protecting Android application from being decompiled. If an Android application is written by C/C++, or partly by Java and partly by C/C++ via JNI, and as Java code and C/C++ code will be compiled in different ways, the final application might not be decompiled easily as mentioned before, at least for the decompilation tools that only able to handle Java code. In this way, JNI might be able to add an effective security layer to the source code of Android applications and the assets stored.

Applying JNI to protect the source code of Android applications and assets has not been thoroughly analyzed, assessed or discussed in the academic community (to the best of our knowledge), though it has been proposed as a solution in some online discussion forums when some developers asked about the source code/assets protection issues online [16][17].

In this research, we'll apply JNI to design and write an Android application in Java and C/C++, and conduct an experiment to assess the effectiveness of this approach in protecting the source code and assets of Android applications.

3. METHOD AND EXPERIMENTATION

3.1. Experiment 1

We first encrypted one image and one text file in AES CBC mode. The encrypted image data and text data are stored in the APK file of an Android app we developed in Java. When this app is launched, it will decrypt the image and text then partially display them on the screen of the Android device. The AES key used to decrypt the image and text data is hard-coded in the Java source code.

A different image and text file were encrypted using a different AES key than the previous one. The encrypted image and text data are stored in the second Android app we developed, which will decrypt and partially display them, just like the first app. The difference is the decryption process is coded in C++, which output only the decrypted result to Java via JNI. The source code of both apps is obfuscated using ProGuard during compilation.

	App One	App Two
Obfuscated by ProGuard?	Yes	Yes
Decryption process written in	Java	C++
Decryption key written in	Java	C++
Display process written in	Java	Java

After compiling these 2 Android apps and having them tested, we uploaded these 2 Android apps to 3 developer themed online discussion forums, which are XDA Developer Forum[18], Stack Overflow[19] (English) and CSDN Forum[20] (Chinese), and invite hackers to try decrypting the image and text content stored in the apps but only partially displayed when the apps are launched.

We can't control hackers' interest in our project and do not know if anyone of them would like to cooperate with us by trying to do the decompiling and decrypting. So we decided to collect the experimental result based on 2 conditions:

If there are hackers participating in our project, making attempts to decompile and decrypt the content in the apps and would like to share with us the result (whether they're successful

or not). We'll try to collect information as follows:

- Is the hacking finally successful or not?
- How long have you spent on it?
- How many attempts have you tried so far?
- Which approaches (tools) have you used?

We'll then use the above result data collected to conduct analysis. If any hacker claimed he/she has succeeded in hacking either one of the 2 apps, we'll ask him/her to provide the decrypted image and text file in this app as a proof.

Otherwise, if no hacker makes any attempt on decompiling the 2 apps, we'll collect their comments/replies, and count how many comments show attitude of supporting JNI as a more effective source code protection approach, how many show attitude of supporting hacking apps written in JNI is as easy (hard) as hacking apps written in Java.

Otherwise, if no hacker has even replied any comment on our posts online, we'll have to conduct pure conceptual analysis without any input from hackers.

3.2. Experiment 2

Furthermore, we developed 2 additional apps to test the runtime efficiency of the C++ based decryption process of Android applications to assess whether the introduction of C/C++, as the non-official programming language for Android applications, has negative effect on performance and speed. The 2 apps are supposed to decrypt 3 encrypted files (1MB, 2MB and 4.3MB) and calculate the time spent on decrypting each of the 3 files. The result time span will be automatically calculated and displayed when the app is launched. The decryption process in the first app is written in Java, while in the second app written in C++.

4. RESULT

4.1. Result of Experiment 1:

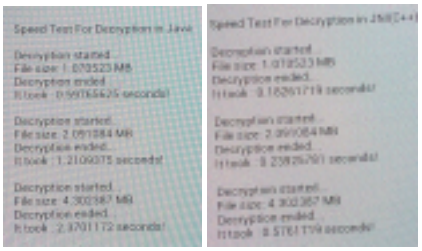
We have not received any sharing from hackers about the hacking result. But we have received a total of 13 comments/replies so far. Based on the attitude of each comment, we divided them into four categories:

- Java Better: thinks Java with obfuscation is better than JNI approach at deterring decompilation.
- JNI Better: thinks C/C++ via JNI is better than Java with obfuscation at deterring decompilation.
- Same Effectiveness: thinks no approach can prevent decompilation.
- Void Comment: contains no proper attitude related to this project.

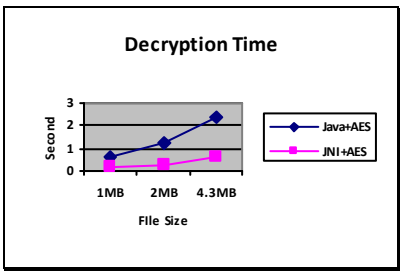
Java Better	JNI Better	Same Effectiveness	Void Comment
0	2	4	7

4.2. Result of Experiment 2:

As coded, the 2 apps calculated the time it took them to decrypt each of the 3 different files and displayed them on the Android screen (the model we used is Samsung Galaxy S1).



	1MB	2MB	4.3MB
Java+AES	0.6	1.21	2.37
JNI+AES	0.18	0.24	0.58



5. ANALYSIS AND DISCUSSION

5.1. Safety and Security Concerned

In Experiment 1, by developing the two Android apps and exposing them to hackers, we planned to evaluate the whether coding in C/C++ with JNI for Android apps has any advantage over coding in Java with obfuscation, in terms of protecting the source code and assets stored in the apps. Decrypted image and text files are supposed to be posted as an indicator of successful hacking, because only by obtaining the source code can hackers find out the decryption key and decryption process to decrypt the image and text files. But instead of making attempts to hack our apps, hackers only left some comments with attitude. Thus we analyzed their attitude for the evaluation.

Of all the non-void comments from hackers, 66.7% think any protection/encryption/anti-decompilation method is useless for some hackers (senior hackers perhaps). But apparently, whether this kind of hacker exists, they are not going to be in the target group of our project.

The rest of hacker comments (33.3%) think JNI will indeed increase the difficulty of decompiling Android applications and increasing the security of the source code and the assets. We did some extended research to find some reasons underlying their opinions.

When the Android application written in Java and C/C++ is compiled, the Java source code is compiled to Java Class files and then to the DEX binary file, which as mentioned before is vulnerable to existing decompilation tools. Meanwhile, the C/C++ source code in the Android app is compiled to .so files, which are essentially Linux Shared Object files for the Android app to use. This is because Android is a Linux based platform [22], which relies on the Linux kernel (level one) to provide basic drivers to interact with hardware. Some C/C++ libraries (level two) such as SSL, WebKit and OpenGL ES that process essential system tasks are built on the Linux kernel. The .so files that are compiled from C/C++ code are actually additional Linux libraries to the second level of Android architecture [21].

Are Linux .so library files easy to be decompiled? So far fortunately the answer is no, because many abstractions in the C/C++ source code are lost when compiled into assembly code, which is then converted to .so files. Those lost abstractions make it difficult to decompile assembly code back to C/C++ source code [23]. While Java on the other hand preserved many of these abstractions in the Java Class files, which makes the decompilation much easier as mentioned before.

Even if there are mature and powerful decompilers for Linux .so files, they might not work properly on Android .so files. The reason is there are many difference between the .so files for Linux and .so files for Android, probably a result of differences between Android and Linux [24]. To compile C/C++ source code into Android-specific .so files, many changes have to be made to the build processes that are originally targeted at Linux [25].

To summarize, hackers tend to support that applying JNI to Android application development is a better solution to source code and assets protection than only using Java with obfuscation. And their opinions are well supported by technical theories behind Android architecture.

5.2. Runtime and Launch Speed Performance

The result of Experiment 2 shows that the decryption performance of Android app written by C++ via JNI has its competitive advantage over the app that uses Java to decrypt the same content. Most of the time, people may think C/C++ is a much faster language than Java [26][27], but some argue that in some cases Java could be faster than C/C++ [28].

We won't discuss so much about which programming language is faster in which cases. So long as for using C/C++ code via JNI to decrypt assets/data for the purpose of protecting source code and assets, JNI

approach shows much better runtime performance than traditional Java approach for the same file size. Moreover, as the size of the file to be decrypted increases, the time needed for decryption by Java increases dramatically while C/C++ in JNI approach only shows insignificant increase of time consumed.

5.3. Effort on Development

Most Android developers are Java developers[29], who only know little or limited about C/C++ programming. However, JNI requires developers to know a lot of C/C++ programming, especially for complicated algorithms such as encryption and decryption in order to protect the assets in Android apps. Although there are some existing open-source security-related C/C++ libraries such as OpenSSL [30] and Crypto++ [31] that we can put into use directly, they are not specifically designed for JNI and Android. To compile them properly for JNI and Android, a lot of modifications are needed on the ordinary building process [25]. Any missing or incorrect modification might lead to error on compilation or crash on launching the Android app.

Being proficient in both Java programming and C/C++ programming is not enough for JNI approach. There are a lot of JNI-specific knowledge for Android developers to learn before they are able to use it comfortably. Such knowledge include multi-thread issues, accessing Java classes and methods from C/C++, charset compatibility, data/variable type conversion, etc. [32] Each of the tiny adjustment must be considered and processed properly otherwise the compiler will always stop working and report errors. [14]

In short, the learning curve of JNI might be long for Java developers who know little about C/C++ programming, and still plenty of new knowledge to learn even for developers familiar with both Java and C++.

6. Conclusion

The analysis of the result of the experiments shows the JNI approach could be an effective solution to the decompilation issue of Android apps. With proper implementation of JNI approach, the source code of the app will not be easily obtained by hackers via the common existing decompilation tools. Additionally, by coding decryption process in C/C++ via JNI, the assets/data stored in the app can be protected because of the protection of C/C++ source code. Thus, Android developers' intellectual property and users' privacy will be more secure.

7. References

[1] Ron Amadeo, "Google's iron grip on Android: Controlling open source by any means necessary", arstechnica.com, Oct. 21, 2013. [Online]. Available: http://arstechnica.com/gadgets/2013/10/googles-iron-grip-on-android-controlling-open-source-by-any-means-necessary/. [Accessed: Nov. 17, 2013].

[2] Robert Diana, "Traditional Programming Language Job Trends-February2012", regulargeek.com , Feb. 10, 2012. [Online]. Available: http://regulargeek.com/2012/02/10/traditional-programming-language-job-trends-february-2012/. [Accessed: Nov. 17, 2013].

[3] Egle Mikalajunaite, "Android Market is the fastest growing mobile content platform since the beginning of 2011", www.research2guidance.com, May. 5, 2011. [Online]. Available: http://www.research2guidance.com/android-market-will-become-the-biggest-mobile-content-platform-in-the-world-by-august-2011/. [Accessed: Nov. 17, 2013].

[4] WordPress.com "AndroidStarted-Note2:Android file.apk decompile", justamomentgoose.wordpress.com, June 4, 2013. [Online]. Available: http://justamomentgoose.wordpress.com/2013/06/04/android-started-note-2-android-file-apk-decompile/. [Accessed: Nov. 17, 2013].

[5] S. Cimato, et al., "Overcoming the obfuscation of Java programs by identifier renaming", The Journal of Systems and Software, vol. 78, pp. 60–72, Jan. 2005.

[6] user818502, "Why is it so easy to decompile Java Code?", stackoverflow.com, Sep. 16, 2012. [Online]. Available: http://stackoverflow.com/questions/12450510/why-is-it-so-easy-to-decompile-java-code/. [Accessed: Nov. 17, 2013].

[7] Chris Davies, "95% Android game piracy experience highlights app theft challenge", www.slashgear.com, May 15th 2013. [Online]. Available: http://www.slashgear.com/95-android-game-piracy-experience-highlights-app-theft-challenge-15282064/. [Accessed: Nov. 17, 2013].

[8] Claburn, Thomas, "Android Survey Highlights Piracy Problem", United Business Media LLC, United Business Media LLC, Sep 8, 2011.

[9] Daniel, "The True Problem With Google's License Verification Library (LVL)", daniel-codes.blogspot.hk, Oct. 11, 2010. [Online]. Available: http://daniel-codes.blogspot.hk/2010/10/true-problem-with-googles-license.html. [Accessed: Nov. 17, 2013].

[10] Youn-Sik Jeong, et al., "A Hybrid Design of Online Execution Class and Encryption-based Copyright Protection for Android Apps", ACM, Proceedings of the 2012 ACM Research in Applied Computation Symposium, RACS '12, pp. 342-343, 2012.

[11] Patrick Schulz, "Code Protection in Android", net.cs.uni-bonn.de , June 7,2012. [Online], Available: http://net.cs.uni-bonn.de/fileadmin/user_upload/plohmann/2012-Schulz-Code_Protection_in_Android.pdf. [Accessed: Nov. 17, 2013].

[12] Youn-Sik Jeong, "An Anti-Piracy Mechanism based on Class Separation and Dynamic Loading for Android Applications", Proceedings of the 2012 ACM Research in Applied Computation Symposium, RACS '12, pp. 328-332, 2012.

[13] Oracle, "Java Native Interface", docs.oracle.com, [Online]. Available: http://docs.oracle.com/javase/6/docs/technotes/guides/jni/. [Accessed: Nov. 17, 2013].

[14] Android, "Android NDK", developer.android.com, [Online]. Available: http://developer.android.com/tools/sdk/ndk/index.html. [Accessed: Nov. 17, 2013].

[15] Oracle, "Java Native Interface Specification—Contents", docs.oracle.com, [Online]. Available: http://docs.oracle.com/javase/6/docs/technotes/guides/jni/spec/intro.html#wp9502. [Accessed: Nov. 17, 2013].

[16] sachin003, "How to avoid reverse engineering of an APK file?", stackoverflow.com, Dec. 13, 2012. [Online]. Available: http://stackoverflow.com/questions/13854425/how-to-avoid-reverse-engineering-of-an-apk-file. [Accessed: Nov. 17, 2013].

[17] [Online]. Available: http://www.linkedin.com/groups/Android-How-can-I-prevent-86481.S.205685643. [Accessed: Nov. 17, 2013].

[18] zopelee, "Welcome to decrypt the App content !!!", forum.xda-developers.com, Nov. 13, 2013. [Online]. Available: http://forum.xda-developers.com/showthread.php?t=2525212. [Accessed: Nov. 17, 2013].

[19] user2990181, "Is ProGuard or JNI capable of protecting assets stored in Android apk file?", stackoverflow.com, Nov. 14, 2013. [Online]. Available: http://stackoverflow.com/questions/19968385/is-proguard-or-jni-capable-of-protecting-assets-stored-in-android-apk-file/. [Accessed: Nov. 17, 2013].

[20] zopelee, bbs.csdn.net, Nov. 14, 2013. [Online]. Available: http://bbs.csdn.net/topics/390642167. [Accessed: Nov. 17, 2013].

[21] Mysterio, "Does Android really use the same kernel as Linux?", unix.stackexchange.com, Nov. 27, 2011. [Online]. Available: http://unix.stackexchange.com/questions/25463/does-android-really-use-the-same-kernel-as-linux. [Accessed: Nov. 17, 2013].

[22] Android, "Android, the world's most popular mobile platform", developer.android.com, [Online]. Available: http://developer.android.com/about/index.html. [Accessed: Nov. 17, 2013].

[23] Vignesh4303, "How to decompile Linux .so library files from a MS-Windows OS?", reverseengineering.stackexchange.com, Aug. 21, 2013. [Online]. Available: http://reverseengineering.stackexchange.com/questions/2664/how-to-decompile-linux-so-library-files-from-a-ms-windows-os. [Accessed: Nov. 17, 2013].

[24] Hadeel Tariq Al-Rayes. "Studying Main Differences between Android & Linux Operating Systems", International Journal of Electrical & Computer Sciences, Vol. 12, No. 05, pp. 46-49, 2012.

[25] Piotr Morgwai Kotarbinski, "Android NDK Advanced Tutorial", morgwai.pl, 2013, [Online]. Available: http://morgwai.pl/ndkTutorial/. [Accessed: Nov. 17, 2013].

[26] kostja, "when is java faster than c++ (or when is JIT faster then precompiled)?", stackoverflow.com, Dec. 23, 2010. [Online]. Available: http://stackoverflow.com/questions/4516778/when-is-java-faster-than-c-or-when-is-jit-faster-then-precompiled. [Accessed: Nov. 17, 2013].

[27] Ben Joan, "Difference Between Java and C++", www.differencebetween.net, July. 27, 2011.

[Online]. Available: http://www.differencebetween.net/technology/difference-between-java-and-c/. [Accessed: Nov. 17, 2013].

[28] Paul Buchheit, "Java running faster than C", paulbuchheit.blogspot.hk, June. 3, 2007. [Online]. Available: http://paulbuchheit.blogspot.hk/2007/06/java-is-faster-than-c.html. [Accessed: Nov. 17, 2013].

[29] [Online]. Available: http://www.infoworld.com/d/application-development/java-remains-most-popular-language-thanks-android-178469-0. [Accessed: Nov. 17, 2013].

[30] [Online]. Available: http://www.openssl.org/. [Accessed: Nov. 17, 2013].

[31] [Online]. Available: http://www.cryptopp.com/. [Accessed: Nov. 17, 2013].

[32] Android, "JNI Tips", developer.android.com, [Online]. Available: http://developer.android.com/training/articles/perf-jni.html. [Accessed: Nov. 17, 2013].