Detecting Java methods called by Java Reflection during Runtime

Sumin Lee
Soongsil University
Seoul, Korea
smlee.oslab@gmail.com

Minho Park
Soongsil University
Seoul, Korea
mhpark.oslab@gmail.com

Jiman Hong*
Soongsil University
Seoul, Korea
jiman@ssu.ac.kr

ABSTRACT

When Java Reflection is used, the class to access and the method to be called are determined at runtime. Although this Java Reflection provides flexibility in system development, it can be used to hide malicious behavior in Android malicious applications, making it difficult to analyze malicious behaviors. In this paper, we propose a tool that records method calls of an Android application in the execution log and detects methods called at runtime through Reflection. We also show that the proposed tool can detect the methods in the application in which Java Reflection is applied.

KEYWORDS

Android, Java Reflection, Dynamic Analysis, Execution Log

1 INTRODUCTION

The Android mobile platform is the most used mobile platform in the world, but due to its versatility and openness, it is vulnerable to mobile devices and personal information leakage. According to AV-Test, a global security product performance evaluation agency in 2022, more than 3 million malicious applications (apps) targeting the Android platform have been developed every year for the past three years [1]. Newly emerging Android malicious apps are becoming more sophisticated and threatening the security of Android by applying various anti-analysis techniques such as code encryption, code obfuscation, and software packing [2].

Among the code obfuscation techniques, Java Reflection used for API hiding is a Java API used to inspect or modify the behavior of classes and methods at runtime, and the class to call or method to be executed is determined at runtime [3]. However, Android malicious apps can use Java Reflection to decide what actions to take at runtime and hide their malicious actions. Therefore, it is important to determine what behavior the code using Java Reflection in the Android app performs. In this paper, we propose a tool for detecting the method called with Java Reflection based on method call information recorded in the Android app execution log.

The remainder of the paper is organized as follows. Section 2 presents background. Section 3 describes the structure and design of the proposed tool. Section 4 presents the validity of the proposed tool by evaluating it in the app using reflection. Finally, concluding remarks are provided in Section 5.

* Corresponding Author

2 BACKGROUND

2.1 Android Execution Environment

The APK, which is the installation file of the Android app, is composed of a dex file that converts the Java code written in the app into Dalvik bytecode, AndroidManifest.xml containing essential information of the app, and resource files required to run the app. Currently, as runtime environments for running Android apps, there are Dalvik Virtual Machine (VM) and Android Runtime (ART) [4]. The runtime environment and compilation method also have been continuously changed as shown in Table 1 to improve Android app execution performance.

Dalvik VM compiles the app when it is running. Then, it interprets the bytecode of the dex file through the interpreter or executes the machine code that interpreted from the frequently used bytecode while executing the app through the Just-In-Time Compiler (JIT). ART interprets the bytecode of the dex file as machine code through the Ahead-of-Time compiler (AOT) when installing the app. Since Android 7.0, ART includes Dalvik VM's JIT compiler, and in the latest version, ART's method has one of the following states: interpreted dex code, JIT compiled, and AOT compiled.

Table 1: Runtime Environment and Compiler based on Android Versions

Android Version	Runtime Environment	Compiler	Remarks
Android 2.2 (Froyo)	Dalvik VM	JIT	JIT added
Android 4.4 (Kitkat)	Dalvik VM	JIT	ART added, but the default VM is Dalvik VM
Android 5.0 (Lollipop)	ART	AOT	-
Android 7.0 (Nougat)	ART	AOT + JIT	introducing a JIT Compiler with code profiling to ART

3 A Tool to detect the method using Reflection Based on Android App Execution Log

The proposed tool records the method called while running the app by modifying the Android Open Source Project (AOSP) [5] 10 version. As in Fig. 1, after installing the app on the mobile device where the modified AOSP is built to record method calls, runs it. Method calls that occur internally while the app is running are recorded in the execution log, and the execution log is extracted after the app is terminated. The proposed tool changes the compilation option of AOSP so that the app can be executed only through the Dalvik interpreter in order to effectively record the method called with Java Reflection at runtime. And to prevent unnecessary method call recording, method calls are recorded only for methods called by the installed app.

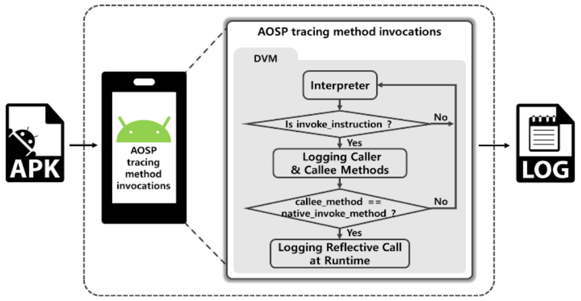

APK
AOSP tracing method invocations
AOSP tracing method invocations
DVM
Interpreter
Is invoke_instruction ?
No
Yes
Logging Caller & Callee Methods
callee_method == native_invoke_method ?
No
Yes
Logging Reflective Call at Runtime
LOG

Figure 1. Overview of the proposed tool

3.1 Logging Caller & Callee Methods

Java Reflection is provided through the java.lang.reflect package. Before calling a method using Java Reflection, it is necessary to check the class name to be accessed and the method in the class. Therefore, in this paper, only several Reflection methods listed in Table 2 are analyzed. Also, to execute a method using Java Reflection, invoke() of java.lang.reflect.Method must be called. At this time, the method called is the Reflective Call, and if we can find the point where the Reflective Call is called, the execution flow of the Android app can be analyzed and can confirm why the Java Reflection is called.

Table 2: Classes and Methods in java.lang.reflect package

Type of Class	Method	Description
Class	getName()	show the name of the entity
	getDeclaredMethod()	show the specified declared method in the class
	getDeclaredMethods()	show all the declared methods in the class
	getMethods()	show all the public methods in the class
Method	Invoke()	invoke a method of the class at runtime

Android app calls a new method via invoke-kind instructions, and the types of invoke-kind instructions are shown in Table 3. The AOSP interpreter handles the invoke-kind instructions with DoInvoke(). In DoCallCommon(), which is called in the process of executing the invoke command, the method that is called the invoke instruction (caller method) and the callee method via the invoke instruction can be detected.

Table 3: invoke-kind instructions

opcode	Instruction	opcode	Instruction
0x6E	invoke-virtual	0x74	invoke-virtual/range
0x6F	invoke-super	0x75	invoke-super/range
0x70	invoke-direct	0x76	invoke-direct/range
0x71	invoke-static	0x77	invoke-static/range
0x72	invoke-interface	0x78	invoke-interface/range

3.2 Logging Reflective Call at Runtime

AOSP handles the class of java.lang.reflect.Method with the Java Native Interface (JNI). In Android, JNI defines how compiled bytecode and native code written in C/C++ interact. The JNI that handles invoke() of java.lang.reflect.Method is Method_invoke(), and the method called at runtime can be checked in invokeMethod() called during instruction processing.

The proposed tool extracts the app execution log through the "adb logcat" command. The format of recording method calls in the modified AOSP is shown in Fig. 2. In Fig. 2, [Call] indicates the method names of the caller and callee called by the invoke instruction and the call relationship between the two methods. Also, [Reflection] indicates a method called at runtime by invoke() of java.lang.reflect.Method.

```
[Call] "Caller Method" -> "Callee Method"
[Reflection] "Reflective Call at Runtime"
```

Figure 2. Execution Log Recording Format

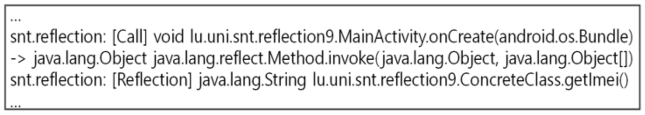
4 EVALUATION

To implement the proposed tool on Google's Pixel 4a, we build the modified AOSP on the device and evaluate whether the proposed tool properly extracts a method called by Java Reflection. Also, among the benchmark apps used to validate the tools proposed by Sun et al. in [6], Reflection9.apk, which uses invoke() of java.lang.reflect.Method, is also used in this experimental evaluation.

Fig. 3 shows a part of the app's MainActivity code. The benchmark app, Reflection9.apk, imports the ConcreteClass class and accesses the fields in the class. And the acquired device identification information is stored in the class internal field "imei". It obtains imei again using the class internal method getImei() and send the information to the outside using SMS. The proposed tool can detect that ConcreteClass.getImei() was called by Java Reflection as shown in Fig. 4.

```
TelephonyManager telephonyManager = (TelephonyManager)
     getSystemService(”phone”);
String deviceId = telephonyManager.getDeviceId();
Class<?> c = getClassLoader().loadClass(”lu.uni.snt.
     reflection9.ConcreteClass”);
BaseClass bc = (BaseClass) c.newInstance();
Field f = c.getField(”imei”);
f.set(bc, deviceId);
Method m = c.getMethod(”getImei”, new Class[0]);
String imei = (String) m.invoke(bc, new Object[0]);
SmsManager sms = SmsManager.getDefault();
sms.sendTextMessage(”+49 1234”, null, imei, null, null);
```

Figure 3. MainActivity of Reflection9.apk



```
...
snt.reflection: [Call] void lu.uni.snt.reflection9.MainActivity.onCreate(android.os.Bundle)
-> java.lang.Object java.lang.reflect.Method.invoke(java.lang.Object, java.lang.Object[])
snt.reflection: [Reflection] java.lang.String lu.uni.snt.reflection9.ConcreteClass.getImei()
...
```

Figure 4. Execution log for Reflecion9.apk

We also experimented with 25 randomly selected real-world apps using reflection to evaluate the tool proposed by Alhanahnah et al. In [7]. However, the proposed tool can detect methods called using Java Reflection in 7 apps out of 25 apps. This is because the proposed tool performs dynamic analysis, so even if the app uses invoke() of java.lang.reflect.Method, if the method is not executed during experimentation, it is not recorded in the execution log. However, as a result of manually analyzing 7 apps, it was confirmed that if the method called by Java Reflection was recorded in the execution log, the called method could be detected even if additional obfuscation and encryption were applied.

5 CONCLUSIONS

In this paper, we proposed a tool to detect the method called by Java Reflection based on the execution log that records the method call of the Android app. The proposed tool records the method called by invoke-kind instruction and invoke() of java.lang.reflect.Method in the execution log while the app is running. The evaluation results showed that the proposed tool can detect the method called by Java Reflection. In the additional experiment with real-world apps, the proposed tool can detect the method called by Java Reflection even when additional obfuscation techniques are used.

ACKNOWLEDGMENTS

This research was supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Education(NRF-2016R1D1A1B01016073)

REFERENCES

[1] AV-Test(2022), Development of Android malware, https://www.av-test.org/en/statistics/malware/ (accessed Aug., 14, 2022)

[2] Rodríguez, R. J., Ugarte-Pedrero, X., & Tapiador, J. (2022). Introduction to the Special Issue on Challenges and Trends in Malware Analysis. *Digital Threats: Research and Practice (DTRAP)*, *3*(2), 1-2.

[3] Sihag, V., Vardhan, M., & Singh, P. (2021). A survey of android application and malware hardening. Computer Science Review, 39, 100365.

[4] Android Runtime(ART) and Dalvik(2022), https://source.android.com/docs/core/dalvik (accessed Aug., 14, 2022)

[5] Android Open Source Project, https://source.android.com/ (accessed Aug., 14, 2022)

[6] Sun, X., Li, L., Bissyandé, T. F., Klein, J., Octeau, D., & Grundy, J. (2021). Taming reflection: An essential step toward whole-program analysis of android apps. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, *30*(3), 1-36.

[7] Alhanahnah, M., Yan, Q., Bagheri, H., Zhou, H., Tsutano, Y., Srisa-An, W., & Luo, X. (2020). Dina: Detecting hidden android inter-app communication in dynamic loaded code. IEEE Transactions on Information Forensics and Security, 15, 2782-2797.