POSTER: Detection of Information Leaks via Reflection in Android Apps

Jyoti Gajrani
MNIT, Jaipur, India
2014rcp9542@mnit.ac.in

Li Li
SnT, University of Luxembourg, Luxembourg
li.li@uni.lu

Vijay Laxmi
MNIT, Jaipur, India
vlaxmi@mnit.ac.in

Meenakshi Tripathi
MNIT, Jaipur, India
mtripathi.cse@mnit.ac.in

M.S. Gaur
MNIT, Jaipur, India
gaurms@mnit.ac.in

Mauro Conti
University of Padua, Italy
conti@math.unipd.it

ABSTRACT

Reflection is a language feature which allows to analyze and transform the behavior of classes at the runtime. Reflection is used for software debugging and testing. Malware authors can leverage reflection to subvert the malware detection by static analyzers. Reflection initializes the class, invokes any method of class, or accesses any field of class. But, instead of utilizing usual programming language syntax, reflection passes classes/methods etc. as parameters to reflective APIs. As a consequence, these parameters can be constructed dynamically or can be encrypted by malware. These cannot be detected by state-of-the-art static tools. We propose EspyDroid, a system that combines dynamic analysis with code instrumentation for a more precise and automated detection of malware employing reflection. We evaluate EspyDroid on 28 benchmark apps employing major reflection categories. Our technique show improved results over FlowDroid via detection of additional undetected flows. These flows have potential to leak sensitive and private information of the users, through various sinks.

Keywords

Dynamic Analysis; Instrumentation; Malware; Reflection; Android

1. MOTIVATION FOR WORK

- Faruki et al. identified reflection as one of the major stealth technique used by Android malware to evade static analysis techniques [6].
- Through an analysis spanned across four years, Andrubis [9] reported that reflection is employed by 57.08% of Android malware samples.
- Highly advance malware families like OBAD, FakeInstallar etc. make use of Java Reflection APIs and nested methods.

Moreover, Android provides Inter-Component Communication (ICC) feature for communication among components of application. `Intent` is a message passing object to request an action from a component (from same or different app) to facilitate ICC. Unfortunately, malware authors are misusing the feature to distribute the leaks over multiple components of given apps [10]. Further, not only classes or methods can be called through reflection but Intents also. Detection and analysis of these malware require ICC supported analysis approach.

```
1 String cls="android.Telephony.
      TelephonyManager";
2 String reverse = new StringBuffer("
      dIeciveDteg").reverse().toString();
3 Class c = Class.forName(cls);
4 tm = (TelephonyManager) this.getSystemService
      (Context.TELEPHONY_SERVICE);
5 Method method = c.getMethod(reverse, new Class<?>[0]);

6 String id=(String) method.invoke(tm);
7 Intent i=new Intent(this, Activity2.class);
8 i.putExtra("imei", id);
9 startActivity(i);

                MainActivity

10 String value= getIntent().getExtras().
      getString("imei");
11 SmsManager sm = SmsManager.getDefault();
12 sm.sendTextMessage(phoneNo,null,value,null,
      null);

                Activity2
```

Listing 1: Reflective APIs (Highlighted) which can be exploited for information leaks

Listing 1 shows the code of `Onlytelephony_reverse` app from the well-known DroidBench benchmark [2]. This app uses reflective APIs (lines highlighted) and ICC (lines 7-9) to leak the sensitive data, e.g., IMEI here. The class `android.Telephony.TelephonyManager` define the method `getDeviceId()`. `MainActivity` instantiates the object of this class using `Class.forName` API (Line 3). The example not only uses reflection but also constructs method-name dynamically by using the `reverse()` function (Line 2). Then, it passes this dynamically constructed method-name to `getMethod()` API (Line 5) which creates the method object.

ASIA CCS '17 April 02-06, 2017, Abu Dhabi, United Arab Emirates

ACM ISBN 978-1-4503-4944-4/17/04.

DOI: http://dx.doi.org/10.1145/3052973.3055162

Finally, it invokes the specified method using reflection API `invoke()` (Line 6). Furthermore, `MainActivity` passes the IMEI value to `Activity2` using `Android Intent` (lines 7-9), and `Activity2` performs the leakage of IMEI using SMS. The method `reverse()` here is just a motivating example, where in other cases more complicated methods could be applied such as encryption, substring, concatenation etc. to subvert static analysis.

We analyzed this application on FlowDroid [5], IccTA, AmanDroid, and DroidSafe static analysis tools developed for detection of privacy leaks. None of these tools were successful in identifying the leak. Also, we analyze the application using DroidRA [8] which is reflection-aware static analysis approach. However, DroidRA also fails to identify the leak here as the app constructs the method name at runtime. The dynamic construction of parameters of reflective APIs is very trivial and can be done using various ways like concatenation, encryption, and substring generation.

2. CONTRIBUTIONS

Our proposed technique is a work in progress with the following contributions:

- We propose EspyDroid, a system that combines static and dynamic analysis to unfold the hidden leaks performed by the app using reflection. Runtime monitoring makes EspyDroid capable of resolving reflection when the arguments of reflective APIs are encrypted, obfuscated or run-time dependent.
- EspyDroid is able to detect leaks distributed over multiple components through Intents. Also, when Intents themselves are called through reflection.
- We tested EspyDroid on widely used DroidBench benchmark and similar work DroidRA.

3. PROPOSED SOLUTION: EspyDroid

The aim of EspyDroid is to unravel leaks in the presence of reflection directly on app bytecode to obliviate the need of source code for this analysis.

3.1 Overall Solution

The overall architecture of EspyDroid is as shown in Figure 1. The complete system of EspyDroid consists of three main modules: `Dynamic Analyzer`, `Log Tracer`, `Instrumentation Agent`.

The `Dynamic Analyzer` module uses APIMonitor [1] which repackages the app to add monitoring code for the specified reflection APIs and then executes the app and collects logs. In this way, it assists in resolving the reflective APIs and associated parameters (Step 1). `Log Tracer` traces the logs and prepares processed input (Step 2) for `Instrumentation Agent` that generates equivalent non-reflective statement(s) for each reflective statement and instruments at appropriate point (Step 3). The `Instrumentation Agent` is developed in Java and performs instrumentation in Jimple, the intermediate code representation of Soot framework [4]. The aim of instrumentation is to prepare enhanced intermediate code (thus an enhanced app) with resolved reflective calls so as to enable precise and accurate taint propagation. For automatic UI exploration of app, Intelligent UI exploration module from [7] is used which is black-box testing approach extended using Robotium framework [3].

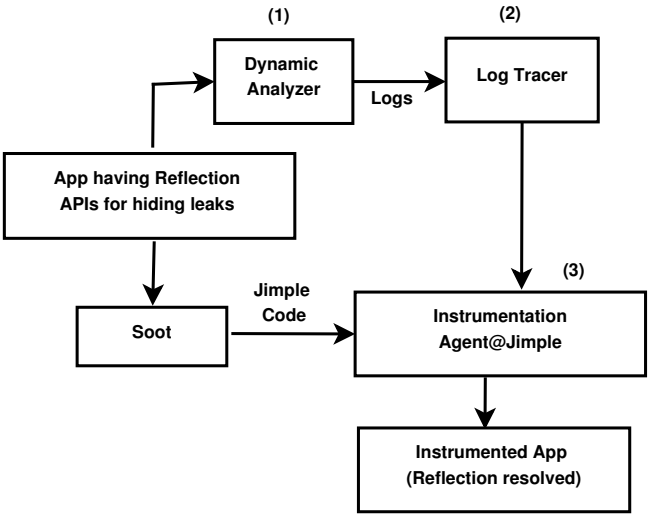

Figure 1: EspyDroid System Architecture

3.2 Illustrative Example

Figure 2 shows the Jimple code snippet of `MainActivity` corresponding to motivating example of Listing 1. The corresponding instrumented code with non-reflective statements is also shown. The three reflective APIs (highlighted), and the corresponding instrumented statements are shown in Figure 2. For the example, two non-reflective statements are constructed. One instrumentation is corresponding to reflectively instantiating class (Line 7) which is useful for inferring the method name and other instrumentation is corresponding to reflectively invoking the method (Lines 10 and 14). The illustrative example explains the approach of Instrumentation Agent with one of the representative app. Some of the representative samples access Intents class and its methods also through reflective APIs. The way Instrumentation Agent is designed handles the instrumentation of Intent-based reflections similarly.

4. EXPERIMENTS

We use 17 DroidBench, 9 DroidRA, two modified versions of DroidRA samples as representative dataset. The major usages of reflection in the representative dataset are:

- Reflective class instantiations to hide the class names that perform sensitive leaks.
- Constructor-based reflective class instantiations to hide the class names that perform sensitive leaks.
- Reflective invocations of methods to hide the malicious methods invoked.
- Field-based reflective accesses to set the class fields with the malicious information to be leaked.
- All above cases with leaks in single component or distributed over multiple components.
- Source and Sink methods accessed by reflective instantiation of their classes to hide the classes and methods accessing sensitive information or leaking it.
- Intent-based reflective access to hide the communication between two components.
- Encrypted/Obfuscated parameters of reflective APIs to fail even reflection aware static analysis based techniques.

EspyDroid is able to identify a number of distinct leakage

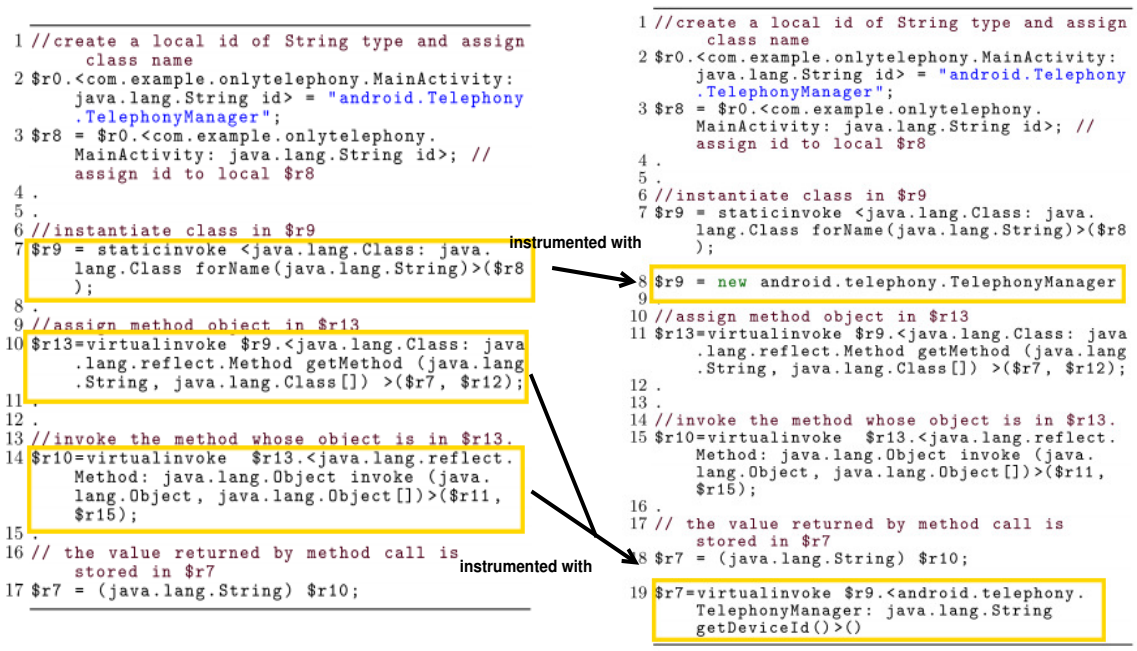

Figure 2: Jimple code of Listing 1, and the instrumented Jimple Code

paths which are not captured by FlowDroid alone. The results show a large improvement in identification of leakages and hence, lead to reduction of false negatives. In the original representative dataset, FlowDroid could identify 19 leakage paths while the same tool could identify 47 leakage paths in the instrumented apps when used in conjunction with EspyDroid. Not only the leakage paths, but the number of identified sources and sinks is also improved. FlowDroid could identify 143 sources and sinks in original dataset while FlowDroid could identify 173 sources and sinks on instrumented dataset. The results demonstrate that the precision of static analysis tools get improved on taking instrumented apps as input instead of original reflection-employed apps.

5. CONCLUSIONS

We propose EspyDroid, a reflection aware technique which is resilient against encryption, obfuscation or run-time dependency of reflection APIs parameters. The results on small representative dataset demonstrate that static analyzers results in large false negatives in the presence of reflection. To improve upon these missed leaks, EspyDroid proposes an hybrid approach to improve false negatives. EspyDroid is a modular work in progres which shall be expanded for handling more advanced cases of reflection.

6. ACKNOWLEDGMENTS

This work is partially supported by Security Analysis Framework for Android Platform (SAFAL, Grant 1000109932) by Department of Electronics and Information Technology, Government of India. The work is also partially supported by DST-CNRS project IFC/DST-CNRS/2015-01/332 at MNIT Jaipur.

7. REFERENCES

[1] *APIMonitor*. https://code.google.com/archive/p/droidbox/wikis/APIMonitor.wiki.

[2] *DroidBench*. https://github.com/secure-software-engineering/DroidBench/tree/develop.

[3] *Robotium*. https://github.com/RobotiumTech/robotium.

[4] The soot framework for java program analysis: a retrospective. In *Cetus Users and Compiler Infastructure Workshop (CETUS 2011)*, 2011.

[5] S. Arzt et al. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. *ACM SIGPLAN Notices*, 2014.

[6] P. Faruki et al. Android security: a survey of issues, malware penetration, and defenses. *IEEE Communications Surveys & Tutorials*, 17(2), 2015.

[7] J. Gajrani et al. spectra: a precise framework for analyzing cryptographic vulnerabilities in android apps. In *to be published in 2017 14th IEEE Annual Consumer Communications & Networking Conference (CCNC)*. IEEE, 2017.

[8] L. Li et al. Droidra: Taming reflection to support whole-program analysis of android apps. In *Proceedings of the 25th International Symposium on Software Testing and Analysis*. ACM, 2016.

[9] M. Lindorfer et al. Andrubis - a tool for analyzing unknown android applications. 2014.

[10] D. Octeau et al. Effective inter-component communication mapping in android: An essential step towards holistic security analysis. In *Presented as part of the 22nd USENIX Security Symposium (USENIX Security 13)*, 2013.