Poster: Automated, Context-Sensitive Analysis of iOS Applications

Dennis Tatang
Horst Görtz Institute for IT-Security (HGI)
Ruhr-Universität Bochum
Email: dennis.tatang@rub.de

***Abstract*—The security of mobile phones and the surrounding ecosystem has attracted a lot of research in the last years. While lots of work has been performed for the Android platform, the security of iOS apps has yet not been explored as much.**

In this paper, we present an automated analysis method for iOS applications with a focus on code coverage. More specifically, we introduce an analysis method that enables a context-sensitive analysis of input fields in order to bypass registration forms and similar user interface elements with the goal of covering more code paths. For this purpose, static and dynamic program analysis methods are used to automatically detect and handle such UI elements. We built a prototype of the proposed method based on the analysis platform DiOS [1]. Our preliminary results based on an analysis of 25 apps from the official Apple App Store suggest that our approach enables on average a 16% increase in code coverage.

I. Introduction

Mobile devices have become an important part of our lives. We use smartphones for private purposes, but also for business aims. Hence, a large amount of private and sensitive information is stored on these devices. For this reason it is important to know what kind of privacy- and security-related data is being collected and processed by individual apps. While lots of work in this area was performed for the Android platform, we focus in this paper on Apple's iPhone and the surrounding ecosystem. Apple delivers the operating system iOS with several security features (e. g., privilege separation, code signing, and sandboxing) [2]. Data manipulations outside of the sandbox are only possible via API calls. As a result, monitoring of these requests is important to understand the behavior of a given app. Note that the use of sensitive API methods must be approved by a user since iOS 6, published in September 2012 [3]. Nevertheless, apps often receive many rights which are not needed for their basic functionality (e. g., freeware apps often try to monetize privacy-related information on a smartphone).

Several analysis tools for iOS apps are available [4]–[6]. There is only one automated, dynamic analysis tool by Kurtz et al. [4] called DiOS which, however, is not able to analyze iOS apps in a context-sensitive way. This leads to limitations when an application requires prior registration actions and other kinds of user interactions. In these cases, the dynamic analysis executes only a limited number of functions and the code coverage is rather low.

In this paper, we extend the work of Kurtz et al. on DiOS [4], a practical system to perform privacy analysis of iOS apps. DiOS provides a highly scalable and fully automated solution to schedule apps from the official Apple App Store for privacy analysis on iOS devices. It uses the *UIAutomation Framework* provided by Apple [7] to automate the analysis on the GUI layer. The framework is a JavaScript library and with the aid of this framework, it is possible to write scripts to perform automated iOS app testing.

II. Contribution

The main deficiency of DiOS in automation capabilities is the missing handling of input fields for registration or authentication. This can cause limitations with respect to the analysis results, for example this lack may lead to incomplete results: if DiOS cannot bypass such an authentication, many functions will not be executed and the analysis report might lack important details. Accordingly, not all sensitive API calls are recorded and analyzed. In this paper, we show that the assumption made by Kurtz et al. that applications usually request all rights on the initial run [4] is not fully correct. Hence, the objective of our work is to execute as much code as possible during the analysis phase.

The code coverage is of particular importance, especially for dynamic analysis, because the higher the code coverage, the better the analysis report since more functions have been executed and probably inspected. As long as the task is to analyze apps without any registration or authentication forms, DiOS will provide good results for the code coverage. Some very popular categories of apps, however, usually need an authentication phase (e. g., shopping, social networking, or banking apps). In addition, privacy-related information is usually used in these apps. Therefore, our contribution is important for analysis of apps which need a prior login.

A. Analyzing social networking apps

Since social networking apps normally need some kind of login phase, the code coverage of a DiOS analysis run for this kind of apps is lacking. Initial investigations confirmed this assumption. Another reason why we have decided to analyze social networking apps is that a short survey carried out as part of this work confirmed that the importance of social networks is high: nearly 90 % of respondents are using at least one social networking app. Therefore, we analyzed the Apple App Store Top 25 social networking apps in order to implement the DiOS extension. Static analysis of these apps show that we can find

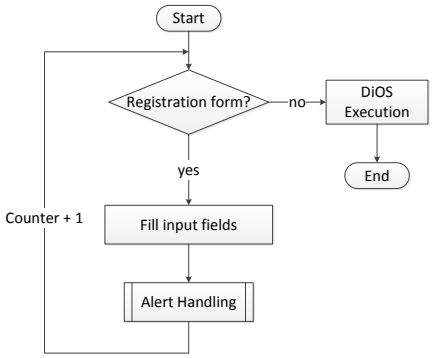

Fig. 1. Schematic overview of handling authentication forms

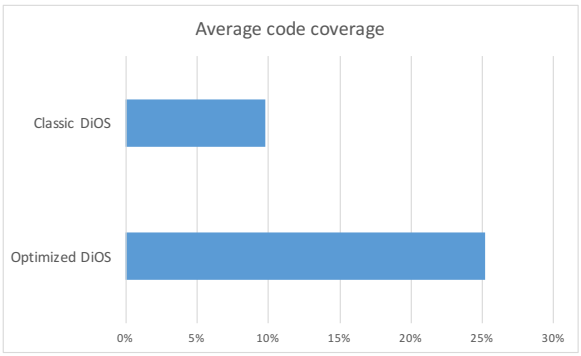

Fig. 2. Comparison of code coverage

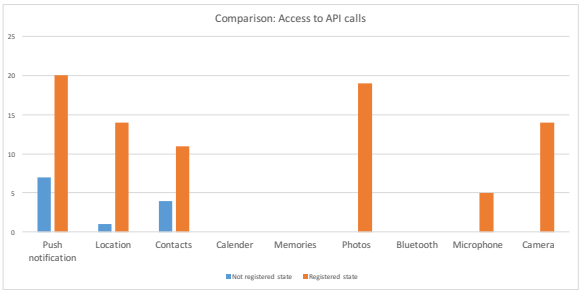

Fig. 3. Registered API calls

primarily four different kinds of input fields: `TextField`, `SecureTextField`, `Picker`, and `TextView`.

B. DiOS Extension

Figure 1 presents the procedure. The idea of our DiOS extension is that it can automatically detect input fields for registration or authentication in an active view. Furthermore, it recognizes what kind of input field is present. After that it will fill the forms with matching, valid credentials, in order to bypass any registration and authentication forms.

Our implementation recognizes input fields with the aid of the *UIAutomation Framework*. It is possible to output all UI elements of an active view. After this we check the types of the input fields. For each field, its own handler was implemented. For example, a `SecureTextField` is usually used for entering a password, thus we enter our password information when detecting this kind of input field.

III. Evaluation

A first implementation, where previously created accounts will be tried for every kind of input field, showed that we can achieve a better code coverage if we successfully log in. In short, we confirmed that dynamic analysis of apps will reach better results if they are in a registered state. We compare the classic DiOS version with the optimized DiOS version. One of the most important parameter is, as already mentioned, the code coverage. The results demonstrate that, if we analyze an application in a registered state, we reach a better code coverage. Figure 2 illustrates this behavior.

We have achieved an increase in code coverage of about 16 %. This result is consistent with other works that deal with the code coverage of dynamic iOS app analyzes [4]. Interesting for privacy purposes are the executed API calls. For app analysis that require a login, we have noticed a great difference between the actually executed calls in a registered (blue) or a not registered state (orange), see Figure 3.

IV. Conclusion

The main part of the work was the static and dynamic analysis of 25 iOS social networking apps in order to improve DiOS. In the first step, we have confirmed that social networking apps need a login and that nearly everyone is using a social network. After that, we statically analyzed the fundamental construction of iOS applications and showed that dynamic analyses of iOS apps in a registered state deliver more accurate results, especially privacy aspects of different API calls change decisively. The assumption from previous work that sensitive API calls are executed immediately after starting an application was refuted for social networking apps.

The context-sensitive filling of all kind of input fields can be improved by taking not just the type, but also the content into account. Furthermore, we could check which content of input fields is privacy-related. The last goal for future work is a GUI automation of a full registration without any previously created user credentials.

References

[1] “DiOS-Analysis,” 2015, ”https://github.com/DiOS-Analysis”.
[2] C. Miller, D. Blazakis, D. DaiZovi, S. Esser, V. Iozzo, and R.-P. Weinmann, *iOS Hacker’s Handbook* -. New York: John Wiley & Sons, 2012.
[3] Apple, “iOS 6.0,” 2016, ”https://developer.apple.com/library/prerelease/ios/releasenotes/General/WhatsNewIniOS/Articles/iOS6.html”.
[4] A. Kurtz, A. Weinlein, C. Settgast, and F. Freiling, “DiOS: Dynamic Privacy Analysis of iOS Applications,” FAU Erlangen-Nürnberg, Dept. of Computer Science, Tech. Rep. CS-2014-03, June 2014.
[5] A. Kurtz, M. Troßbach, and F. Freiling, “Snoop-it: Dynamische Analyse und Manipulation von Apple iOS Apps,” in *Sicherheit 2014 – Sicherheit, Schutz und Zuverlässigkeit*, ser. Lecture Notes in Informatics, Bonn, 2014.
[6] M. Egele, C. Kruegel, E. Kirda, and G. Vigna, “PiOS: Detecting Privacy Leaks in iOS Applications,” in *Network and Distributed System Security Symposium (NDSS)*, 2011.
[7] Apple, “UI Automation JavaScript Reference for iOS,” 2012, ”https://developer.apple.com/library/ios/documentation/DeveloperTools/Reference/UIAutomationRef/”.