Lalaine: Measuring and Characterizing Non-Compliance of Apple Privacy Labels

Yue Xiao, Zhengyi Li, and Yue Qin, *Indiana University Bloomington;* Xiaolong Bai, *Orion Security Lab, Alibaba Group;* Jiale Guan, Xiaojing Liao, and Luyi Xing, *Indiana University Bloomington*

https://www.usenix.org/conference/usenixsecurity23/presentation/xiao-yue

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 32nd USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 32nd USENIX Security Symposium.

August 9–11, 2023 • Anaheim, CA, USA

978-1-939133-37-3

USENIX'23 Artifact Appendix: Measuring and Characterizing Non-Compliance of Apple Privacy Labels

Yue Xiao[1], Zhengyi Li[1], Yue Qin[1], Xiaolong Bai[2], Jiale Guan[1], Xiaojing Liao[1], Luyi Xing[1]

[1]Indiana University Bloomington, [2]Orion Security Lab, Alibaba Group

A Artifact Appendix

A.1 Abstract

[Mandatory] The downloader is utilized to download the app along with its corresponding privacy label. The static analyzer is used to screen apps that make calls to iOS system APIs. The dynamic analysis pipeline is employed to verify whether an app's code behavior complies with its privacy label. To use the tool, you need to provide the binary code of the app in .ipa format, as well as its privacy label from the Apple store (if it is not present in our 366,697 app privacy label dataset). The tool will then identify and output any inconsistencies it detects. The dynamic analysis pipeline is composed of three stages:

- *End-to-end execution, which includes fully automated app UI execution, dynamic instrumentation, and network monitoring.*
- *Inferring data and purpose from the call trace and network traffic information.*
- *Conducting a compliance check to identify any inconsistencies*

A.2 Description & Requirements

[Mandatory] To utilize the tool, the system requirements include Mac OS and a rooted iOS device. We have tested the tool on Mac OS version 12.6.2, which is the minimum version we recommend. Additionally, the iOS device must be running version 12.2 or higher, although lower versions may also be compatible with the tool. We have only listed the minimum versions we have tested, but other versions may still be compatible.

A.2.1 Security, privacy, and ethical concerns

[Mandatory] Rooting an iOS device can create security, privacy, and ethical issues. It grants administrative access to the device's operating system, enabling customization and control over the device's functionality but bypassing built-in security features, which may expose the device to security threats. Rooting can also compromise privacy, granting unauthorized access to personal information and data, particularly when installing third-party software from untrusted sources. Additionally, rooting violates Apple's terms of service, which may lead to legal consequences and could undermine the efforts of developers and manufacturers to create secure devices.

A.2.2 How to access

[Mandatory] You can access the source code in github: https://github.com/xiaoyue10131748/Lalaine/tree/LalaineStable

A.2.3 Hardware dependencies

[Mandatory] The tool requires a rooted iOS device to run, which may present a hardware dependency issue.

A.2.4 Software dependencies

[Mandatory] We use Macaca, an open-source automation testing framework that supports different types of applications and provides automation drivers, environment support, peripheral tools, and integration solutions to handle challenges such as test automation and client-side performance. We also set up NoSmoke, a cross-platform UI crawler that scans view trees, performs OCR operations, and creates and runs UI test cases.

- install macaca https://macacajs.github.io/guide/environment-setup.html#macaca-cli
- install nosmoke https://macacajs.github.io/NoSmoke/guide/

We utilize Frida, a dynamic code instrumentation toolkit. We inject snippets of JavaScript into native apps on iOS. We built our hooking framework on top of the Frida API.

- install Frida's CLI tools on MacOS: https://frida.re/docs/installation/

- configure Frida on your rooted iOS device: `https://frida.re/docs/ios/`

We utilized Fiddler, which is a web debugging proxy tool that monitors, analyzes and modifies the traffic on iOS device.

- install Fiddler in your MacOS: `https://docs.telerik.com/fiddler/configure-fiddler/tasks/configureformac`
- configure your rooted iOS device: `https://docs.telerik.com/fiddler/configure-fiddler/tasks/configureforios`

A.2.5 Benchmarks

[Mandatory] The privacy label of apps we crawled from app store are needed to put it under data folder. Please download it from `https://drive.google.com/file/d/1k3FulkLvOhgLV_hU-hkxnuvnP4FF3tXz/view?usp=share_link`. If the app you want to test is not on the list, you can mannully add it to this file to allow further complaince check.

A.3 Set-up

[Mandatory] This section should include all the installation and configuration steps required to prepare the environment to be used for the evaluation of your artifact.

A.3.1 Installation

[Mandatory] Please download the source code `https://github.com/xiaoyue10131748/Lalaine.git` and follow the README to setup environment.

A.3.2 Basic Test

[Mandatory] Check that Macaca, Frida and Fiddler are successfully installed. First, run the command `nosmoke -u <device id>`, to see if nosmoke can launch an app and automatically execute UI events. Second, run the command `frida-ps -U`, to see if the frida can list all the apps installed on the iPhone. Third, launch Fiddler and open any app on the iPhone to see if the traffic generated by the app can be captured.

A.4 Evaluation workflow

[Mandatory for Artifacts Functional & Results Reproduced, optional for Artifact Available] This section should include all the operational steps and experiments which must be performed to evaluate if your your artifact is functional and to validate your paper's key results and claims. For that purpose, we ask you to use the two following subsections and cross-reference the items therein as explained next.

A.4.1 Major Claims

[Mandatory for Artifacts Functional & Results Reproduced, optional for Artifact Available] Enumerate here the major claims (Cx) made in your paper. Follows an example:

(C1): *The tool is able to download the binary of app and its privacy label.*

(C2): *The tool is able to screen apps that call sensitive iOS system APIs.*

(C3): *The tool is able to gather call trace and network traffic by dynamically executing an app in rooted device.*

(C4): *The tool is able to analyze call trace and network traffic to extract (data, purpose) from code behavior.*

(C5): *The tool is able to perform complaince check.*

A.4.2 Experiments

[Mandatory for Artifacts Functional & Results Reproduced, optional for Artifact Available]

(E1): *[download app binary and crawl privacy label] [1 human-minutes + 3-4 compute-minutes + 10GB disk]: In this step, the tool will crawl privacy label from apple store and download its binary ipa file.*
How to: Collect the info of app you want to test. And execute the following down-loader command.
Preparation: *Put the information of the app that you want to download in app_info.json*
Execution: (1) `python privacy_label_crawler.py -input_file ./app_info.json -result_dir ./label/ -driver_path ./chromedriver` (2) `python app_binary_downloader.py -input_file ./app_info.json -result_dir ./ipa/`
Results: *The app binary will be in the folder /ipa/ and the privay label will be in the folder /label/*

(E2): *[screen apps binary ipa file] [0.5 human-minutes + 0.5 compute-minutes + 10GB disk]: In this step, the tool will screen apps that make calls to iOS system APIs.*
How to: Execute the following static analyzer (SAF) command.
Preparation: *Put the binary of app (.ipa) you want to static scan under the folder /app*
Execution: (1) `python find_in_decrypted_ipas.py -f ./API_List.txt -i ./app/`
Results: *The results will be in the file find_in_decrypted_ret.txt*

(E3): *[call trace and traffic gathering] [1 human-minutes + 3-4 compute-minutes + 10GB disk]: In this step, the tool will gather call trace and network traffic by dynamically executing an app in the rooted device.*
How to: *This step will take 3-4 mins. It takes three steps: (1) launch the iPhone, network monitor, and Macaca server for the reviewers. (2) the reviewers run the command `python batch_ui_frida_test.py 0 . -i`*

`<device id>` *(3) close the iPhone and macaca server, dump the traffic from the network monitor*

Preparation: *Make sure the iPhone, network monitor and Macaca server are launched.*

Execution: `python batch_ui_frida_test.py 0 . -i <device id>`

Results: *The results about are under* `result/0/` *folder.*

(E4): *[(data, purpose) inference] [0 human-hour + 0.5 compute-minutes]: Analyze call trace and network traffic to extract (data, purpose) from code behavior.*

How to: *Execute the following command.*

Execution: `python analyze_log.py 0 .`

Results: *Analyzing result can be found in* `./result/0/prediction_output/`

(E5): *[Compliance check] [0 human-hour + 0.5 compute-minutes]: Perform compliance check to find any inconsistencies between (data,purpose) extracted from call trace and network traffic and privacy label in its privacy label.*

How to: *Execute the following command.*

Execution: `python compliance_check.py 0 .`

Results: *Analyzing result can be found in* `./result/0/inconsistency_output/`