MAD-IOS: dynamic app vulnerability analysis in non-jailbroken devices

Alfonso Solimeo[1], Luca Capacci[2], Stefano Taino[2], Rebecca Montanari[1]
[1]: University of Bologna, Italy, alfonso.solimeo@studio.unibo.it, rebecca.montanari@unibo.it
[2]: CryptonetLabs, Italy, luca.capacci@cryptonetlabs.it, stefano.taino@cryptonetlabs.it

Abstract

Mobile apps are pervasive in our life supporting us from simple actions, such as photo sharing, to more important ones, such as banking transactions. Security around these operations and data is crucial, making app vulnerability analysis and code review fundamental. Android and iOS split the mobile market share each other. However, while the first can rely on many analysis tools, for iOS it is not the same. Not only there is erroneously the idea about the immunity of iOS from malware and bad coding, but also it is challenging to *jailbreak* iOS devices. In this paper, we present MAD-IOS, a novel framework for dynamic iOS app vulnerability analysis that does not rely on jailbreaking techniques, making it possible to work also for non-jailbroken devices. Exploiting dynamic analysis and without breaking iOS security model, it is possible to embrace iOS-based devices audience as wide as possible and to provide a security assessment through a normal use of the app.

1 Introduction

In recent years, the adoption of mobile devices has exponentially increased. By 2021 estimated devices per capita are expected to be 1.5 and mobile data traffic will reach the amount of 49 exabytes [1]. Within this context the applications (app), running on major mobile operating systems (Android and iOS), are playing a crucial role in leveraging the widespread use of smartphones. App development is gradually becoming easier and the number of developers is growing substantially with 2.7 million and 2.2 million apps, respectively in the Play Store [2] and in the App Store [3] published at the beginning of 2017.

Unfortunately, developers' security expertise has not grown equally. Increased popularity of Android/iOS devices along with the capability of apps of manipulating and working on various personal and sensitive data and associated monetary benefits have attracted a great number of malware developers to target mobile platforms. Kaspersky Lab, in 2016, detected more than 8 million malicious installation packages and almost four hundred thousand among mobile banking trojans and ransomware [4].

Given the threats posed by the increasing number of malicious apps, academia and industry researchers have proposed methodologies and techniques for automated analysis of app vetting and for malware detection. In particular, methodologies for malware detection can be based on either *static* or *dynamic* analysis. Static analysis consists of inspecting the entire code without executing the

app, in a non-runtime environment. This inspection can take place on the source code, when it is available, or on bytecode/binaries, making use of techniques, such as reverse engineering, control flow analysis and permission checking. On the other hand, analyzing the actions performed by an app code while it is being executed is called dynamic analysis. App code inspection can take place by modifying the executable or using a specific environment. Although static analysis is an optimal first step during a code review process and is useful for resource constrained devices, it does not evaluate execution context and all its related aspects: multiple entry points, asynchronously running app components, callbacks [5], data dynamically downloaded, objects created during runtime [6], etc. Moreover, in many cases the binaries may be obfuscated or the source code may be not available.

Because of the widespread adoption of Android and its high degree of openness compared to iOS, the Google's platform is the most attacked [7] and therefore, great research efforts have been directed toward the design/development of tools/techniques for supporting Android malware analysis. In the field of static analysis many tools are available, such as *RiskRanker*, *FlowDroid*, *AmanDroid*, whereas in the field of dynamic analysis we have *AppsPlayground*, *VetDroid*, *KBTA IDS*, *Crowdroid* [5].

Unfortunately, the same attention has not been directed toward iOS devices [8] because of both the market share and of the wrong idea that iOS less susceptible by malware infection. On the contrary, malware on iOS devices continues to increase [9]. Few analysis tools are available and the iOS world also witnesses the lack of auditing tools and techniques, which would be fundamental to perform accurate penetration tests and to mitigate the unsecure programming methods that can be present also in apps that have passed the Apple checks. Some remarkable works on iOS side are: *iRET* [10], *SnoopIt* [11] and *Introspy-iOS* [12]. iRET provides static analysis features, such as binary analysis, keychain analysis and Theos tweaks management. SnoopIt and Introspy-iOS provide dynamic analysis support, tracing iOS sensitive calls and collecting various kinds of data. All of these solutions share a common characteristic, they work only on *jailbroken* devices. Jailbreaking allows to remove sandbox restrictions via privilege escalation by modifying the iOS system kernel in order to permit reading/writing access to file system. The privilege escalation allows custom software installation and device behavior modification. Jailbroken devices can bypass, disable or patch the code-signing mechanism that otherwise would allow to load only Apple-approved applications.

However, jailbreaking techniques are becoming increasingly difficult day by day. On the one hand, the current iOS release is 11.2 and there are no jailbreaks available for iOS 11.2. We have to go back in time to version 10.2, thirteen versions before, to find the latest jailbreak which is, by the way, only a *semi-untethered*[*] jailbreak. On the other hand, we have to consider also penetration tests in corporate environment that can have more complications or bans in performing the jailbreak. There is now a common agreement that jailbreaking cannot be considered a reliable procedure to rely on for app analysis.

In this demo paper, we present a novel dynamic app vulnerability analysis framework, called **MAD-IOS (M**obile **A**pp **D**riller for **iOS),** that, unlike state-of-the-art solutions, does not rely on jailbreaking techniques to profile app secure assessment, thus avoiding the breakage of the iOS Security Model. MAD-IOS provides also a non-invasive technique to trace sensitive calls: the entire analysis work can be performed during the normal use of the app, without any particular actions from end-user side. In this way, MAD-IOS execution is independent from possible jailbreak releases and all analysis operations can be conducted on pure iOS devices, reproducing the natural use, avoiding any contaminations due to the breakage of iOS Security Model.

[*] With a semi-untethered jailbreak, on every device reboot, you have to repeat the jailbreak procedure, usually via a specific app.

2 MAD-IOS Architecture

MAD-IOS is a dynamic app vulnerability analysis framework, developed to work with every kind of device powered by iOS. Essential feature is the capability to work with non-jailbroken devices. It is composed of some existing tools and by specific *hooks*[†], built on top of *Theos* [13], *Theos Jailed* [14] and *Fastlane* [15]. The basic idea is to provide a tool able to instrument a decrypted iOS application Archive (IPA) to intercept and collect data from sensitive calls, during its normal use, such as hashing functions, cryptographic key generation, keychain items storing, URLs creation, database calls. The IPA can be installed only on iOS devices and contains all the iOS application components: binaries, images, meta-data, etc.

Figure 1 shows MAD-IOS architecture composed of several modules.

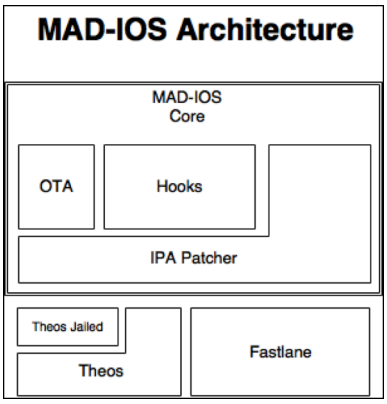

Figure 1 - MAD-IOS Architecture

IPA Patcher module comprises several bash-scripts, that automate the patching and making of instrumented IPAs. Its task is to make the *tweak* and inject it into the IPA. Usually tweaks refer to extensions for jailbroken devices, provided individually or as additional code injected into existing apps, that modify the normal behavior. Our MAD-IOS tweaks are, instead, designed for non-jailbroken devices and without any impact on the app. We consider the use of bash-scripts a good choice not only because all the involved tools are command-line tools but also because they provide good performances and quick and easy configuration.

IPA Patcher uses two additional modules: *Hooks* and *OTA*. The Hooks module is composed of many theos-based hooks. They leverage both fishhook [16] and *Logos* [17], that provide method hooking mechanisms, and they compose the code to inject, aka the tweak. OTA module, instead, permits to generate what it is necessary to implement an Over The Air Installation of patched IPA, to make a remote installation via *www* possible: a *plist* file and a related web-page. They need to use the *itms-services* protocol, which is the official Apple way for OTA distribution of iOS apps [18] since iOS 7 to the latest iOS 11 release.

MAD-IOS Core takes advantage of Theos, Theos Jailed and Fastlane. Theos is "a cross-platform suite of tools for building and deploying software for iOS and other platforms" [13] widely used to build tweaks for jailbroken devices. Theos Jailed is a module to develop tweaks in a jailed environment, which meets our main goal. Fastlane helps developers to automate the building of iOS apps: we are using it to manage all issues related to the *provisioning profile*, an Apple certificate that is essential to implement the code-signing mechanism in iOS.

[†] In this context, we intend for *hook* a code block designed to intercept a specific function.

3 MAD-IOS at work

MAD-IOS operates on a decrypted IPA and makes use of an Apple Developer Profile. Let us recall that Apple encrypts any app released on the App Store, and Theos-based code injection is possible only on decrypted IPAs. The Apple Developer Profile is, instead, needed to sign and install the app correctly. Since this tool is conceived for the target app's developer and for pentesters authorized by the app's developer, gaining access to a non-encrypted IPA of the target app is not an issue.

The operating principle underlying MAD-IOS is quite simple. Once obtained a decrypted app we first need to setup *env_var.sh*, an environment variable file, that contains some information such as *Apple Developer Username*, *App Group ID*, Theos working path, *Device UDID* and project name. Then, by running *init.sh* we produce a patched IPA, correctly signed and ready to be installed on the selected device. The patched app will be the same as the original one, with only a different *Bundle ID* which we chose to call *it.cryptonetlabs.<Project Name>*.

When running, MAD-IOS intercepts a big variety of sensitive calls on the device. Data collected are stored in a *SQLite Database* linked to the single app. The information is stored following the schema *Library – Function – Args – dataOut – Stacktrace – timestamp*. *Library* and *Function* will be respectively the name of the class or library (e.g. *NSURLRequest*, in case of Apple API call, *SQLite* in case of an external library) and the name of the method or function intercepted (e.g. *initWithURL* for *NSURLRequest*, *sqlite3_open_v2* for *SQLite*). *Args* will be the arguments related and *dataOut* will be the return value, if present. The *Stacktrace* and *timestamp* fields save calls chain that pointed to the intercepted function and the timing of the call. Figure 2 shows some database entry examples.

	library	function	args	dataOut	stackTrace	timestamp
	Filter	Filter	Filter	Filter	Filter	Filter
14	Randomization Services	arc4random_uniform	upper_bound: 2147483646	722663544	(0 [redacted].dylib ...	1505498208.0...
15	SQLite	sqlite3_open_v2	filename: /var/mobile/ Containers/Data/Applicatio...	0	(0 [redacted].dylib ...	1505498208.3...

Figure 2 - Database Entries

The hooks cover many calls that are relevant for security: cryptography (*MD5, Sha, RNG, ...*), networking (*NSURL, CFNetwork, ...*), storage (*Keychain, SQLite, ...*), logging (*NSLog, ...*), IPC (*URL Scheme, UIPasteboard, ...*). Every app call to a target function is intercepted by MAD-IOS. After saving the call-related data MAD-IOS returns the control without significantly decreasing the app performance, as a key characteristic. In order to share this database, without breaking the Apple Security Model, we used the *App Group* capability thus enabling data sharing. Another app developed by the same developer can retrieve the database. This second app sends the result to the *MAD backend* to review it. The backend consists of a web server, which automatically parses the collected data and shows the results in a human-readable format. Collected data is structured in such a way as to be easily analyzed by automatic tools, thus allowing the analyst to easily develop scripts to display the results of the analysis in any format (e.g. html, docx ...). For instance, in our environment, we developed a python-based application on the backend, which analyzes the data collected and generates an HTML-based report. The results are classified according to the *OWASP Top Ten Mobile Risks* [19] using four severity levels: *High*, *Medium*, *Low* and *Information*.

Furthermore, injected hooks can intercept any function/method in the app's sandbox so false positives/negatives and the overall result of the analysis are not influenced by the limits imposed by iOS security model. The only downside in working on jailed devices consists of an incomplete filesystem visibility, which is limited to the very same paths accessible to the target app, but, once again, this does not influence the results, since during the analysis we are only interested in files read and written by the target app. Figure 3 exhibits some details about the generated report. MAD-IOS enables users to detect, for example, if there are HTTP calls that are not using SSL. Figure 4 highlights the advantage of relying on a dynamic approach, by showing the interception of the

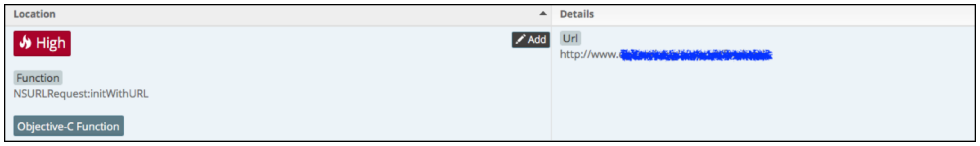

Figure 3 - NSURLRequest detail

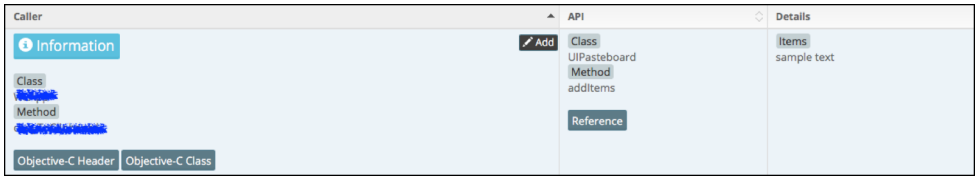

Figure 4 - UIPasteboard detail

UIPasteboard:addItems method as an example. As it stems from Figure 4 the analyzed app shares sensitive data, such as passwords. With a static approach, we would have not been able to obtain this information (*Items* in the Figure 4), because it is generated only at runtime.

Figure 5 shows another feature of MAD-IOS: with the help of *Hopper* [20] and the stack trace previously saved, we are able to detect in most cases the *caller*, addressing the *symbolicating* issue, which consists in retrieving the function/method names from iOS stack traces, that mostly contains only memory addresses. We are able to obtain the desired information by comparing the data contained inside the stack traces with data extracted from the disassembled binary.

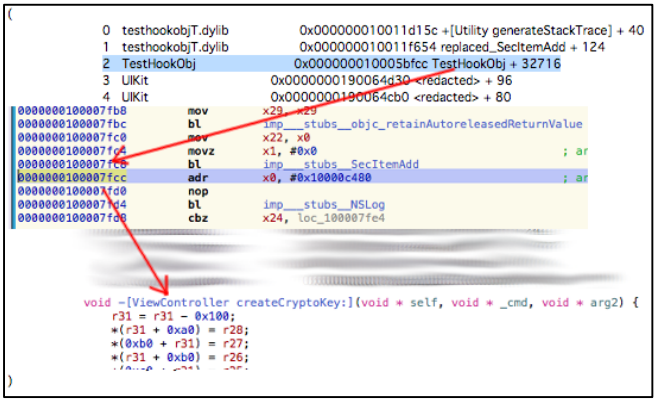

Figure 5 - Symbolicating the Stack trace

As outlined in Figure 6, we correctly retrieved the *class* and the *method* that called *SecItemAdd*.

Figure 6 - SecItemAdd detail

4 Conclusion

MAD-IOS is a complete dynamic analysis tool to assess possible threats caused by wrong coding, exploiting the power of dynamic analysis, in the iOS ecosystem. In particular, as key distinctive feature, MAD-IOS makes possible to execute all the *pentest* process steps, without having to depend on third party procedures and jailbreaking. In addition, our app vulnerability analysis approach does not break the iOS Security Model. We believe that MAD-IOS can provide a contribution in the iOS Security field that, as showed, is still lacking this kind of solution. The encouraging results we obtained are pushing our efforts to further refine MAD-IOS and to expand its features, with the main focus on Swift-based method interceptions.

References

[1] CISCO, "Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2016–2021," 7 February 2017. [Online]. Available: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.pdf.

[2] AppBrain, "Google Play Stats," January 2017. [Online]. Available: http://www.appbrain.com/stats.

[3] Apple Inc., "App Store shatters records on New Year's Day," [Online]. Available: https://www.apple.com/newsroom/2017/01/app-store-shatters-records-on-new-years-day.html.

[4] Kaspersky Lab, "Mobile Malware Evolution 2016," 2016. [Online]. Available: https://securelist.com/files/2017/02/Mobile_report_2016.pdf.

[5] Sufatrio, D. J. J. Tan, T.-W. Chua and V. L. L. Thing, "Securing Android: A Survey, Taxonomy, and Challenges.," *ACM Computing Surveys,* vol. 47, no. 4, p. 45, May 2015.

[6] S. Hao, B. Liu, S. Nath, W. G. Halfond and R. Govindan, "PUMA: Programmable UI-automation for Large-scale Dynamic Analysis of Mobile Apps," *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services,* pp. 204-217, 2014.

[7] F-Secure, "MOBILE THREAT REPORT," 2014. [Online]. Available: https://www.f-secure.com/documents/996508/1030743/Mobile_Threat_Report_Q1_2014.pdf?_ga=2.228340765.1471073704.1506532054-664434203.1506532054.

[8] L. García and R. J. Rodríguez, "A Peek under the Hood of iOS Malware," *Availability, Reliability and Security (ARES), 2016 11th International Conference on,* 2016.

[9] Skycure, "Mobile Threat Intelligence Report," 2017. [Online]. Available: https://cg9j53d64gz46qncx41jvxq16p-wpengine.netdna-ssl.com/wp-content/uploads/2017/07/Skycure-10YrsofiOS-MobileThreatIntelligenceReport_2017Q1.pdf.

[10] S. Jensen. [Online]. Available: https://github.com/S3Jensen/iRET.

[11] A. Kurtz. [Online]. Available: https://code.google.com/archive/p/snoop-it/.

[12] T. Daniels and D. Alban. [Online]. Available: https://github.com/iSECPartners/Introspy-iOS.

[13] [Online]. Available: https://github.com/theos/theos.

[14] K. Oberai. [Online]. Available: https://github.com/kabiroberai/theos-jailed.

[15] Fabric, [Online]. Available: https://github.com/fastlane/fastlane.

[16] Facebook, [Online]. Available: https://github.com/facebook/fishhook.

[17] [Online]. Available: http://iphonedevwiki.net/index.php/Logos.

[18] Apple inc., "Install in-house apps wirelessly," [Online]. Available: https://help.apple.com/deployment/ios/#/apda0e3426d7.

[19] OWASP Foundation, 2016. [Online]. Available: https://www.owasp.org/index.php/OWASP_Mobile_Security_Project#tab=Top_10_Mobile_Risks.

[20] Cryptic Apps SARL, [Online]. Available: https://www.hopperapp.com.

[21] International Data Corporation, "Smartphone OS Market Share, 2017 Q1," 2017. [Online]. Available: https://www.idc.com/promo/smartphone-market-share/os.

[22] Gartner, "Gartner Says Worldwide Sales of Smartphones Grew 7 Percent in the Fourth Quarter of 2016," 2016. [Online]. Available: http://www.gartner.com/newsroom/id/3609817.

A. MAD-IOS walk-through

In this appendix, we show a simple walk-through for MAD-IOS.

The main idea is to show the inner workings of modules composing the framework via the usual steps of a typical use case scenario.

First of all, we will show the preliminary operations, to be performed only once before the first usage:

- Retrieve *Device UDID*;
- Install on the device the Agent-App to manage the process device-side;
- Make a server side setup modifing `env_var` file with the following content:
  - Theos bin path;
  - Apple Developer Profile username;
  - App Group ID.

After that, every time the security analyst wants to analyze an app, has to perform the following steps:

1) Upload a decrypted IPA to the MAD-IOS backend:
   - MAD-IOS backend instruments the IPA and makes the download available via OTA.
2) Download via OTA the patched IPA on the device, using the Agent-App;
3) Use and deeply explore the patched IPA, focusing on sensitive actions. The analysis coverage strongly depends on how much the security analyst browses the app's GUI;
4) Via the Agent-App, upload to the MAD-IOS backend the collected data:
   - MAD-IOS backend evaluates this data and provides a security assessment of it;
   - MAD-IOS backend symbolicates the collected stack traces, leveraging Hopper.
5) Connect to the MAD-IOS backend's web GUI to examine the security report.