Security Vulnerabilities in Hotpatching Mobile Applications

Sarah Ford, Aspen Olmsted
Department of Computer Science
College of Charleston
Charleston, SC
fordsr@g.cofc.edu, olmsteda@cofc.edu

Abstract— **The need for developers to be able to update mobile apps immediately on discovery of a critical bug is something the Apple iOS software patching system does not allow through their traditional app patching lifecycle. Two tools have been developed to solve this problem, one commercial and one open-source. Both employ JavaScript and dynamic code downloads and provide a method for users to receive immediate updates, but both have the potential to be abused and open the user to multiple security vulnerabilities. This paper will discuss how the tools JSPatch and Rollout.io, open-source and commercial respectively, enable quick updates but also expose users to multiple security vulnerabilities and argue for why Apple should not allow them; it proposes a better solution using the same technology that preserves security.**

Keywords- Javascript; iOS; patching; mobile computing; open-source tools; Apple; security

I. Introduction

There is a strong business need for developers to be able to quickly and safely patch their iOS Apps. In the past, the only option for developers was to submit their updated app to the Apple store, who reviewed the changes and then allowed the app to be included in the 'Updates' section of a user's phone for the user to download.

Though most apps still employ this method to update their source code, some developers, wanting to patch apps immediately, have begun to employ commercial and open source tools, which allow developers to include a small amount of code in the source code of their app upon its initial submission to Apple's App Store, which makes a call to a remote server that returns executable JavaScript code. The tool then converts the JavaScript to Objective-C or Swift and adds it to the original source at runtime.

These tools provide a much-needed solution to developers who find critical bugs or security vulnerabilities in their apps after they have been deployed on the app store, but they also create security vulnerabilities and allow malicious developers to evade Apple's strict app review process, which has previously kept the iOS app environment relatively safe for users and their information.

This paper examines how JavaScript hot patching works and documents the vulnerabilities associated with both the commercial and the open-source tool. We demonstrate the dangers and conclude that Apple has an urgent need to change its security policy but also a great opportunity to adopt this technology into its app review process with its full security measures.

The organization of the paper is as follows. Section II describes the related work and the limitations of current methods. In Section III, we document three example use cases as a motivating example. Section IV explains how the hotpatching works technically. Section V explores the commercial tool available for hotpatching. Section VI takes a look at the open source tool available for hotpatching. In Section VII, we explain our test implementation. Section VIII looks at the policy of the Apple, the owner of the phone operating system. Section IX looks at the core problem that led to the situation we find ourselves in. In Section X, we propose a solution. We conclude and discuss future work in Section XI.

II. Related Work

The need to enable users to have access to an app update quickly is not just a need in iOS. More research, in fact, has been done in the other chief mobile operating system, Android. Previous work has formulated various solutions to the need to patch apps quickly and prevent crashes.

Bissyandé et. al. [1] formulated a solution to the need for app users to quickly have access to app updates through a peer-to-peer, network-based update propagation system using a middleware. They were able to demonstrate its effectiveness at a large conference.

In a different approach to the update problem, Azim, Meamtiu, and Marvel [2] propose a solution to allow smartphone apps to "self-heal" by detecting when an app is crashing and altering the byte code to prevent it from interacting with the crashing part of the app and allow the user to continue using other parts of the app.

Both solutions provide options to the need to update quickly to preserve application functionality, but neither allows for the developer to immediately patch their own code as soon as the user opens the broken app.

III. Motivating Example

For our motivating example, we propose three variations of the following scenario: a student developer develops a simple game, which is accepted by Apple's App Store.

In the first scenario, our developer is well meaning: she simply wants to update her app if there are bugs quickly. She includes the open-source hot patching tool: JSPatch. JSPatch makes a call to a remote server every time the app runs and downloads executable JavaScript code. Though her intentions are good, she exposes her users to the danger of the well-known Man-in-the-Middle attack (MitM) [3]. If her user is using her app on an unencrypted or dangerous network, an attacker could intercept and modify the JavaScript and maliciously attack the game user's phone or our developer's app functionality.

In our second scenario, our developer is also well-intentioned, and also in need of income, because she is, after all, a university student; therefore, she includes an advertising software developer kit (SDK) in her app in order to make some money from her app. The advertising SDK, however, is from a malicious developer and includes JSPatch. When a user runs the app, the advertising SDK may employ some private iOS APIs, which make us of private APIs to steal personal information from the user's device [3] .

In our third scenario, our developer is malicious. She wants to steal her user's information to sell to interested third parties. She includes JSPatch in her app with no malicious code downloading at first, but once her app is already in the app store, she modifies the JavaScript to include an iOS private API, which accesses the user's personal information and stores it on her remote server to sell to third parties.

IV. How JavaScript Hotpatching works

JavaScript injection at runtime is possible in the iOS operating system because of the JavaScriptCore framework and a technique called method swizzling [4].

The JavaScriptCore Framework "allows you to evaluated JavaScript programs from within an Objective-C or C-based program. It also lets you insert custom objects into the JavaScript environment" [5].

The code to be excuted by the JavaScriptCore framweork gets into the app through a call to a remote server, which downloads the Javascript and then executes it with a technique known as method swizzling. Method swizzling "is the process of changing the implementation of an existing selector. It's a technique made possible by the fact that method invocations in Objective-C can be changed at runtime, by changing how selectors are mapped to underlying functions in a class's dispatch table" [6].

Both the use of the JavaScriptCore and method swizzling are compliant with Apple's development guidelines because the JavaScriptCore is a public API and method swizzling does not alter the binary of the app [7]. See Figure 1 for a visual representation of the process.

V. The Commercial Tool: Rollout.io

Rollout.io is an Israeli startup company, which offers a tool to implement all phases of hot patching [8]. They provide not only the code to be put in the source code of the app but also an interface and server from, which to push these code updates to your app. Because they have direct control over the server pushing the code, they also have fewer security vulnerabilities than the open-source tool (described below).

The most major vulnerability in Rollout.io is the ability to load an "arbitrary public framework" and use the associated APIs with malicious intent [9]. For example, to access sensitive user data and export it without the user's knowledge. Though many apps access sensitive user data (photos, contacts, etc.) with a clear purpose, Apple's review

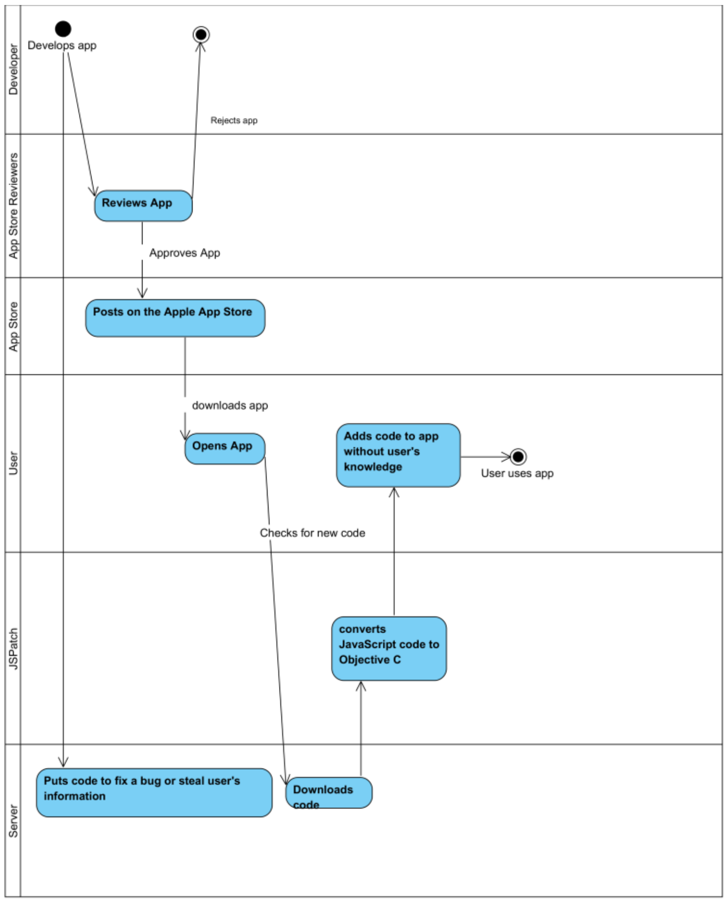

Figure 1. How JSPatch works.

ensures that these apps do not export private user data or access user data without a legitimate reason [7].

After security researchers at Fire Eye [9] identified that Rollout.io could be used maliciously through the use of private APIs, Rollout.io responded that they would be preventing users from accessing private APIs when submitting patches through their system article. Developers do not need to use private APIs to gain access to sensitive user information and abuse it, however, so this is not a perfect solution.

VI. The open-source Tool: JSPatch

JSPatch is an open source project created by a Chinese developer in 2014 [10]. It is regularly updated and has more than 30 contributors. It is similar in functionality to its commercial equivalent (discussed above). However, it has two additional security vulnerabilities, which Rollout.io does not.

The first major problem occurs when the developer is malicious. The developer can invoke a private Apple API in the JSPatch code without Apple's knowledge [3]. Apple does not allow for private APIs to be invoked in any app that is on the app store, but they only check for it in the app review process [3].

A non-malicious developer could still be put at risk by using JSPatch if they do not “protect the communication from client to the server for JavaScript content” and thus open themselves up to a man-in-the-middle attack (MITM) [3]. The attacker could then modify the JavaScript and attack the host app and the user’s device in a variety of different ways.

VII. IMPLEMENTATION

To test the usability of these tools, JSPatch was chosen to test the ease of implementation, since, because it is open-source, it is more accessible to the normal developer, and more widely used in the App Store. We found it relatively easy to implement JSPatch using its documentation (though it should be noted that the Chinese documentation is more detailed, so it would probably be easier for a native Chinese speaker).

The exploit we chose to replicate was adopted from researchers at FireEye [3] who provided multiple compelling examples of the dangers of JSPatch. The exploit chosen was trivial one but emblematic of the problems, which can occur in JSPatch. We were able to load an arbitrary public framework which, once loaded, grants the script access to any private APIs which the framework has access to. Thus, without going through any review by Apple, privacy violations or bad practices, which would be grounds for an application to be rejected by official Apple reviwers, can be carried out without their knowledge.

VIII. APPLE’S POLICY

Rollout.io and JSPatch claim their tool is being accepted by Apple. JSPatch does not make an explicit legal claim, but in a GitHub issue thread, one user complains their app was rejected based on its inclusion of JSPatch, they include text from their rejection notification: “app contains an SDK designed to update the app outside of the App Store process. It would be appropriate to remove this SDK before resubmitting for review” [11]. In the same thread, user bang590, creator of JSPatch, claims Apple has been accepting apps, which include JSPatch, so there is no reason for it to be rejected and makes some suggestions for things to change, so the user will be accepted [11].

User bang590 is correct, according to FireEye’s analysis as of January 2016, 1,200 apps in the app store contained JSPatch [3]. Rollout.io claims to be used on over 370 apps with a total device count of over 50 million [12].

Rollout.io, as a company, must have some sort of legal precedent to sell their product. They claim that according to Apple’s developer guidelines 3.3.2 and 3.3.3 [7], they are not in violation of the rules because “ 1. The code is run by Apple’s built-in WebKit framework and JavascriptCore. The code does not provide, unlock or enable additional features or functionality” [13]. The author also claims that no app has ever been rejected for containing Rollout.io.

Rollout.io is correct that its product is not designed to add new feature or functionality. However, that does not prevent it or JSPatch from being used to do just that: to use it for the addition of functionality without the user's knowledge, which violates the user’s privacy or puts them at risk.

Since this is not a discussion of the legality but the security of this policy, regardless of whether or not this exploit is within Apple’s developer guidelines, Apple has been allowing apps with both JSPatch and Rollout.io on its app store for several years now.

Clearly, both Rollout.io and JSPatch pose major problems to Apple’s supposedly stringent security policies. Apple’s security is often praised as superior to Google’s Android because of their strict review process and single, proprietary app store. However, tools like JSPatch and Rollout.io directly undermine the review process, which is supposedly keeping apps secure.

IX. PROBLEM THAT LEADS TO CURRENT STATE

The problem, which these tools are trying to address, though, is not creating a way to undermine the app review process, but creating a way to avoid the time delay, which Apple’s review process creates for developers who are anxious to keep users if their app is crashing and users who are irritated by apps they want to use but are crashing. Apple [14] provides a way for developers to request an expedited review for fixing a critical bug, but, of course, it is not guaranteed that your request will be granted.

Comparably, the Google [15] play store, implemented a similar app review process. However, the times dramatically differ. In fact, Google rolled out the app review process in 2015 without notifying developers, and there was no noticeable change in rollout time because review times remained, on average, under an hour. They automate part of the process before submitting it to app reviewers. Therefore, they can do it much more quickly.

Apple [16] has significantly improved its review time since the invention of JSPatch and Rollout.io, shrinking it from an average of 8.8 days in 2015 to 1.95 days in May 2016. However, this is still significantly longer than Google’s, their main competitor. Since Apple does not publicize information about their review process, it is unknown what is taking so long compared to Google.

X. TOWARDS A BETTER SOLUTION

Apple has an urgent need to change their review process to make it comparable to Google’s, perhaps automating parts of it to speed up a review, to eliminate the need for tools like JSPatch and Rollout.io. Though they have decreased the review time (see above) since the invention of these tools, they have not decreased it to an acceptable level for developers who want to patch immediately.

With this lag time and their allowance of JSPatch and Rollout.io, they have undermined their entire security process, and these tools should be banned from use but not without a quicker patching solution.

The technology of Rollout.io and JSPatch represent a creative and easy solution to this problem, which should not be disregarded, however. To secure the process, Apple could

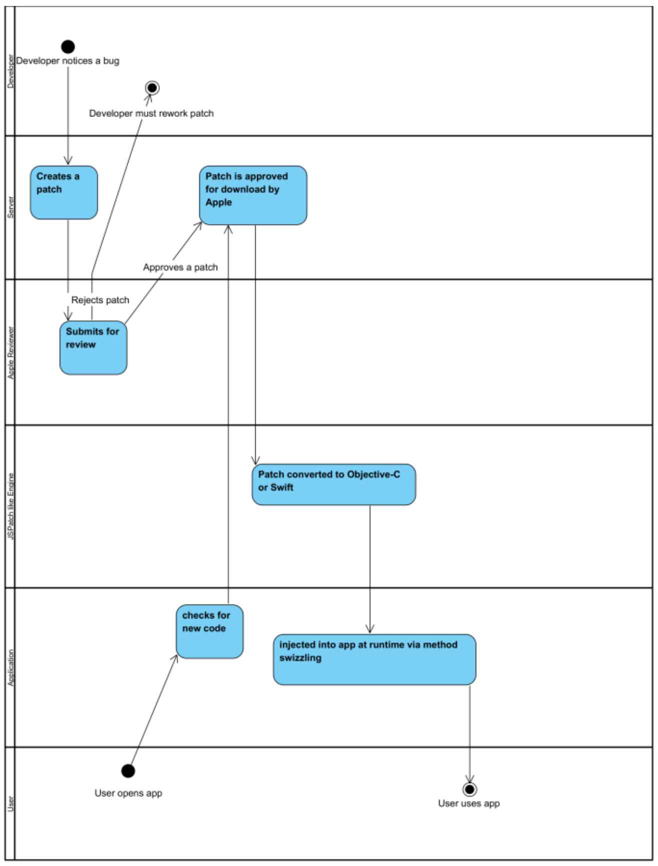

Figure 2. A better solution.

require submission of these patches to undergo review before they are actually downloaded to the app. These patches are not designed to be large scale changes to the entire app, but small hotfixes to bugs. The code being added when the tool is used correctly, should be relatively few lines and thus easy for an Apple app reviewer to approve within minutes. To protect against MITM attacks, developers who submit with this technology could be required to add code to ensure that the JavaScript being downloaded was protected and Apple could reject apps, which did not protect the network communication between the app and the server.

If developers began making their small patches through this technology and not resubmitting their entire app for even the smallest of bug patches, it would, in theory, free up the time of the Apple app reviewers to review initial app submissions and large updates more quickly. Therefore, this solution solves the problem of secure hot-patching and the problem of long submission wait times while maintaining the clean iOS app environment; see Figure 2 for a visual representation of the suggested process.

XI. CONCLUSION AND FURTHER RESEARCH

The need to be able to patch apps immediately is vital to developers and has driven them to create tools that expose loopholes in Apple's otherwise strict development guidelines and inconsistencies in its review process. Rollout.io and JSPatch provide significant benefits to developers and users when they are used safely and responsibly, but when they are in the hands of a malicious developer or if JSPatch is used without proper encryption, malicious code can enter the otherwise clean iOS environment.

On March 7, 2017, Apple began sending emails to developers using both Rollout.io and JSPatch to warn them that apps containing these tools will no longer be accepted on the app store. However, there has been no change in Apple's official development guidelines with regards to the language used [17]. It seems like Apple is moving in the right direction in terms of securing their ecosystem. However, there remains no good solution for quickly patching iOS apps. It is also worth noting that Apple has taken an alarmingly long time to recognize the problem with these tools, despite extensive reporting on it from security researchers. Apple's review system is slow and inefficient and also seems to lack efficacy and consistency.

Despite these concerns, Apple has an opportunity to make developers and users happy while maintaining security through the solution proposed here. It would allow them to review apps more quickly by relegating small changes to the hotpatching fixes, which would require much less time to review than the whole app code base that is resubmitted through the current Apple app update process.

REFFERENCES

[1] T. F. Bissyandé, L. Réveillère, J.-R. Falleri and Y.-D. Bromberg, "Typhoon: a middleware for epidemic propagation of Software Updates," in *Proceedings of the Third International Workshop on Middleware for Pervasive Mobile and Embedded Computing*, Lisbon, 2011.

[2] M. T. Azim, I. Neamtiu and L. M. Marvel, "Towards self-healing smartphone software via automated patching," in *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*, New York, 2014.

[3] J. Xie, Z. Chen and J. Su, "Hot or Not? The Benefits and Risks of IOS Remote Hot Patching," 2016 January 2016. [Online]. Available: https://www.fireeye.com/blog/threat-research/2016/01/hot_or_not_the_bene.html. [Accessed 11 December 2016].

[4] Rollout.io, "Rollout Under The Hood – 2016 Update," 22 March 2016. [Online]. Available: https://blog.rollout.io/under-the-hood-2016-update/. [Accessed 1 December 2016].

[5] Apple, "JavaScriptCore," [Online]. Available: https://developer.apple.com/reference/javascriptcore. [Accessed 11 December 2016].

[6] M. Thompson, "Method Swizzling," 17 February 2014. [Online]. Available: http://nshipster.com/method-swizzling/. [Accessed 13 December 2016].

[7] Appe, "iOS Developer Program Information," 3 4 2015. [Online]. Available: https://developer.apple.com/programs/ios/information/iOS_Program_Information_4_3_15.pdf. [Accessed 11 December 2016].

[8] Rollout.io, "About Rollout," [Online]. Available: https://rollout.io/about/. [Accessed 17 December 2016].

[9] J. Xie and J. Su, "Rollout or Not: The Benefits and Risks of IOS Remote Hot Patching," 4 April 2016. [Online]. Available: https://www.fireeye.com/blog/threat-research/2016/04/rollout_or_not_the.html. [Accessed 30 November 2016].

[10] bang590, "JSPatch," 7 August 2016. [Online]. Available: https://github.com/bang590/JSPatch. [Accessed 13 December 2016].

[11] xnth97, "Rejected by App Store," 15 September 2015. [Online]. Available: https://github.com/bang590/JSPatch/issues/111. [Accessed 2017 April 2017].

[12] Rollout.io, "Our Customers Fix Things Faster and Get More 5 Star Reviews," 2016. [Online]. Available: https://rollout.io/success-stories/. [Accessed 17 April 2017].

[13] O. Prusak, "Update Native iOS Apps without the App Store. How is this Legit?," 27 January 2016. [Online]. Available: https://rollout.io/blog/updating-apps-without-app-store/. [Accessed 17 April 2017].

[14] Apple, "App Review Support," [Online]. Available: https://developer.apple.com/support/app-review/. [Accessed 17 April 2017].

[15] S. Perez, "App Submissions On Google Play Now Reviewed By Staff, Will Include Age-Based Ratings," 17 March 2015. [Online]. Available: https://techcrunch.com/2015/03/17/app-submissions-on-google-play-now-reviewed-by-staff-will-include-age-based-ratings/. [Accessed 17 April 2017].

[16] O. Raymuldo, "Apple is approving apps for the iOS App Store much faster now," 12 May 2016. [Online]. Available: http://www.macworld.com/article/3070012/ios/apple-is-approving-apps-for-the-ios-app-store-much-faster-now.html. [Accessed 17 April 2017].

[17] E. Rusovsky, "Rollout's Statement on Apple Guidelines," 13 March 2017. [Online]. Available: https://rollout.io/blog/rollout-statement-on-apple-guidelines/. [Accessed 19 April 2017].