Automation in iOS Application Assessments

SiRA Team:

Justin Engler
Seth Law
Josh Dubik
David Vo

Contact:

1.sira.tool@gmail.com
Twitter: @siratool

Legal Notice

Third-party sources are quoted as appropriate. Authors are not responsible for the content of the external sources including external websites referenced in this publication.

This publication is intended for educational and informational purposes only. The authors are not responsible for the use that might be made of the information contained in this publication.

Table of Contents

Introduction

Apple's App Store continues to grow in popularity, and iOS devices continue to have a high perception of security from both users and experts. However, applications on the App Store often have security or privacy flaws that are not apparent, even to sophisticated users. Security experts can find these flaws via manual tests, but the enormity of the App Store ensures that only a small minority of applications could ever be manually tested. This whitepaper explores the issues associated with assessing iOS applications and proposes techniques to inject automation into the assessment methodology.

Why Evaluate Mobile Application Security?

According to ABI Research mobile application downloads are expected to reach five billion by 2014 (ABIresearch, 2009). With wide adoption of mobile devices from both users and corporations, mobile applications become increasingly targeted for attackers and developers to craft mobile applications to collect private data such as geographic location, login credentials, or confidential corporate data. These kinds of data can also be compromised via an application, which is not malicious, but does not provide adequate security.

Many enterprises do not do security evaluations of 3rd party desktop software in use in the enterprise because other defenses (firewalls, proxies, AV, IPS, etc.) are in place. However, not only are those tools completely lacking in a mobile environment, the devices themselves are less secure and are more prone to loss or theft than a corporate desktop (or laptop). Furthermore, mobile applications have the same capabilities as any other applications. They can access the corporate network over SSL or HTTP, access backend databases, place orders, etc.

Organizations need to have a strong understanding of the security posture of their processes and data. As these migrate to mobile platforms, organizations need to be aware of the security of applications in place on their mobile devices.

Current State of iOS Application Assessment

Application security assessments on other, more established platforms are often highly automated, and the skill sets required to perform them are relatively well-known. Many consultancies, large and small, can provide these services.

For iOS application security assessments, the state of the art is much more primitive. Software tools are limited to a few forensics or developer tools repurposed for security assessments. Only in the past few years has a consistent knowledge base and methodology appeared to allow clients to have a strong understanding of what to expect from an iOS application security assessment. Although specifics of methodology vary, the following is a representative sample of assessment methodologies merged together from (OWASP, 2012; Zdziarski, 2012; Chell, 2012; van Sprundel, 2011; Esser, 2012; Thiel, 2011), and shows what a typical black box iOS application security assessment methodology might look like.

1. Enable network proxy
2. Snapshot file system
3. Install application

4. Decrypt application
5. Snapshot file system
6. Binary analysis
7. Runtime analysis

Figure 1: iOS Application Security Assessment Methodology Overview

Prerequisites

Before starting, the assessment requires a computer running BackTrack 5r2, a jailbroken iOS device, and a means to obtain the application to be assessed. More importantly, the permission to assess the application.

Preparation

Steps 1-5 are preparatory. At step 1, a network proxy of some sort is needed to intercept traffic transmitted and received by the application. This proxy needs to be installed on a separate machine, not the device under test (DUT). Proxies with storage, interception, modification, and SSL man-in-the-middle capabilities, such as Burp Suite or Zed Attack Proxy, are good choices. The DUT must also be configured to pass traffic through the proxy provided. In step 2, an initial file system snapshot can be taken to give a baseline for later comparisons, to see what files were written by the application under test (AUT) during the assessment. The application is installed for step 3, either via the official Apple App Store or from one of several other installation channels. Applications from some sources may be encrypted. Step 4 is application decryption, to allow for later analysis. Another file system snapshot (full or differential) is taken at step 5 to show what files were created by the installation process. Step 6 marks the beginning of analysis.

Binary Analysis

6. Binary analysis:
    a. PIE enabled?
    b. Stack smashing protection enabled?
    c. Reference counting enabled?
    d. Class-dump or class-dump-z
    e. XML processors installed?
    f. Jailbreak Detection?
        i. (if yes, disable)

Figure 2: Binary Analysis Detail

A decrypted application binary is inspected for several security features. Modern iOS applications should be compiled with the Position Independent Executable (PIE) feature (HLFS Development Team, 2008), which allows for more effective Address Space Layout Randomization (ASLR). This is checked in step 6a. Step 6b checks for stack smashing protection, and step 6c checks to see if iOS5's automatic memory management has been enabled. These three compilation settings can help prevent successful memory corruption attacks. Step 6d will produce information about the program's internal class names and references to external libraries, which can be useful in later analysis phases. Of particular note here is to determine if the application is using an XML processing library. Finally, some

applications include code to try to detect jailbroken devices and prevent the application from functioning if a jailbroken device is found. If jailbreak detection is enabled, it must be removed before the assessment can continue. In some cases, jailbreak detection might not be discovered until later phases of the assessment.

Runtime Analysis

Use Application

The first step in runtime analysis is to use the application as a legitimate user would. Most applications have an initial install state that is somewhat different than the more typical application usage that will follow. This often includes signing up for or signing into a cloud service, agreeing to terms of use, or a tutorial that explains the operation of the app. After initial setup is complete, the analyst should continue to use the application as a normal user would, and attempt to use each function or setting present in the application at least once. If any network traffic is generated by the application during this phase, it will be stored by the proxy set up in step 1.

Man-in-the-Middle detection

Use of a man-in-the-middle proxy should be detected and disallowed by the application for any sensitive data transmissions. However, the use of the man-in-the-middle proxy will be needed for other parts of the assessment, so this otherwise desirable behavior of the application will need to be disabled for the remainder of testing.

Sensitive Data - File system

After the analyst has finished using the application as a normal user would, another file system snapshot is taken. The snapshot is compared to the previous ones to determine what information was stored by the application. If sensitive information was stored, the analyst should confirm that the application stores the information in a secure manner. Network traffic can also be analyzed at this point to determine if any sensitive data was transmitted, and if that data was transmitted securely. The file system should also be examined for evidence that the application registered URL handlers, which will be further analyzed in later steps.

Sensitive Data - Network

Analysis of the network traffic generated during testing will determine if any sensitive data (passwords, keys, etc.) or personal information (contact lists, calendar entries, GPS coordinates, unique identifiers, etc.) was sent over the network. It should be clear from examining the logs as to whether the information was transmitted securely or in the clear.

7. Runtime Analysis:
    a. Use the application and record data
    b. Certificate enforcement?
        i. If yes, bypass (import cert, hook cert functions, activate debug mode, etc.)
    c. Snapshot file system
    d. Analyze snapshot diffs
        i. Locate storage of sensitive data
            1. Was it stored securely?
        ii. Protocol handlers installed?
    e. Locate transmission of sensitive data
        i. Was it transmitted securely?
        ii. Privacy Analysis

1. Did the app transmit Contacts?
2. Did the app transmit Calendar?
3. Did the app transmit Location?
4. Did the app store a location log?
5. What was the granularity of the location?
6. Did the app transmit UDID?

e. Abuse the application and record data
   i. If protocol handlers in use, can they be abused?
   ii. UIWebView in use?
      1. Attempt XSS
         a. Attempt to exploit objc bridge
   iii. XML in use?
      1. Attempt local XML attacks
   iv. Attempt buffer overflows
   v. Attempt format string attacks
   vi. Attempt local file traversal
   vii. Attempt local SQLi
   viii. Logic flaw abuse
   ix. (If in scope - Server-side analysis)
f. Snapshot file system
g. Analyze snapshot diffs
h. As findings are discovered, repeat any steps in 7. as needed.

Figure 3: Runtime Analysis Detail

Application Abuse

At this point, the analyst begins to actively misuse the application. A variety of malformed inputs can be used to test for various client-side application vulnerabilities. See (Zdziarski, 2012; Chell, 2012) for more specific details. The analyst will also attempt to exploit logic flaws in the application to attempt to perform actions that would not otherwise be allowed. Finally, for some assessments the analyst will attempt to attack the server-side portions of the application, which are often web services. This testing is common when the analyst is hired by or works for the application author. When an application is analyzed for a third-party client, server-side testing is not performed unless the application's author has been notified of the testing and permits it.

Repeat and Wrapup

A final file system snapshot and comparison are performed. This information, and any network logs, is examined for any new data. This might require the analyst to perform further work and repeat one or more substeps in step 7.

Drawbacks

The methodology shown in Figure 4, while comprehensive, has several drawbacks:

- Manual setup and analysis is time-intensive
- Specialized knowledge and skill sets are required for both setup and analysis

1. Enable network proxy
2. Snapshot file system
3. Install App
4. Decrypt app
5. Snapshot file system
6. Binary analysis:
    a. PIE enabled?
    b. Stack smashing protection enabled?
    c. Reference counting enabled?
    d. Class-dump or class-dump-z
    e. XML processors installed?
    f. Jailbreak Detection?
        i. (if yes, disable)
7. Runtime Analysis:
    a. Use the app and record data
    b. Certificate enforcement?
        i. if yes, bypass (import cert, hook cert functions)
    c. Snapshot file system
    d. Analyze snapshot diffs
        ii. Locate storage of sensitive data
            1. Was it stored securely?
        iii. Protocol handlers installed?
    f. Locate transmission of sensitive data
        x. Was it transmitted securely?
        xi. Privacy Analysis
            1. Did the app transmit Contacts?
            2. Did the app transmit Calendar?
            3. Did the app transmit Location?
            4. Did the app store a location log?
                b. What was the granularity of the location?
            5. Did the app transmit UDID?
    i. Abuse the app and record data
        i. If protocol handlers in use, can they be abused?
        ii. UIWebView in use?
            1. Attempt XSS
                c. Attempt to exploit objc bridge
        iii. XML in use?
            1. Attempt local XML attacks
        iv. Attempt buffer overflows
        v. Attempt format string attacks
        vi. Attempt local file traversal
        vii. Attempt local SQLi
        viii. Logic flaw abuse
        ix. (If in scope - Server-side analysis)
    j. Snapshot file system
    k. Analyze snapshot diffs
    l. As findings are discovered, repeat any steps in 7. as needed.

Figure 4: Full iOS Application Assessment Methodology

The Problem of Scale

Apple's iOS App Store contains over 680,000 applications as of 2012 (148Apps, 2012). New apps are added to the App Store at a rate of 1,121 per day (148Apps, 2012). A standard manual assessment methodology, performed by those people currently capable, will never be able to catch up. Furthermore, many app projects are small and do not have the resources to hire out a full manual security assessment. One survey of iOS developers showed their average development cost to be $6,453 per app (Ahlund, 2010). Though this datapoint appears low compared to other anecdotal information, it can still be used as a ballpark figure for simple applications.

On the flip side, enterprises continue to roll out more iOS devices, either via traditional corporate purchasing, or via Bring Your Own Device (BYOD) policies. According to a 2012 Q2 earnings call transcript, Apple CEO, Tim Cook stated that "94% of the Fortune 500 are testing or deploying iPads and 75% of the Global 500 are testing or deploying the iPad" (Seeking Alpha, 2012). In either case, business users of iOS devices will clamor for use of apps to help them be more efficient in business. Some enterprises restrict their users to a trusted set of apps (less likely in BYOD environments), but these strategies face pressure from corporate users to open up to a broader variety of apps. It seems unlikely that every corporate IT mobility department will be able to afford custom manual assessments for each application being used on devices that also touch corporate data.

Towards Automation: SiRA

A good portion of the work involved in an iOS application assessment is setup and other tedious repeatable tasks. Many of these tasks could be easily automated.

Our proof of concept tool, "Semi-automated iOS Rapid Assessment" (SiRA), implements many of the automated and partially automated steps in Figure 5.

1. Enable network proxy
2. Snapshot file system
3. Install App
4. Decrypt app
5. Snapshot file system
6. Binary analysis:
    a. PIE enabled?
    b. Stack smashing protection enabled?
    c. Reference counting enabled?
    d. Class-dump or class-dump-z
    e. XML processors installed?
    f. Jailbreak Detection?
        ii. (if yes, disable)
7. Runtime Analysis:
    a. Use the app and record data
    b. Certificate enforcement?
        i. if yes, bypass (import cert, hook cert functions)
    c. Snapshot file system
    d. Analyze snapshot diffs
        i. Locate storage of sensitive data
            1. Was it stored securely?
        ii. Protocol handlers installed?
    e. Locate transmission of sensitive data
        i. Was it transmitted securely?
        ii. Privacy Analysis
            1. Did the app transmit Contacts?
            2. Did the app transmit Calendar?
            3. Did the app transmit Location?
            4. Did the app store a location log?
                a. What was the granularity of the location?
            5. Did the app transmit UDID?
    f. Abuse the app and record data
        i. If protocol handlers in use, can they be abused?
        ii. UIWebView in use?
            1. Attempt XSS
                a. Attempt to exploit objc bridge
        iii. XML in use?
            1. Attempt local XML attacks
        iv. Attempt buffer overflows
        v. Attempt format string attacks
        vi. Attempt local file traversal
        vii. Attempt local SQLi
        viii. Logic flaw abuse
        ix. (If in scope - Server-side analysis)
    g. Snapshot file system
    h. Analyze snapshot diffs
    i. As findings are discovered, repeat any steps in 7. as needed.

Easily Automated
Automatable
Partially Automatable
Not Automatable

Figure 5: Full iOS Application Assessment Methodology

SiRA Features

SiRA is able to automate or semi-automate many of the steps involved in an application assessment. SiRA includes some assistance for all 7 of the major methodology steps outlined above. Not all automatable substeps are currently implemented, but work is ongoing. In addition, SiRA provides a convenient single location for a variety of manual and semi-automated functionalities. Finally, SiRA can automate your automation by providing a step-by-step guided methodology walkthrough with SiRA CruiseControl. SiRA CruiseControl will automatically perform some steps, and will inform the user when it is time to perform a manual step. When the user indicates the step is complete, SiRA moves on to the next phase of the methodology. All file system changes are catalogued and tricky file formats are automatically decoded for ease of reference. In addition, automatic analysis will attempt to find security issues automatically during the assessment.

Why Can't We Automate Everything?

A variety of issues make fully automated testing infeasible. In most cases this is because a program cannot understand the semantics of the AUT. Some activities in an assessment, as well as many of the possibilities for vulnerabilities, rely on an understanding of the meaning of the data found and its relationship to other parts of the system.

Jailbreak Detection and Bypass

A typical application with jailbreak detection will crash or display an error message if it is run on a jailbroken device. This is likely to happen very early in the execution of the app, but could happen anywhere (or even multiple places) in an app. An automated program is incapable of discerning an error message regarding jailbreaking detection from any other message. Furthermore, a program is not likely to be able to automatically evade a jailbreak detection. A human expert will be needed to hook or patch the application to bypass the detection.

Application Autopilot

It is theoretically possible that an app could be operated by another program. This is analogous to the web crawlers or discovery tools present in some web application scanners. The program would need to be able to identify what different user interface controls are present on a given "screen", interact with each one appropriately (click buttons, enter text, perform gestures, etc.) and keep track of "screen changes" to avoid repeatedly testing the same areas, infinite loops between a series of screens, etc. It is likely that a tool like this would be useful in the general case, but would fail completely on certain kinds of applications and would be seriously incomplete on others.

Some rudimentary research has already been done in this area (Szydlowski, Egele, Kruegel, Vigna, 2011), but existing solutions are neither general nor robust. The SiRA team has also been working in this area and hopes to have something interesting to show soon.

File and Network Analysis

While both files and network logs can be searched and parsed for known strings, an application could perform any number of simple encodings that would fool a program, but be easily noticeable by a human. Human intuition in addition to automated mass analysis is the best

approach.

Application Abuse

A variety of fuzzing techniques can be used to attempt to uncover client-side flaws, but a human operator is needed to confirm that the attack succeeded. In addition, SiRA is intended to specifically examine client-side issues and minimize the risk of accidentally attacking server-side functionality. No fuzzing or attack functionality is included in this version of SiRA, though it may be added in future releases.

Logic Flaws

Because logic flaws are so varied, and because they require understanding of what correct or expected behavior looks like, automated techniques cannot be successful in detecting generalized logic flaws. It is possible that some specific logic flaws could be detected by a program like SiRA if they were already well known and understood (perhaps stemming from a common misuse of a framework API). This is less the discovery of a new flaw as it is discovering the existence of a known flaw type in a new application.

Automation in the Future

Crowd-driven Assessment

While a professional analyst with deep knowledge will always be preferable, at some level semi-automated tools like SiRA open up iOS application assessment to anyone with a jailbroken iOS device. More people will assess applications, including the kinds of applications that don't usually receive professional attention. More exposure and discussion will raise the bar for iOS development security practices. Cloud services could facilitate a ratings system where a security-competent user can rate an application on various security or privacy metrics, so that other less technical users can make more informed choices regarding application use.

Tomorrow's SiRA

SiRA will be extended to provide more automation possibilities.

- More detection of more issues during file analysis
- Library hooking to provide bypasses for certificate pinning and apps that do not obey system proxy settings
- SiRA AutoPilot - Automatically operate apps for step 7a. Detect onscreen controls and manipulate them intelligently (subject to the limitations described above)
- SiRA Smartbomb - Intelligent fuzz generation based on known inputs and likely attack vectors. Automatic or assisted detection of probable security issues caused by malformed data.
- SiRA Drone - Fire and forget, fully automated assessment from selection of apps, installation, use, fuzzing, and analysis
- Tell us what else you'd like to see!

As of July 26th, you can download the latest SiRA here:
https://bitbucket.org/siratool/sira-release/downloads

References

148Apps. (2012) “App Store Metrics” Retrieved from http://148apps.biz/app-store-metrics/?mpage

ABIresearch. (2009) “Mobile Application Downloads to Hit Five Billion in 2014” Retrieved from http://www.abiresearch.com/press/1569-Mobile+Application+Downloads+to+Hit+Five+Billion+in+2014

HLFS Development Team. (2008) “Position Independent Executables” Retrieved from http://linuxfromscratch.xtra-net.org/hlfs/view/unstable/glibc-2.4/chapter02/pie.html

OWASP. (2012) “Mobile Security Testing” Retrieved from https://www.owasp.org/index.php/OWASP_Mobile_Security_Project

Seeking Alpha. (2012) “Apple's CEO Discusses Q2 2012 Results - Earnings Call Transcript” Retrieved from http://seekingalpha.com/article/524451-apple-s-ceo-discusses-q2-2012-results-earnings-call-transcript?part=single

Martin Szydlowski, Manuel Egele, Christopher Kruegel, and Giovanni Vigna. (2011) “Challenges for Dynamic Analysis of iOS Applications” - iNetSec2011 Open Research Problems in Network Security, Luzerne, Switzerland, 2011 - Retrieved from http://iseclab.org/papers/iphone-dynamic.pdf

Jonathan Zdziarski. (2012) “Hacking and Securing iOS Applications: Stealing Data, Hijacking Software, and How to Prevent It.” O’Reilly Media. ISBN-10: 1449318746

David Thiel. (2011) “Secure Development on IOS” Presented at Source Boston 2011. Retrieved from http://www.isecpartners.com/presentations/secure-development-on-ios-1.html

Steffan Esser. (2012) “iOS5_An_Exploitation_Nightmare” Presented at CanSecWest 2012. Retrieved from http://antid0te.com/CSW2012_StefanEsser_iOS5_An_Exploitation_Nightmare_FINAL.pdf

Dominic Chell. (2012) “iOS Application (In)Security” Retrieved from http://www.mdsec.co.uk/research/

Ilja van Sprundel. (2011) “iPhone and iPad Hacking” Presented at CanSecWest 2011.

Ilja van Sprundel. (2011) “Hacking Smart Phones” Presented at CCC 27.

Alex Ahlund. (2010) “iPhone App Sales, Exposed” retrieved from http://techcrunch.com/2010/05/16/iphone-app-sales-exposed/