Forensic Science International: Digital Investigation 52 (2025) 301871

Contents lists available at ScienceDirect

Forensic Science International: Digital Investigation

journal homepage: www.elsevier.com/locate/fsidi

DFRWS EU 2025 - Selected Papers from the 12th Annual Digital Forensics Research Conference Europe

Tapping .IPAs: An automated analysis of iPhone applications using apple silicon macs

Steven Seiden [a,b,*], Andrew M. Webb [b], Ibrahim Baggili [a,b]

[a] Baggil(i) Truth (BiT) Lab, Center of Computation & Technology Baton Rouge, LA, USA
[b] Division of Computer Science & Engineering Louisiana State University Baton Rouge, LA, USA

ARTICLE INFO

Keywords:
Digital forensics
Digital investigations
iOS forensics
Apple forensics
Artifact collection
Automated analysis
iOS security
Apple silicon
Application analysis
Virtualization
iPhone

ABSTRACT

Dynamic analysis of iOS applications poses significant challenges due to the platform's stringent security measures. Historically, investigations often required jailbreaking, but recent enhancements in iOS security have diminished the viability of this approach. Consequently, alternative methodologies are necessary. In this study, we explore the feasibility of automated iOS application analysis on the ARM-based M1 Mac platform. To do so, we utilized an ARM-based Mac to install several popular iOS applications. Our manual analysis using existing macOS tools demonstrated the potential to uncover artifacts such as chat messages and browsing history. To streamline this process, we developed a tool, *AppTap*, which facilitates the entire forensic procedure from installation to artifact extraction. AppTap enables analysts to quickly install, test, and retrieve file system artifacts from these applications and allows for the easy checkpointing of user files generated by iOS apps. These checkpoints help analysts correlate artifacts with user actions. We tested AppTap with the top 100 iPhone apps and top 100 iPhone games from the U.S. App Store (n=200). Our results showed that 46 % of these applications were installed and operated as expected, while 30.5% failed to install, likely due to the older macOS version—a necessary condition for this study. We discuss several strategies to enhance application support in the future, which could significantly increase the number of supported applications. Applying our methodologies as-is to the M1 Mac platform has significantly streamlined the forensic process for iOS applications, saving time for analysts and expanding future capabilities.

1. Introduction

Dynamic analysis involves executing software and observing its behavior under various conditions. Unlike static analysis, which entails examining the code, dynamic analysis enables analysts to uncover obfuscated activities and observe the application's response to inputs in real-time. This method is crucial in the digital forensics field, as it allows forensic analysts to detect malicious activities and collect evidence from benign applications that are utilized maliciously (Shijo and Salim, 2015; Subedi et al., 2018).

In this work, we explore the dynamic analysis of applications developed for iOS, the operating system used by iPhones and iPads[1]. Since its initial release, iOS has been intentionally designed with strict security features to enhance user security. Key among these features is sandboxing, which restricts the ability to use tools for inspecting an application's activity (Apple, Inc., n.d. d).

Until recently, analysts required a physical iOS device (either an iPhone or iPad) to execute iOS applications, as these could only run on official hardware. The platform's inherent software restrictions made performing dynamic analysis of iOS applications challenging. Consequently, jailbreaking—an act of exploiting a security vulnerability to escalate user privileges—was necessary for conducting in-depth application analysis (Ali et al., 2019; Lindorfer et al., 2015; AL-Dowihi et al., 2023). However, for various reasons discussed in Section 2, jailbreaking has become less feasible. Thus, alternative means of analysis are needed. Currently, one platform, *Corellium*, allows analysts to emulate iPhone hardware and run iOS applications in a controlled, unrestricted environment. However, the high cost and limited customizability of Corellium has rendered it insufficient for many forensic analysts. In this paper, we explore an alternative methodology for analyzing iOS

* Corresponding author. Baggil(i) Truth (BiT) Lab, Center of Computation & Technology Baton Rouge, LA, USA.
E-mail addresses: sseide3@lsu.edu (S. Seiden), andrewwebb@lsu.edu (A.M. Webb), ibaggili@lsu.edu (I. Baggili).

[1] The operating system for iPads has recently been renamed to iPadOS. However, for simplicity, we will refer to both operating systems as iOS throughout this paper.

https://doi.org/10.1016/j.fsidi.2025.301871

Available online 24 March 2025

applications by leveraging the recently introduced ARM-based Mac platform.

The release of ARM-based Macs enables running iOS applications on the macOS platform, offering a new way to simplify the live analysis process of these applications. However, the steps needed to run iOS apps on ARM-based Macs remain complex, with numerous restrictions limiting use. Thus, we developed a new tool, *AppTap*, to streamline live analysis of iOS applications on ARM-based Macs. In our evaluation, we successfully ran 46% of the iOS applications we tested from the App Store on the ARM-based Mac platform. Our case study demonstrated that this methodology enables the effective collection of digital forensic artifacts from iOS applications, affirming its viability for forensic investigations. Our research seeks to address the following research questions (RQs):

RQ1 How can an ARM-based Mac be prepared for analyzing iOS applications?
RQ2 How can the analysis of iPhone applications using ARM-based Macs be streamlined?
RQ3 How effective is this methodology in a digital forensic case study? Is the methodology widely applicable in a real-life situation?
RQ4 What are the benefits of analyzing iOS applications on the macOS platform?

By answering the aforementioned RQs, our work makes the following contributions:

1. We evaluate the practicality of running iOS applications on ARM-based Macs. *To the best of our knowledge, we are the first to apply the capabilities of the new ARM-based Macs to run iOS applications in a forensics context.*
2. We synthesize and evaluate a framework for the forensic analysis of iOS applications' user files on ARM-based Macs.
3. We developed a open-sourced tool, *AppTap*,[2] that automates the forensic examination and artifact discovery process for iOS applications.

The rest of the paper is organized as follows: Section 2 provides a brief history, giving context to the increasing difficulties of iPhone application analysis and illustrating the benefits of our work. In Section 3, we review previous work in the area of digital forensics, emphasizing its significance and how it has been applied to iPhones. Section 4 dives into our new methodology for analyzing iPhone applications, including a technical overview, preparation of the platform for analysis, acquisition of iPhone applications, and the analysis itself. Using these initial findings, we develop a forensic framework in Section 4.5 to standardize the analysis process, which also leads to the creation of AppTap. Section 5 assesses the effectiveness of our methodology. In Section 6, we apply this methodology to a broad spectrum of popular App Store applications, with an evaluation of these results presented in Section 7. Based on our findings, Section 8 outlines potential future directions, and the paper concludes in Section 9.

2. History

Jailbreaking emerged as a method for customizing user iPhone experience, including altering system user interface (UI) elements and accessing settings not typically available. The development of jailbreak tools has declined over time. Most modern jailbreak tools only support iOS devices released up until 2017 (The Apple Wiki, 2024 a). These supported devices are expected to stop receiving updates from Apple in the coming years (The Apple Wiki, 2024 b, 2023 b). This trend is visualized in Fig. 1.

We hypothesize that this trend occurs for a few reasons:

- As the security of iOS has improved, the likelihood of discovering exploitable security vulnerabilities has significantly decreased.
- Fewer iPhone users are jailbreaking their personal devices due to enhancements in the user experience of iOS devices, resulting in reduced profitability from the monetization of jailbreaking tools (via advertisements, donations, etc.).

In addition to decreasing availability of jailbreak tools, it is crucial to consider the integrity of a forensic investigation when using these tools. As noted by The Apple Wiki (2023 a), some jailbreak tools may contain malicious code. Employing such a tool in a forensic examination could compromise the investigation's integrity. To support security analysts, Apple recently introduced its own security analysis platform, the Apple Security Research Device (ASRD). The ASRD allows security researchers to request a jailbroken iOS device directly from Apple, thereby eliminating the need for third-party jailbreak tools (Apple Security Engineering and Architecture (SEAR), 2023). However, access to this platform is heavily restricted, and the terms of service impose limitations on the types of analysis that can be performed.

3. Related work

In this section, we review prior work related to digital forensics, software analysis on physical iOS devices (both jailbroken and non-jailbroken), and software analysis on virtual iOS devices.

3.1. Digital forensics overview

Digital forensics plays a key role in preventing and investigating cybercrime. As described by Casey (2009), digital forensics allows investigators to analyze crimes involving computer systems by extracting digital evidence from said systems. This evidence is referred to by the digital forensics community as forensic *artifacts*. Digital forensics and the collection of artifacts has been previously applied to nearly all platforms (Manna et al., 2021; Ligh et al., 2014; Hoog, 2011).

Once artifacts are collected, they can be shared and utilized by other analysts to showcase examples of data gathered as evidence for an investigation (Grajeda et al., 2018). For example, Grajeda et al. (2023) introduced the *Artifact Genome Project*, allowing forensics investigators from around the world to collaborate and share artifacts they have found

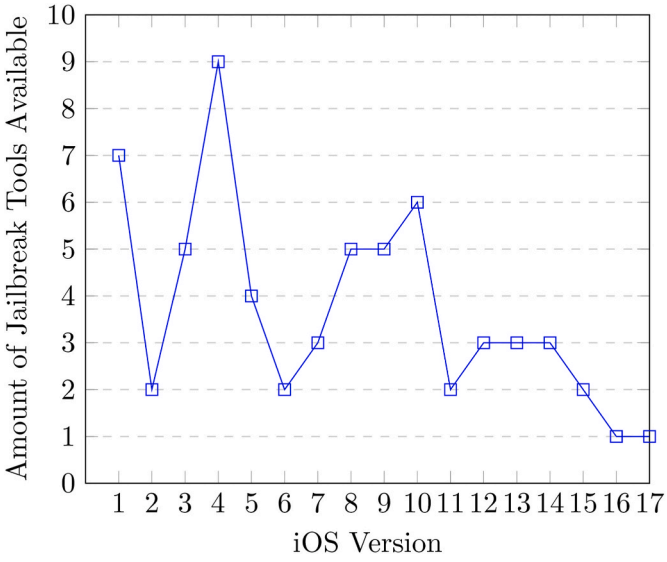

Fig. 1. Trend in jailbreak tools available per iOS update.

[2] *AppTap* source code is available at https://github.com/StevenSeiden/AppTap.

during research and investigations. When it comes to performing forensics investigations involving iOS applications, the methodologies can be separated into those that employ physical devices, and those that use hardware virtualization.

3.2. Physical device analysis

When performing application analysis using physical devices, there are two categories: *jailed* and *jailbroken*. A *jailed* device refers to one that operates on an unmodified version of Apple's operating system. Conversely, a *jailbroken* device has undergone modification through a software exploit to unlock or alter the operating system via privilege escalation (Chang et al., 2015).

Performing application analysis on jailed devices is limited by constraints imposed by Apple. Previous work with jailed devices utilized tools such as *Charles Proxy* to capture network packets from applications (Bhatt et al., 2018). Another popular tool, *Frida*, facilitates application testing through code injection. While Frida is compatible with jailed devices, many of its capabilities are only fully accessible on jailbroken devices (Ravnås, n.d.). Furthermore, researchers have employed iPhone and iPad backups as a method for extracting data in a forensically sound manner from jailed devices (Bader and Baggili, 2010; Husain et al., 2011; Ali et al., 2012; Baggili et al., 2014).

On jailbroken devices, application analysis is well-supported. Jonathan Zdziarski is often credited as the first to utilize physical acquisition on an iPhone using the Zdziarski method (Zdziarski, 2008). Currently, many analysis tools require a jailbroken device to function properly (Beijnum, 2023). However, the declining feasibility of jailbreaking devices has necessitated the exploration of alternative analysis methods (Ricco, 2023). Furthermore, forensic companies that discover zero-days for forensic acquisition on iPhone devices typically do not disclose them publicly, as these discoveries provide a competitive edge.

3.3. Third-party virtualization and other tools

Due to the challenges posed by restricted iOS device hardware, analysts have begun to shift away from using physical hardware for analysis. One of the most significant advancements in iPhone application analysis is the platform Corellium (Brewster, 2018). Since its launch in 2018, Corellium has had a substantial impact on the security analysis of iOS applications (Brewster, 2021). Corellium's platform provides analysts with virtualized iOS devices, allowing for malware analysis without the need for a physical device (Corellium, n.d.). Because these virtualized devices are under Corellium's control, typical restrictions can be easily bypassed as needed. These virtualized devices can test third-party applications (such as those from the App Store), but only after they have been decrypted, which can be challenging to achieve (Corellium, Inc., n.d.). Additionally, customizing the platform is difficult, limiting the range of analysis tools that analysts can use. The high licensing costs further impose a burden on analysts.

A few other tools exist to emulate iOS applications, though these tools are not designed for digital forensics investigations. For example, No Yume (2024) developed *TouchHLE*, an emulator for running older iOS applications on Intel x64 platforms. However, TouchHLE is quite limited as it only supports applications written for iOS version 2. The popular open-source ARM emulator QEMU has been used to emulate older iOS devices, but so far, it has only successfully emulated up to iOS version 2.1 (de Vos, 2022). Projects like Nguyen et al.'s *TruEMU* have also started work on emulating iOS, but they are currently very limited and lack key features, such as displaying applications' user interfaces (Nguyen et al., 2022).

Other tools have been developed for iOS application analysis but have limitations. For example, *Cellebrite* and *Magnet* provide forensic analysis tools for iOS devices (Bays and Karabiyik, 2019; Chamberlain and Azhar, 2019). However, these tools can only extract artifacts after the application has been run and data has been collected from a physical device, and therefore do not support live analysis. Additionally, *MobSF*, a popular mobile application security suite, is capable of performing static analysis of iPhone applications (Bergadano et al., 2020). However, since MobSF does not support real-time application analysis, it cannot uncover digital forensic artifacts for analysis.

Thus, in this paper, we explore the feasibility of virtualizing iOS applications through alternative means using the built-in capabilities of ARM-based Macs. In doing so, we developed a new tool, *AppTap*. AppTap integrates various existing technologies and applications, enabling analysts to easily perform iPhone application analysis through the ARM-based Mac platform. Unlike Corellium, our methodology allows the application to run locally on bare metal within the macOS environment. This approach offers several benefits, including the ability to utilize the numerous macOS forensic tools available today and automate the process. This process is similar to what has been done with Android before, but our methodology adapts this for iOS applications (Anglano et al., 2020).

4. Methodology

In this section, we present the steps required to run iOS applications on macOS for forensic analysis. First, we provide a technical overview for those unfamiliar with iOS/macOS systems in Section 4.1. Next, we prepare the macOS environment for experimentation in Section 4.2. Then, we acquire applications for testing in Section 4.3 and analyze these applications in Section 4.4. Using the knowledge gained from this process, we formulate a forensic framework in Section 4.5. Implementing this framework, we build AppTap to automate the analysis process in Section 4.6. All tools used throughout this section are presented in Table 1.

4.1. Technical overview

Before installation, iOS applications are distributed through iOS Package Archive (IPA) bundles. The bundles take the form of a LZFSE (Lempel–Ziv Finite State Entropy) compressed folder, which can be decompressed with many standard decompression applications by renaming the file from *example.ipa* to *example.zip* (Ke, 2017; Apple, Inc., n.d. a). Decompressing these files can allow us to manually analyze the bundle's contents. The file structure of these bundles is shown in Fig. 2.

Contained within this bundle are several metadata files and the application itself (**.app*). This application file is a bundle that contains an executable and necessary resources (Apple, 2017). The contents within IPA bundles downloaded from Apple come encrypted with *FairPlay* (The Apple Wiki, 2010). Therefore, you cannot execute the bundled application as you would with a standard macOS application. To install

Table 1
A list of tools used for application testing.

Tool	Purpose
Apple Configurator[a]	Imaging Hardware Firmware
AppTap	Installation & File Analysis
cliclick[b]	Automating User Interaction
DB Browser for SQLite[c]	Reading Application Files
DirEqual[d]	Checkpoint Comparison
fs_usage	Filesystem Activity Tracking
Hex Fiend[e]	Reading Hex-Encoded Files
ProvisionQL[f]	Analyzing .IPA Properties
Sideloadly[g]	Sideloading .IPA Bundles

[a] Version 2.17, available at https://apps.apple.com/us/app/apple-configurator/id1037126344?mt=12.
[b] Version 5.1, available at https://github.com/BlueM/cliclick.
[c] Version 3.12.2, available at https://sqlitebrowser.org/dl/.
[d] Version 3.12.2, available at https://sqlitebrowser.org/dl/.
[e] Version 2.17.1, available at http://hexfiend.com/.
[f] Version 1.6.4, available at https://github.com/ealeksandrov/ProvisionQL.
[g] Version 0.24, available at https://sideloadly.io/.

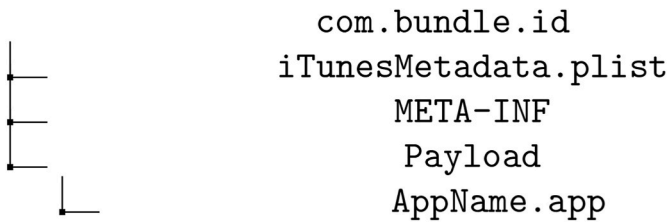

Fig. 2. Typical content structure of IPA

these bundles on an ARM-based macOS device, decryption is required. This process typically occurs automatically on devices like iPhones as part of application installation.

With the release of macOS 11.3, the built-in ability to decrypt and install IPA files has been restricted to authorized applications, such as those downloaded from the Mac App Store (MAS). Many iPhone applications are not authorized to be installed via the MAS (as explained in Section 4.3), meaning macOS 11.3 and later significantly limit the range of applications that can be freely installed. It is currently unknown how to break FairPlay's hardware decryption keys, so we must rely on Apple's built-in decryption tools (Khoa, 2021). Thus, the system must be running macOS 11.2.3 or earlier to successfully perform application installation. Once installed, these applications run natively on the macOS platform, utilizing iOS dependencies that Apple has ported to macOS (Apple, Inc., n.d. *c*). When running these applications on a Mac, macOS emulates many features of a typical iOS device, such as the touchscreen. Additionally, macOS relays emulated device specifications to these applications, including the version of iOS it is emulating.

Note, iOS applications store user data within a single directory, called a group container or a sandbox. On macOS, an application's sandbox is located at `~/Library/Group Containers/group.com.bundle.name`. For our study, we acquire artifacts by analyzing these sandbox files.

4.2. Platform setup

The first step is installing a capable version of macOS on a compatible ARM-based device. For our purposes, we are using an M1 Apple Mac Mini (A2348). To downgrade the device to our desired firmware version, macOS 11.2.3, we utilized Apple Configurator and another Mac. Using the appropriate firmware bundle, we successfully flashed the Mac Mini. Once this was complete, the computer was ready to locally decrypt and install IPA files for analysis.

4.3. Application acquisition

There are many ways to acquire iOS applications. The easiest method is via the MAS. With the release of ARM-based M1 Macs, Apple made iPhone applications available for download through the MAS, as seen in Fig. 3. This allows users to easily run these applications on macOS (Apple, Inc., 2020). Shortly after this release, developers were provided with the option to opt-out of having their applications listed on the MAS, and many developers have chosen to do so (Espósito, 2020). Another method is to use a jailbroken iOS device to acquire these IPA files, but since this research aims to move away from jailbreaking, this method is insufficient. Consequently, acquiring iOS applications on macOS has become a challenge.

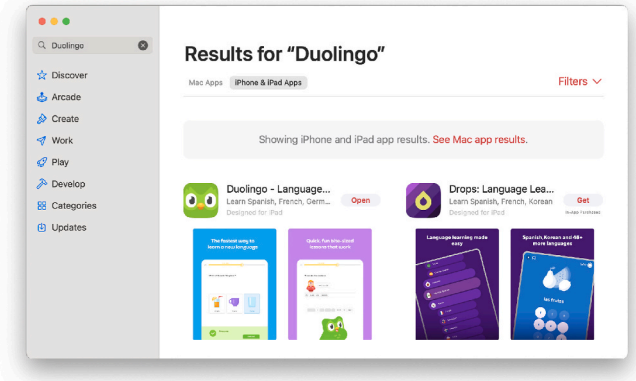

Fig. 3. Searching for an iOS app on the Mac App Store.

To circumvent these challenges, one option is to use an application like the open-source *IPATool*, which allows users to easily download IPA files directly from Apple (Alfhaily, 2023). Alternatively, to avoid using third-party tools, one can utilize the official *Apple Configurator*, requesting the tool to install an application on an iOS device, and then use the downloaded IPA from the tool's cached files (Anderson, 2020). Of course, these bundles are encrypted and unauthorized by the MAS. *However, since we downgraded the machine to macOS 11.2.3, these applications will be decrypted and installed by macOS regardless of authorization status.* Therefore, installing these applications is as simple as opening them with the built-in iOS App Installer.

4.4. Application analysis

Once we have installed our applications, we begin analysis. It is important to note that while many iOS applications will run successfully on the macOS platform, some applications have dependencies that do not exist on macOS and thus may fail to install or crash upon launching. To automate application analysis, we decided to use the open-source application *cliclick*. This program allows for automating cursor interactions with applications. By using cliclick in conjunction with the built-in *fs_usage* (a simple utility that tracks processes' file system activity), we can monitor the files being actively read and written by iOS applications, as seen in Fig. 4. These files can be collected and stored as artifacts automatically for later analysis. In our preliminary study, we found that many applications we tested, such as *Firefox* and *Discord*, stored user files in SQLite format. User data from these databases can be extracted and visualized, as seen in Fig. 5. This data can contain valuable artifacts for potential use in a forensic investigation.

4.4.1. Virtualization

Running iOS applications within macOS additionally allows for environment virtualization. Virtualization of applications on macOS has several benefits, the primary being the ease of performing memory dumps to conduct memory forensics. Memory forensics is an extremely important part of application analysis, as it enables analysts to find non-persistent data that only exists in memory (Ligh et al., 2014).

Popular virtualization tools, such as VMware Fusion, can be utilized to create ARM-based macOS Virtual Machine (VM)s (Broadcom, 2022). Using a VM, a memory dump can be acquired and forensically analyzed. Virtualization support for ARM macOS was introduced in macOS 12 with the addition of the series of *VZMac* API series (Apple, Inc., n.d. e). ARM versions of macOS 11 do not support this and thus cannot be virtualized. However, as previously mentioned, only these earlier versions of macOS 11 allow for decryption of these unauthorized IPAs. In order to virtualize iOS applications not available on the MAS, an IPA must be decrypted outside of the VM, then transferred to the VM and sideloaded. There are several methods for exporting decrypted applications, such as employing a macOS 11 installation or a jailbroken iOS device. However, one must always ensure to abide by local laws and regulations surrounding decryption and DRM. We will not touch on this process as it falls beyond the scope of this work.

Once a decrypted application is copied to the VM, it can be sideloaded. Sideloading is a process that involves *signing* an IPA with an Apple Developer account and installing it. There are several third-party applications for sideloading IPAs on macOS; for our purposes, we

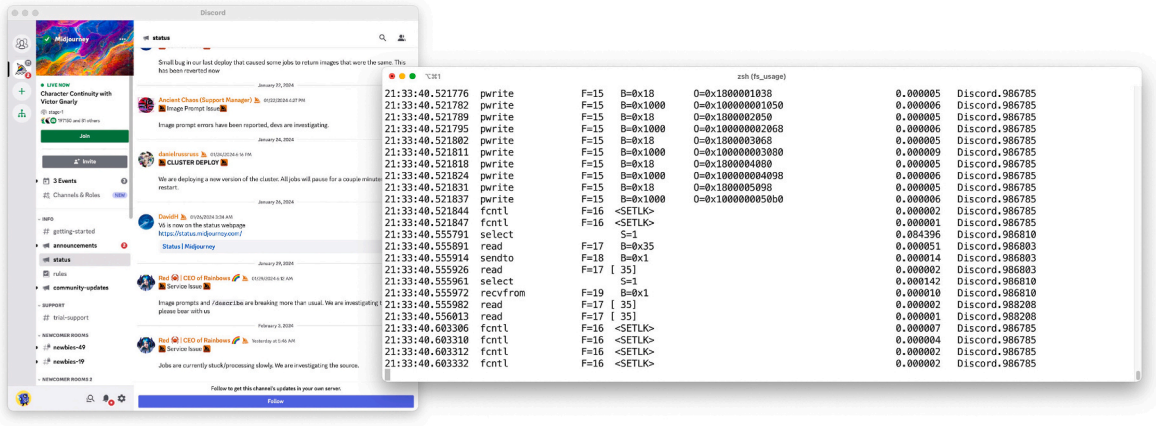

Fig. 4. Automated Analysis of the Discord iPhone Application. User Input Is Automated Via *cliclick*, and Filesystem Activity Is Tracked With *fs_usage*.

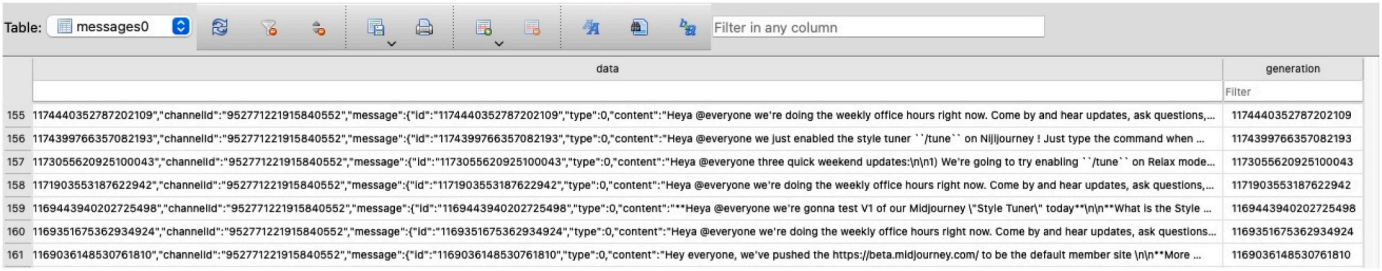

Fig. 5. User messages being read from the discord Application's saved files with DB browser.

decided to use *Sideloadly* (Goodwin and Woolley, 2022; Sideloadly, n. d.). Once an application has been sideloaded, it can be run normally within the VM.

4.5. Forensic framework

To streamline the application analysis process, we developed a forensic framework for iOS applications. This framework outlines the process of collecting artifacts from iOS applications and associating them with user actions. To attribute artifacts to specific user actions, we monitor the creation and modification of artifacts, timestamping these actions to establish associations. To accomplish this, we break down the process step-by-step.

First, we capture and intercept user input to an application (in our case, a mouse click). This input is not transmitted to the application but is recorded for future reference. Next, we ask the user to specify their intended action within the application. Using this information, we create a checkpoint of the application's sandbox, labeling it as the state before the user's intended action. Finally, we forward the user's original input to the application, allowing it to create and modify any new artifacts. This process is repeated several times. By using a series of checkpoints, an analyst can determine which user actions result in the creation of specific artifacts.

We formalize this framework, representing successful artifact collection $AC()$ of artifacts ar. This collection methodology relies on several filesystem snapshots fs to be taken, capturing the files used by a process at various times. First, the application to be analyzed (target application) must be running as a process p. Next, the analyst should begin recording the filesystem's activity f and the user's actions a with the target application, (such as recording a mouse click as cursor location coordinates).

Once this recording begins, the analyst must undertake an action a within the target application via an interaction i. The interaction i describes the user input itself, while the action a describes the intended consequences of said user input. For example, when clicking on a send button, the action a would be sending a message, and the interaction i would be clicking at a specific location on the screen (where the send button is).

When the user commits an interaction i on an application they are analyzing, this interaction should be intercepted and thus **disallowed from reaching the target application p**. Instead, a filesystem snapshot fs should be taken, storing all relevant information for this interaction. Only once all this information is collected should the target application p receive the interaction. In taking the snapshot, the following should occur:

- Ask the user to specify the action a in a string format. For example, "clicking on the send button".
- Take filesystem snapshot fs, copying all application files to a directory that will not be modified further. Label this snapshot as containing the filesystem information immediately prior to interaction i being sent to target application p. Therefore, this snapshot will be associated the state of the filesystem immediately before action a takes place.
- Forward interaction i to target application p, causing action a to occur as normal.

This process of interacting and snapshotting will repeat n times, until the analyst has collected sufficient data. From there, the analyst will perform differential forensic analysis to compare the differences between distinct filesystem snapshots $fs_n - fs_{n-1}$. This reveals the changes in the filesystem between each snapshot. In doing so, the examiner can attribute artifacts ar extracted from individual snapshots to different intended actions a. We share the mathematical formulation of the procedure below:

$$AC(ar, f, fs, n, p, a, i) = AR(FS(f, a, i, n), n) \wedge \sum_{0}^{n} FS(f, a, i, n) \quad (1)$$

$$AR(f, n) = FS(n)_f - FS(n-1)_f \quad (2)$$

$$FS(f, a, i, n) = F(n), \wedge A(n) \wedge I(n) \quad (3)$$

where:

$AC()$ is the successful collection of artifacts
$AR()$ is a collection of artifacts from the filesystem f at time n
$FS()$ is an individual filesystem snapshot
f is the overall filesystem activity
n is the desired snapshot index
p is the process of the target application
a is the action associated with the snapshot
i is the intention associated with the snapshot

4.6. Automation

To expedite the process of acquiring, installing, and testing applications, we constructed a custom application, *AppTap*. AppTap automates different existing tools, such as *ipatool*, which enables acquiring IPA files from the application store (Alfhaily, 2023). The program was written in Swift and features a command-line interface, as shown in Fig. 6. An overview of the installation of AppTap is shown in Fig. 7.

Fig. 8 shows an overview of *AppTap*'s filesystem analysis framework. The actual implementation of this framework within the macOS platform is illustrated in Fig. 9. All artifacts are pulled from the sandbox location mentioned in Section 4.1. These filesystem analysis features allow the analyst to take snapshots of the application's files between each interaction. Attribution of artifacts is performed automatically as the application is run. This form of differential forensic analysis, described by Garfinkel et al. (2012), enables an examiner to correlate different actions with the creation of specific artifacts.

When launching AppTap, the program first asks the user to select a sandbox location to analyze. Next, the user interacts with the application. As described above, each time the user interacts (e.g., clicking a button), the input is delayed and a snapshot occurs. AppTapp stores these snapshots in the directory of its execution.

5. Case study

To evaluate the efficacy of AppTap and our methodology, we conducted a case study in which an analyst is seeking artifacts from a custom-made iOS application. We created a simple iOS application, *NewList*, in Swift and Swift UI. This application stores a list of strings

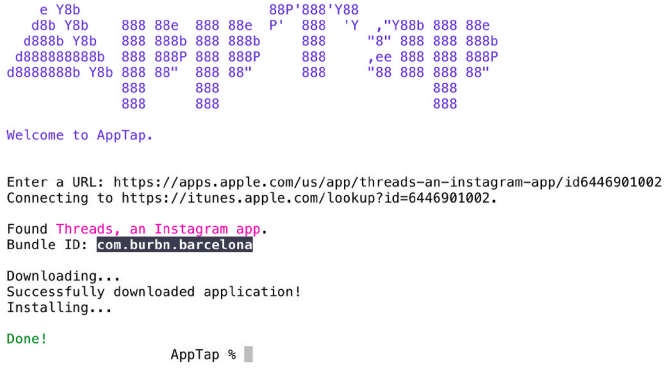

Fig. 6. Downloading and installing apps using AppTap.

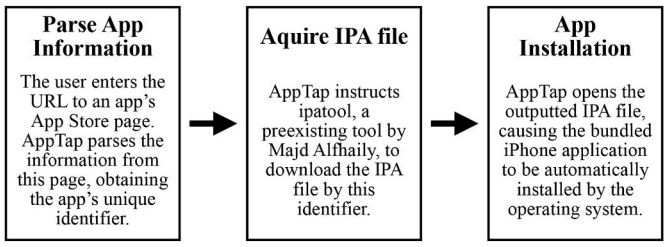

Fig. 7. The streamlined acquisition and installation process provided by AppTap.

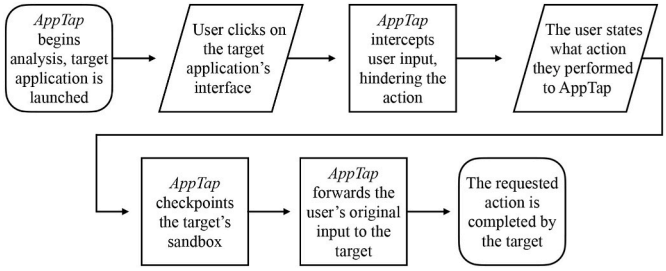

Fig. 8. An overview of AppTap's application filesystem activity analysis process, following our forensic framework.

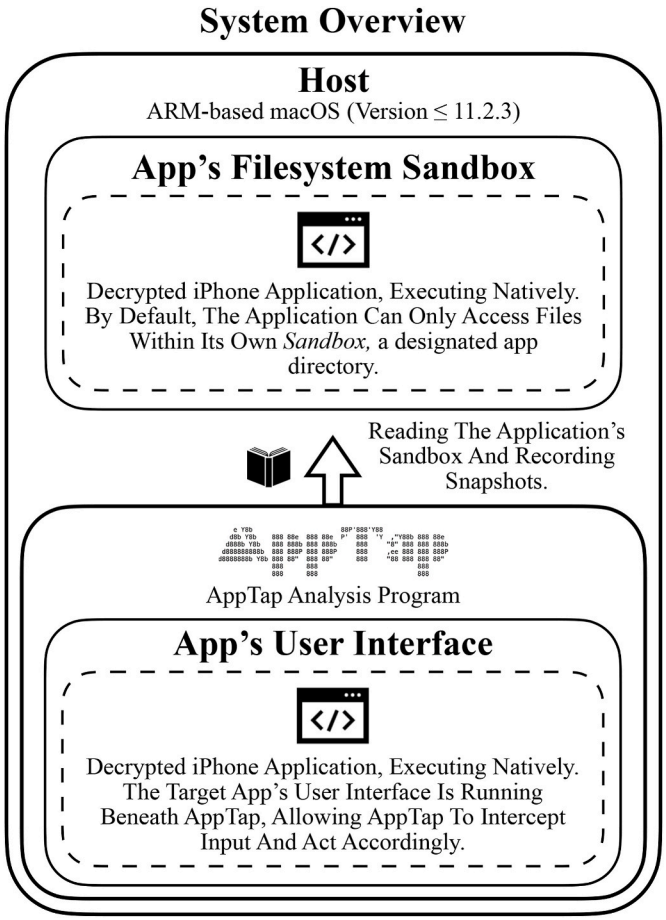

Fig. 9. AppTap analysis system overview.

from which the user can add and remove entries. This list is persistent and stored within NewList's sandbox. Using this simple program, we can generate filesystem artifacts on command to test whether or not our

artifact attribution is successful.

The process is as follows: First, we launched NewList. Then, we launched the *AppTap Interaction Logger*. With the running Interaction Logger, we initiate creating a new entry in NewList with the title "AddingAnArtifact". The Interaction Logger takes a snapshot, Snapshot 1. Next it follows through on the request, creating the "AddingAnArtifact" entry within NewList. Finally, we closed NewList, which creates Snapshot 2.

Next, we explored differences between the snapshots. We utilized *DirEqual*, a tool for comparing differences between two directories. As seen in Fig. 10, the tool highlighted a difference between the two snapshots with file `~/Data/Library/Preferences/x.NewList.plist`. We opened the two `x.NewList.plist` files in *Hex Fiend* and compared them. As shown in Fig. 11, Snapshot 1 does not contain the string "AddingAnArtifact", while Artifact 2 does. Thus, we can attribute the action taken between Snapshot 1 and Snapshot 2 to the creation of this artifact. By examining how *AppTap* works in a realistic scenario, we show how our approach can be used to successfully allow analysts to perform a live analysis of applications to determine when artifact creation takes place.

6. Results

We started our evaluation by testing the success of our analysis methodology with different applications on the iOS App Store. To do this, we examined installing the top free apps and games from the United States iOS App Store (Apple separates these into two *Top 100* lists, so n=200). This list was obtained from (Apple, Inc., 2024 a,b). The raw data of this evaluation is detailed in Appendix Appendix A, and the results are shared in Fig. 12. Overall, we successfully ran 92 (46%) applications. In our testing, the largest point of failure was applications refusing to install, accounting for 61 (30.5%) of the examined applications. Upon examining the properties of these applications with *ProvisionQL*, we found that many required a later version of iOS than the version macOS reports to applications during emulation. Our macOS 11.2.3 installation reported support for up to iOS 14.4, an operating system older than that supported by several applications that refused to install. For example, the Airbnb and Amazon Prime Video applications specify a minimum of iOS 16.0 and iOS 15.0 respectively, surpassing the specifications of our macOS installation. We posit that this is the cause of applications failing to install. We also found that 42 (16%) applications installed but exhibited an exception error when running. The other 15 (7.5%) applications exhibited numerous other errors, such as freezing, missing dependencies or detecting that they were running in an environment with elevated user privileges (often called jailbreak detection).

In an attempt to address the most prominent issue—applications refusing to install—we theorized two methodologies: modifying the applications themselves or modifying the operating system. First, we explored modifying the applications. However, we found this approach to be unfeasible. The applications are encrypted when acquired and are only decrypted upon installation. This creates a paradoxical situation where modification cannot occur until they are decrypted, but they are not decrypted until they are installed.

Next, we explored modifying the operating system to allow application installation. In the past this has been possible on both iOS and macOS. One way to do this is to modify the version number reported by the operating system, which is stored at `~/System/Library/CoreServices/SystemVersion.plist` on both iOS and macOS. However, doing this also proved difficult. macOS comes with many built-in security protections, such as System Integrity Protection (SIP), which prevents the operating system from being modified (Apple, Inc., n.d. b). These features can be disabled. However, numerous attempts to modify `~SystemVersion.plist` resulted in rendering the system unbootable. Therefore, this approach was also unsuccessful. In the future, we would like to explore this circumvention further, as we believe that this is a promising route to running applications if successfully implemented.

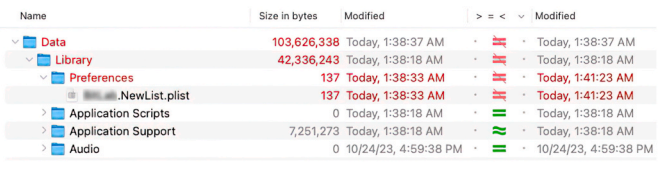

Fig. 10. Finding the Changes Between Two AppTap Snapshots using *DirEqual*.

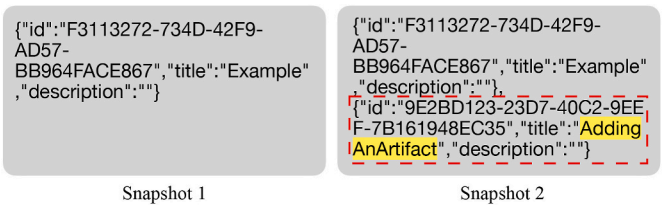

Fig. 11. Comparing the data within two AppTap snapshots. Snapshot 2 contains a new artifact from snapshot 1.

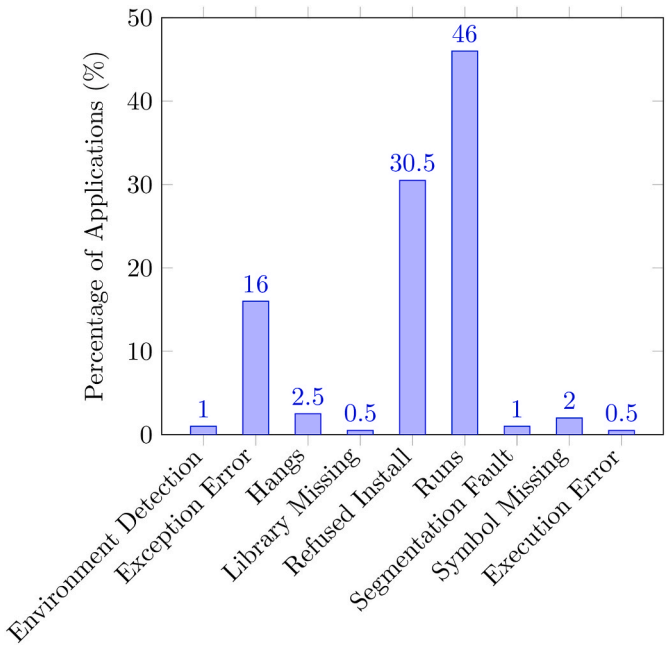

Fig. 12. Evaluation Results Against Popular iOS Apps (Percentages).

7. Discussion

RQ1: How can you prepare an ARM-based Mac for analyzing iOS applications? First, an analyst should acquire the appropriate hardware. As described in Section 4.1, the primary requirement for hardware is an ARM-based Mac that can run macOS 11.2.3 or earlier, which can be determined using a variety of third-party technical specification websites. Next, the appropriate version of macOS and relevant analysis applications must be installed. This includes installing AppTap, ipatool, and their dependencies. As a result, an appropriate environment for iOS application acquisition and analysis will be achieved.

RQ2: How can the analysis of iPhone applications using ARM-based Macs be streamlined? There are two main hurdles in iPhone application analysis on macOS: App acquisition, and the analysis itself. In Section 4.3, we discussed the difficulty in acquiring iPhone applications, and described how ipatool can simplify this process. By hooking the existing ipatool application into AppTap, we were able to simplify the process of acquiring applications, enabling the analyst to automate

this process. To simplify the analysis process, we built AppTap's filesystem analysis feature around the forensic framework we defined in Section 4.5. With this tool, we are able to determine at what time different filesystem artifacts are created automatically. By using AppTap, application acquisition and artifact attribution is facilitated for the investigator automatically, streamlining this process.

RQ3: How effective is this methodology in a digital forensic case study? Is the methodology widely applicable in a real-life situation? Beginning in Section 4.4, we explored what data could be collected from iPhone applications using ARM-based Macs. We chose to focus on filesystem data, and used a methodology typical for Mac application analysis. Having successfully located many user files, such as browsing history and chat messages, we decided to focus on filesystem artifact attribution. In Section 5, we examined our filesystem analysis methodology in a controlled case study. By creating a custom made iPhone application and testing it through AppTap, we were able to say with full confidence that the artifacts were created at the time AppTap specified. In doing so, we were able to see that AppTap successfully identifies artifacts created and attributes them to the appropriate user action. In the context of a forensics investigation, this can be utilized to tell what actions cause an application to behave maliciously. It can also be utilized to discover the context behind how a benign application was used maliciously. Testing this methodology against a wide variety of App Store applications, we found that 46% of these applications ran successfully and could be analyzed. Therefore, our methodology allows an analyst to save time and more easily conduct an analysis about half of the time, making this platform a worthwhile investment.

RQ4: What benefits does analyzing iOS applications on the macOS platform bring? Finally, we evaluate the benefits that using this methodology provides when compared to existing methods. Given the success in running numerous popular applications, we compare AppTap to other tools available on the market (Table 2). We find that our AppTap analysis methodology provides several advantages: Unlike tools such as Correllium, our open-source methodology provides a modular and cost effective way to extract artifacts from iOS applications. Compared to jailbreaking a device or acquiring an ASRD, our methodology's platform is more accessible, since no software exploit is required and ARM-based Macs are available to purchase by the public. Additionally, we compare our tool to Cellebrite and Magnet, which are developed for the collection of preexisting forensic evidence, rather than live artifact collection. Compared to these other tools, our platform allows for a live analysis, enabling an analyst to easily attribute different actions to different artifact collection.

8. Future work

In the future, we would like to incorporate additional forensic analysis tools within *AppTap*. For example, we are interested in adding the ability to analyze network traffic and I/O activities. Further automating the filesystem checkpoints would also prove useful. For example, one area we would like to improve is the means of providing context behind each snapshot. Implementing an automatic checkpoint labeling system would allow us to eliminate the interruption caused by asking the user to specify what action they are taking between each interaction with the target application.

Furthermore, future work should increase the amount of applications supported by *AppTap*. Although not easy, we posit that this can be performed by spoofing the operating system version reported by macOS. However, forcing applications to run in an unsupported environment can lead to unexpected results. Therefore, we must proceed carefully, considering supporting applications on a case-by-case basis. For example, if the implementation of application dependencies has changed, spoofing version numbers should be avoided.

Finally, we would like to do an in-depth comparison between artifacts generated on iOS devices and macOS devices and determine if any are generated by background noise. This is to see what potential discrepancies may exist, as this is a limitation within our current work.

9. Conclusion

Overall, we found that utilizing the ARM-based Mac platform allowed for the successful dynamic analysis of iOS applications. After performing an extensive platform setup and exploring the manual analysis of iOS applications, we were able to formulate a forensic framework for the live analysis of iOS applications. Using this framework, we were able to develop *AppTap*, a tool that not only enables analysts to easily acquire and install applications, but perform live analysis and filesystem artifact attribution. In our evaluation, we showed that nearly half of the tested iPhone applications can be analyzed using our methodology. Despite not supporting various applications, this approach can still prove very beneficial, saving analysts time and money. Further research is required to expand the amount of supported iOS applications. Future work should explore additional analysis techniques, strengthening the benefits that *AppTap* can provide to the cybersecurity community.

Appendix A

Top free iPhone applications and games in the United States App Store

Table A.3
Using *AppTap* to install the top 100 free iPhone apps and top 100 free iPhone games from the App Store, then running these applications to see how they react in the macOS environment. Our results can be interpreted as follows: *Runs* indicates that the application ran successfully. *Refused Install* indicates that macOS refused to install the application, often due to a mismatched firmware version requirement. *Environment Detection* indicates that the application detected an anomaly in its runtime environment and voluntarily blocked its usage. *Hangs*,

Table 2
Comparing AppTap to alternatives.

	AppTap	Jailbreaking	Correllium	ASRD	Cellebrite	Magnet
Virtualization	Full	None	Full	None	N/A	N/A
Third-Party Tools	Full	Limited	None	None	None	None
Artifact Attribution	Full	Limited	Limited	Limited	Limited	Limited
Availability	Easy	Hard	Easy	Hard	Hard	Hard
Cost	Minimal	Minimal	High	Minimal	High	High

Symbol Missing, *Library Missing*, *Segmentation Fault*, and *Execution Error* all indicate a generic error preventing analysis.

Application Name	Version Number	Analysis Result
2 Player Games: the Challenge	6.7.2	Exception Error
8 Ball Pool™	55.2.1	Exception Error
Airbnb	24.11	Refused Install
Alien Invasion: RPG Idle Space	3.0.52	Exception Error
AliExpress Shopping App	8.94.0	Runs
Amazon Prime Video	10.17	Refused Install
Amazon Shopping	23.6.0	Refused Install
American Airlines	2024.10	Refused Install
Among Us!	2024.3.5	Runs
Angry Birds 2	3.20.0	Exception Error
Animal Hunter: Wild Shooting	4.0.2	Exception Error
AXS Tickets	6.6	Runs
BitLife - Life Simulator	3.13.1	Runs
Block Blast!	3.9.1	Runs
Blood Strike - FPS for all	1.003.40	Runs
BMI Calculator – Weight Loss	1.8.9	Refused Install
Brain Test: Tricky Puzzles	2.748.0	Runs
Braindom: Brain Games Test Out	2.1.9	Runs
Brawl Stars	54.243	Execution Error
Bridge Race	3.46	Exception Error
Build A Queen	0.81	Exception Error
Business Empire: RichMan	1.11.20	Runs
Call of Duty®: Mobile	1.0.43	Runs
Candy Crush Saga	1.273.0	Runs
Canva: Design Art & AI Editor	4.102.0	Runs
CapCut - Video Editor	10.7.0.62	Hangs
Capital One Mobile	6.4.0	Refused Install
Capital One Shopping: Save Now	2.49.0	Runs
Cash App	4.39	Refused Install
Charades - Best Party Game!	2.7.13	Refused Install
ChatGPT	1.2024.073	Refused Install
Chess - Play & Learn	4.3.5	Refused Install
Chick-fil-A	2024.4.0	Refused Install
Clash of Clans	16.137.13	Exception Error
Clash Royale	6.256.1	Exception Error
Clean It: Restaurant Cleanup!	1.3.4	Exception Error
Coin Master	3.5.1530	Runs
CookieRun: Witch's Castle	1.0.201	Exception Error
Costco	24.2.2	Runs
Crossy Road	6.2.0	Exception Error
Cryptogram: Word Brain Puzzle	2.0.4	Exception Error
Dice Dreams	1.74.2	Runs
Discord - Chat Talk & Hangout	201.0	Runs
Disney+	3.0.2	Refused Install
Domino's Pizza USA	11.2.0	Refused Install
DoorDash - Dasher	2.315.0	Refused Install
DoorDash - Food Delivery	6.10.0	Refused Install
Doorman Verify Neighbor Game	1.7	Runs
DraftKings Sportsbook & Casino	4.33.1	Refused Install
Duolingo - Language Lessons	7.15.0	Refused Install
EA SPORTS FC	20.1.02	Runs
eBay Marketplace: Buy and Sell	6.151.0	Refused Install
ESPN: Live Sports & Scores	7.2.0	Refused Install
Etsy: Home Style & Gifts	6.65.1	Refused Install
Eventbrite	9.67.0	Refused Install
Evony	4.68.1	Runs
Expedia: Hotels Flights & Car	2024.10	Refused Install
Facebook	455.0.0	Exception Error
Falling Art Ragdoll Simulator	0.11.0	Exception Error
FanDuel Sportsbook & Casino	1.91.0	Runs
Fly Delta	5.42.1	Refused Install
Gardenscapes	7.7.5	Runs
Geometry Dash Lite	2.21.3	Runs
Gmail - Email by Google	6.0.231224	Refused Install
Going Balls	1.1.102	Runs
Google	298	Refused Install
Google Chrome	123.6312.52	Refused Install
Google Drive	4.2411.11802	Refused Install

(*continued on next page*)

Table A.3 (*continued*)

Application Name	Version Number	Analysis Result
Google Maps	6.107.3.49540	Runs
Google Meet	235.0	Refused Install
Google Photos: Backup & Edit	6.73.0	Refused Install
Google Translate	8.4.0	Refused Install
GossipMaster	11.1	Segmentation Fault
GTA: San Andreas - NETFLIX	1.72.42919648	Refused Install
Hexa Sort	1.7.03	Exception Error
Hulu: Watch TV shows & movies	8.3.2	Runs
I Want Watermelon	1.0	Runs
Idle Racer - Tap Merge & Race	0.9.103	Exception Error
Impulse - Brain Training	1.30.11	Hangs
Indeed Job Search	200.0	Refused Install
Instagram	306.0.0	Refused Install
Intuit Credit Karma	24.12	Runs
Kingdom Guard:Tower Defense TD	1.0.409	Runs
Klondike Adventures: Farm Game	2.119.1	Runs
Last War: Survival	1.0.193	Runs
Lemon8 - Lifestyle Community	5.9.1	Runs
Life360: Find Friends & Family	24.10.0	Refused Install
LinkedIn: Network & Job Finder	2024.0314.0932	Refused Install
Little Caesars Pizza	24.3.0	Runs
Lyft	15.50.3	Refused Install
Magic Tiles 3: Piano Game	11.021.008	Runs
Makeup ASMR: Makeover Story	2.2	Runs
Match Factory!	1.13.81	Runs
Math Puzzle Games - Cross Math	3.3.0	Runs
Max: Stream HBO TV & Movies	3.4.0	Symbol Missing
McDonald's	8.0.0	Refused Install
Messenger	449.0.0	Runs
Microsoft Authenticator	6.8.6	Refused Install
Microsoft Copilot	420313002.28.0	Refused Install
Mob Control	2.67.3	Exception Error
Modern Community	1.1008.81088	Runs
MONOPOLY GO!	1.19.2	Runs
Monster Never City	1.05.230	Runs
My Perfect Hotel	1.8.5	Runs
NBA 2K24 MyTEAM	204.03.223770223	Exception Error
Netflix	16.22.0	Refused Install
Nike: Shoes Apparel Stories	24.18.0	Refused Install
NYT Games: Word Games & Sudoku	4.70.1	Refused Install
Offline Games - No Wifi Games	1.7.2	Exception Error
Outlets Rush	1.42.0	Runs
Paper.io 2	3.17.0	Runs
Paramount+	15.0.04	Symbol missing
ParkMobile: Park. Pay. Go.	24.7.0	Refused Install
PayPal - Send Shop Manage	8.58.0	Runs
Peacock TV: Stream TV & Movies	5.3.11	Runs
Pinterest	12.9	Runs
Pixel Heroes: Tales of Emond	1.2.2	Runs
Pizza Hut - Delivery & Takeout	5.33.0	Runs
Pizza Ready!	1.2.0	Runs
PlayStation App	24.2.0	Runs
Pocket FM: Audio Series	2.7.4	Refused Install
Pokemon GO	0.305.1	Refused Install
Project Makeover	2.84.1	Exception Error
PUBG MOBILE	3.1.0	Runs
Reddit	2024.12.0	Exception Error
ReelShort	1.5.05	Runs
Riddle Test: Brain Teaser Game	1.8.4	Exception Error
Ring - Always Home	5.70.0	Refused Install
Roblox	2.616.655	Runs
Royal Match	20500	Runs
Shazam: Find Music & Concerts	17.6.0	Refused Install
SHEIN - Shopping Online	10.6.2	Runs
Shop: All your favorite brands	2.148.0	Runs
ShortTV - Watch Dramas & Shows	1.7.1	Runs
Snake Clash!	1.10.0	Runs
Snake.io - Fun Online Snake	1.19.93	Runs
Snapchat	12.78.0.32	Runs

(*continued on next page*)

Table A.3 (*continued*)

Application Name	Version Number	Analysis Result
Solar Smash	2.3.4	Exception Error
Solitaire	8.0.0	Runs
Solitaire Cash	7.6.0	Hangs
Solitaire Clash: Win Real Cash	1.1.44	Environment Detection
Solitaire Grand Harvest	2.360.0	Exception Error
Spotify - Music and Podcasts	8.9.19	Refused Install
Starbucks	6.68	Refused Install
Stumble Guys	0.67	Exception Error
Subway Surfers	3.26.0	Runs
Super Slime - Black Hole Game	2.13.0	Hangs
Survivor!.io	2.6.3	Runs
Taco Bell Fast Food & Delivery	8.40.3	Refused Install
Target	2024.11.0	Refused Install
Teacher Simulator	1.8.1	Runs
Telegram Messenger	10.9.2	Runs
Temple Run	1.25.1	Runs
Temu: Shop Like a Billionaire	2.35.0	Runs
Tetreault	2.0.0	Hangs
Tetris	5.13.1	Exception Error
The Roku App Official	10.1.0	Symbol Missing
Threads an Instagram app	313	Runs
Ticketmaster–Buy Sell Tickets	249.0	Refused Install
TikTok	33.8.0	Runs
Tile Family: Match Puzzle Game	1.46.3	Runs
Township	17.1.0	Runs
Traffic Escape!	3.1.0	Runs
Train Miner: Idle Railway Game	1.6.5	Runs
Travel Town - Merge Adventure	2.12.511	Exception Error
Trivia Star: Trivia Games Quiz	1.278	Runs
Tubi: Movies & Live TV	8.5.1	Refused Install
Twisted Tangle	1.43.1	Runs
Twitch: Live Streaming	18.7.2	Runs
Uber - Request a ride	3.606.10000	Refused Install
Uber Eats: Food Delivery	6.208.10000	Refused Install
UNO!	1.12.5498	Runs
Venmo	10.36.0	Runs
VPN - Super Unlimited Proxy	1.9.38	Runs
Vrbo Vacation Rentals	2024.11	Refused Install
Walmart: Shopping & Savings	24.10	Refused Install
War of Evolution	70075	Runs
Water Sort Puzzle	16.0.0	Runs
Water Sort Puzzle: Get Color	5.2.3	Exception Error
Waze Navigation & Live Traffic	4.102.0	Refused Install
We Are Warriors!	1.23.0	Exception Error
Weapon Master: Gun Shooter Run	2.7.2	Runs
Wendy's	10.4.0	Runs
WhatsApp Messenger	23.25.85	Environment Detection
Whiteout Survival	1.15.1	Runs
Wood Nuts & Bolts Puzzle	3.7	Exception Error
Word Search Explorer: Fun Game	1.58.0	Runs
Wordle!	1.54.0	Runs
Words of Wonders: Crossword	4.5.22	Exception Error
Words With Friends 2 Word Game	21.60	Runs
Wordscapes	2.16.0	Runs
X	10.33	Refused Install
Yelp: Food Delivery & Reviews	24.12.0	Refused Install
YouTube Music	6.44.2	Library Missing
YouTube: Watch Listen Stream	19.10.5	Symbol Missing
Yuka - Food & Cosmetic scanner	4.34	Runs
Zelle	8.6.1	Segmentation Fault
Zen Word - Relax Puzzle Game	1.33.0	Runs
Zillow Real Estate & Rentals	16.74.1	Refused Install
Zoom - One Platform to Connect	5.17.11	Runs

References

AL-Dowihi, L.W., Alogaiel, R.M., Alomari, M.M., Alahmadi, R.N., Alsadah, S.K., Alghulayqah, H.S., Alattas, H.T., 2023. Mobile Investigation; Forensics Analysis of Ios Devices.

Alfhaily, M., 2023. Ipatool. https://github.com/majd/ipatool.

Ali, A.A., Cahyani, N., Jadied, E., 2019. Digital forensic analysis on idevice: jailbreak ios 12.1. 1 as a case study. Indonesia J. Computing (Indo-JC) 4 (2), 205–218.

Ali, S., AlHosani, S., AlZarooni, F., Baggili, I., 2012. ipad2 logical acquisition: automated or manual examination?. In: Proceedings of the Conference on Digital Forensics, Security and Law', pp. 113–128.

Anderson, M., 2020. How to Download Ipa Files for Ios. https://appletoolbox.com/download-ipa-files-ios/.

Anglano, C., Canonico, M., Guazzone, M., 2020. The android forensics automator (anfora): a tool for the automated forensic analysis of android applications. Comput. Secur. 88, 101650.

Apple, I., 2017. Bundle Programming Guide. https://developer.apple.com/library/archive/documentation/CoreFoundation/Conceptual/CFBundles/Introduction/Introduction.html.

Apple, Inc, 2020. Sessions Wwdc20: Platforms State of the Union — Apple Event 2020 in Detail. https://youtu.be/VAcVal5HWbE?si=09RCwv2vR4ygwyIf.

Apple, Inc, 2024a. Top Free Iphone Apps on the App Store. https://apps.apple.com/us/charts/iphone/top-free-apps/36.

Apple, Inc, 2024b. Top Free Iphone Games on the App Store. https://apps.apple.com/us/charts/iphone/top-free-games/6014.

Apple, Inc. (n.d.a), 'Algorithm.lzfse'. URL: https://developer.apple.com/documentation/compression/algorithm/lzfse.

Apple, Inc. (n.d.b), 'Disabling and enabling system integrity protection'. URL: https://developer.apple.com/documentation/security/disabling_and_enabling_system_integrity_protection.

Apple, Inc. (n.d.c), 'Running your ios apps in macos'. URL: https://developer.apple.com/documentation/apple-silicon/running-your-ios-apps-in-macos.

Apple, Inc. (n.d.d), 'Security of runtime process in ios and ipados'. URL: https://support.apple.com/guide/security/security-of-runtime-process-sec15bfe098e/web.

Apple, Inc. (n.d. e), 'Vzmacplatformconfiguration'. URL: https://developer.apple.com/documentation/virtualization/vzmacplatformconfiguration.

Apple Security Engineering and Architecture (SEAR) (2023). URL: https://security.apple.com/blog/security-research-device-program-2024/.

Bader, M., Baggili, I., 2010. Iphone 3gs Forensics: Logical Analysis Using Apple Itunes Backup Utility.

Baggili, I., Awawdeh, S.A., Moore, J., 2014. Life (Logical Iosforensics Examiner): an Open Source Iosbackup Forensics Examination Tool.

Bays, J., Karabiyik, U., 2019. Forensic analysis of third party location applications in android and ios. In: IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS). IEEE, pp. 1–6.

Beijnum, A., 2023. Haly: Automated Evaluation of Hardening Techniques in Android and Ios Apps. University of Twente. Master's thesis.

Bergadano, F., Boetti, M., Cogno, F., Costamagna, V., Leone, M., Evangelisti, M., 2020. A modular framework for mobile security analysis. Inf. Secur. J. A Glob. Perspect. 29 (5), 220–243.

Bhatt, A.J., Gupta, C., Mittal, S., 2018. Network forensics analysis of ios social networking and messaging apps. In: 2018 Eleventh International Conference on Contemporary Computing (IC3). IEEE, pp. 1–6.

Brewster, T., 2018. This super stealth startup has built an apple hacker's paradise. Forbes. https://www.forbes.com/sites/thomasbrewster/2018/02/15/corellium-virtual-apple-iphones-for-hacking/?sh=16e370ac4a3b.

Brewster, T., 2021. Corellium Scores $25 Million Funding to Build Android and Iphone Cyber Research Juggernaut. https://www.forbes.com/sites/thomasbrewster/2021/12/16/corellium-announces-25-million-investment-from-cisco-paladin-to-build-iphone-and-android-research-heaven/.

Broadcom, 2022. System Requirements for Vmware Fusion 13.X (90114). https://kb.vmware.com/s/article/90114.

Casey, E., 2009. Handbook of Digital Forensics and Investigation. Academic Press.

Chamberlain, A., Azhar, M., 2019. Comparisons of forensic tools to recover ephemeral data from ios apps used for cyberbullying. Proc. IARIA 1–6.

Chang, Y.-T., Teng, K.-C., Tso, Y.-C., Wang, S.-J., 2015. Jailbroken iphone forensics for the investigations and controversy to digital evidence. J. Comput. 26 (2), 19–33.

Corellium (n.d.), 'Mobile malware & threat research'. URL: https://www.corellium.com/solutions/malware-research.

Corellium, Inc. (n.d.), 'Testing third-party ios apps'. URL: https://support.corellium.com/features/apps/testing-ios-apps.

de Vos, M., 2022. Emulating an Ipod Touch 1g and Iphoneos 1.0 Using Qemu (Part I). https://devos50.github.io/blog/2022/ipod-touch-qemu/.

Espósito, F., 2020. Ios Apps Will Run on Apple Silicon Macs, but Major Developers Have Already Opted Out of the Mac App Store. 9to5Mac. https://9to5mac.com/2020/11/09/ios-apps-will-run-on-apple-silicon-macs-but-major-developers-have-already-opted-out-of-the-mac-app-store/.

Garfinkel, S., Nelson, A.J., Young, J., 2012. A general strategy for differential forensic analysis. Digit. Invest. 9, S50–S59. The Proceedings of the Twelfth Annual DFRWS Conference. https://www.sciencedirect.com/science/article/pii/S174228761200028X.

Goodwin, C., Woolley, S., 2022. Sideloading: an exploration of drivers and motivations. In: 35th International BCS Human-Computer Interaction Conference 35. ', pp. 1–6.

Grajeda, C., Berrios, J., Benzo, S., Ogunwobi, E., Baggili, I., 2023. Expanding digital forensics education with artifact curation and scalable, accessible exercises via the artifact genome project. Forensic Sci. Int.: Digit. Invest. 45, 301566.

Grajeda, C., Sanchez, L., Baggili, I., Clark, D., Breitinger, F., 2018. Experience constructing the artifact genome project (agp): managing the domain's knowledge one artifact at a time. Digit. Invest. 26, S47–S58.

Hoog, A., 2011. Android Forensics: Investigation, Analysis and Mobile Security for Google Android. Elsevier.

Husain, M.I., Baggili, I., Sridhar, R., 2011. A simple cost-effective framework for iphone forensic analysis. In: Digital Forensics and Cyber Crime: Second International ICST Conference, ICDF2C 2010, Abu Dhabi, United Arab Emirates, October 4-6, 2010, Revised Selected Papers 2'. Springer, pp. 27–37.

Ke, M., 2017. Unzip with Lzfse Support (Archived). https://web.archive.org/web/20210729084720/. https://sskaje.me/2017/08/unzip-with-lzfse-support/.

Khoa, N.ê.A., 2021. Fairplay. https://blog.efiens.com/post/luibo/osx/fairplay/.

Ligh, M.H., Case, A., Levy, J., Walters, A., 2014. The Art of Memory Forensics: Detecting Malware and Threats in Windows, Linux, and Mac Memory. John Wiley & Sons.

Lindorfer, M., Neugschwandtner, M., Platzer, C., 2015. Marvin: efficient and comprehensive mobile app classification through static and dynamic analysis. In: 2015 IEEE 39th Annual Computer Software and Applications Conference, vol. 2. IEEE, pp. 422–433.

Manna, M., Case, A., Ali-Gombe, A., Richard III, G.G., 2021. Modern macos userland runtime analysis. Forensic Sci. Int.: Digit. Invest. 38, 301221.

Nguyen, T., Kim, K., Bianchi, A., Tian, D.J., 2022. Truemu: an Extensible, Open-Source, Whole-System Ios Emulator. Blackhat USA'22.

No Yume, H., 2024. Touchhle - more information. https://touchhle.org/.

Ravnås, O. A. V. (n.d.), 'ios — frida'. URL: https://frida.re/docs/ios/.

Ricco, A., 2023. The importance of jailbreaks for ios security work. United States Cybersecutiy Magazine. https://www.uscybersecurity.net/ios-security-research-jailbreaks/.

Shijo, P., Salim, A., 2015. Integrated static and dynamic analysis for malware detection. Procedia Comput. Sci. 46, 804–811.

Sideloadly (n.d.). URL: https://sideloadly.io/.

Subedi, K.P., Budhathoki, D.R., Dasgupta, D., 2018. Forensic analysis of ransomware families using static and dynamic analysis. In: 2018 IEEE Security and Privacy Workshops (SPW). ', IEEE, pp. 180–185.

The Apple Wiki, 2010. Fairplay. https://theapplewiki.com/wiki/FairPlay.

The Apple Wiki, 2023a. Scam Jailbreaks and Unlocks. https://theapplewiki.com/wiki/Scam_Jailbreaks_and_Unlocks.

The Apple Wiki, 2023b. 'T8015'. https://theapplewiki.com/wiki/A11.

The Apple Wiki, 2024a. Jailbreak. https://theapplewiki.com/wiki/Jailbreak.

The Apple Wiki, 2024b. palera1n. https://theapplewiki.com/wiki/Palera1n.

Zdziarski, J., 2008. iPhone Forensics: Recovering Evidence, Personal Data, and Corporate Assets. O'Reilly Media, Inc.