R+R: State of the Application Sandboxing on macOS: A Differentiated Replication

Iryna Pastukhova
iryna.p@macpaw.com
MacPaw
Kyiv, Ukraine

Ivan Synytsia
sip@macpaw.com
MacPaw
Kyiv, Ukraine

Nataliia Stulova
nata.stulova@macpaw.com
MacPaw
Kyiv, Ukraine

ABSTRACT

Background. Application sandboxing is a common practice in mobile and desktop operating systems, employed to prevent unauthorised resource access and protect users from malicious apps. In the context of macOS, the second most widespread desktop operating system, little has been studied on the topic of application security outside of malware research. *Blochberger et al.* [3] performed an exploratory study of two application marketplaces for macOS in 2019, comparing sandboxing adoption in apps from the official app marketplace and a third-party one.

Aim. While their study provides very useful insights both for researchers and practitioners, by design, it excludes open-source apps, a limitation we aim to address in our current work.

Methods. We conduct a differentiated replication study, extending the application dataset to include open-source apps. Given the time difference between the original study and ours, we also check if the original results hold in the newer macOS version.

Results. We confirm the original observations on sandbox adoption practices, noticing a significant improvement in sandboxing mechanism adoption in the closed-source apps from a third-party store. We also demonstrate the similarity of the results between sandbox adoption in closed- and open-source applications.

Conclusions. Our work confirms the applicability of the original sandboxing analyses in different contexts, such as dataset or test OS environment changes. We also demonstrate that open-source applications could be included in datasets used for the Apple ecosystem analysis.

CCS CONCEPTS

• **Software and its engineering** → **Software creation and management**; • **Security and privacy** → **Software security engineering**.

KEYWORDS

differential replication, sandbox, macOS, open source, privilege separation

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

ACM Reference Format:
Iryna Pastukhova, Ivan Synytsia, and Nataliia Stulova. 2024. R+R: State of the Application Sandboxing on macOS: A Differentiated Replication. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Application sandboxing is one of the universally established mitigation techniques to prevent unauthorized resource access by applications, available in all major mobile and desktop operating systems, including Windows[1], macOS[2], Linux[3], iOS[4], and Android[5]. While the Android sandboxing mechanism historically receives the most attention from the research community, and iOS is also studied often, given the prevalence of mobile devices, studies on desktop application sandboxing mechanisms are rather scarce. In the case of macOS, the second most used desktop operating system according to Statcounter[6] and Statista[7] data, only a few studies[3, 13] have researched the app sandboxing state in the wild.

The most complete study on application sandboxing adoption on macOS to date is *Blochberger et al.* [3] from 2019, which also analyzes the larges dataset of macOS applications. However, the study design in this work omits open-source apps, that are often studied in the context of general malware and vulnerability research. The absence of such apps hinders the generalization of the reference study results for the open-source apps, and *vice versa* — it cannot be said if the results obtained on open-source apps will generalize to closed-source software, which is dominant in the Apple ecosystem.

In our differentiated replication study of [3], we focus on macOS application sandboxing adoption 5 years later, expanding the dataset to include more third-party app installation sources, both open source in nature. This allows us both to see the dynamics compared to the original study, and check the generalizability of the previous research results to the open-source apps and vice versa.

We confirm results of the reference work in what concerns the application sandbox adoption in the official and third-party application stores, noticing a significant improvement in the latter one — over 15% more apps are now properly sandboxed in accordance

[1] https://learn.microsoft.com/en-us/windows/security/application-security/application-isolation/windows-sandbox/windows-sandbox-overview
[2] https://developer.apple.com/documentation/xcode/configuring-the-macos-app-sandbox
[3] e.g., Ubuntu https://ubuntu.com/core/docs/security-and-sandboxing
[4] https://support.apple.com/guide/security/security-of-runtime-process-sec15bfe098e/web
[5] https://developers.google.com/privacy-sandbox/overview/android
[6] https://gs.statcounter.com/os-market-share/desktop/worldwide/#monthly-200901-202404
[7] https://www.statista.com/statistics/268237/global-market-share-held-by-operating-systems-since-2009/

with the official Apple guidelines. We demonstrate that their results are observable even on a different application dataset and, in some aspects, carry over to the open-source apps distributed non-commercially.

2 RELATED WORK

2.1 macOS Security

Studies considering security issues with macOS apps focus on malware analysis, relying on a mix of binaries of open and closed-source apps. In [10], authors collect 1000 macOS app binaries from OS X Mavericks and open-source software (OSS). While the proportions are not specified, we can assume that the OSS apps are a minority in that dataset, as the `/usr/bin/` directory on macOS Sonoma 14.1.1 contains 980 binaries. In [12], the authors collect 460 app binaries from the Mac App Store (MAS) exclusively. In [7], authors work with 461 app binaries from MAS and the Homebrew package manager is an almost equal proportion. In [6], authors use 853 application binaries retrieved in an unspecified proportion from the `/usr/bin/` system directory, MAS, and the Softonic[8] platform. To the best of our knowledge, none of the previous studies investigates whether there are significant differences in application behavior and resource use between OSS and closed-source apps. Moreover, dataset sizes are rather small and might not be representative of the big picture of any of the platforms/stores.

The body of work on vulnerability research considers larger app datasets, and in contrast with malware studies, authors collect not just binaries but the whole *application bundles*[9]. In [13], authors collect 1612 apps from MAS exclusively, and in [3], which we take as reference, authors collect 13038 apps from MAS and the MacUpdate platform. In it, OSS is not even considered, being a clear research gap, contrasting with the malware research situation.

2.2 Application Sandboxing

Majority of the academic studies explore sandboxing in depth on the mobile operating systems (Android, iOS) in the first place, with Linux recently starting also to get attention. We already considered studies on macOS in the previous subsection, and for Windows sandboxing seems to be applied more often on different abstraction levels[10].

2.2.1 iOS. In [8], authors present the first systematic analysis of the iOS container sandbox profiles; their SandScout framework automatically extracts and decompiles binary sandbox profiles. In [5], authors present the design and implementation of a novel and efficient iOS app hardening service, XiOS, that enables fine-grained application sandboxing.

2.2.2 Android. In [4], authors propose an Android Application Sandbox (AASandbox), which is able to perform both static and dynamic analysis on Android programs to automatically detect suspicious applications.

In [2], the authors describe the approach that is based on application virtualization and process-based privilege separation to securely encapsulate untrusted apps in an isolated environment. In [14] authors propose AppCage, a system that thoroughly confines the run-time behavior of third-party Android apps without requiring framework modifications or root privilege. AppCage leverages two complimentary user-level sandboxes to interpose and regulate an app's access to sensitive APIs.

In [1], authors perform an extensive analysis of the native code usage in 1.2 million Android apps. They first used static analysis to identify a set of 446k apps potentially using native code and then analyzed this set using dynamic analysis.

2.2.3 Linux. In [9], authors provide the first analysis of sandbox policies defined for Flatpak and Snap applications, covering 283 applications contained in both platforms.

The [11] examines current mobile sandboxing techniques and specifies the requirements to propose a trustworthy mobile sandbox methodology that deals with the lack of real user behavior and overcomes the risk of sandbox evasion.

A unifying trait in the above-referenced works is the combined use of static and dynamic analyses for sandbox operation, which is in line with both the methodology of our reference work and our own replication study.

3 REFERENCE WORK

Our reference work is the paper by *Blochberger et al.* [3], in which they evaluate general sandbox mechanism adoption in macOS applications. First, the authors analyze the sandbox mechanism of macOS and identify a critical sandbox-bypass. Additionally, the authors evaluate the general adoption of the sandbox mechanism, as well as app-specific sandbox configurations.

The following subsections provide an overview of the methodological details of reference work and discuss the limitations we identified.

3.1 Context

The context of the reference work consists of the 13038 applications retrieved from two primary sources:

- Mac App Store (MAS), the official macOS application distribution platform (8366 applications downloaded)
- MacUpdate (MU), a third-party application distribution platform (4672 applications downloaded)

In particular, authors have selected applications compatible with and runnable on macOS Mojave (10.14), the fifteenth major OS version, which was released to the public on September 24, 2018.

3.2 Design

Intending to study the sandbox adoption across the downloaded app dataset, *Blochberger et al.* [3] first investigated the general adoption of sandboxing and contrasted the findings between different versions of an app as well as between the two sources MAS and MU. Second, they evaluated app-specific sandbox configurations. Third, they investigated privilege separation use. Finally, we describe a sandbox bypass vulnerability, which has since been fixed.

[8] https://en.softonic.com/

[9] https://developer.apple.com/library/archive/documentation/CoreFoundation/Conceptual/CFBundles/BundleTypes/BundleTypes.html#//apple_ref/doc/uid/10000123i-CH101-SW13

[10] https://learn.microsoft.com/en-us/windows/security/application-security/application-isolation/windows-sandbox/windows-sandbox-overview

3.3 Data collection and analysis

MAS crawling was performed between November 2017 and September 2018. In that period, they scanned the MAS daily and downloaded all free apps and subsequent updates thereof. Due to region-locking, only apps available to the German market could be downloaded. During the MU crawl, they detected 37 238 apps and attempted to download the latest available version per app. Most of them had to be filtered out, due to non-downloadable or invalid URLs, corrupted or otherwise unreadable files, or because the apps were outdated and not supported by modern versions of macOS This resulted in 6164 downloaded apps, out of which 1492 were not compatible with their test system and could not be launched, further lowering the sample size to 4672 compatible apps as of September 2018.

They extracted sandbox configurations and additional metadata for each obtained app. In a first step, they statically analyzed the apps and extracted (i) entitlements from the binaries of the app, and (ii) app categories from the information property list (Info.plist) of the app. The same extraction has been performed for each XPC helper that is part of the app. In a second step, they conducted a dynamic feature extraction to verify if sandboxing is indeed active in the running app. They launched each app, waited for about 30 seconds to allow initialization routines to finish, and then asked the operating system whether the app is sandboxed.

For general sandbox adoption, they considered an app as sandboxed if the sandbox entitlement is included in its binary and enabled, and dynamic analysis showed that sandboxing is indeed active during runtime. They then reported statistics for MAS and MU applications separately, also taking into account apps present in both subsets.

For the evaluation of particular entitlements, they only included entitlements that can be easily configured by developers in the App Sandbox section of the app's capabilities configuration in Xcode. They then compared statistics for MAS and MU applications between themselves.

For privilege separation analysis, they investigated whether XPC helpers have more, less, or the same privileges as the main app.

3.4 Limitations

Our main concern is with the dataset structure of *Blochberger et al.* [3]. Their dataset by design only includes closed-source apps, whereas as we show in the section 2.1, it is common to include OSS applications into application datasets. Our suggestion for the differentiated replication is to extend the dataset, including the OSS software, and see how it compares with the reference study.

We were also alarmed by the high rate of apps discarded from the MU dataset. As a part of a preliminary investigation into the OSS apps for macOS, we collected a small dataset of apps from GitHub and Homebrew Cask of 4663 apps and compared them with the MU apps in terms of the date of the last app update.

We observe that while at least 1/3 of the apps are maintained up-to-date across all datasets, almost 1/5 of the apps in two of our datasets might represent abandonware, especially prominent this issue becomes for the MU apps (Table 1). In the context of application security, where vulnerabilities tend to be patched in the newer app versions, including a dataset of apps polluted with abandonware might significantly bias analyses.

Table 1: Bundle last update date

Dataset	30 days	6 months	1 year	3+ years	N/A
GitHub	**59.55%**	10.98%	5.89%	21.14%	2.44%
Brew Cask	**41.08%**	14.94%	6.39%	18.49%	19.1%
MacUpdate	25.79%	5.70%	3.35%	22.67%	**42.49%**

4 DIFFERENTIATED REPLICATION

4.1 Data Collection and Processing

4.1.1 Application Crawling. We extend the original dataset of [3] that contains Mac App Store (MAS) and MacUpdate (MU) applications to include Open Source Software (OSS) applications. Fig 1 presents the initial distribution by the store type. It is worth mentioning that in the sandbox state extraction phase, the number of applications significantly reduced (see 4.1.5 for more details).

Similar to the original paper, we do not remove duplicated applications across the sources for the initial analysis, as they do not necessarily have the same sandbox status and are different apps in the sense of our experiment.

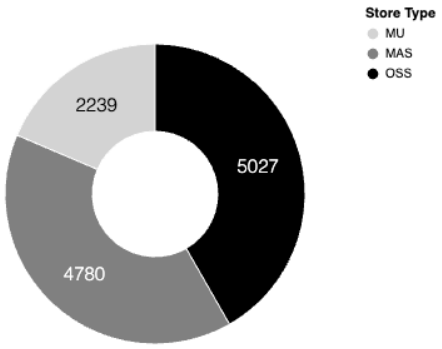

Figure 1: Dataset shares, per provenance source.

4.1.2 Mac App Store (MAS).

Getting application list. The MAS is the main store on the macOS platform and hosts the largest collection of available apps for this operating system. One way to get an extensive list of apps from this source is to enroll into the Apple's Enterprise Partner Feed[11], and the other is to use the iTunes Search API[12] to fetch app details.

The Enterprise Partner Feed provides periodic metadata updates of the content hosted on various Apple platforms, including apps, books, music etc. While collecting MAS application metadata only through this mechanism would have been straightforward, in practice, there are some notable limitations

[11] https://performance-partners.apple.com/epf
[12] https://developer.apple.com/library/archive/documentation/AudioVideo/Conceptual/iTuneSearchAPI/Searching.html#//apple_ref/doc/uid/TP40017632-CH5-SW1

- it is not open-access, which makes it hard to use for app research purposes, as previously reported by the *Blochberger et al.* [3],
- the feed updates can experience delays,
- the metadata does not include some more recent content characteristics (e.g., various ratings), and,
- most critically, this metadata does not include application `bundleID` identifier, which is necessary for application download from the MAS.

Nevertheless, given access to this feed, we used it as a first source of collecting the metadata of applications of interest (e.g., excluding games), and used it to refine our subsequent queries to the iTunes Search API to collect the complement application metadata and then get the applications themselves.

Collecting application metadata. Similarly to the reference study, we queried[13] iTunes search API to obtain full application metadata. As a URL for the API, we used the following query:

```
https://itunes.apple.com/search?term={term}
&country={country}&entity={entity}&genreId=genre
&limit={limit}&offset={offset}}
```

- For the parameter `term`, a list of the most popular words (around 180 items) was selected to maximize the number of app types and skip games.
- We had to specify a single `country` due to geographic restrictions of MAS, it is only possible to download apps available for a specific region
- To allow data to be collected in batches, the undocumented parameter in the API `offset` was used, and later, all these batches were merged into a single list. This allowed us to avoid the limitation of at most 200 apps returned for a query by the iTunes Search API, faced by the *Blochberger et al.* [3].

Downloading applications. We wrote a custom Python script to crawl the MAS and download the applications with the help of the mas-cli[14] tool, also used by the *Blochberger et al.* [3]. The tool because of known issues[15] was launched on macOS 10.14.6, which limited the number of supported apps, as not all of them have backward compatibility with this OS version (we discuss this issue in more detail in Section 6). We have scheduled our downloads to be performed at a rate of 1 app metadata download per second; this way, we did not encounter the limitation of the iTunes Search API that blocks too many requests in a short time.

The collection process was performed with a few attempts between December 2023 and January 2024. In total, 5310 free applications were downloaded and installed. The crawling was performed as much as possible for the specific region, which was set in an Apple ID on the system and launched for the *BLINDED* region.

4.1.3 MacUpdate (MU). As in the original paper, we included the MacUpdate applications in our analysis. To collect the available applications, the official site[16] was scraped for the relevant metadata. While initially collecting over 7k applications, we experienced the same issues as the *Blochberger et al.* [3], such as many apps not launching and not being compatible with our test setup. After filtering our results, a total of 2239 applications were selected for subsequent analysis.

4.1.4 Open Source Software (OSS). We define the programs crawled from GitHub and Brew Cask as the OSS dataset in our work. Of the 5027 OSS apps, 1162 are GitHub, and 3865 are Brew Cask, respectively.

The Brew Cask[17] complements the popular package manager for macOS Homebrew[18] with the set of GUI applications. Collecting the Brew Cask applications is as simple as going through the list of all the available provided in a JSON file[19] and downloading them.

GitHub part of the dataset consists of two sources: Awesome macOS open-source applications[20] and apps collected by the direct querying the GitHub API[21]. The queries for the latter were formed based on the search string `macos` in the repository name, description, or readme and additionally filtered by the repository's creation date, last commit date, and number of stars. The latest creation date, 01-01-2019, was determined by the last update of the data reported in [3]; we consider the repository active if the last commit date was no later than 2023. The number of stars, 153, was chosen based on the first quartile statistic of the Awesome macOS open-source applications, which, being a manually collected dataset, can be, in some sense, treated as a good representative of the indeed used and installed apps. The query script was run with the help of the `PyGitHub` library[22]; the repositories URLs were additionally filtered to minimize potential intersections with the Awesome macOS open-source. The search results were additionally checked and filtered accordingly for release existence, and downloaded archives were scanned for the `.app` extension file's presence. Several apps (primarily miners) were removed due to security issues: they were tracked by our checks as malicious.

In such a way, the final GitHub dataset contains 1162 apps: 520 Awesome and 642 queried apps, respectively.

4.1.5 Feature Extraction. In general, for the feature extraction, we followed a strategy similar to the original paper but performed both static and dynamic extraction in a single step and extended the features with a part of the application bundle info. In particular, the following data was extracted:

- App binaries entitlements
- App bundle's information from the `Info.plist` file: app category, disk size, minimum OS version, and `plist` modification date
- Dynamic sandbox state, extracted with the provided in [3] code after launching the app

The same data, except the dynamic sandbox state, was extracted for each XPC service.

Due to our previous belief that modern applications had reduced the launch time, the initial intention was to skip waiting 30 seconds to finish the initialization routines after launching, as mentioned in [3]. However, the experiments revealed the necessity of this step:

[13] https://developer.apple.com/library/archive/documentation/AudioVideo/Conceptual/iTuneSearchAPI/Searching.html#//apple_ref/doc/uid/TP40017632-CH5-SW1
[14] https://github.com/mas-cli/mas
[15] https://github.com/mas-cli/mas?tab=readme-ov-file#%EF%B8%8F-known-issues
[16] https://www.macupdate.com/
[17] https://formulae.brew.sh/cask/
[18] https://brew.sh/
[19] https://formulae.brew.sh/api/cask.json
[20] https://github.com/serhii-londar/open-source-mac-os-apps/tree/master
[21] https://docs.github.com/en/rest?apiVersion=2022-11-28
[22] https://pygithub.readthedocs.io/en/stable/index.html

even after the process is launched and assigned with a PID, the sandbox state can be determined with some delay; similar behavior can be observed while manually launching the app and checking the sandbox state in the Activity Monitor app. As a result, each app waited for up to 30 seconds to be launched and an additional 10 sec to get the sandbox state – and this appeared to be enough.

Of the all considered apps, more than 28% failed to launch:

- more than 12% – due to the unsuitable format of the executable files or their absence after the source extraction,
- nearly 11% – due to damaged or incompatible .app bundle files, and
- the rest – due to the absence of the macOS folder in the application bundle, troubles with mounting or licenses.

4.2 Evaluation of Sandbox Adoption

In this section, we report the sandbox adoption state of each of the considered apps' sources separately, as well as the statistics for the applications present in all the sources. Additionally, the current adoption state is compared to the one reported in the original paper.

We stick to the definition introduced in the original paper and treat the application sandboxed if it is both *statically* and *dynamically* sandboxed: it has enabled entitlement and has an active sandboxed state in the running state.

4.2.1 Sandbox Adoption in General. Of the total 8908 launched apps, for 6635 (almost 75%) we managed to extract both the dynamic and static sandboxed states; the remaining 25% had at least one of the states equal to null due to different extraction failures, which can be a matter of future studies.

Fig 2 reports the sandbox adoption state for each of the sources.

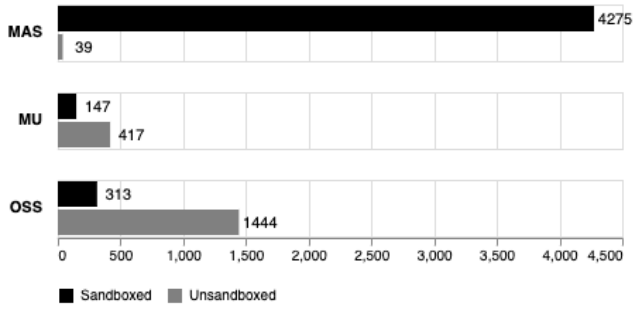

Figure 2: Sandbox state by the store type.

As expected, MAS is the most 'sandboxed' application provider: of the 4314 analyzed apps, 4275 (99 %) are sandboxed. The next place takes MU: out of 564 analyzed apps, 147 (26 %) are sandboxed. OSS appeared to be the most 'unsandboxed' store: only 313 (nearly 18 %) out of the 1757 apps are sandboxed.

4.2.2 Common Applications. The common applications were identified by their bundle identifiers. Only 9 apps with sandbox states were present in all three stores: 4 apps were sandboxed in all versions, 1 app was unsandboxed in all versions, and for the remaining 4 apps, only the MAS versions were sandboxed.

Due to the small number of apps in the intersection of all stores, we also analyzed them pairwise. OSS and MAS share 50 mutual apps: 26 are sandboxed, 1 is unsandboxed in both versions, while 23 are sandboxed only in the MAS version. Out of 27 apps present in both MU and MAS, 12 are sandboxed and one is unsandboxed in both versions, while 14 are sandboxed only in the MAS version. Among the 170 mutual apps between OSS and MU, 28 are sandboxed and 139 are unsandboxed in both providers, 2 are sandboxed only in OSS, and 1 is sandboxed only in the MU version.

We do not undertake to compare these results with the original ones due to the significantly smaller sample of data.

4.2.3 Bypassing the Sandbox. Let us first recall that while only MAS and MU apps were analyzed in the original paper, in the current work OSS apps are added to the consideration. Table 2 compares the currently obtained and those of the 2019 sandbox adoption states. In particular, for both MAS and MU, a significant improvement of more than 5% and 15%, respectively, is observed. While in the case of MAS, it is quite an expected behavior due to the twice longer time period passing since sandboxing became mandatory for the Mac App Store distribution, the MU case allows us to assume that the sandboxing obligation has a positive impact on other application stores too. Although we can not track the dynamic in OSS sandbox adoption state changes, the currently observed of almost 18% seems quite comparative.

Table 2: Sandbox adoption state comparison

	MAS	MU	OSS
2019	93.53 %	10.94 %	*N/A*
2024	99.10 %	26.06 %	17.81 %

In the original work, the issue of bypassing the sandbox was reported. Specifically, six MAS and eleven MU applications had their sandbox static state enabled. However, upon dynamic extraction, it was found that these applications did not maintain their sandbox constraints. This discrepancy highlighted a critical security vulnerability. The issue was immediately reported to Apple, and with the release of macOS 10.13.5, such sandbox evasions should not occur in subsequent versions [3].

In our analysis, we concentrated on MAS apps more thoroughly. Our findings revealed that the applications in our sample maintained their sandbox integrity as expected. This suggests that the measures implemented in macOS 10.13.5 and later versions have effectively mitigated the vulnerability.

4.2.4 Entitlements. In this section, we analyze the entitlements. In order to compare and observe the dynamic of change, similar to the original paper, only the sandboxed apps and the set of easily configured via Xcode entitlements are taken into account. However, we have not analyzed the changes in the entitlements number between the MAS apps' versions. We also skip the investigation of the interplay between entitlements and MAS apps' popularity, as no such correlation was observed.

Distribution. We consider only entitlements present in at least 5% of all applications. Table 3 presents the distribution of the considered entitlements and compares them to those reported in 2019.

Table 3: Entitlements used by more than 5% of all apps

	MAS		OSS		MU		All	
Entitlement	**2024**	**2019**	**2024**	**2019**	**2024**	**2019**	**2024**	**2019**
Client	65.92 %	65.21 %	69.97 %	-	78.91 %	76.13 %	66.59%	65.88 %
User Selected Files (Read-write)	52.75 %	43.07 %	56.55 %	-	80.27 %	81.41 %	53.85 %	45.42 %
User Selected Files (Read-only)	16.65 %	7.86 %	20.13 %	-	8.84 %	3.72 %	16.64 %	7.60 %
Security-scoped Bookmarks	14.22 %	9.24 %	20.04 %	-	37.41 %	31.12 %	15.46 %	10.58 %
Server	14.34 %	12.87 %	20.13 %	-	13.61 %	14.68 %	14.70 %	12.98 %
Print	13.54 %	12.79 %	15.65 %	-	29.25 %	32.09 %	14.17 %	13.97 %
App Groups	11.53 %	7.97 %	23.0 %	-	20.41 %	10.18 %	12.57 %	8.11 %
Notifications	6.78 %	-	7.67 %	-	8.16 %	-	6.89 %	-
Downloads (Read-write)	5.87 %	5.37 %	12.78 %	-	8.84 %	7.44 %	6.42 %	5.49 %
iCloud	5.33 %	-	11.82 %	-	15.65 %	-	6.08 %	-
Camera	5.52 %	-	4.79 %	-	6.80 %	-	5.51 %	-

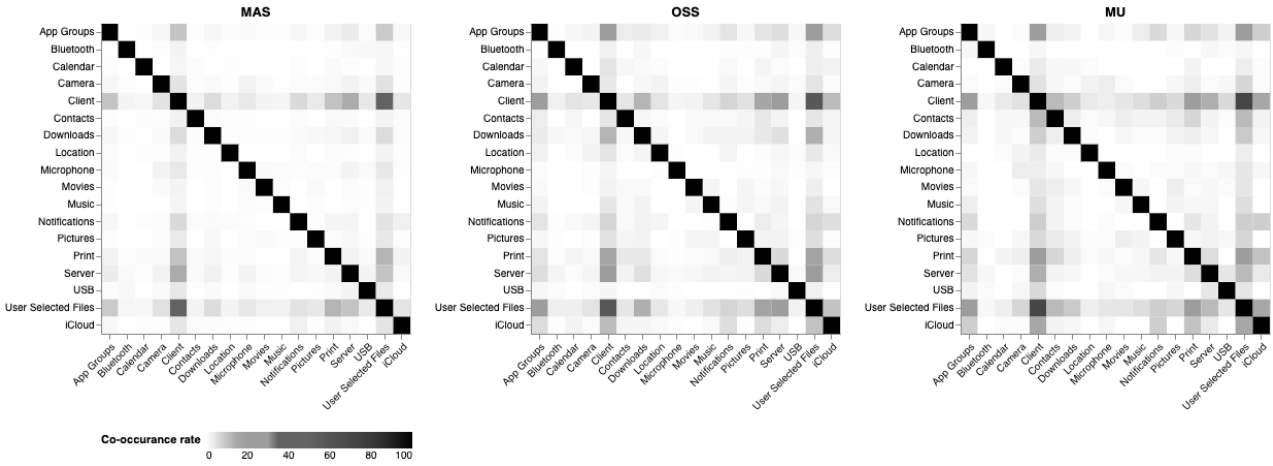

Figure 3: Sandbox state by the store type.

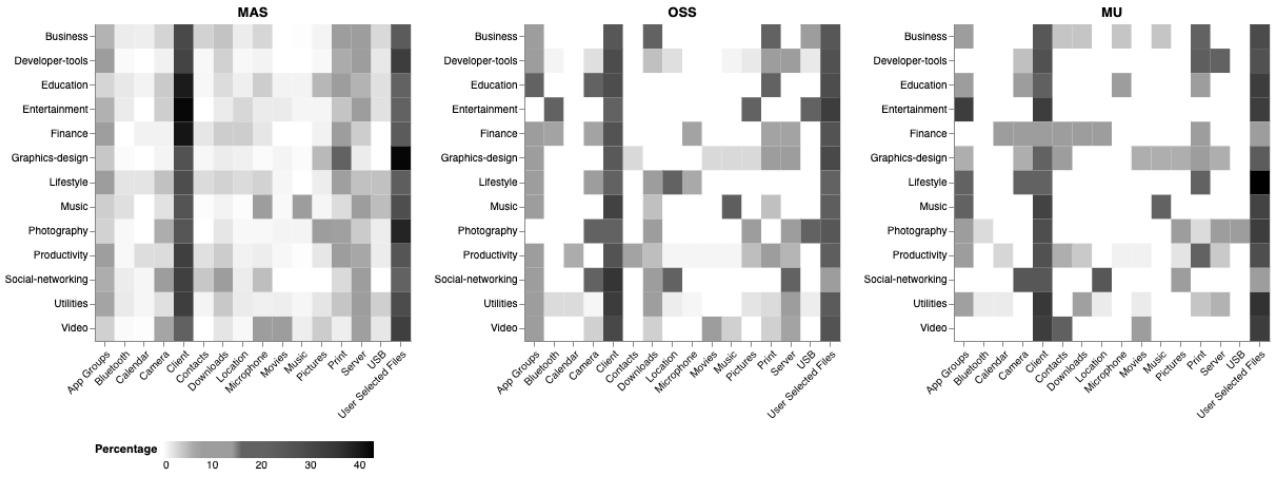

Figure 4: Percentage of entitlements by category for each store type.

The top two leaders stay the same: *Client* and *User Selected Files (Read-write)* ahead by a wide margin; *User Selected Files (Read-only)* and *Security-scoped Bookmarks* displaced *Print* and *Server*. Contrary to the original results, in our sample, *iCloud*, *Notifications*, and *Camera* managed to overcome the 5 % limit, but the *Microphone* did not. These changes likely represent changes in user requests for features.

Co-occurrences. To maintain consistency with the original study, the co-occurrences of entitlements were analyzed. Figure 3 shows the corresponding results in percentages: overall, all store types display similar trends.

Categories. The interplay between entitlements and app categories is analyzed based on entitlements present in more than 5% of apps and categories present in at least 1% of all apps. The *Games* category was excluded due to its presence only in MAS apps. As shown in Figure 4, the stores exhibit somewhat different trends: the distribution for MAS is the most uniform, while OSS and MU display more contrasting trends.

Temporary Exception and Private Entitlements. All the sandboxed apps were also analyzed for the *Temporary Exception* and *Private* entitlements presence.

For the former, we obtained pretty close results to the original ones: *Apple Event* remains the most popular with a small change from 3.2 % in 2019 to 3.8% now. *Global Mach Service* temporary exception, enabling lookup of one or more global Mach services[23], is the next most used (2.3%).

As the latter is not officially documented, we tried to identify them by scanning the entitlements list for the ones starting with *com.apple.private*, and none were found even, regardless of the sandbox state.

4.2.5 Privilege Separation. Of all the sandboxed apps, 420 have helpers with non-empty entitlement sets. We consider the entitlement sets as they are, without any filtering. Each service with a non-empty set of entitlements was classified as *mixed* if its entitlements differed from the app's, and *same* if they were similar to the app's and matched in values.

Of the 420 apps considered, 31 have helpers with the same privileges (2 in MAS, 24 in OSS, and 5 in MU). For 79 apps (32 in MAS, 32 in OSS, and 15 in MU), each helper has *mixed* privileges, meaning they differ from the app's entitlements.

Finally, 92 apps (1 in MAS, 62 in OSS, and 29 in MU) have entitlements, but none of their helpers do; the opposite case was not observed. These numbers indicate that developers in MAS apps most consistently follow the privilege separation strategy.

5 SECURITY AND PRIVACY IMPROVEMENTS IN MACOS SONOMA

Given the difference in time between our study and the work of *Blochberger et al.* [3], we discuss here the difference in operating systems, Apple released the current macOS Sonoma (14) on September 26, 2023. Compared to macOS Mojave (10.14), used in the reference work [3], the company has introduced and enforced several major security improvements. We highlight below major changes in the Apple security architecture, noting which could directly affect the reproducibility of the reference work.

Dropping support for 32-bit applications. Apple has definitely removed support[24] for 32-bit applications starting from macOS Catalina (10.15), which was released on October 7, 2019. From that OS version on, only 64-bit applications, which are generally more efficient, can run on macOS from version 14. While older 32-bit apps can still be listed on and downloaded from the MAS, users report[25] that such apps would not launch.

In the context of the replication of the reference work, it means the following:

- as it is not possible to obtain from the MAS accurate information on whether the app is 32-bit or 64-bit, any automated app download process might get a share of the application that would not launch and could not be subject to any runtime analysis (such as sandboxing status check, network traffic monitoring, etc.), thus making dataset comparison with the reference work inaccurate;
- researchers and practitioners will have to account for this share of non-launchable apps when designing any studies involving running applications;
- depending on the version of the macOS, that will be used for running the experiments, different subsets of the original app superset might launch and introduce bias

Switching from Kernel to System Extensions. Kernel extensions in macOS are loadable kernel modules in the format of object files, with which developers can add custom functionality to the OS kernel, and later rely on that functionality when writing apps. Given that such extensions operate in high privilege mode with direct low-level access to hardware and system resources, since macOS Catalina (10.15) Apple is requiring third party developers to switch to System Extensions frameworks[26], where each framework only has access to its respective resource (e.g., Network Extension will not have access to the file system).

In the context of the replication of the reference work, it means that new apps or updated app versions running on macOS versions 10.15+ could require more entitlements than before.

Enhancing the Gatekeeper. For an application to run on macOS, it has to pass a notarization[27] process, which includes scanning for malware signs and code signing checks. Code signing is a macOS security technology used to certify that an app was created by an application developer and help the system detect any change to the app — whether the change is introduced accidentally or by malicious code (e.g., asking additional entitlements). Gatekeeper[28] is a security mechanism on macOS, that checks application notarization status. Already macOS Catalina (10.15) required all software

[23] https://developer.apple.com/library/archive/documentation/Miscellaneous/Reference/EntitlementKeyReference/Chapters/AppSandboxTemporaryExceptionEntitlements.html

[24] https://support.apple.com/en-us/103076

[25] https://discussions.apple.com/thread/251009413?sortBy=best

[26] https://support.apple.com/guide/deployment/system-and-kernel-extensions-in-macos-depa5fb8376f/web

[27] https://developer.apple.com/documentation/security/notarizing_macos_software_before_distribution

[28] https://support.apple.com/guide/security/gatekeeper-and-runtime-protection-sec5599b66df/web

for macOS to be notarized, but since macOS Ventura (13), the Gatekeeper will check the notarization status every time the app is launched, not just the first one, avoiding exploits where after the first launch an app would escalate its privileges, and with network connections off try to access extra resources. Developers must ensure that all application binaries and bundles are validly signed for the first release and make sure those signatures remain valid after updates. Otherwise, Gatekeeper would block an application from launching. These additional integrity checks prevent the app from being modified in certain ways.

Increasing the granularity of the application permissions. Since macOS version Sonoma (14), Apple introduced[29] more fine-grained control over app permissions and data access. Applications must explicitly ask the user for permission to access files and directories individually.

Enhancing application sandboxing. Since 2021[30] App Sandbox associates an app with its sandbox container using its code signature. The operating system asks the person using an app to grant permission if it tries to access a sandbox container associated with a different app. This futher limits cross-app resource access flows.

6 LIMITATIONS

In this section we would like to list some methodological decisions, that might affect the comparison of our results and the results of the reference work.

Most issues are with the MAS app collection, like in the previous study. The later macOS versions do not allow the 'purchase' operation. For macOS 10.14 and older, mac-cli can perform automatically the full flow of selecting, 'purchasing', and installing an app (the flow is the same for both free and paid apps, the only difference being the price). However, For macOS 10.15+, the 'purchase' step is always manual due to security considerations, so automated apps installations are not possible anymore, which limits our data collection — if a modern app does not have backward compatibility with the 10.14 OS version, this app collection method will skip it. Another issue is the MAS region limit (app can differ between regions due to legal reasons), which could also affect differences between datasets.

We also did not check how much our datasets overlap, which is hard due to the limitations in application collection.

However, our results are encouraging, and their stability over variations of the dataset content indicates that the impact of the above-mentioned discrepancies is not significant.

Replication is also hard because sharing the app datasets is not possible, so metadata dumps will have to be created at best in the future for any cross-dataset content comparisons and app evolution studies.

7 CONCLUSIONS

Our study extends the work of the *Blochberger et al.* [3] by examining the state of application sandboxing on macOS with an updated dataset that includes open-source applications. We performed a differentiated replication to verify the robustness of the original findings across different application sources and newer macOS versions.

Our results align with the original findings that the official Mac App Store has a high adoption rate of sandboxing, with 99.10% of apps being sandboxed in our dataset compared to 93.53% in the 2019 study. This indicates that Apple's policies have continued to encourage widespread adoption of sandboxing. We also observe a notable improvement in the adoption of sandboxing among applications from the MacUpdate platform. Our study shows a significant increase to 26.06% from the 10.94% reported in 2019. This suggests that even third-party platforms are increasingly adhering to sandboxing practices, likely influenced by broader industry trends and user expectations for security. For the first time, our study includes open-source macOS applications, revealing that 17.81% of these apps are sandboxed. While this is lower than the figures for closed-source apps, it highlights that a substantial portion of the open-source community is also adopting sandboxing practices. This inclusion broadens the scope of security analysis on macOS, demonstrating that open-source apps can and should be considered in future studies.

The distribution and co-occurrence of entitlements have evolved, reflecting shifts in application functionality and security practices. Notable changes include the increased use of *User Selected Files (Read-only)* and *Security-scoped Bookmarks*, suggesting developers are adopting more fine-grained control over app permissions. Our analysis shows that the Mac App Store apps consistently follow privilege separation strategies, with a clear separation of entitlements between the main app and its helpers. This practice is less consistent among the open-source and MacUpdate apps.

ACKNOWLEDGMENTS

We thank the Armed Forces of Ukraine for providing security to complete this work.

We thank Maximilian Blochberger for providing the data used to prepare the original paper, Andrey Tkachenko for the assistance in gathering data for this replication, and Ivan Taranenko for assistance in running the local experimental setup.

REFERENCES

[1] Vitor Afonso, Antonio Bianchi, Yanick Fratantonio, Adam Doupe, Mario Polino, Paulo De Geus, Christopher Kruegel, and Giovanni Vigna. 2016. Going Native: Using a Large-Scale Analysis of Android Apps to Create a Practical Native-Code Sandboxing Policy. In *Proceedings 2016 Network and Distributed System Security Symposium*. Internet Society, San Diego, CA. https://doi.org/10.14722/ndss.2016.23384

[2] Michael Backes, Sven Bugiel, Christian Hammer, Oliver Schranz, and Philipp von Styp-Rekowsky. 2015. Boxify: Full-fledged App Sandboxing for Stock Android. 691–706. https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/backes

[3] Maximilian Blochberger, Jakob Rieck, Christian Burkert, Tobias Mueller, and Hannes Federrath. 2019. State of the Sandbox: Investigating macOS Application Security. In *Proceedings of the 18th ACM Workshop on Privacy in the Electronic Society (WPES'19)*. Association for Computing Machinery, New York, NY, USA, 150–161. https://doi.org/10.1145/3338498.3358654

[4] Thomas Bläsing, Leonid Batyuk, Aubrey-Derrick Schmidt, Seyit Ahmet Camtepe, and Sahin Albayrak. 2010. An Android Application Sandbox system for suspicious software detection. In *2010 5th International Conference on Malicious and Unwanted Software*. 55–62. https://doi.org/10.1109/MALWARE.2010.5665792

[5] Mihai Bucicoiu, Lucas Davi, Razvan Deaconescu, and Ahmad-Reza Sadeghi. 2015. XiOS: Extended Application Sandboxing on iOS. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security (ASIA CCS '15)*. Association for Computing Machinery, New York, NY, USA, 43–54. https://doi.org/10.1145/2714576.2714629

[29] https://developer.apple.com/videos/play/wwdc2023/10266/
[30] https://developer.apple.com/documentation/updates/security

[6] Kimo Bumanglag. 2022. *An Application of Machine Learning to Analysis of Packed Mac Malware.* PhD thesis. Dakota State University, Madison, SD, USA. https://scholar.dsu.edu/theses/381

[7] Caio Augusto Pereira Burgardt. 2022. *Malware detection in macOS using supervised learning.* Master's thesis. Universidade Federal de Pernambuco, Recife, Brasil. https://repositorio.ufpe.br/handle/123456789/46235

[8] Luke Deshotels, Razvan Deaconescu, Mihai Chiroiu, Lucas Davi, William Enck, and Ahmad-Reza Sadeghi. 2016. SandScout: Automatic Detection of Flaws in iOS Sandbox Profiles. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16).* Association for Computing Machinery, New York, NY, USA, 704–716. https://doi.org/10.1145/2976749.2978336

[9] Trevor Dunlap, William Enck, and Bradley Reaves. 2022. A Study of Application Sandbox Policies in Linux. In *Proceedings of the 27th ACM on Symposium on Access Control Models and Technologies (SACMAT '22).* Association for Computing Machinery, New York, NY, USA, 19–30. https://doi.org/10.1145/3532105.3535016

[10] Elizabeth Walkup. 2014. Mac Malware Detection via Static File Structure Analysis. https://cs229.stanford.edu/proj2014/Elizabeth%20Walkup,%20MacMalware.pdf

[11] Ezgi Gucuyener and M. Amac Guvensan. 2024. Towards Next-Generation Smart Sandboxes: Comprehensive Approach to Mobile Application Security. In *2024 12th International Symposium on Digital Forensics and Security (ISDFS).* 1–6. https://doi.org/10.1109/ISDFS60797.2024.10527282 ISSN: 2768-1831.

[12] Hamed Haddad Pajouh, Ali Dehghantanha, Raouf Khayami, and Kim-Kwang Raymond Choo. 2018. Intelligent OS X malware threat detection with code inspection. *Journal of Computer Virology and Hacking Techniques* 14, 3 (Aug. 2018), 213–223. https://doi.org/10.1007/s11416-017-0307-5

[13] Luyi Xing, Xiaolong Bai, Tongxin Li, XiaoFeng Wang, Kai Chen, Xiaojing Liao, Shi-Min Hu, and Xinhui Han. 2015. Unauthorized Cross-App Resource Access on MAC OS X and iOS. https://doi.org/10.48550/arXiv.1505.06836 arXiv:1505.06836 [cs].

[14] Yajin Zhou, Kunal Patel, Lei Wu, Zhi Wang, and Xuxian Jiang. 2015. Hybrid User-level Sandboxing of Third-party Android Apps. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security (ASIA CCS '15).* Association for Computing Machinery, New York, NY, USA, 19–30. https://doi.org/10.1145/2714576.2714598