State of the Sandbox: Investigating macOS Application Security

Maximilian Blochberger
blochberger@informatik.uni-hamburg.de
University of Hamburg
Germany

Jakob Rieck
jakobrieck@gmail.com
University of Hamburg
Germany

Christian Burkert
burkert@informatik.uni-hamburg.de
University of Hamburg
Germany

Tobias Mueller
mueller@informatik.uni-hamburg.de
University of Hamburg
Germany

Hannes Federrath
federrath@informatik.uni-hamburg.de
University of Hamburg
Germany

ABSTRACT

Sandboxing is a way to deliberately restrict applications accessing resources that they do not need to function properly. Sandboxing is intended to limit the effect of potential exploits and to mitigate overreach to personal data. Since June 1, 2012, sandboxing is a mandatory requirement for apps distributed through the Mac App Store (MAS). In addition, Apple has made it easier for developers to specify sandbox entitlements – capabilities that allow the app to access certain resources. However, sandboxing is still optional for macOS apps distributed outside Apple's official app store. This paper provides two contributions. First, the sandbox mechanism of macOS is analyzed and a critical sandbox-bypass is identified. Second, the general adoption of the sandbox mechanism, as well as app-specific sandbox configurations are evaluated. For that purpose all 8366 free apps of the MAS, making 25 % of all apps available on the MAS, as well as 4672 apps retrieved from MacUpdate (MU), a third-party app store, were analyzed dynamically. The dataset is over eight times larger than the second biggest study of macOS apps. It is shown that more than 94 % of apps on the MAS are sandboxed. However, more than 89 % of apps distributed through MU do not make use of sandboxing, putting users' data at risk.

CCS CONCEPTS

• **Security and privacy → Operating systems security**; **Software security engineering**; • **Software and its engineering → Extra-functional properties**.

KEYWORDS

sandboxing; capabilities; entitlements; macOS

ACM Reference Format:
Maximilian Blochberger, Jakob Rieck, Christian Burkert, Tobias Mueller, and Hannes Federrath. 2019. State of the Sandbox: Investigating macOS Application Security. In *18th Workshop on Privacy in the Electronic Society (WPES '19), November 11, 2019, London, UK*. ACM, New York, NY, USA, 12 pages. https://doi.org/110.1145/3338498.3358654

WPES '19, November 11, 2019, London, United Kingdom

ACM ISBN 978-1-4503-6830-8/19/11…$15.00
https://doi.org/110.1145/3338498.3358654

1 INTRODUCTION

In an effort to protect the privacy of its millions of customers, Apple has over time added capability restrictions to apps that are distributed via their official App Store for iOS and macOS. On iOS, these restrictions are mandatory, as the App Store is the sole option to install apps. On macOS however, apps can still be installed from third-party sources, where such restrictions might not be enforced. Nonetheless, macOS apps and updates that are distributed through the App Store are checked by Apple for compliance with their submission policy [4]. Since there are millions of apps available in the App Store, mostly for iOS, app review is only feasible by automation. Previous work has shown that static analysis employed by Apple is not without error [18, 56], allowing attackers to publish non-compliant apps. Kurtz [33] has shown that Apple selectively performs dynamic analysis to mitigate this risk.

One restriction employed by Apple is sandboxing, a well-known and established mitigation technique that restricts apps from accessing resources not required for their functionality, thus supporting the principle of least privilege.

In this paper, we analyze whether Apple's sandboxing restriction is effective with respect to macOS, where it is still an optional security feature in contrast to iOS. For that purpose, we surveyed apps obtained from the MAS as well as from MU, a third-party app store. Compared to the next most comprehensive app study on macOS [61], our evaluation considers eight times as many apps and analyzes more than 25 % of all apps available on the MAS. We found that 89 % of apps on the third-party store are not sandboxed, which indicates that sandboxing is rarely employed voluntarily. In addition, we investigated app-specific sandbox configurations, distribution of entitlements, and privilege separation. Our results indicate further that developers choose entitlements sensibly once their app is restricted by a sandbox. We provide details about a critical vulnerability that was discovered while evaluating the apps from our dataset. The vulnerability has already been addressed, thus improving the security of millions of systems worldwide.

The remainder of the paper is structured as follows: First, we provide necessary background on macOS' sandboxing internals.

Second, we present related work on the topic, before we explain our methodology of obtaining the apps and extracting relevant metadata. Next, we evaluate the sandbox adoption among these apps and discuss our findings. Finally, we outline areas for future work based on security and privacy improvements made by macOS Mojave and conclude the paper.

2 FOUNDATIONS: THE APP SANDBOX

In the context of software security, *sandboxing* refers to the isolation of apps by restricting their access to system resources. This limits the potential damage that apps – even when compromised – can inflict on the system [28].

Sandboxing on macOS was first introduced with Mac OS X Leopard (10.5) in 2007 as an optional security feature, which initially could be bypassed easily [46]. Originally configurable only using an undocumented Scheme-like language dubbed Sandbox Profile Language (SBPL), the feature was almost exclusively used by Apple for system processes. Then in 2011, with Mac OS X Lion (10.7), Apple introduced the *App Sandbox* as a usable interface to the sandbox, intended to be employed by third-party apps. Essential to the App Sandbox is the notion of *containers*, which add dedicated home directories for each app. Container data is located at `~/Library/Containers/<bundleId>`. Apps can only access the `Data` sub-directory of the container, which acts as a dedicated home directory. Since 2012, sandboxing is mandated for apps offered via the MAS [26]. For apps distributed outside the store, it remains an optional security feature to this day (April 2019). Though nowadays an important component used to secure both system services and third-party apps, the sandboxing implementation remains closed source and is not documented on a technical level.

The remainder of this section provides background information about the App Sandbox on macOS. First, we investigate the kinds of threats it is supposed to defend against and then describe how the system fits into the broader development and runtime lifecycles of apps.

2.1 Threat Model

An official threat model for the App Sandbox is not provided by Apple. Here, we instead deduce our own informal model from available marketing materials [40], developer documentation [3, 5], and sandbox-related patents [16, 30, 31, 32, 48].

Public marketing material states that sandboxing keeps "your computer and your information safe" from apps misbehaving due to inadvertent programming defects ("bugs") and even from apps that are "compromised by malicious software" [40]. Developer documentation echoes the same sentiments: the App Sandbox is "designed to contain damage to the system and the user's data if an app becomes compromised" [5]. Intentionally malicious software on the other hand is explicitly excluded from the threat model [3]. There is no practical difference between *intentionally malicious* apps and apps that *become compromised*, as both result in malicious code being executed. This seeming contradiction therefore can be interpreted as an admission that sandboxing in itself cannot stop malicious apps from abusing their officially granted privileges. As an example, a malicious instant messenger allowed to access a user's contact database can silently upload and steal this information without the user ever knowing. Sandboxing however should confine even intentionally malicious software to their (hopefully) limited capabilities. This interpretation is consistent with Apple's patents on the topic, which motivate the need for sandboxing by stating that a "program may be a malicious program that is developed to intentionally cause damages" and that by using sandboxing, "such damages can be greatly reduced" [31].

2.2 Sandbox Lifecycle

This section covers three key aspects of the App Sandbox: Configuration, Initialization, and Enforcement. Sandbox configuration describes the process by which sandboxing is enabled and configured during the development phase of an app. During an app's runtime, the sandbox is first initialized. Whenever an app then requests resources protected by sandboxing, the operating system verifies that the app is allowed to perform the requested operation.

2.2.1 Configuration. The App Sandbox is enabled and configured via *entitlements*, which confer specific resources and capabilities to programs. Entitlements are specified as key-value pairs. Strings are used as keys to identify a capability. Values can be used to further configure each entitlement and can have various types – boolean ones in the simplest case.

To satisfy the mandatory sandbox requirement for submissions to the MAS, developers enable the *Sandbox entitlement*: `com.apple.security.app-sandbox`. The App Sandbox is enabled by default for apps created with Xcode, Apple's IDE. Sandboxed apps are then constrained in their execution and for example cannot access a user's files, microphone, or the camera. Most apps cannot perform their functions in such a constrained environment. Therefore, developers can specify additional entitlements in order to regain the capability to access specific resources: for example, enabling the `com.apple.security.device.microphone` entitlement allows access to the microphone, while adding the `com.apple.security.network.client` entitlement grants network access. In total, roughly 50 such sandbox-related entitlements are documented [6, 9]. Entitlements are embedded as a property list in the app's executable and their integrity is protected by the app's code signature.

Since only relatively few entitlements are available and their configurability is limited, they offer a high-level and easy to use, albeit rather inflexible interface to the sandboxing mechanism. They are the only official way to configure sandboxing on macOS. Entitlements on iOS are different, hence a direct comparison would not make sense.

Prior to their release on the MAS, apps go through App Review, where they are analyzed statically, as well as dynamically [33]. App Review looks at an app's entitlements. If it is unclear why certain entitlements are needed, the developer might need to provide a written explanation. Apple reserves the right to reject app submissions if they consider them malicious or overreaching in their usage of certain entitlements [3].

2.2.2 Initialization. On macOS, all apps start out unsandboxed and may *become* sandboxed only later. Early in the startup process, the

dynamic linker checks whether the app has the sandbox entitlement enabled. If enabled, `libsandbox`[1] is invoked and extracts the app's embedded entitlements to compile the final sandboxing profile. This profile is cached and used directly on subsequent launches, alleviating the need to compile it multiple times. With the compiled profile obtained in the previous step, the dynamic linker then invokes a system call to initialize the sandbox before handing over control to the app code, i. e., to its `main()` function or to shared library constructors.

Since sandboxing is normally initialized *before* transferring control to third-party code, it should protect against malicious app code. Note however that the dynamic linker runs in the context of the app itself. If sandbox initialization fails for any reason or if it is possible for app code to run before the initialization is complete, all safeguards provided by the App Sandbox are rendered ineffective. In contrast to macOS, the sandbox on iOS is enforced by the kernel. Here, apps without a container – which is not created for unsandboxed processes – will be terminated [34].

2.2.3 Enforcement. During normal operation, apps call on the operating system to perform fundamental actions on its behalf such as opening files, sending or receiving network packets, or communicating with other software. In a sandboxed environment, the operating system needs to ensure the caller is allowed to perform requested actions. To do this, macOS's kernel calls out to the sandbox kernel extension[2], which uses the compiled sandbox profile installed during app initialization to decide whether to allow the request. The requested resource or an error code is then returned to the app, respectively [14].

3 RELATED WORK

Maass et al. [39], Schreuders et al. [49], and Shu et al. [50] provide structured overviews of sandboxing mechanisms in general. Related work on the App Sandbox, Apple's App Review process, and permissions is explained in detail below.

App Sandbox. In 2011, Blazakis [14] detailed the early implementation of the sandbox on macOS and released a set of useful tools for security researchers. In the same year, Vilaça [55] released reverse-engineered documentation of the underlying sandbox configuration language SBPL. It remains the only publicly available, although outdated, documentation to date. Sandbox configurations are compiled into an opaque binary format. A number of works, most recently Deaconescu et al. [17] and Esser [22], have investigated and attempted to decompile the binary format used on iOS. Deshotels et al. [19, 20] analyze semantic flaws in sandbox profiles, which weaken the security provided by default sandbox profiles, and might be used as gadgets in exploit chains. Edge and O'Donnell [21] provide practical examples that show how custom sandbox profiles can be used to harden apps. Levin [34, 35, 36] and Miller et al. [45] offer insights into the inner workings of macOS and iOS, including implementation details of the sandboxing mechanism.

In 2012, Swami [53] presented a practical attack that disabled sandboxing for apps at runtime. The attack made use of pre-loading a dynamic library, which in turn replaces a symbol required for initializing the sandbox. The attack is prevented by System Integrity Protection (SIP) on current versions of macOS.

App Review. Deng et al. [18], Kurtz [33], and Wang et al. [56] investigate Apple's App Review process and present flaws that lead to malicious apps being accepted into the MAS. We discuss a vulnerability in Section 5.4, which also passed the review process.

Permissions. Android or iOS permissions are quite similar to entitlements. They both are used to protect users' private information. Nowadays most permissions are *runtime permissions* that ask the user for consent before accessing a resource. *Install-time permissions* are functionally equivalent to entitlements. The only difference is that permissions are displayed to the user before installing the app. Acar et al. [1] provide a structured overview outlining the field of permission research for Android. Liu et al. [37] present a recommendation system for privacy preferences based on users' permission choices per category amongst other things. Recommendations could take the expected *permissions per category* into account, just as we have analyzed for entitlements in Section 5.2. Further work in the field is used to compare our results to findings already scrutinized [15, 23, 59] or to put our perspective in context [24, 27, 60, 62].

4 METHODOLOGY

In this section, we describe the method used to gather macOS apps, followed by the extraction of features regarding sandboxing and version metadata.

4.1 Application Crawling

In this initial step, we download, install and extract features from as many macOS apps as were available on two different sources: First, the MAS, Apple's official app store which is an integral part of macOS, and second, MU, a third-party app store. Note that downloading the apps is necessary because the app-specific sandbox configuration is not available via any publicly available metadata, but has to be read from the downloaded app. In the following, we describe the crawling process for both sources.

4.1.1 Mac App Store (MAS). The MAS is only available on macOS and offers no way of comprehensively listing all available apps. Apple's Enterprise Partner Feed [8] would likely provide the listing functionality, but is not available to researchers. We instead make use of the *Mac App Store Preview* page, which lists available apps by genre and can be viewed with a web browser [11]. Utilizing the web interface, we developed a custom metadata crawler, which periodically scans the store for new apps. For each listed app, we query the iTunes Search API [10] to obtain its full metadata. Due to rate-limiting, each crawl takes approximately three hours to complete. Note that purchasing, downloading, and installing apps is only possible using the official desktop client app. Therefore, we utilized a modified version of the `mas` command line utility [44] to automatically *purchase* and install detected apps. Note that free apps need to be *purchased* as well, i. e., tied to the user's Apple ID (user account). We also use this utility to check for updates, because the official MAS app becomes unresponsive if too many apps are installed and shows at most 200 available updates.

[1] `/usr/lib/libsandbox.dylib`
[2] `/System/Library/Extensions/Sandbox.kext`

The crawling was performed between November 2017 and September 2018. In that period, we 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. The resulting dataset consists of 8366 apps with a total of 15 832 versions. As of January 15, 2019, this also includes 518 paid apps that had been free sometime during our crawl, probably due to promotional sales. Note that we were not able to obtain historic app versions that predate the version at the time of the app purchase. As a consequence, versions of temporarily free apps might not be represented throughout the whole crawling period. In addition, updates that were quickly superseded might have been missed, as we crawled once a day. Some app versions were uploaded as a separate app, thus interrupting the traceable version history. Uploading separate apps is a known technique to allow paid upgrades, which are otherwise not supported. We used natural sort order to create a reproducible version history, although it might not reflect the actual order of release. Out of a random sample of 250 apps only one app did change the version format in between releases.

We believe that we were able to obtain a close to complete list of free macOS apps available on the MAS. This is supported by the fact that the total number of apps found by our scanner exceeds the number of apps reported by *AppShopper* [13].

4.1.2 MacUpdate (MU). In addition to MAS, we also obtained apps listed on MU [41], a website offering downloads for many macOS apps. In contrast to MAS, MU apps are not required to be sandboxed.

During our crawl, we 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, so called *Classic apps*. This resulted in 6164 downloaded apps. Out of these apps, 1492 were not compatible with our test system and could not be executed. Thus, a total of 4672 compatible apps could be obtained from MU.

Note that the crawling of MU was performed in September 2018. In contrast to the MAS, we limited crawling to a single pass through and did not repeatedly scan for new apps or version updates. This is due to the fact that apps' bundle IDs, which are curated by Apple for applications with valid code-signatures, cannot be reliably used as app identifiers outside of the MAS. Therefore, there is no reliable way to detect different versions of the same app.

4.2 Feature Extraction

We extracted sandbox configurations and additional metadata for each obtained app. In a first step, we 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 case of MAS apps, the following additional metadata was extracted using the iTunes Store API on September 19, 2018: (iii) version release dates, and (iv) version ratings. In a second step, we conducted a dynamic feature extraction to verify if sandboxing is indeed active in the running app. Since the sandbox is initialized during app startup, we launched each app, waited for about 30 seconds like Kurtz [33] to allow initialization routines to finish, and then asked the operating system whether the app is sandboxed, as depicted in Listing 1.

```
// Apple private API declares sandbox_check, which is implemented
// in /usr/lib/system/libsystem_sandbox.dylib.
int sandbox_check(pid_t pid, const char *operation, int type, ...);

// This can be used to determine if a process pid is sandboxed.
int process_is_sandboxed(pid_t pid) {
  return sandbox_check(pid, NULL, 0);
}
```

Listing 1: C code that determines whether a process is sandboxed.

5 EVALUATION OF SANDBOX ADOPTION

In this section, we evaluate the sandbox adoption based on the data extracted from the crawled macOS apps. First, we investigate the general adoption of sandboxing and contrast the findings between different versions of an app as well as between the two sources MAS and MU. Second, we evaluate app-specific sandbox configurations. Third, we investigate whether privilege separation is used as intended. Finally, we describe a sandbox bypass vulnerability.

Note that we did not investigate differences between free and paid apps, as related work shows that such differences are negligible [38, 43, 54, 58].

5.1 Sandbox Adoption in General

For the purposes of this evaluation, we consider 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.

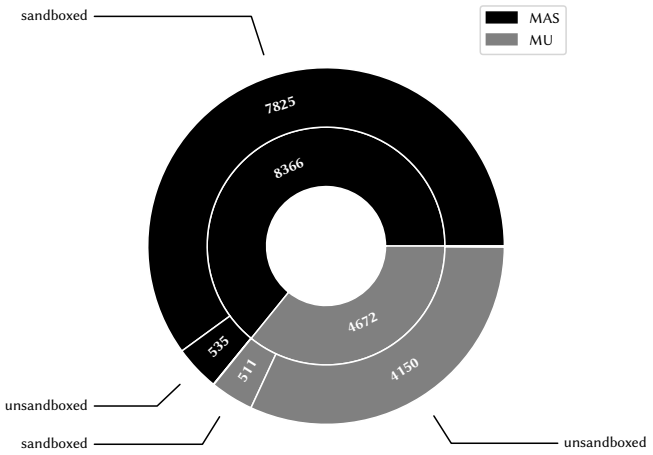

Figure 1: Comparison of MAS and MU apps and their sandbox status. Six MAS and eleven MU apps were not sandboxed during runtime despite having the sandbox entitlement.

5.1.1 Mac App Store (MAS). As depicted in Figure 1, 7825 (93.53 %) out of 8366 apps distributed through the MAS were sandboxed. 535 (6.39 %) apps were not sandboxed. Six apps did contain the

sandbox entitlement but were not sandboxed during runtime, a behavior which we investigate in Section 5.4. For 7818 (93.45 %) apps, all collected versions were sandboxed, whereas 539 (6.44 %) apps had no sandboxed version. Only nine apps had both sandboxed and unsandboxed versions. Almost all of the unsandboxed apps were released before sandboxing became mandatory as depicted in Figure 2.

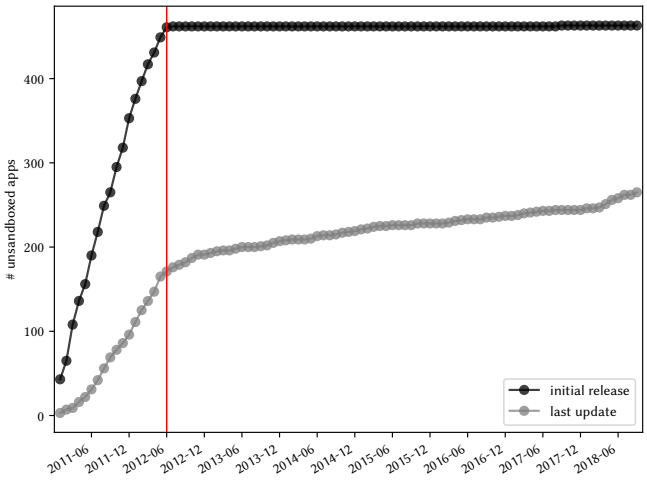

Figure 2: Cumulative distribution of MAS apps' initial releases and their last updates, starting from the inception of the MAS until September 2018. The red line marks the time from which on Apple has mandated sandboxing on the MAS.

5.1.2 MacUpdate (MU). In case of MU, 511 (10.94 %) out of 4672 apps were sandboxed, whereas 4150 (88.83 %) apps were not sandboxed. Eleven of the unsandboxed apps contained the sandbox entitlement, see Section 5.4 for details.

5.1.3 Common Applications. Based on the app bundle identifiers, 173 apps (1.29 % of all apps) were contained in both datasets. Only the latest collected versions were taken into account. Out of these apps, 53 (30.64 %) apps were sandboxed for both sources. For 94 (54.34 %) apps, only the MAS variant was sandboxed. The remaining 26 apps were not sandboxed for both sources.

5.1.4 Discussion. Effectively all newly released MAS apps are sandboxed, while the majority of MU apps are not. The unsandboxed MAS apps were released before sandboxing became mandatory (legacy apps). Since App Review is performed for app submissions, App Store restrictions usually are not applied to existing apps but only to new apps and updates. Updates to legacy apps without adopting current restrictions seem to be tolerated, as can be seen in Figure 2. Six unsandboxed applications were released in 2017. All of them were affected by a bug, for which details are described in Section 5.4.

While sandbox adoption of MAS apps is high, future work should identify and address reasons for the low adoption of the sandboxing mechanism of apps distributed outside of the MAS.

We were surprised by the low number of apps contained in the intersection of both datasets, which might be due to our simple bundle identifier matching strategy. Since about two thirds of the apps contained in the intersection were sandboxed in the MAS but not in MU, it might be worth investigating whether the unsandboxed variants are outdated or whether they contain additional features that require more privileges. The latter case might point out that Apple's restrictions are too prohibitive for some use-cases.

5.2 Entitlements

For the evaluation of particular entitlements, only the most recent version of each app was included. As entitlements are only enforced if the app is sandboxed, only sandboxed apps were included. For simplification, we only include entitlements that can be easily configured by developers in the App Sandbox section of the app's capabilities configuration in Xcode (see Table A.1 for details), except for the sections *Temporary Exception Entitlements* and *Private Entitlements*. Entitlements that add the capability of accessing files or folders were grouped together, regardless of whether they are set to *Read Only* or *Read/Write*.

5.2.1 Distribution. As can be seen in Table 1, 5493 (65.88 %) apps contain the *Client* and 3787 (45.42 %) apps contain the *User Selected Files* entitlements. The *Client* entitlement allows apps to access resources over the Internet or other network connections. Felt et al. [23] have shown that 62 % of Android apps crawled in October 2010, use the `INTERNET` permission, which coincides with our results. The *User Selected Files* entitlement enables a mechanism called *Powerbox* [5, 49], which presents an Open/Save dialog to the user running outside the application context. Note that the read-only variant of that entitlement is enabled by default in Xcode. With *User Selected Files* enabled, the app can access files or directories selected in the file dialog. Note that selecting a directory does not necessarily grant access to all files within that directory. Some files need to be selected explicitly, like the Transparency Consent and Control (TCC) database[3], or permission protected resources.

5.2.2 Co-occurrences. Next, we analyzed which entitlements co-occurred with others. For that purpose, apps were grouped by a base entitlement. For each additional entitlement we calculated the percentage of apps within that group possessing both entitlements, which is visualized in Figure 3. Base entitlements used by fewer than 25 apps were removed, as they are not representative. Ignoring entitlements used by many apps such as *Client*, *User Selected Files*, *Print*, and *Server*, co-occurrences above 40 % are *Camera* and *Microphone*, where nearly half of the apps containing one also contain the other. Apps that have the *Bluetooth*, *Calendar*, or *Movies Folder* entitlement, also have the *USB*, *Contacts*, or *Pictures Folder* entitlement respectively.

5.2.3 Popularity. Entitlements are not visible to the user. Nonetheless, we investigated whether application popularity could be affected by application privileges as has been suggested by related work on Android permissions [27, 62]. For that purpose, we grouped MAS apps by their respective rating in the MAS. However, entitlements were distributed evenly between half-star ratings from 0 to

[3] `~/Library/ApplicationSupport/com.apple.TCC/TCC.db`

Table 1: Overview of sandbox-related entitlements which are used by more than 5 % of all sandboxed apps.
†**The entitlements are not in the App Sandbox section in Xcode. While *Application Groups* are still supported [9], *Security-scoped Bookmarks* are only mentioned in deprecated documentation [6].**

Entitlement	MAS		MU		All	
	#	%	#	%	#	%
Client	5104	65.21 %	389	76.13 %	5493	65.88 %
User Selected Files (Read/Write)	3371	43.07 %	416	81.41 %	3787	45.42 %
Print	1001	12.79 %	164	32.09 %	1165	13.97 %
Server	1007	12.87 %	75	14.68 %	1082	12.98 %
Security-scoped Bookmarks†	723	9.24 %	159	31.12 %	882	10.58 %
Application Groups†	624	7.97 %	52	10.18 %	676	8.11 %
User Selected Files (Read Only)	615	7.86 %	19	3.72 %	634	7.60 %
Downloads (Read/Write)	420	5.37 %	38	7.44 %	458	5.49 %
Microphone	402	5.14 %	25	4.89 %	427	5.12 %

Figure 3: Percentage of sandboxed apps containing an additional entitlement in relation to a base entitlement. Base entitlements with fewer than 25 apps (*Camera*, *Microphone*, *Bluetooth*, *Location*, *Calendar*, and *Movies Folder* in MU) were removed, as they are not representative.

5 stars, indicating that non-visible app privileges gained through entitlements did not affect users' rating of the app.

5.2.4 Categories. We grouped apps per MAS category and evaluated, which entitlements were used by apps within that category, as can be seen in Figure 4. Note that 22 sandboxed apps from MAS had no category information and were excluded. Ignoring common entitlements such as *Client*, *User Selected Files*, *Print*, and *Server*, nearly 60 % of apps within the *Weather* category contain the *Location* entitlement. About 25 % of *Social Networking* apps have the *Microphone* and *Camera* entitlement and access to the *Downloads Folder*. Above 20 % of apps in the *Music*, *Photography*, or *Video* category have access to the *Music*, *Pictures*, or *Movies Folder* respectively. 22.33 % of *Video* apps also have the *Microphone* entitlement. Apps from the *Games* category have the least privileges.

5.2.5 Historical Development. In Figure 5, we looked at which entitlements got added or removed for different versions of an app. Prior work analyzed the evolution of permission usage for Android apps [59] and ad libraries [15] and found that privileges increase over time. We come to the same conclusion, since more than twice as many entitlements get added than get removed. Due to the lack of app versions in the MU dataset, this evaluation was only performed for MAS apps.

5.2.6 Temporary Exception Entitlements. Temporary exception entitlements have a dual use [6] (see Table A.3 for details). On one hand, they show problems with Apple's restrictions by apps not being able to function properly with the given sandboxing mechanism. On the other hand, they allow an incremental transition for apps to employ the sandboxing mechanism. The first needs to be addressed by changes to the sandboxing mechanism and hence the operating system itself. The latter can be addressed by apps gradually applying additional restrictions, which is a necessity due to the difficulty of adding security mechanism to an existing app. Only a few apps use temporary exception entitlements. The most commonly used temporary exception allows for scripting and automating of other apps, and is used by 3.2 % of apps.

5.2.7 Private Entitlements. Private entitlements are not documented by Apple and developers are not allowed to utilize them. They can provide elevated sandbox exceptions, such as access to the TCC database, where users' decisions for permission requests are stored. In 2017, Strafach [51] uncovered that the *UBER* iOS app was granted such an entitlement. In our dataset, only Apple's apps use private entitlements.

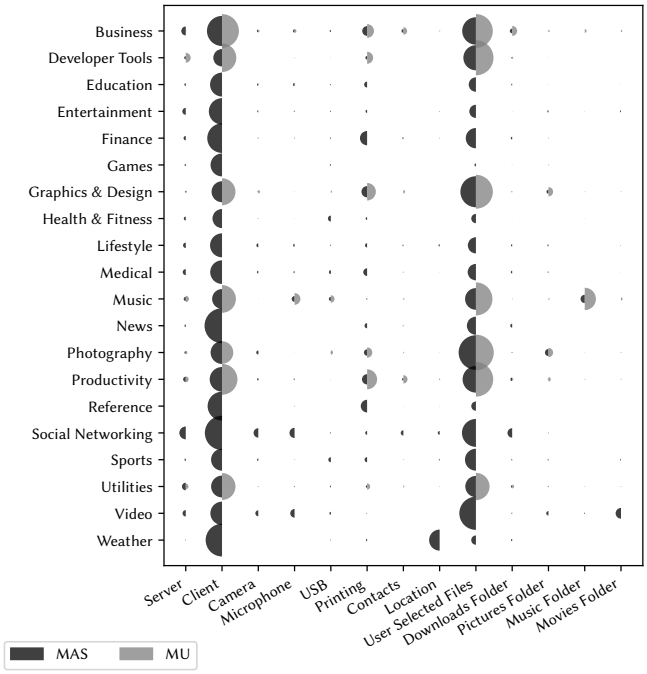

Figure 4: Percentage of entitlements per MAS category. In addition we excluded the *Bluetooth* and the *Calendar* entitlements, as they occurred less than 10 % in each category. Categories with less than 25 apps (*Education*, *Entertainment*, *Finance*, *Games*, *Health & Fitness*, *Lifestyle*, *Medical*, *News*, *Reference*, *Social Networking*, *Sports*, *Video*, and *Weather* in MU and *Travel* in both datasets) were removed as well, as they are not representative.

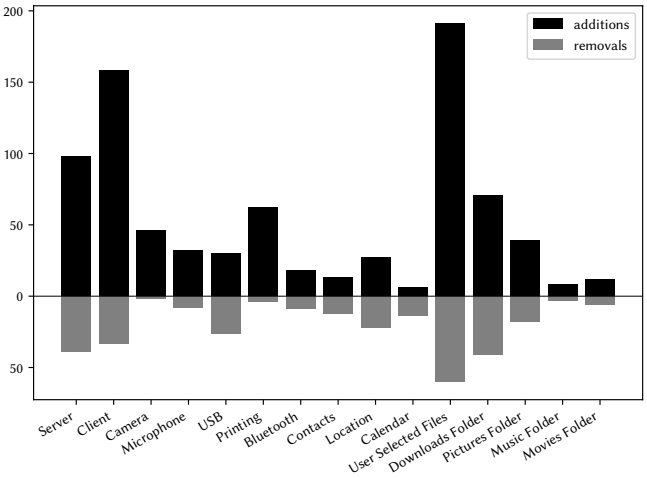

Figure 5: Entitlements added and removed between versions of MAS apps.

5.2.8 Discussion. Entitlement co-occurrences as well as entitlements per category look sensible, meaning that we found no noticeable problems. We tried to identify anomalous entitlements in order to determine over-privileged apps, by clustering entitlements with respect to their co-occurrences, the app's categories, and all applications. However, the dataset was too diverse, so that no sensible results could be obtained. We manually inspected apps with unique combinations of entitlements and with the most privileges and found no suspicious configuration. This indicates that once restricted to sandboxing, developers make sensible choices regarding the privileges an app requires. This hypothesis is supported by Jain and Lindqvist [29], who have already shown that developers make sensible choices regarding resource access, if the options presented to them are easy to use and comprehend.

Our analysis only includes entitlements that have been granted to apps. Future work should identify whether entitled resources are actually used by the app or whether they are over-privileged.

5.3 Privilege Separation

Privilege separation is a "generic approach that lets parts of a process run with different levels of privilege" [47]. The technique can be used on top of sandboxing to further reduce the impact of compromise by limiting the attack surface to separate app components (helpers).

On macOS, XPC provides the mechanism for inter-process communication (IPC) and provides an interface for developers to subdivide their app into multiple helper components referred to as XPC services. Since sandboxed apps cannot start child-processes with higher privileges, XPC services are started by the operating system. Each helper has its own sandbox and a sandbox configuration that is specifically tailored to that helper's functionality. In contrast to apps, the App Sandbox is *disabled* by default for XPC services created with Xcode. Apple describes the technology as being one of the pillars of proper sandboxing and encourages developers to make extensive use of XPC services for stability and security reasons [7].

We investigated whether XPC helpers have more, less, or the same privileges as the main app. If a helper has some entitlements granting privileges the main app does not have and at the same time lacks entitlements the main app possesses, then the privileges cannot be compared. We consider these helpers to have mixed privileges. Table 2 shows that the majority of MAS apps with XPC helpers use lower privileges for their helpers. Most helpers do not have the *User Selected Files*, *Client*, or *Printing* entitlements, which the app has. The majority of MU apps however use helpers that have more privileges than the app itself; nearly all of these helpers are not sandboxed at all.

5.3.1 Discussion. Since app components can potentially access resources to which the app binary is not entitled to, the app can still gain access to these resources by communicating via IPC. Hence, the entitlements of the helpers need to be included in the overall privileges of an app. We re-evaluated *co-occurrences* and *entitlements per category* with respect to the overall privileges of the app and were not able to find significant differences compared to the findings presented in Section 5.2 that only made use of the entitlements of the app binary.

Many XPC helpers outside the MAS are not sandboxed at all. The usage of sandboxing on the MAS however indicates that Apple ensures helpers are sandboxed as well. Note that no dynamic

Table 2: Comparing privileges of XPC helpers to the privileges of the main app.
† **Note that an app is added to all categories for which it has at least one helper.**

	MAS		MU	
	Apps	Helpers	Apps	Helpers
Same privilege	11	13	4	6
Helper more privileged	4	4	95	199
Helper less privileged	46	94	24	51
Mixed privileges	14	24	11	27
Overall	64†	135	110†	283

analysis has been performed for helpers. However, they might be affected by problems during sandbox initialization as well.

Our analysis only includes helpers that exist inside the app bundle, right after the app was installed (MAS) or extracted (MU). Apps might prompt the user to install additional helpers. Future work should investigate how prevalent additional helpers are and how their privileges compare to the privileges of helpers contained in app bundles.

5.4 Bypassing the Sandbox

In Section 2.2.2, we discussed that the sandbox is initialized in the context of the app. We argued already that this can potentially disable sandbox protection. During the dynamic analysis we performed, six MAS and eleven MU apps were identified that were not sandboxed at runtime, which we did not expect. After further investigation, we discovered that a bug in the property list parser led to the sandbox not being correctly initialized.

```
xpc_object_t raw_entitlements =
↪  xpc_copy_entitlements_for_pid(getpid());
if (raw_entitlements) {
  const void *data = xpc_data_get_bytes_ptr(raw_entitlements);
  size_t len = xpc_data_get_length(raw_entitlements);
  ctx->entitlements = xpc_create_from_plist(data , len);
  xpc_release(raw_entitlements);
}
```

Listing 2: Reverse-engineered excerpt from the sandbox initialization process

5.4.1 Vulnerability Description. During initialization of the sandbox, the `libsystem_secinit`[4] library gets linked into the app context. Amongst other things, it reads and parses the entitlements from the app's binary. If the app does not possess entitlements, the `xpc_copy_entitlements_for_pid()` function returns `NULL`, causing `ctx->entitlements` not to be populated, as can be seen in Listing 2. This leads to the sandbox initialization being skipped. For the six apps retrieved from the MAS, the entitlements started with a Byte order mark (BOM), i. e., `0xEFBBEF`, which indicates that the file is UTF-8-encoded. A bug within the `xpc_create_from_plist()` function caused it to return `NULL` as well, which in turn had the same effect. App developers could simply add a BOM at the beginning of their entitlements and bypass the sandbox completely. The issue was reported to Apple[5] and has been addressed in macOS 10.13.5.

5.4.2 Discussion. The vulnerability affected all six apps from the MAS. Visual similarities indicate four out of the six affected apps share substantial code. It is therefore conceivable that they were built from the same underlying project, which may have included a modified entitlement file from the start. We therefore assume that this did not happen with malicious intent.

The eleven apps from MU however did not contain a BOM and we are still investigating the reason why the sandbox is not being properly initialized for these apps. In addition, we checked whether entitlements of XPC helpers did start with a BOM and found none.

The vulnerability is critical, as it can be exploited by malicious developers without much effort. By simply adding a BOM at the beginning of their entitlements, they could circumvent the mandatory sandboxing requirement and still submit their application to the MAS. Users might believe that apps retrieved from the MAS are restricted by the sandbox mechanism. Unfortunately, there is no easy way for users to see whether an app is actually sandboxed, aside from a column in the *Activity Monitor*, which is not visible by default. We suggest that a warning should be displayed to the user for unsandboxed apps, informing him about the implications. However, the effectiveness of security indicators and warnings is a major topic of discussion [2, 24, 25, 52, 60], but might provide an incentive for developers to adopt sandboxing even outside the MAS. With macOS Mojave (10.4), permissions, similar to iOS or Android, were added that provide an additional layer of protection. Even unsandboxed apps are restricted in accessing user data without the user's consent, mitigating the impact of sandbox bypasses, see Section 6 for details.

We argue that the issue is a consequence of the sandbox being initialized during the app startup in contrast to being enforced by the operating system.

6 SECURITY & PRIVACY IMPROVEMENTS IN MACOS MOJAVE

With macOS Mojave (10.14), changes were made to the underlying security mechanisms. As Mojave was released recently, our dataset did not contain enough apps utilizing the improvements to infer meaningful results. However, Mojave's additions do not fundamentally change the impact of the results presented in this paper, and are discussed here as a basis for future work on this topic.

Permissions. Apps ask users for their consent to access private information since macOS Mountain Lion (10.8) with the introduction of Gatekeeper. Gatekeeper requires apps to be signed by registered developers [12]. In addition, apps linked against the Mojave SDK are required to define *usage descriptions* in the app's `Info.plist`. If there is no such description and the app tries to access the resource, it will be terminated. This is enforced for both API calls and direct access to the data through the file system.

Hardened Runtime. The Hardened Runtime can be used to restrict potentially harmful operations during runtime, such as dynamically loading libraries, executing unsigned code pages, or accessing

[4] `/usr/lib/system/libsystem_secinit.dylib`

[5] CVE-2018-4229

certain resources such as the user's address book. Similar to the App Sandbox the hardened runtime requires code-signing and is configured through entitlements embedded in the app's binary (see Table A.2 for details). If an entitlement is missing that guards access to a certain resource, Gatekeeper then suppresses the respective permission dialog, thus preventing the user from consenting. As a consequence, the app will not be able to access the resource, even if it is unsandboxed.

App Notarization. App Notarization requires an additional certificate issued from the *Apple Notarization Service*, which is called a *Notarization Token* and gets stapled to the app's original code signature. This allows Apple to check non-MAS apps for malware. If malware is found, a Notarization Token is not issued and the developer is notified. When App Notarization becomes mandatory as announced [42], the distribution of malware that can be identified by Apple is prevented before it is installed on the user's machine.

App Notarization requires apps to have the Hardened Runtime enabled, which in turn requires configuring entitlements in order to access certain resources. With App Notarization becoming mandatory in the near future, we expect future results of non-MAS apps to change drastically.

7 CONCLUSION

Apple enforces macOS apps to be sandboxed, when they are distributed through the MAS. In addition, Apple has made the configuration of the sandbox easier for developers. We performed a large-scale analysis of macOS apps and found that almost all apps retrieved outside of the MAS lack sandboxing. Therefore, we think that Apple's enforcement is the main reason why macOS apps are sandboxed. However, since entitlements for apps from both MAS and MU are distributed similarly, we think that easy configuration supports developers in making sensible choices. Developers might lack incentives for utilizing the App Sandbox voluntarily and further work is needed to address this problem. Forcing developers to use sandboxing might be effective, but impacts the freedom to develop apps with sophisticated features, which might not be possible in an otherwise too restrictive environment.

REFERENCES

[1] Yasemin Acar, Michael Backes, Sven Bugiel, Sascha Fahl, Patrick D. McDaniel, and Matthew Smith. 2016. Sok: lessons learned from android security research for appified software platforms. In *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 433–451.

[2] Devdatta Akhawe and Adrienne Porter Felt. 2013. Alice in warningland: A large-scale field study of browser security warning effectiveness. In *USENIX Security Symposium*. USENIX Association, 257–272.

[3] Apple Inc. 2011. App sandbox and the mac app store. In *WWDC*. https://developer.apple.com/videos/play/wwdc2011/204/.

[4] Apple Inc. [n. d.] App store review guidelines. Retrieved April 24, 2019 from https://developer.apple.com/app-store/review/guidelines/.

[5] Apple Inc. 2016. Apple developer documentation: app sandbox design guide. (September 13, 2016). Retrieved September 18, 2018 from https://developer.apple.com/library/archive/documentation/Security/Conceptual/AppSandboxDesignGuide.

[6] Apple Inc. 2017. Apple developer documentation: entitlement key reference. (March 27, 2017). https://developer.apple.com/library/archive/documentation/Miscellaneous/Reference/EntitlementKeyReference.

[7] Apple Inc. 2016. Creating xpc services. Retrieved April 24, 2019 from https://developer.apple.com/library/content/documentation/MacOSX/Conceptual/BPSystemStartup/Chapters/CreatingXPCServices.html.

[8] Apple Inc. [n. d.] Enterprise partner feed relational - affiliate resources. Retrieved April 24, 2019 from https://affiliate.itunes.apple.com/resources/documentation/itunes-enterprise-partner-feed/.

[9] Apple Inc. [n. d.] Entitlements - bundle resources | apple developer documentation. Retrieved April 25, 2019 from https://developer.apple.com/documentation/bundleresources/entitlements.

[10] Apple Inc. [n. d.] Itunes search api - affiliate resources. Retrieved April 24, 2019 from https://affiliate.itunes.apple.com/resources/documentation/itunes-store-web-service-search-api/.

[11] Apple Inc. [n. d.] Mac app store preview. Retrieved April 24, 2019 from https://itunes.apple.com/us/genre/id39.

[12] Apple Inc. 2018. macOS Security – Overview for IT. White paper. (November 2018). https://www.apple.com/business/resources/docs/macOS_Security_Overview.pdf.

[13] [n. d.] Appshopper. Retrieved April 24, 2019 from http://appshopper.com/mac/.

[14] Dionysus Blazakis. 2011. The Apple Sandbox. White paper.

[15] Theodore Book, Adam Pridgen, and Dan S. Wallach. 2013. Longitudinal analysis of android ad library permissions. *CoRR*, abs/1303.0857.

[16] Simon Cooper, Nick Lane-Smith, and Joshua Osborne. 2014. Restriction of program process capabilities. (2014). Patent No. US Patent 8,635,663 B2. https://patents.google.com/patent/US8635663B2.

[17] Razvan Deaconescu, Luke Deshotels, Mihai Bucicoiu, William Enck, Lucas Davi, and Ahmad-Reza Sadeghi. 2016. Sandblaster: reversing the apple sandbox. *CoRR*, abs/1608.04303.

[18] Zhui Deng, Brendan Saltaformaggio, Xiangyu Zhang, and Dongyan Xu. 2015. Iris: vetting private API abuse in ios applications. In *ACM Conference on Computer and Communications Security*. ACM, 44–56.

[19] Luke Deshotels, Razvan Deaconescu, Costin Carabas, Iulia Manda, William Enck, Mihai Chiroiu, Ninghui Li, and Ahmad-Reza Sadeghi. 2018. Ioracle: automated evaluation of access control policies in ios. In *AsiaCCS*. ACM, 117–131.

[20] 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 *ACM Conference on Computer and Communications Security*. ACM, 704–716.

[21] Charles Edge and Daniel O'Donnell. 2016. *Enterprise Mac Security*. (3rd ed.). Apress.

[22] Stefan Esser. 2014. Ios 8: containers, sandboxes and entitlements. Retrieved February 12, 2018 from https://www.slideshare.net/i0n1c/ruxcon-2014-stefan-esser-ios8-containers-sandboxes-and-entitlements.

[23] Adrienne Porter Felt, Kate Greenwood, and David A. Wagner. 2011. The effectiveness of application permissions. In *WebApps*. USENIX Association.

[24] Adrienne Porter Felt, Elizabeth Ha, Serge Egelman, Ariel Haney, Erika Chin, and David A. Wagner. 2012. Android permissions: user attention, comprehension, and behavior. In *SOUPS*. ACM, 3.

[25] Adrienne Porter Felt, Robert W. Reeder, Alex Ainslie, Helen Harris, Max Walker, Christopher Thompson, Mustafa Emre Acer, Elisabeth Morant, and Sunny Consolvo. 2016. Rethinking connection security indicators. In *SOUPS*. USENIX Association, 1–14.

[26] Chris Foresman. 2012. Apple's latest sandboxing deadline delay signals moving goalposts for devs. (February 22, 2012). Retrieved March 14, 2018 from https://arstechnica.com/?p=36730.

[27] Bin Fu, Jialiu Lin, Lei Li, Christos Faloutsos, Jason I. Hong, and Norman M. Sadeh. 2013. Why people hate your app: making sense of user feedback in a mobile app store. In *KDD*. ACM, 1276–1284.

[28] Ian Goldberg, David A. Wagner, Randi Thomas, and Eric A. Brewer. 1996. A secure environment for untrusted helper applications. In *USENIX Security Symposium*. USENIX Association.

[29] Shubham Jain and Janne Lindqvist. 2014. Should i protect you? understanding developers' behavior to privacy-preserving apis. In *USEC '14*, 1–10.

[30] Ivan Krstić and Love Hörnquist Åstrand. 2016. File system access for one or more sandboxed applications. (2016). Patent No. US Patent 9,342,689 B2. https://patents.google.com/patent/US9342689B2.

[31] Ivan Krstić, Austin G. Jennings, and Richard L. Hagy. 2016. Methods for restricting resources used by a program based on entitlements. (2016). Patent No. US Patent App. 15/060,837. https://patents.google.com/patent/US20160321471A1.

[32] Ivan Krstić and Pierre-Olivier J. Martel. 2013. System and method for preserving references in sandboxes. (2013). Patent No. US Patent 8,601,579 B2. https://patents.google.com/patent/US8601579B2.

[33] Andreas Kurtz. 2016. *Dynamic analysis and privacy implications of Apple iOS apps*. Ph.D. Dissertation. University of Erlangen-Nuremberg, Germany.

[34] Jonathan Levin. 2017. **OS Internals: Security & Insecurity*. Volume 3.

[35] Jonathan Levin. 2017. **OS Internals: User Space*. Volume 1.

[36] Jonathan Levin. 2012. *Mac OS X and iOS Internals: To the Apple's Core*. Wiley.

[37] Bin Liu, Mads Schaarup Andersen, Florian Schaub, Hazim Almuhimedi, Shikun Zhang, Norman M. Sadeh, Yuvraj Agarwal, and Alessandro Acquisti. 2016.

Follow my recommendations: A personalized privacy assistant for mobile app permissions. In *SOUPS*. USENIX Association, 27–41.

[38] Wei Liu, Ge Zhang, Jun Chen, Yuze Zou, and Wenchao Ding. 2015. A measurement-based study on application popularity in android and ios app stores. In *Mobidata@MobiHoc*. ACM, 13–18.

[39] Michael Maass, Adam Sales, Benjamin Chung, and Joshua Sunshine. 2016. A systematic analysis of the science of sandboxing. *PeerJ Computer Science*, 2.

[40] [n. d.] Macos - security - apple. Retrieved September 18, 2018 from https://www.apple.com/macos/security/.

[41] [n. d.] Macupdate. Retrieved April 24, 2019 from https://macupdate.com.

[42] Pierre-Olivier Martel, Kelly Yancey, and Garrett Jacobson. 2018. Your apps and the future of macos security. In *WWDC*. https://developer.apple.com/videos/play/wwdc2018/702/.

[43] William Martin, Federica Sarro, Yue Jia, Yuanyuan Zhang, and Mark Harman. 2017. A survey of app store analysis for software engineering. *IEEE Trans. Software Eng.*, 43, 9, 817–847.

[44] [n. d.] Mas-cli/mas. Retrieved April 1, 2019 from https://github.com/mas-cli/mas.

[45] Charlie Miller, Dionysus Blazakis, Dino Dai Zovi, Stefan Esser, Vincenzo Iozzo, and Ralf-Philipp Weinmann. 2012. *iOS Hacker's Handbook*. Wiley, (May 2012). ISBN: 978-1118204122.

[46] Charlie Miller and Dino A. Dai Zovi. 2009. *The Mac Hacker's Handbook*. (1st ed.). Wiley. ISBN: 978-0-470-39536-3.

[47] Niels Provos, Markus Friedl, and Peter Honeyman. 2003. Preventing privilege escalation. In *USENIX Security Symposium*. USENIX Association.

[48] David Rahardja, Toby C. Paterson, and Anthony D'Auria. 2018. Mediated data exchange for sandboxed applications. (2018). Patent No. US Patent 9,898,355 B2. https://patents.google.com/patent/US9898355B2.

[49] Z. Cliffe Schreuders, Tanya McGill, and Christian Payne. 2013. The state of the art of application restrictions and sandboxes: A survey of application-oriented access controls and their shortfalls. *Computers & Security*, 32, 219–241.

[50] Rui Shu, Peipei Wang, Sigmund Albert Gorski III, Benjamin Andow, Adwait Nadkarni, Luke Deshotels, Jason Gionta, William Enck, and Xiaohui Gu. 2016. A study of security isolation techniques. *ACM Comput. Surv.*, 49, 3, 50:1–50:37.

[51] Will Strafach. 2017. Archived. (October 3, 2017). Retrieved April 25, 2019 from https://web.archive.org/web/20180207074222/https:/twitter.com/chronic/status/915281242597314560.

[52] Joshua Sunshine, Serge Egelman, Hazim Almuhimedi, Neha Atri, and Lorrie Faith Cranor. 2009. Crying wolf: an empirical study of SSL warning effectiveness. In *USENIX Security Symposium*. USENIX Association, 399–416.

[53] Yogesh Swami. 2012. Axelexic/sanboxinterposed. Retrieved April 1, 2019 from https://github.com/axelexic/SanboxInterposed.

[54] Nicolas Viennot, Edward Garcia, and Jason Nieh. 2014. A measurement study of google play. In *SIGMETRICS*. ACM, 221–233.

[55] Pedro Vilaça. 2011. Apple's sandbox guide. Version 0.1. (September 3, 2011).

[56] Tielei Wang, Kangjie Lu, Long Lu, Simon P. Chung, and Wenke Lee. 2013. Jekyll on ios: when benign apps become evil. In *USENIX Security Symposium*. USENIX Association, 559–572.

[57] Patrick Wardle. 2018. Escaping the microsoft office sandbox. (August 15, 2018). Retrieved April 24, 2019 from https://objective-see.com/blog/blog_0x35.html.

[58] Takuya Watanabe, Mitsuaki Akiyama, Fumihiro Kanei, Eitaro Shioji, Yuta Takata, Bo Sun, Yuta Ishii, Toshiki Shibahara, Takeshi Yagi, and Tatsuya Mori. 2017. A study on the vulnerabilities of mobiles apps associated with software modules. *CoRR*, abs/1702.03112.

[59] Xuetao Wei, Lorenzo Gomez, Iulian Neamtiu, and Michalis Faloutsos. 2012. Permission evolution in the android ecosystem. In *ACSAC*. ACM, 31–40.

[60] Primal Wijesekera, Arjun Baokar, Ashkan Hosseini, Serge Egelman, David A. Wagner, and Konstantin Beznosov. 2015. Android permissions remystified: A field study on contextual integrity. In *USENIX Security Symposium*. USENIX Association, 499–514.

[61] Luyi Xing, Xiaolong Bai, Tongxin Li, XiaoFeng Wang, Kai Chen, Xiaojing Liao, Shi-Min Hu, and Xinhui Han. 2015. Cracking app isolation on apple: unauthorized cross-app resource access on MAC OS X and ios. In *ACM Conference on Computer and Communications Security*. ACM, 31–43.

[62] Hengshu Zhu, Hui Xiong, Yong Ge, and Enhong Chen. 2014. Mobile app recommendations with security and privacy awareness. In *KDD*. ACM, 951–960.

A ENTITLEMENTS

Table A.1: Sandbox entitlements that can be configured through Xcode

†Each key starts with `com.apple.security`.

‡These entitlements are boolean entitlements as well. However, their value can either be set to *Read Only* or to *Read/Write* in Xcode. Depending on the chosen value, the suffix `.read-only` or `.read-write` is added to the key. If the entitlement is not set (-), access to the resource is not granted.

Group	Key†	Xcode Label	Default Value
Network	`network.server`	Incoming Connections (Server)	-
	`network.client`	Outgoing Connections (Client)	-
Hardware	`device.camera`	Camera	-
	`device.microphone`	Microphone	-
	`device.usb`	USB	-
	`print`	Printing	-
	`device.bluetooth`	Bluetooth	-
App Data	`personal-information.addressbook`	Contacts	-
	`personal-information.location`	Location	-
	`personal-information.calendars`	Calendar	-
File Access‡	`files.user-selected`	User Selected Files	Read Only
	`files.downloads`	Downloads Folder	-
	`assets.pictures`	Pictures Folder	-
	`assets.music`	Music Folder	-
	`assets.movies`	Movies Folder	-

Table A.2: Hardened Runtime entitlements that can be configured through Xcode

†Each key starts with `com.apple.security`.

‡This is similar to the *Microphone* sandbox entitlement. Unsure whether there is a specific reason for not re-using the *Microphone* entitlement, as is done with the *App Data* sandbox entitlements.

*These keys are also in the set of Sandbox entitlements. If an unsandboxed but hardened app has no such entitlement, the permission dialog is not displayed to the user. Hence, the app cannot be added to the TCC database, which is required for accessing the resource.

★This is similar to the *Pictures Folder* sandbox entitlement. Instead of granting access to the user's `~/Pictures/` directory, this is required for showing a permission dialog to the user, if the app tries to access pictures from the *Photos* app through the `PhotoKit` API. This entitlement grants read and write access. The *Pictures Folder* in contrast, can be set to read access only as well.

Group	Key†	Xcode Label
Runtime Exceptions	`cs.allow-jit`	Allow Execution of JIT-compiled Code
	`cs.allow-unsigned-executable-memory`	Allow Unsigned Executable Memory
	`cs.allow-dyld-environment-variables`	Allow DYLD Environment Variables
	`cs.disable-library-validation`	Disable Library Validation
	`cs.disable-executable-page-protection`	Disable Executable Memory Protection
	`cs.debugger`	Debugging Tool
Resource Access	`device.audio-input`‡	Audio Input
	`device.camera`*	Camera
	`personal-information.location`*	Location
	`personal-information.addressbook`*	Address Book
	`personal-information.calendars`*	Calendar
	`personal-information.photos-library`★	Photos Library
	`automation.apple-events`	Apple Events

Table A.3: Temporary exception entitlements (not shown in Xcode)

†**Each key starts with `com.apple.security.temporary-exception.`**

‡**These entitlements are boolean entitlements as well. However, their value can either be set to *Read Only* or to *Read/Write* by appending the suffix `.read-only` or `.read-write` to the key.**

*__This allows to specify an arbitrary sandbox profile in the SBPL, which is not documented and presumably only allowed for a good reason. Only four third-party apps in our dataset make use of this entitlement. Mistakes in the SBPL configuration can render the sandbox protection ineffective, as was recently shown for Microsoft Office [57].__

Group	Key†	Name
	`apple-events`	Apple Events
	`audio-unit-host`	Audio Unit Hosting
	`mach-lookup.global-name`	Global Mach Service
	`mach-register.global-name`	Global Mach Service Dynamic Registration
	`iokit-user-client-class`	IOKit User Client Class
File Access‡	`files.home-relative-path`	Home Relative Path
	`files.absolute-path`	Absolute Path
Shared Preference Domain‡	`shared-preference`	Shared Preference Domain
Undocumented	`sbpl`*	SBPL