macOS Mojave gets new APIs around AppleEvent sandboxing – but AEpocalypse still looms ()

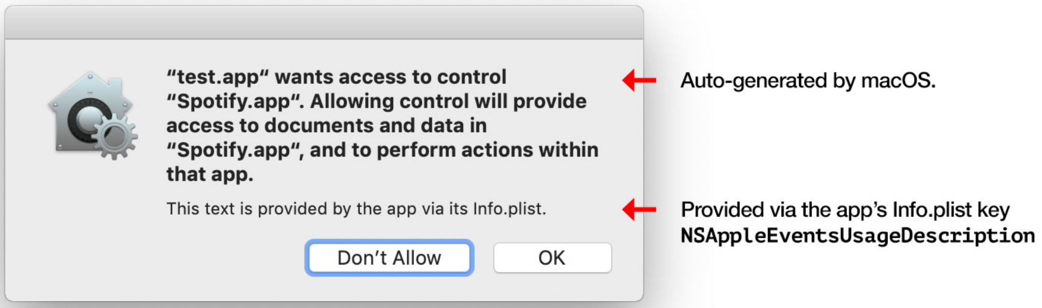
BY FELIX SCHWARZ (HTTPS://TWITTER.COM/INTENT/USER?SCREEN_NAME=FELIX_SCHWARZ) / AUG 30, 2018 / IN MACOS (/BLOG/CATEGORIES/MACOS), MAC (/BLOG/CATEGORIES/MAC), SECURITY (/BLOG/CATEGORIES/SECURITY), PRIVACY (/BLOG/CATEGORIES/PRIVACY)

AppleEvent sandboxing – which was announced in the WWDC 2018 session "Your Apps and the Future of macOS Security" (https://developer.apple.com/wwdc18/702) – affects every app that sends AppleEvents to other apps.

Back in June, I wrote that the new Apple Event sandboxing in macOS Mojave lacks essential APIs (/blog/2018/06/apple-event-sandboxing-in-macos-mojave) and highlighted problems with the implementation.

Since then, Apple has made a couple of changes to AppleEvent sandboxing.

Usage Descriptions for AppleEvents


"test.app" wants access to control "Spotify.app". Allowing control will provide access to documents and data in "Spotify.app", and to perform actions within that app.

This text is provided by the app via its Info.plist.

Don't Allow
OK
Auto-generated by macOS.
Provided via the app's Info.plist key NSAppleEventsUsageDescription

Usage descriptions provide context around an app's request for a particular permission. Typically, it is shown as part of authorization prompts that ask for permission to access sensitive areas like photos, camera, contacts or calenders.

Since macOS Mojave beta 6, a usage description for AppleEvents can (and often must) now be provided for the `NSAppleEventsUsageDescription` key in an app's Info.plist. Daniel Jalkut has an excellent blog post (https://indiestack.com/2018/08/apple-events-usage-description/) showing you how.

Caveats

- If you build your app with the 10.14 SDK, the `NSAppleEventsUsageDescription` key is **required**. If it is not provided, trying to send an AppleEvent to another, running app will always return error `errAEEventNotPermitted` (-1743). The key is only **optional** when using older SDKs.
- If you use AppleEvents in a helper app that's contained in your app's bundle, macOS may expect and use the value for the `NSAppleEventsUsageDescription` key in *the app's*

Info.plist, not the helper app's Info.plist.

- Apps can only provide one usage description. It is not possible to provide different usage descriptions for every target app. I still hope we'll get that capability at some point in the future.

Determining and prompting for user consent

Two new APIs now enable apps to time when the user is asked for consent and to determine the current authorization status.

AEDeterminePermissionToAutomateTarget

macOS Mojave beta 7 introduced `AEDeterminePermissionToAutomateTarget`, a new API that allows apps to determine whether they're authorized to send an AppleEvent to another app:

```
OSStatus AEDeterminePermissionToAutomateTarget( const AEAddressDesc* target,
                                                AEEventClass theAEEventClass,
                                                AEEventID theAEEventID,
                                                Boolean askUserIfNeeded );
```

The new function expects a reference to the target app to be provided as `AEAddressDesc`.

Permissions can be determined for a specific `AEEventClass` / `AEEventID` combination. Alternatively, passing `typeWildCard` for both checks if *every* AppleEvent can be sent to the target app. This distinction is important since certain AppleEvents don't require authorization from the user.

Finally, apps can choose to just check their current authorization status - or prompt the user to obtain permission as needed.

Here's a simple example that determines if the calling app currently has permission to send AppleEvents to iTunes - and prompts if not:

```
OSStatus status;
NSAppleEventDescriptor *targetAppEventDescriptor;

targetAppEventDescriptor = [NSAppleEventDescriptor descriptorWithBundleIdentifier:@"com.apple.iTunes"];

status = AEDeterminePermissionToAutomateTarget(targetAppEventDescriptor.aeDesc, typeWildCard, typeWildCard,
true);
```

Possible return values of `AEDeterminePermissionToAutomateTarget` include:

- `errAEEventNotPermitted` (-1743): the user has declined permission.
- `errAEEventWouldRequireUserConsent` (-1744): user consent is required for this, but the user has not yet been prompted for it. You need to pass `false` for `askUserIfNeeded` to get this.
- `noErr` (0): the app is authorized to send AppleEvents to the target app.
- `procNotFound` (-600): the specified target app is not currently running.

kAEDoNotPromptForUserConsent

Also new in macOS Mojave beta 7 is `kAEDoNotPromptForUserConsent`, a flag that can be added to the `AESendMode` that's passed to `AESend()`.

If the AppleEvent sent that way would require user consent, the user is not prompted for consent. Instead `AESend()` will return `errAEEventWouldRequireUserConsent`.

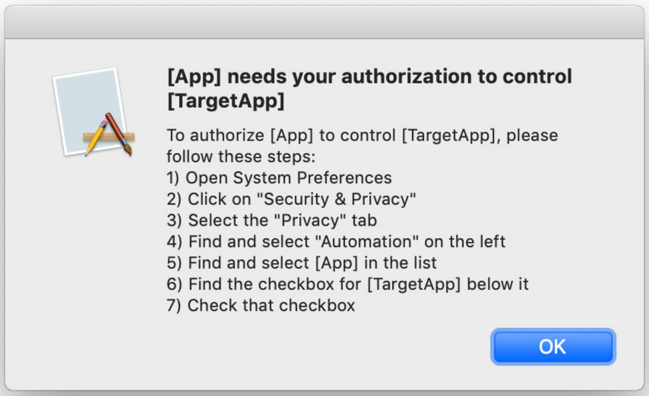
Caveats

- **target apps need to run:** to determine the current status or request authorization to send AppleEvents to another app, that app still needs to be running. Unfortunately, this limitation stands in the way of good and consistent user experiences:
  - **apps can't adapt their UI:** if an app wants to adapt its UI depending on whether it's authorized to send AppleEvents to a target app, it can't determine its current status unless that target app is running.

    A UI, however, that only dynamically adapts *sometimes* (= while the target app is running) is inconsistent and confusing.
  - **apps can't preauthorize:** apps that want to provide a good user experience around authorizations as part of onboarding would greatly benefit from the ability to prompt for authorization without having to launch the targeted app. It just isn't a good user experience having to.

    Unfortunately, for apps like mine that target a large number of apps and are meant to be useable without a keyboard, mouse or trackpad attached to the Mac, there are no realistic alternatives to preauthorization during onboarding. Right now that means launching each targeted app and then prompting the user for consent.

    Authorization "on-the-go" - prompting for authorization when target apps are used with them for the first time - is not an option here.
  - **caching results is not a solution:** users can grant or revoke their authorization to control another app in System Preferences at any time, so that any previously cached return values from `kAEDoNotPromptForUserConsent` no longer reflect the status quo, leading to inconsistencies in apps relying on them.
- **prompts can't be repeated:** macOS will show only one prompt per target app. If users decline an authorization in error, or change their minds later, the only way to do so is in a remote corner of System Preferences. The best possible user experience then looks somewhat like this:



[App] needs your authorization to control [TargetApp]

To authorize [App] to control [TargetApp], please follow these steps:
1) Open System Preferences
2) Click on "Security & Privacy"
3) Select the "Privacy" tab
4) Find and select "Automation" on the left
5) Find and select [App] in the list
6) Find the checkbox for [TargetApp] below it
7) Check that checkbox

Bringing up the prompt only once per target app definitely makes sense when the prompt is triggered *implicitly*, by way of sending an AppleEvent. Otherwise, apps unaware of AppleEvent sandboxing that keep sending AppleEvents to a target app could bring up the prompt very frequently.

However, when using `AEDeterminePermissionToAutomateTarget` to *explicitely* request the prompt, it should be possible to show the prompt more than once to avoid an unnecessarily bad user experience. If abuse of the ability to repeatedly show the prompt is a concern, a "Don't show this again" checkbox could be added to it to deal with that.

- **current status can't be determined without (risking to) prompt**: As of 10.14 beta 9, you need to pass `true` for `askUserIfNeeded` to get `errAEEventNotPermitted`, which can bring up the prompt.

  Passing `false`, `errAEEventWouldRequireUserConsent` is returned even if the user was previously prompted and declined permission.

  In consequence, even for running applications, `AEDeterminePermissionToAutomateTarget` can't be used to silently determine the current status. To be usable for that, it would need to return `errAEEventNotPermitted` in case consent is required *and* the user has already been prompted about it and denied. This feels like a bug. Paulo Andrade has filed a radar (http://www.openradar.me/radar?id=4945773766639616) about it.

- **calls can be blocking:** if `AEDeterminePermissionToAutomateTarget` prompts the user for authorization, it blocks the calling thread until the user has made a choice. If this is an issue in your app, consider moving the call to a different thread: `AEDeterminePermissionToAutomateTarget` is thread-safe.

- **usage description required:** if `AEDeterminePermissionToAutomateTarget` always returns `errAEEventNotPermitted` (-1743) even if you target apps for which no prompt has been shown so far, your Info.plist (or the Info.plist of the app containing the app) likely lacks a usage description for AppleEvents (see above for more on this).

- **documentation is hard to find:** at the time of writing, no documentation - or even a hint at the existence of the new API - exists in Apple's new documentation. Detailed documentation can, however, be found in the `AppleEvents.h` header file.

Configuration Profiles

Apple recently updated their Configuration Profile Reference (https://developer.apple.com/enterprise/documentation/Configuration-Profile-Reference.pdf), which now includes a *Privacy Preferences Policy Control Payload*.

Next to `AppleEvents`, it also allows providing an array of *Identity Dictionaries* for each of these *Services*: `AddressBook`, `Calendar`, `Reminders`, `Photos`, `Camera`, `Microphone`, `Accessibility`, `PostEvent`, `SystemPolicyAllFiles`, `SystemPolicySysAdminFiles`. That should cover pretty much all of the new sandboxing features in Mojave.

For `AppleEvents`, detailed information like the code requirements of the sender and the target app is required in the *Identity Dictionary*. Here's an example excerpt of a payload that would allow Remote Buddy to send AppleEvents to iTunes:

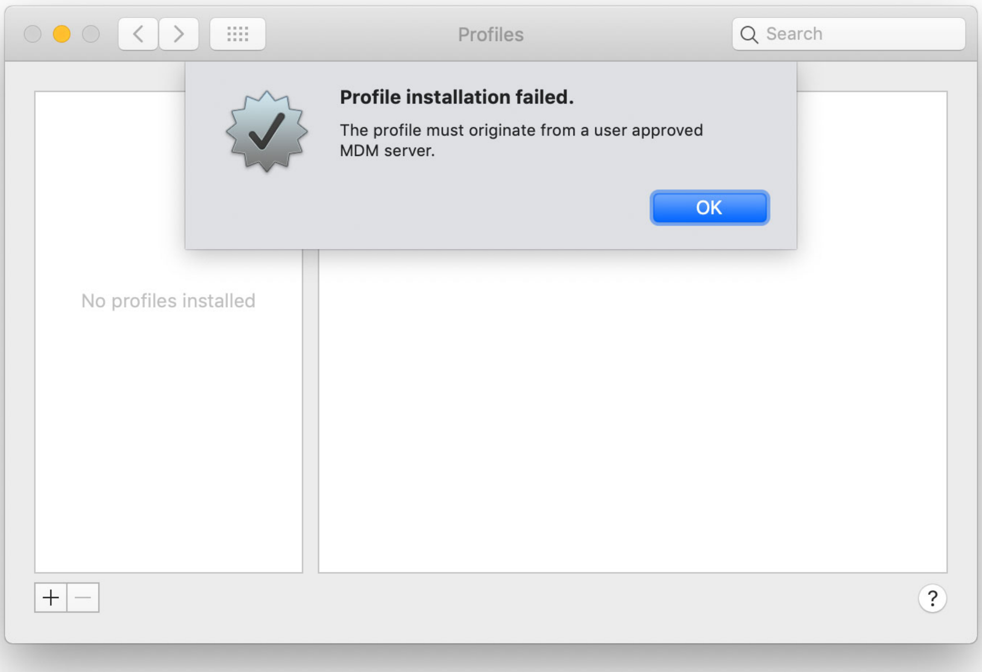
```
<key>PayloadType</key>
<string>com.apple.TCC.configuration-profile-policy</string>
<key>Services</key>
<dict>
        <key>AppleEvents</key>
        <array>
                <dict>
                        <key>Identifier</key>
                        <string>com.iospirit.remotebuddy</string>
                        <key>IdentifierType</key>
                        <string>bundleID</string>
                        <key>CodeRequirement</key>
                        <string>anchor apple generic and identifier "com.iospirit.remotebuddy" and
(certificate leaf[field.1.2.840.113635.100.6.1.9] /* exists */ or certificate
1[field.1.2.840.113635.100.6.2.6] /* exists */ and certificate leaf[field.1.2.840.113635.100.6.1.13] /*
exists */ and certificate leaf[subject.OU] = UH96K9N25C)</string>

                        <key>AEReceiverIdentifier</key>
                        <string>com.apple.iTunes</string>
                        <key>AEReceiverIdentifierType</key>
                        <string>bundleID</string>
                        <key>AEReceiverCodeRequirement</key>
                        <string>identifier "com.apple.iTunes" and anchor apple</string>

                        <key>Allowed</key>
                        <true/>
                </dict>
        </array>
</dict>
```

With no way to whitelist apps and no way to ask the user for authorization without the targeted apps running, building a configuration profile for users and saving them a ton of clicks and frustration is definitely a tempting idea. I've also had it.

However, profiles with this payload fail to install as they "*must originate from a user approved MDM server*":



Profile installation failed.

The profile must originate from a user approved MDM server.

Updated authorization prompts

When AppleEvent authorization prompts first appeared in beta 2, the title just read:

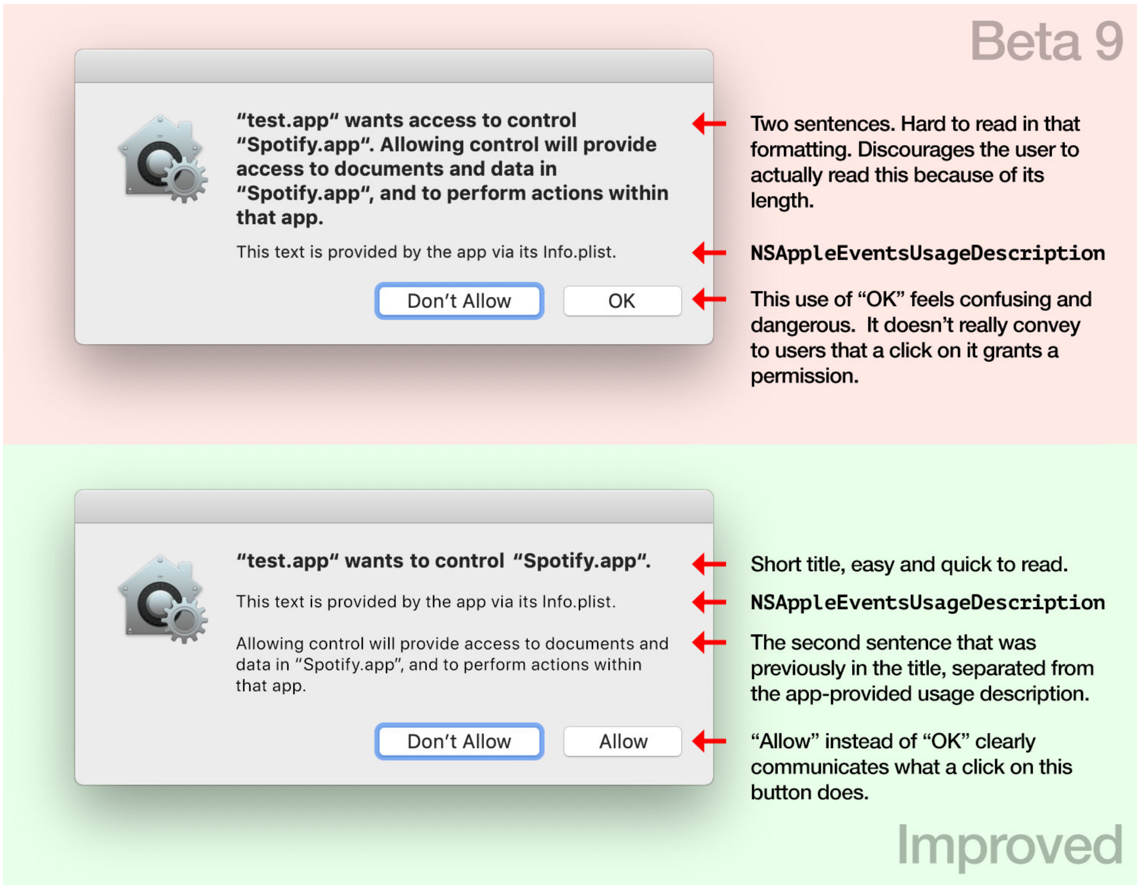
"Some.app" would like to control the application "Other.app".

In the current beta, the title is now a lot longer:

"Some.app" wants access to control "Other.app". Allowing control will provide access to documents and data in "Other.app", and to perform actions within that app.

The additional sentence provides valuable context to inform the decision of people who aren't as familiar with macOS security, but it makes the title way too long.

Below is an image that shows this and other issues I found with the current authorization prompt, as well as suggestions on how it could be improved.


Beta 9

"test.app" wants access to control "Spotify.app". Allowing control will provide access to documents and data in "Spotify.app", and to perform actions within that app.

This text is provided by the app via its Info.plist.

Two sentences. Hard to read in that formatting. Discourages the user to actually read this because of its length.

NSAppleEventsUsageDescription

This use of "OK" feels confusing and dangerous. It doesn't really convey to users that a click on it grants a permission.

"test.app" wants to control "Spotify.app".

This text is provided by the app via its Info.plist.

Allowing control will provide access to documents and data in "Spotify.app", and to perform actions within that app.

Short title, easy and quick to read.

NSAppleEventsUsageDescription

The second sentence that was previously in the title, separated from the app-provided usage description.

"Allow" instead of "OK" clearly communicates what a click on this button does.

Improved

I've filed a bug on this (Radar (rdar://43897386), Open Radar (https://openradar.appspot.com/radar?id=5063613442162688)).

Resetting the privacy database

Apple provides `tccutil` - a utility to manage the privacy database. To reset AppleEvent authorizations for all apps, use:

```
tccutil reset AppleEvents
```

Going by the man page, it should also be possible to reset the AppleEvent authorizations for a single app by appending its bundle identifier like this:

```
tccutil reset AppleEvents com.iospirit.remotebuddy
```

However, that command currently fails with an error, stating there is no such bundle identifier.

What's still missing?

Apple's willingness to act on the issues of the original AppleEvent sandboxing implementation (/blog/2018/06/apple-event-sandboxing-in-macos-mojave) in a matter of weeks deserves respect and my heartfelt thanks goes out to everyone at Apple who made that possible.

The new APIs solve the *most* pressing issues.

For many apps, however, it remains impossible to provide a user experience on par with that of previous macOS versions. Still missing for that:

- the ability *for users* to **whitelist apps** to exempt them from AppleEvent sandboxing - much like what "Full Disk Access" does for file system access. This is especially important for apps that target many apps - or an open-ended list of apps. Also, users of legacy software that's not updated for macOS Mojave may find themselves in need of a whitelisting option.
- **an improved version of `AEDeterminePermissionToAutomateTarget`** that can
    - determine the current authorization status and prompt for authorization **without requiring the target app to run**
    - show the authorization **prompt for a target app more than once**

AEpocalypse still looms

AppleEvents are a powerful and important IPC mechanism, differentiator and defining feature of the Mac. They're at the heart of AppleScript, Mac automation, utilities, accessibility tools and pro workflows.

The introduction of sandboxing will have a lasting impact on how AppleEvents work and how they can be used by apps. It has the potential to change the character of the technology as well as the Mac platform as a whole.

Summary of the current status

As of beta 9 - possibly the last beta before Mojave is likely released to the public in roughly a month - here's where things stand:

- **AppleEvent sandboxing still lacks essentials like whitelisting** or pre-authorization to target apps that are not currently running (see above)
- other than for a short segment in WWDC 2018 session 702 (https://developer.apple.com/wwdc18/702), **there's no documentation, guide, release note, tech note or transition guide** about AppleEvent sandboxing.
- **building an app with the macOS 10.14 SDK can break its use of AppleEvents**: AppleEvent and AppleScript APIs will return `errAEEventNotPermitted` - until a `NSAppleEventsUsageDescription` – which isn't mentioned or documented anywhere in Apple's documentation or headers – is added to the correct Info.plist.
- the **new APIs (http://codeworkshop.net/objc-diff/sdkdiffs/macos/10.14b6/CoreServices.html), added in beta 7**, are a relatively recent addition that many developers will find useful. They are, however, **not obvious to find** - and documentation for them is only available in the headers, but not Apple's developer documentation.

Practical impact

Control over the timing of authorization prompts is now possible, but still constrained.

No support for whitelisting apps means users of utilities that target many apps will not be able to escape a Vista-esque (https://www.youtube.com/watch?v=VuqZ8AqmLPY) "bombardment" with authorization prompts that keep interrupting their work. It's not hard to see that affected apps will get one-star reviews, loose users and sales - through no fault of their own.

No support for whitelisting apps also means there's no fallback for legacy apps that - for any reason - don't work well, or at all, with AppleEvent sandboxing.

The lack of documentation around AppleEvent sandboxing means developers either aren't aware of the changes - or can't know in time how they can prepare their apps for them.

Apple's options from here

AppleEvent sandboxing, as of Mojave beta 9, is not in a good shape. The addition of new APIs in beta 7 telegraphed that Apple is still working on it. But it's unclear what changes are still in the pipeline - and whether Apple can make enough progress before Mojave's public release.

I hope Apple can at least squeeze in support for whitelisting before public release as it's really a catch-all for issues with the new mechanism.

I feel, though, that the most responsible way for Apple to handle this situation would follow the playbooks for Group FaceTime and 32 Bit deprecation: postpone the feature (https://www.macrumors.com/2018/08/13/group-facetime-removed-ios-12-and-macos-mojave/) until it has matured - and make it available to developers behind a feature toggle (https://developer.apple.com/library/archive/releasenotes/General/RN-macOS-10.13.4/index.html#//apple_ref/doc/uid/TP40017702-CH1-DontLinkElementID_1) until then.

Ultimately, I'd like Apple to reconsider its approach when making changes like these to the foundation of macOS: introduce it at WWDC, but put it behind a feature toggle (https://developer.apple.com/library/archive/releasenotes/General/RN-macOS-10.13.4/index.html#//apple_ref/doc/uid/TP40017702-CH1-DontLinkElementID_1). Leverage the developer community. Enter into a dialogue to learn about unintended effects of the change, missing or bad APIs. Iterate. Make changes where needed. Provide comprehensive documentation well in advance. Then, at WWDC the year after, remove the feature toggle and make the change permanent.

Giving foundational changes like these a full year to mature – and developers to adapt their apps to them – would lead to all-around better results, less stress, anxiety and frustration for developers, a higher degree of stability and a better experience for users.

I'd like to thank Andreas Hegenberg (https://twitter.com/folivora_ai), Daniel Alm (https://twitter.com/daniel_a_a) and Jens Miltner (https://twitter.com/DebugStr) for sharing their discoveries and insights with me. This blog post would be far less comprehensive without them.

Updates

How to open System Preferences' Automation section

Sep 2, 2018

Using the `x-apple.systempreferences` URL scheme (https://macosxautomation.com/system-prefs-links.html), it's possible to open System Preferences right at "Security & Privacy > Privacy > Automation" by opening this URL:

`x-apple.systempreferences:com.apple.preference.security?Privacy_Automation (x-apple.systempreferences:com.apple.preference.security?Privacy_Automation)`

In code:

```
[[NSWorkspace sharedWorkspace] openURL:[NSURL URLWithString:@"x-apple.systempreferences:com.apple.preference.security?Privacy_Automation"]];
```

Thanks to Paulo Andrade (https://twitter.com/pfandrade_/status/1036028158150094848) and Shane Stanley for independently sharing this hint with me.

`AEDeterminePermissionToAutomateTarget` can't be used to "silently" determine authorization status – not even for running apps

Sep 2, 2018

As of 10.14 beta 9, you need to pass `true` for `askUserIfNeeded` to get `errAEEventNotPermitted`, which can bring up the prompt.

Passing `false`, `errAEEventWouldRequireUserConsent` is returned even if the user was previously prompted and declined permission.

In consequence, even for running applications, `AEDeterminePermissionToAutomateTarget` can't be used to silently determine the current status. To be usable for that, it would need to return `errAEEventNotPermitted` in case consent is required *and* the user has already been prompted about it and denied. This feels like a bug. Paulo Andrade has filed a radar (http://www.openradar.me/radar?id=4945773766639616) about it.

Thanks to Paulo Andrade (https://twitter.com/pfandrade_/status/1036028158150094848) for pointing this out to me.

Further reading

Sep 2, 2018

- AEDeterminePermissionToAutomateTarget Added, But AEpocalypse Still Looms (https://mjtsai.com/blog/2018/08/31/aedeterminepermissiontoautomatetarget-added-but-aepocalyse-still-looms/) – Michael J. Tsai's excellent mix of quotes, links and commentary on the topic
- Creating Privacy Preferences Policy Control profiles for macOS (https://derflounder.wordpress.com/2018/08/31/creating-privacy-preferences-policy-control-profiles-for-macos/) – Rich Trouton
- macOS Mojave: Opening New Vistas in Security for Mac Users (https://www.shirt-pocket.com/blog/index.php/shadedgrey/comments/macos_mojave_opening_new_vistas_in_security_for_mac_us – Dave Nanian, Shirt Pocket
- A looming Mac automation apocalypse? (https://sixcolors.com/link/2018/08/a-looping-mac-

automation-apocalypse) – Jason Snell, SixColors

PREVIOUS POST
Apple Event sandboxing in macOS Mojave lacks es...

Imprint (/imprint) | Privacy (/privacy)