

## STARTING THE SWITCH TO PYTHON 3

28 JUNE 2019

CATEGORIES: MAC OS

TAGS: PYTHON

With Python (and other scripting languages) being ~~dropped from~~ deprecated in macOS Catalina\*, and [Python 2.7 maintenance ending in 2020](#), it's time to start updating any Python 2 scripts so they are compatible with Python 3.

This might seem like a big task, and it could very well be for your code-base, however there are a few things you can do to start the process.

If you don't already have Python 3 installed on your Mac, you'll need to get it installed.

You can install it using `homebrew`, or you can install it from the [Python Releases for Mac OS X page](#). It's also provided by Apple, but currently you need an Apple Developer account to access those command line tool betas.

There's a great guide on what to do and what *not* to do when installing Python 3 on macOS [here](#).

No matter which way you get it installed, your Python 3 interpreter will be in a new location, and you'll definitely want to update `pip3` for package management.

You can also expect a lot, if not all of the "standard" packages that Apple included with their Python 2.7 framework to *not* be installed, so you'll need to make sure you've installed those as well. [This is a great opportunity to start using virtual environments](#).

For the process of migrating your code, there's a very handy [quick start guide](#), as well as a lot of in depth coverage about. There is a process to automatically update Python 2 code to Python 3 compatible code, which is [covered here](#).

Test all the changes made, particularly those made by `futurize`, to ensure that the scripts function as expected and that no "undocumented features" are present.

A few things that are important:

- A `homebrew` installation of Python 3 might install it to different locations as compared to installing Python 3 from the official packages.
- Python 3 won't necessarily end up in your path, so you'll either need to add it to your path, or set up some aliases to relevant binaries.
- Using `#!/usr/bin/env python` as the shebang in your script doesn't automatically mean a script will be executed with the Python 3 interpreter.
- Python 3 installed by the official Python installer is *not* code-signed or notarised.

\*Correction (2019-06-29): Python 2 is deprecated and will be removed in a future release of macOS. That could be a future dot release of macOS Catalina, or even later.

# PANDAS

29 MARCH 2019

CATEGORIES: PROGRAMMING

TAGS: PANDAS, PYTHON

Recently I've needed to wrangle a large data dump from a database system that manages to mangle exports of data in certain circumstances.

In this particular instance, the data exported out of the database had the `ItemID` missing from every 'record' after the first record exported for each unique `ItemID` (as per the example table below).

ItemID	Date	Content
012345	28/03/2019	Hello World.
	27/03/2019	The quick brown fox jumps over the lazy dog.
	26/03/2019	Lorem ipsum dolor sit amet, consectetur adipiscing elit.
543210	28/01/2018	Danish pastries are deliciously delightful and delicate.
	27/01/2018	Excepteur sint occaecat cupidatat non proident.
	26/01/2018	A lazy dog lets the quick brown fox jump over him.

Thankfully, this particular issue was fairly easy to fix with some very trivial python coding using the [pandas](#) package.

```
#!/usr/bin/python

# A quick thing to note about this solution, is that it has the
# potential to take quite some time on large++ datasets.
import pandas as pd

# Create a data frame from the export CSV file.
df = pd.read_csv('export.csv')

# Use a 'forward fill' to fill in the gaps for the 'ItemID' column in the
# data frame.
df.fillna(method='ffill', inplace=True)

# Convert the 'numpy.float64' type to a 'str' type.
# This is necessary in this particular instance.
# There may be a trailing '.0', so strip this out.
df = df.astype(str).replace('\.0', '', regex=True)
```

```
# Convert the date into ISO 8601 - the one true date format
df = pd.to_datetime(df, format='%d/%m/%Y')

# The date conversion will output the date in an Excel file
# as YYYY-mm-dd HH:MM:SS' - aka 2019-03-29 16:32:32
# If you only need the date, get rid of the time with this:
df = df.dt.date

# Save back out to an Excel file because this was the target file
# required.
write_xlsx = pd.ExcelWriter('Report_Sample.xlsx')

# By default, pandas Excel output includes the data frame index and
# header, this is not necessary in the target Excel document.
df.to_excel(write_xlsx, 'Sample', index=False, header=False)
write_xlsx.save()
```

The `pandas` package is one that often comes highly recommended for manipulating data sets by many `python` users, and in this simple scenario it is easy to see why this is the case. It certainly is possible to write something in `python` that would not have required the installation of a package, but there are many circumstances where packages like `pandas` are easier to use to manipulate data than having to write your own package/module.



# UPDATING BASH AND VIM

1 MARCH 2019

CATEGORIES: MAC OS

TAGS: MAC\_OS\_X

Armin Briegel's [post on updating the version](#) of `bash` on macOS to `bash 5.0` prompted me to create a repo for the scripts I use to keep `bash` and `vim` updated on my Mac's. You can find them in this [git repo](#).

Both update scripts require the macOS Command Line Tools installed. Additionally, there are some basic details about usage in each file.

I also dislike `homebrew`, so for other neat little binaries that I tend to use, I've found the [rudix package git repo](#) to be pretty handy (check the `ports` folder for macOS installer package files).

The tools I predominantly find myself using are:

- `mtr` - `ping` and `traceroute` in one neat binary
- `wget` - because sometimes `curl` is a pain
- `dos2unix` - those damn `^M` characters!
- `rsync` - this version includes support for some of the macOS specific attributes and what not
- `bash-completion` - because `tab complete` can always be better

Not all the `rudix` packages are compiled for the most recent version of macOS, but in some cases they'll still work just fine.

# TCC IN MOJAVE SLIDES FROM BAW MEETUP

23 NOVEMBER 2018

CATEGORIES: MAC OS

TAGS: BRISBANE APPLE WRANGLERS, DEP, JAMF, MAC, MACADMINS, MACOS, MACOS  
10.14, MEETUP

The Brisbane Apple Wranglers met last night, pizza and chilled beverages on hand. There was a presentation from JAMF (thanks for the sponsorship for the night) about the recent 10.8 release, and the upcoming 10.9 release, a presentation from Ryan on his organisations Mac on-boarding process for their students with the JAMF 10.9 beta, and I gave a run down on TCC in Mojave and the new Privacy Preferences Policy Control Profiles.

Thanks to @locee for organising the events, looking forward to the next one. If you're in Brisbane or South East QLD and want to join, head on over to the [Brisbane Apple Wranglers Meetup](#) to signup.

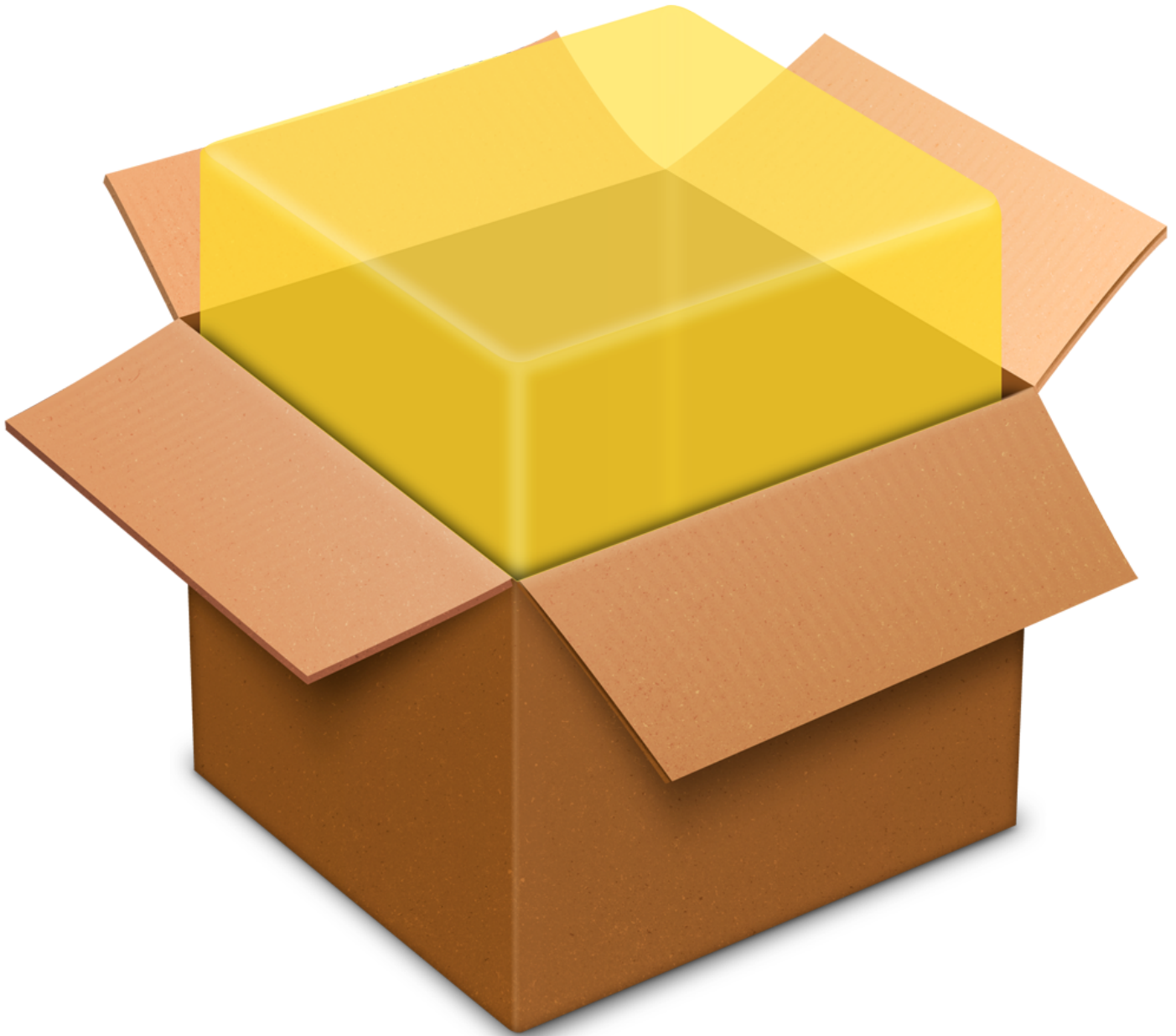
Slides from my talk on TCC in Mojave are available [here](#).

# PRESERVING XATTR WITH PKGBUILD

3 NOVEMBER 2018

CATEGORIES: MAC OS

TAGS: CODE SIGNING, MACOS, PACKAGING, PRIVACY PREFERENCES



If you're deploying a script or other file (text, image, etc) that is code signed, you will need to make sure the code sign requirements are kept ([see here as to why](#)).

If you use `pkgbuild`, you'll need to make sure you include the `--preserve-xattr` argument in the build options. This argument is undocumented, and may not be supported in older versions of `pkgbuild`.

Note, If you use the Packages app to build Mac `pkg` files, it doesn't preserve extended attributes when it builds a `pkg` file, so other work arounds will be required if you're using that to build your software packages.

Using [outset](#) as an example of building a package with `pkgbuild` and preserving the extended attributes, here's a quick example of modifying the `Makefile` to include this argument.

In the block below, the repo is cloned, then `sed` is used to modify the line containing `pkgbuild`, adding the `--preserve-xattr` argument after the `--ownership recommended` argument, `git diff` is then run to verify the change.

```
:git # git clone https://github.com/chilcote/outset
Cloning into 'outset'...
remote: Enumerating objects: 515, done.
remote: Total 515 (delta 0), reused 0 (delta 0), pack-reused 515
Receiving objects: 100% (515/515), 273.71 KiB | 479.00 KiB/s, done.
Resolving deltas: 100% (226/226), done.
:git # cd outset/
:outset # sed -i Makefile.backup '/pkgbuild/s/ownership recommended/ownership recommended
--preserve-xattr/g' Makefile
:outset # git diff Makefile
:outset # git diff Makefile
diff --git a/Makefile b/Makefile
index 7decb72..872e583 100644
--- a/Makefile
+++ b/Makefile
@@ -16,7 +16,7 @@ clean:
    ## pkg - Create a package using pkgbuild pkg: clean
-    pkgbuild --root pkgroot --scripts scripts --identifier ${PKGID} --version ${PKGVE
RSION} --ownership recommended ./${PKGTITLE}-${PKGVERSION}.component.pkg
+    pkgbuild --root pkgroot --scripts scripts --identifier ${PKGID} --version ${PKGVE
RSION} --ownership recommended --preserve-xattr ./${PKGTITLE}-${PKGVERSION}.component.pkg
    productbuild --identifier ${PKGID}.${PKGVERSION} --package ./${PKGTITLE}-${PKGVER
SION}.component.pkg ./${PKGTITLE}-${PKGVERSION}.pkg
    rm -f ./${PKGTITLE}-${PKGVERSION}.component.pkg
:outset #
```

**Note:** this is an example only that's fairly specific to modifying the `outset Makefile` for repeatable builds that preserve the extended attributes. If you want to preserve extended attributes for other files when your packages are built, you may need to manually modify the `Makefile` or adjust your command line `pkgbuild` arguments to include the `--preserve-xattr` command.

Greg Neagle has [added this flag](#) into the `munki-pkg` utility.

To code sign, an example is provided below.

In this example, the `codesign` command is used with the relevant certificate details. The `-i` flag is used to specify an identifier for the code sign requirements. This is not a required argument, but

if it isn't supplied, `codesign` will use the filename to determine the identifier. Lastly, the path to the object being signed is provided.

The `xattr` command is used to demonstrate that for plain text files such as this, the code sign requirements are actually written out as extended attributes.

The `codesign -dr - /path/to/file` command outputs the code sign requirements for the file just signed.

```
:outset # codesign -s "Mac Developer: jappleseed@example.org (QED00ABDEC)" -i com.github.outset
outset pkgroot/usr/local/outset/outset
:outset # xattr pkgroot/usr/local/outset/outset
com.apple.cs.CodeDirectory
com.apple.cs.CodeRequirements
com.apple.cs.CodeRequirements-1
com.apple.cs.CodeSignature
:outset # codesign -dr - pkgroot/usr/local/outset/outset
Executable=/Users/jappleseed/Desktop/git/outset/pkgroot/usr/local/outset/outset
host => identifier "com.apple.pythonw" and anchor apple
designated => identifier "com.github.outset" and anchor apple generic and certificate leaf = "Mac Developer: jappleseed@example.org (QED00ABDEC)" and certificate 1 /* exists */
:outset #
```

From here, you can follow your normal build process.

In the case of this outset example it's simply a matter of running `make pkg`.

```
:outset # make pkg
rm -f ./outset*.{dmg,pkg}
rm -f ./pkgroot/usr/local/outset/FoundationPlist/*.pyc
pkgbuild --root pkgroot --scripts scripts --identifier com.github.outset --version "2.0.6" --ownership recommended --preserve-xattr ./"outset"-2.0.6.component.pkg
pkgbuild: Inferring bundle components from contents of pkgroot
pkgbuild: Adding top-level postinstall script
pkgbuild: Wrote package to ./outset-2.0.6.component.pkg
productbuild --identifier com.github.outset."2.0.6" --package ./"outset"-2.0.6.component.pkg ./"outset"-2.0.6.pkg
productbuild: Wrote product to ./outset-2.0.6.pkg
rm -f ./"outset"-2.0.6.component.pkg
```

**Note:** these `Makefile` changes will be removed if the `outset` repo is updated, and this will preserve *all* extended attributes for files in the build process, so you may want to remove *specific* extended attributes such as `com.apple.quarantine` attributes.

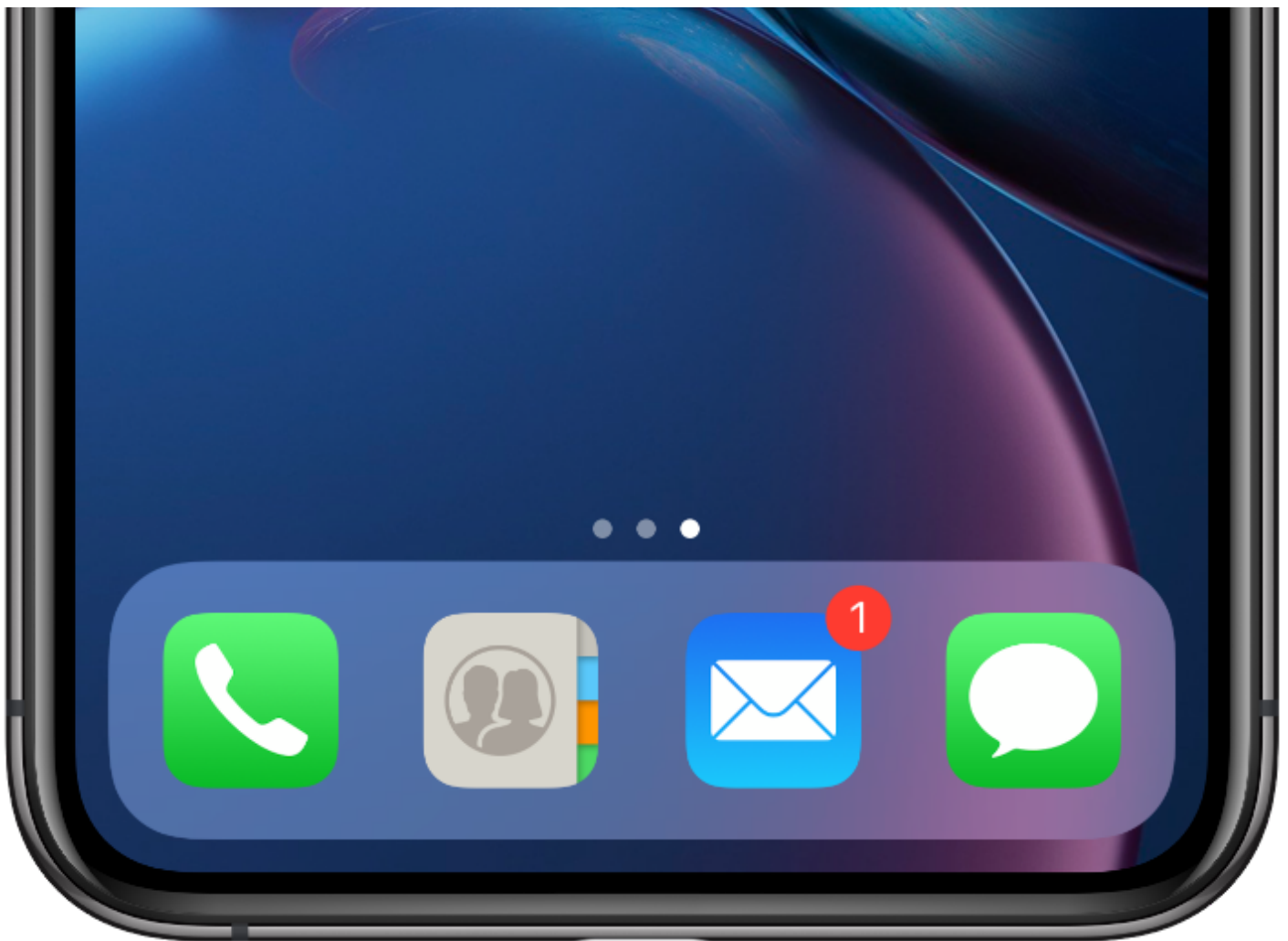
# IPHONE OVERLAY ON SCREEN RECORDINGS

21 OCTOBER 2018

CATEGORIES: IOS

TAGS: DEVICE\_FRAME, FFMPEG, IOS, IPHONE XS MAX, IPHONEOVERLAY.PY, PYTHON





This script was inspired by an [iOS Shortcut](#) – taking a device frame and overlaying it on an image.

[iphoneoverlay.py](#) will take an iOS screen recording (currently only the iPhone XS Max is supported), and overlay the hardware frame (device frame) onto the recording.





## Requirements

- Tested on `python 2.7.10` on macOS, and with `ffmpeg 4.0.2-tessus` for macOS
- `ffmpeg` must be installed in `/usr/local/bin` – available from <https://www.ffmpeg.org/download.html>

## Usage



# Clone

```
:Documents # git clone https://github.com/carlashley/iphone_overlay
:Documents # cd iphone_overlay
:Documents # chmod +x iphoneoverlay.py
```

## View Help

```
:iphone_overlay # ./iphoneoverlay.py -h
usage: iphoneoverlay.py  -i
                        --overlay
```

optional arguments:

-h, --help	show this help message and exit
-a, --keep-audio	Keep audio.
-i, --input	Screen recording to add device frame to.
-o, --output	Destination video filename.
-c, --bg-colour "#ffffff"	Background colour. If specifying RGB code, quote the code. For example: "#ffffff"
--overlay	Device frame to use as overlay.
--orientation	Orientation of final video. Defaults to portrait.
-d, --debug	Debug output.
-v, --version	show program's version number and exit

iOS screen recordings will need to be saved to your Mac in order to create the overlay.

## Usage Examples

### Required arguments

Both `-i, --input` and `--overlay` are required arguments.

### Basic run with only input video file and overlay

Resizing source video to match device frame iPhone-XS-Max-Portrait-Space-Gray.png image size in portrait orientation and applying overlay.

```
:iphone_overlay # ./iphoneoverlay.py -i ~/Downloads/Portrait.mp4 --overlay iphoneXSmaxRes
izing source video to match device frame iPhone-XS-Max-Portrait-Space-Gray.png image size
in portrait orientation.
frame= 490 fps= 45 q=-1.0 Lsize=      680kB time=00:00:08.11 bitrate= 686.1kbits/s dup=2
```

```
drop=0 speed=0.746x
Video saved to: /Users/jappleseed/Downloads/Portrait_overlay.mp4
```

Specify input video file, overlay, and background colour, no output filename

```
:iphone_overlay # ./iphoneoverlay.py -i ~/Downloads/Portrait.mp4 --overlay iphoneXSmax --
bg-colour="#b00d23"
Resizing source video to match device frame iPhone-XS-Max-Portrait-Space-Gray.png image s
ize in portrait orientation and applying overlay.
frame= 490 fps= 45 q=-1.0 Lsize=      693kB time=00:00:08.11 bitrate= 699.2kbits/s dup=2
drop=0 speed=0.752x
Video saved to: /Users/jappleseed/Downloads/Portrait_overlay.mp4
```

Specify input video file, output video file, background colour, overlay, and orientation

```
:iphone_overlay # ./iphoneoverlay.py -i ~/Downloads/Portrait.mp4 -o HelloWorld_Landscape.
mp4 --bg-colour="#ffffff" --overlay iphoneXSmax --orientation landscape
Resizing source video to match device frame iPhone-XS-Max-Portrait-Space-Gray.png image s
ize in landscape orientation and applying overlay.
frame= 490 fps= 55 q=-1.0 Lsize=      703kB time=00:00:08.11 bitrate= 709.5kbits/s dup=2
drop=0 speed=0.912x
Video saved to: /Users/jappleseed/Documents/git/Portrait_overlay.mp4
```

## Demo





# Limitations

- Currently only works for video recorded on an iPhone XS Max.

# TCC ROUND UP

28 SEPTEMBER 2018

CATEGORIES: MAC OS

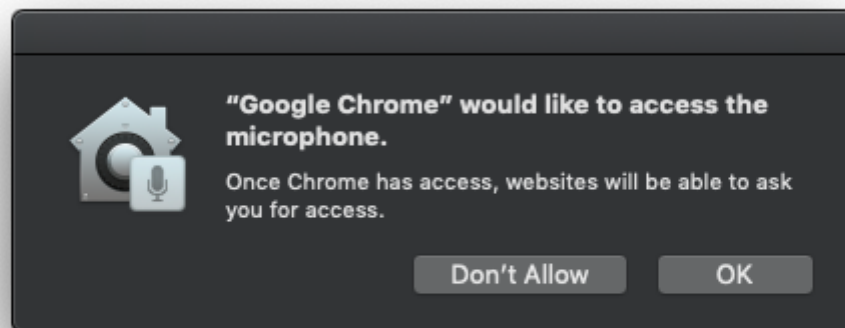
TAGS: CODE SIGNING, MAC OS, MACOS, MACOS 10.14, MACOS MOJAVE, MDMD, MOJAVE, PPPCP, PRIVACY PREFERENCES, PROFILES

Now that macOS Mojave has been released into the wild, a quick recap of changes relating to Transparency, Consent, and Control (TCC).

## What changes?

In certain circumstances, the first time you run an app, you may be prompted to allow it to access data such as your contacts, or maybe hardware such as the camera, or possibly even control your computer.

This doesn't have to be the first time you run an app either. For example, a web browser such as Google Chrome won't access your camera or microphone, but at some point it may, and when it does, that is when you'll see the prompt.



Additionally, if the user doesn't acknowledge the consent prompt, by clicking either of the buttons, and leaves it alone, the consent prompt will eventually disappear after a certain "time out" period, it will also assume that the user does not want to provide consent.

Worse, a typical schedule runs when the user isn't even present, and so the prompts go without response, and the events time out.

Worse still, a timeout (the system defaults to two minutes) doesn't re-prompt, but assumes the answer is "no".

*Dave Nanian @ Shirt Pocket*

## Can it be disabled?

Unlike System Integrity Protection (SIP), these protections cannot be disabled.

## Accessing protected folders from the command line.

You will no longer be able to access certain protected folders in user profiles that contain protected information.

To access these folders, or other TCC protected folder locations, you need to add the Terminal (or whatever your preferred terminal app is) to the Full Disk Access section of the Privacy tab in the Security & Privacy preference pane, or by using a Privacy Preferences Policy Control Payload (PPPCP for short).



A list of protected locations collated by a number of macadmins members (check the pinned items in #tcc for the Google Sheet created by @erik):

- /Users/username/Library/Application Support/CallHistoryTransactions
- /Users/username/Library/Application Support/com.apple.TCC
- /Users/username/Library/Application Support/AddressBook
- /Users/username/Library/Application Support/CallHistoryDB
- /Users/username/Library/IdentityServices

- /Users/username/Library/Calendars
- /Users/username/Library/Preferences/com.apple.AddressBook.plist
- /Users/username/Library/Messages
- /Users/username/Library/Mail
- /Users/username/Library/Safari
- /Users/username/Library/Suggestions
- /Users/username/Library/Containers/com.apple.Safari
- /Users/username/Library/PersonalizationPortrait
- /Users/username/Library/Metadata/CoreSpotlight
- /Users/username/Library/Cookies
- /Users/username/Library/Caches/CloudKit/com.apple.Safari
- /private/var/db/dslocal/nodes/

## How do I manage it?

Apple have provided new MDM [Configuration Profile](#) payloads.

Profiles for PPPCP can be built by using my [tccprofile.py](#) tool, Erik Burglund's [Profile Creator](#), JAMF's [PPPC-Utility](#), or by artisinally hand crafting them (though you're likely to drive yourself mad if you do this).

You will *absolutely* need an MDM to deploy these profiles as they cannot be deployed direct to a machine through a package, or other installation method.

This will mean either a DEP to MDM enrolment workflow, or through User Approved MDM (users manually enrol their Mac into MDM).

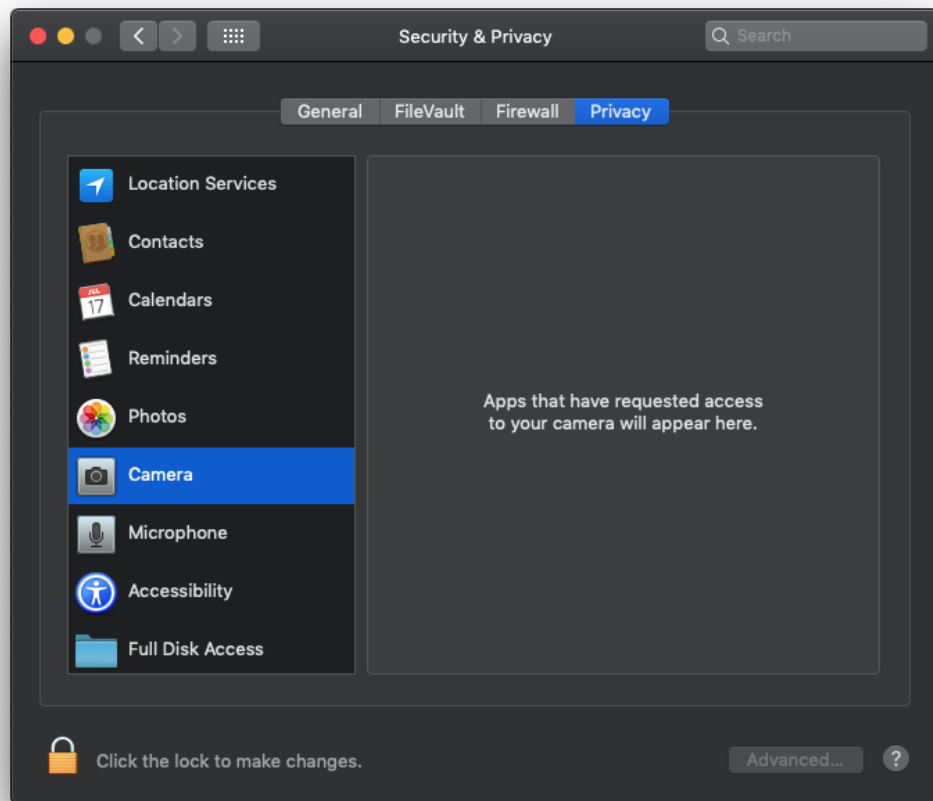
Apple introduced the concept of User Approved MDM in macOS High Sierra. It's here to stay, and we will see more profiles require this in the future.

Rich Trouton has a [git repo](#) with a number of profiles for various applications, and [JAMF has a profile](#) that has been prepared for deployment in JAMF environments to pre-approve their binaries.

## Why can't I remove items from Microphone, Camera, etc?

Not all items can be removed from the relevant privacy setting.

For example, apps that are allowed or have blocked access to the camera or microphone can only have their check box ticked or unticked. It's also not possible to add apps to the Camera or Microphone preferences directly, and any PPPCP profile can only explicitly deny an app access to the camera or microphone.



You can however reset the TCC database (both the user and system database) by using the `tccutil` tool. However, this will require providing Full Folder Access to the Terminal app (or terminal app of your choice).

```
@pegasus: /Users/ /Desktop/git/profilr [tty002] — less · man tccutil

tccutil(1)          BSD General Commands Manual          tccutil(1)

NAME
  tccutil -- manage the privacy database

SYNOPSIS
  tccutil command service [bundle_id]

DESCRIPTION
  The tccutil command manages the privacy database, which stores decisions the user has made about whether apps may access personal data.
  One command is currently supported:

  reset    Reset all decisions for the specified service, causing apps to prompt again the next time they access the service. If a bundle identifier is specified, the service will be reset for that bundle only.

EXAMPLES
  To reset all decisions about whether apps may access the address book:

  tccutil reset AddressBook
  tccutil reset All com.apple.Terminal

Darwin               April 3, 2012               Darwin
```

Haircut from the macadmins slack has [this great gist](#) to reset it with a simple python script, as well as [this great write up](#) on making it a Self Serve item in JAMF.

## How will I know what to approve?

You will need to test existing management scripts and the applications/tools that get deployed to your Mac fleet. Not everything requires approval, but there are apps that will surprise you.



To help understand what you need to approve, you can parse the TCC log stream (just don't cross the streams).

[illegible]

Importantly, you can *only* approve apps or scripts that have been code signed.

It can't be that simple?

Unfortunately, this is true. It *isn't* all that simple.

Over the last few weeks as this has been tested out by members of the macadmins slack ([join up if you haven't already!](#)), there have been many discoveries about apps that ask for access to control themselves, or scenarios where the parent process isn't identified correctly in the TCC User Consent dialog.

There are also scenarios where if a python script is called from a Launch Daemon or Agent, the user will be prompted to allow the specific script control of the computer or access to protected data (depending on what is being processed).

PPPCP profiles should not be installed on a Mac that isn't running macOS Mojave 10.14 as they will not be applied after upgrading to macOS Mojave 10.14\*. Check your MDM to see how you can “scope” or “tag” profiles/devices to avoid deploying PPPCP profiles to Mac hardware that isn't running macOS Mojave, or to Mac hardware that is running macOS Mojave.

**\*Note:** This highlights a need for Apple to implement either a “check in with MDM after upgrade” routine, or a means by which a Mac system can be told to check in with an MDM.

## Should I approve shells or interpreters?

Generally speaking, it's best to avoid being too general with what is approved in a PPPCP profile. For example, if you approve `/usr/bin/python` in a PPPCP profile to control the Mac, then *any* `python` script that runs code to control your computer will be able to do that.

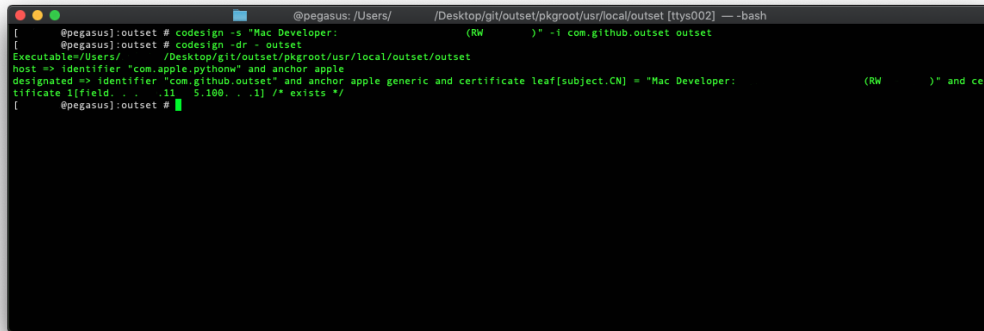
If at all possible, only whitelist what needs to be approved.

## Can I approve scripts?

Yes you can, however you will need to make sure that the script/tool being approved is code signed, and you will need to create a PPPCP profile for that script.

Code signed scripts have the code signing details stored in extended attributes ( `xattr` ). When you deploy the script, these attributes must be preserved when put onto the target Mac.

You will also need a code signing certificate in order to code sign scripts, software, etc. Apple provides more information about code signing here.

A terminal window with a dark background and light green text. The prompt is '@pegasus:~/outset'. The user enters 'codesign -s "Mac Developer: (RW)" -i com.github.outset outset'. The terminal shows the execution of the code signing command, including the path to the executable and the identifier 'com.apple.pythonw'. The command completes successfully, and the prompt returns to '@pegasus:~/outset #'.

```
@pegasus:~/outset # codesign -s "Mac Developer: (RW)" -i com.github.outset outset
[
@pegasus:~/outset # codesign -dr - outset
Executable/Users/~/Desktop/git/outset/pkgroot/usr/local/outset/outset
host => identifier "com.apple.pythonw" and anchor apple
designated => identifier "com.github.outset" and anchor apple generic and certificate leaf[subject.CN] = "Mac Developer: (RW)" and cer
tificate ifield. . . . 11 5.100. . . 1] /* exists */
[
@pegasus:~/outset #
```

## Can I code sign third party apps that aren't code signed?

Technically yes, however it is strongly recommended that if at all possible, rely on the developer to code sign the app.

Code signing is a way of indicating the app has not been changed since it was signed, identify where the app came from or who signed it, and if the app is trustworthy.

By code signing an app that you have not created or are the developer of, you effectively take responsibility for making sure it does not act maliciously, etc.

## Should I create a single profile, or multiple profiles?

It's ultimately a decision to make based on the environment that these are deployed in.

The granularity of multiple profiles is a good thing, but you do have to be mindful of how profiles are applied on a macOS system. If there are conflicting payloads of the same type, then the most restrictive payload wins.

Creating a large profile with multiple payloads in it will minimise the chance of having conflicting payloads in multiple profiles, however it will be more cumbersome to make changes to specific payloads, especially when the profile applies to multiple machines that may have different needs.

PPPCP profiles have been deployed, but nothing shows up in the Security & Privacy preferences pane.

This seems to be by design (presently), but does not help the user or administrator know if a profile is in place or deployed correctly.

The only time an app will show up in the Security & Privacy preference pane, is when it has been approved directly by the user.

If this is something that you believe should be changed, then submit feedback/RADARs to Apple.

An app that used to work fine in macOS High Sierra, no longer works, and randomly crashes.

This may be because the app hasn't been compiled/built properly for macOS Mojave 10.14.

[This blog post](#) goes into more detail about this issue.

## What can I do to avoid triggering a TCC User Consent dialog?

For third party apps that are out of your control, the only course you can take is to send feedback to the app developer. In some cases, they will be able to re-write code to avoid code that will trigger these consent dialogs, but in some cases that won't be possible.

If you are writing your own scripts for managing macOS, or just to make your life easier, then you may need to re-write your scripts to avoid things that will trigger a consent dialog. You will need to test your scripts to find out if it will trigger a consent dialog.

This is still all confusing.

The [macadmins slack](#) is a fantastic resource for Mac admins, especially for those that are one man shops. Join the #tcc channel and join in the discussions there.

## Feedback.

The best way that we can affect change is by submitting feedback to Apple through either the Apple Seed program or the Developer program.

If you're not in the Apple Seed program, then reach out to your Apple SE and ask for an invite to join in.

Provide thorough details of the issue, steps to replicate, screenshots, videos, etc, and include impact statements that describe what the impact is to IT staff, end users, as well as the number of

devices this applies to.

Joining the Developer program is also a good idea if you can't join the Apple Seed program.

# CODE SIGNING SCRIPTS FOR PPC WHITELISTING

23 SEPTEMBER 2018

CATEGORIES: MAC OS

TAGS: MAC OS, MACOS, MACOS 10.14, PRIVACY PREFERENCES, PROFILES, TCC

One of the issues with creating a Privacy Preferences Policy Control Payload (PPPCP) profiles is working out whether you will whitelist an entire shell or interpreter (for example, `/bin/bash` or `/usr/bin/python`) or go down the route of code signing scripts that trigger a TCC User Consent dialog.

On the face of it, whitelisting an entire shell or interpreter seems like the easiest way to handle this situation, especially if you deploy a tool like `outset`. However, there is the possibility that some malicious app might drop a `LaunchDaemon` or `LaunchAgent` onto a macOS system that executes a shell script, and suddenly, you've got a script that is essentially allowed to execute any action that might ordinarily have prompted a consent dialog.

The alternative is to be diligent with what gets whitelisted, and rather than whitelisting `/usr/bin/python` (as an example), you may choose to code sign your scripts and generate PPPCP profiles to whitelist those scripts. This would allow you to be somewhat more granular in what is pre-approved for running on a system without generating consent dialogs.

## *However.*

Code signing a script does not work in the same way that code signing an app bundle does.

When an app bundle is code signed, the details/requirements are put in a folder called `_CodeSignature`.

When a plain text file is code signed, the signature ends up in an extended attribute, specifically four different attributes.

```
:outset # codesign -s "Mac Developer: foo@example.org (ABC01FFFGH)" -i com.github.outset
outset
:outset # ls -lha
total 8
drwxr-xr-x  4 carl  staff   128B 23 Sep 12:53 .
drwxr-xr-x  4 carl  staff   128B 23 Sep 12:30 ..
drwxr-xr-x  4 carl  staff   128B 23 Sep 12:09 FoundationPlist
-rwxr-xr-x@ 1 carl  staff   1.0K 23 Sep 12:20 outset
:outset # xattr outset
com.apple.cs.CodeDirectory
com.apple.cs.CodeRequirements
com.apple.cs.CodeRequirements-1
com.apple.cs.CodeSignature
```

```
:outset # codesign -dr - outset
Executable=/Users/carl/Desktop/git/outset/pkgroot/usr/local/outset/outset
designated => identifier "com.github.outset" and anchor apple generic and certificate lea
f = "Mac Developer: foo@example.org (ABC01FFFGH)" and certificate 1 /* exists */
```

This presents no problem as long as the code signed file stays where it is, or is moved around by tools that keep the extended attributes attached to the file (such as `mv`, `cp`, or `rsync` with appropriate flags, or even certain compressed file formats).

It *is* a problem if you need to build a package to deploy the file to another machine.

So far in testing, I've found that any package built with Packages, or `pkgbuild` / `productbuild`, the extended attributes for files are stripped when the package is built, and it seems there are no command line arguments for `pkgbuild` / `productbuild` to keep the extended attributes (update: this is not strictly true, see the update below). The Packages app also strips extended attributes when building a package (including when a file is embedded as a resource for the installer).

## Work around

---

Update: [Greg Neagle replied to a tweet](#) about the issue of the extended attributes missing after building a package. If you use `pkgbuild`, simply add the `--preserve-xattr` flag to the `pkgbuild` command to preserve the extended attributes.

This is an undocumented flag/feature, so older versions of `pkgbuild` may not support it. This was tested on macOS Mojave 10.14, with Xcode 10 installed.

Once I know how to do the same thing with Packages, I'll be sure to update, other wise, the work around below will work in a pinch.

---

To get around this, there are several compressed file formats that preserve the extended attributes of a file, one of which is the handy `tar.gz` file type.

In this example, the [outset git repo](#) has been cloned and changes are being made to the files within that directory. This also presumes you are familiar with the [Packages](#) application and building a package with it.

```
:~ # cd ~/Desktop/git
:git # git clone https://github.com/chilcote/outset
Cloning into 'outset'...
remote: Enumerating objects: 16, done.
remote: Counting objects: 100% (16/16), done.
remote: Compressing objects: 100% (14/14), done.
remote: Total 531 (delta 4), reused 8 (delta 0), pack-reused 515
```

```
Receiving objects: 100% (531/531), 281.03 KiB | 456.00 KiB/s, done.  
Resolving deltas: 100% (230/230), done.
```

Code sign the `/usr/local/outset/outset` script:

```
:outset # codesign -s "Mac Developer: foo@example.org (ABC01FFFGH)" -i com.github.outset  
outset
```

If you want to verify the `codesign` result:

```
:outset # codesign -dr - outset  
Executable=/Users/carl/Desktop/git/outset/pkgroot/usr/local/outset/outset  
designated => identifier "com.github.outset" and anchor apple generic and certificate lea  
f = "Mac Developer: foo@example.org (ABC01FFFGH)" and certificate 1 /* exists */
```

Next, a tarball needs to be created containing *just* the script that has been code signed:

```
:outset # pwd  
/Users/carl/Desktop/git/outset  
:outset # cd pkgroot/usr/local/outset/  
:outset # tar -cvf outset.tar.gz outset  
a outset  
:outset # ls -l  
total 32  
drwxr-xr-x  4 carl  staff   128 23 Sep 12:09 FoundationPlist  
-rwxr-xr-x@ 1 carl  staff  1024 23 Sep 12:20 outset  
-rw-r--r--  1 carl  staff  8704 23 Sep 13:49 outset.tar.gz
```

From here, include the `outset.tar.gz` file as a resource in a Packages project that replicates the `outset` installer, and modify the `postinstall` script that is used by the normal `outset` package to look like this:

```
#!/bin/bash  
  
#reference: https://github.com/google/macops/blob/master/keychainminder/Package/postinsta  
ll  
  
resources=$(dirname $0)  
target_vol=$3  
package_bundle_id=$INSTALL_PKG_SESSION_ID  
  
] && exit 0  
  
/bin/launchctl load /Library/LaunchDaemons/com.github.outset.boot.plist  
/bin/launchctl load /Library/LaunchDaemons/com.github.outset.cleanup.plist  
/bin/launchctl load /Library/LaunchDaemons/com.github.outset.login-privileged.plist
```

```

user=$( /usr/bin/stat -f '%u' /dev/console)
] && exit 0
/bin/launchctl asuser ${user} /bin/launchctl load /Library/LaunchAgents/com.github.outset.login.plist
/bin/launchctl asuser ${user} /bin/launchctl load /Library/LaunchAgents/com.github.outset.on-demand.plist

/usr/bin/tar -xpf ${resources}/outset.tar.gz -C /usr/local/outset/
/usr/sbin/chown root:wheel /usr/local/outset/outset

exit 0

```

The additional lines in the script are ( ... denotes code from the original script):

```

...
resources=$(dirname $0)
target_vol=$3
package_bundle_id=$INSTALL_PKG_SESSION_ID
...
/usr/bin/tar -xpf ${resources}/outset.tar.gz -C /usr/local/outset/
/usr/sbin/chown root:wheel /usr/local/outset/outset
...

```

Make sure the `postinstall` script is included in the Packages file, build and test. If all has gone correctly, running `codesign -dr - /usr/local/outset/outset` should result in the code sign details returned to `stdout` and you should be able to create a PPPCP profile with the code sign details of that script.

## Note

Currently, the [tccprofile.py](#) script doesn't used the code sign requirements of a script that has been code signed. This capability is coming soon.

Update: `tccprofile.py` now supports scripts that have been code signed.

## Reference

Details about how code signing works for text files was found here – [Preserving Extended Attributes on OS X](#).



# READING TCC LOGS IN MACOS

6 SEPTEMBER 2018

CATEGORIES: MAC OS

TAGS: LOGGING, LOGS, MACOS, MOJAVE, PRIVACY PREFERENCES, TCC, USER CONSENT

```
Filtering the log data using "subsystem == "com.apple.TCC" AND composedMessage BEGINSWITH "AttributionChain"
Timestamp      Thread      Type      Activity      PID      TTL      tcccd: [com.apple.TCC:access] AttributionChain: ACC:{ID: com.apple.systemevents, PID[2348], auid: 501, euid: 501, bin
ary path: '/System/Library/CoreServices/System Events.app/Contents/MacOS/System Events'}, REQ:{ID: com.apple.WindowServer, PID[219], auid: 88, euid: 88, binary path: '/System/Library/PrivateFrameworks/Sky
Light.framework/Versions/A/Resources/WindowServer'}
2018-09-05 11:33:16.474912+1000 0x6c79      Info      0x763b      234      0      tcccd: [com.apple.TCC:access] AttributionChain: RESP:{ID: com.apple.Terminal, PID[2307], auid: 501, euid: 501, respon
sible path: '/Applications/Utilities/Terminal.app/Contents/MacOS/Terminal'}, ACC:{ID: com.apple.osascript, PID[2347], auid: 501
, euid: 501, binary path: '/usr/bin/osascript'}, REQ:{ID: com.apple.WindowServer, PID[219], auid: 88, euid: 88, binary path: '/System/Library/PrivateFrameworks/SkyLight.framework/Versions/A/Resources/WindowServer'}
2018-09-05 11:33:16.481370+1000 0x6c79      Info      0x763c      234      0      tcccd: [com.apple.TCC:access] AttributionChain: RESP:{ID: com.apple.Terminal, PID[2307], auid: 501, euid: 501, respon
sible path: '/Applications/Utilities/Terminal.app/Contents/MacOS/Terminal'}, ACC:{ID: com.apple.osascript, PID[2347], auid: 501
, euid: 501, binary path: '/usr/bin/osascript'}, REQ:{ID: com.apple.appleeventsd, PID[68], auid: 55, euid: 55, binary path: '/System/Library/CoreServices/appleeventsd'}
2018-09-05 11:33:16.494363+1000 0x6b6f      Info      0x0      234      0      tcccd: [com.apple.TCC:access] AttributionChain: RESP:{ID: com.apple.Terminal, PID[2307], auid: 501, euid: 501, respon
sible path: '/Applications/Utilities/Terminal.app/Contents/MacOS/Terminal'}, ACC:{ID: com.apple.osascript, PID[2347], auid: 501
, euid: 501, binary path: '/usr/bin/osascript'}, REQ:{ID: com.apple.appleeventsd, PID[68], auid: 55, euid: 55, binary path: '/System/Library/CoreServices/appleeventsd'}
2018-09-05 11:33:16.624863+1000 0x6c79      Info      0x0      234      0      tcccd: [com.apple.TCC:access] AttributionChain: ACC:{ID: com.apple.systemevents, PID[2348], auid: 501, euid: 501, bin
ary path: '/System/Library/CoreServices/System Events.app/Contents/MacOS/System Events'}, REQ:{ID: com.apple.appleeventsd, PID[68], auid: 55, euid: 55, binary path: '/System/Library/CoreServices/appleeven
tsd'}
2018-09-05 11:33:16.625863+1000 0x6c79      Info      0x0      234      0      tcccd: [com.apple.TCC:access] AttributionChain: ACC:{ID: com.apple.systemevents, PID[2348], auid: 501, euid: 501, bin
ary path: '/System/Library/CoreServices/System Events.app/Contents/MacOS/System Events'}, REQ:{ID: com.apple.appleeventsd, PID[68], auid: 55, euid: 55, binary path: '/System/Library/CoreServices/appleeven
tsd'}
2018-09-05 11:33:16.628124+1000 0x6c83      Info      0x7d01      281      0      tcccd: [com.apple.TCC:access] AttributionChain: ACC:{ID: com.apple.systemevents, PID[2348], auid: 501, euid: 501, bin
ary path: '/System/Library/CoreServices/System Events.app/Contents/MacOS/System Events'}, REQ:{ID: com.apple.systemevents, PID[2348], auid: 501, euid: 501, binary path: '/System/Library/CoreServices/Syste
m Events.app/Contents/MacOS/System Events'}
2018-09-05 11:33:16.673744+1000 0x6b6f      Info      0x763d      234      0      tcccd: [com.apple.TCC:access] AttributionChain: ACC:{ID: com.apple.FolderActionsDispatcher, PID[2349], auid: 501, eui
d: 501, binary path: '/System/Library/CoreServices/FolderActionsDispatcher.app/Contents/MacOS/FolderActionsDispatcher'}, REQ:{ID: com.apple.WindowServer, PID[219], auid: 88, euid: 88, binary path: '/Syste
m/Library/PrivateFrameworks/SkyLight.framework/Versions/A/Resources/WindowServer'}
2018-09-05 11:33:16.698061+1000 0x6b6f      Info      0x0      234      0      tcccd: [com.apple.TCC:access] AttributionChain: ACC:{ID: com.apple.FolderActionsDispatcher, PID[2349], auid: 501, eui
d: 501, binary path: '/System/Library/CoreServices/FolderActionsDispatcher.app/Contents/MacOS/FolderActionsDispatcher'}, REQ:{ID: com.apple.appleeventsd, PID[68], auid: 55, euid: 55, binary path: '/System
/Library/CoreServices/appleeventsd'}
2018-09-05 11:33:16.698519+1000 0x6b6f      Info      0x0      234      0      tcccd: [com.apple.TCC:access] AttributionChain: ACC:{ID: com.apple.FolderActionsDispatcher, PID[2349], auid: 501, eui
d: 501, binary path: '/System/Library/CoreServices/FolderActionsDispatcher.app/Contents/MacOS/FolderActionsDispatcher'}, REQ:{ID: com.apple.appleeventsd, PID[68], auid: 55, euid: 55, binary path: '/System
/Library/CoreServices/appleeventsd'}
```

Logging user consent events in macOS Mojave to understand what is happening when a process requests access to control another app, or access to data can be done by using a one line `log` command (credit to [@bp](#) on the [macadmins](#) Slack for the command, and the idea to use the phrase *user consent* in relation to these changes).

Run this command, and then execute the script, or launch the app that you need to test.

```
log stream --debug --predicate 'subsystem == "com.apple.TCC" AND eventMessage BEGINSWITH
"AttributionChain"'
```

## Example using osascript

Example `log` output using `osascript -e 'tell app "System Events" to display dialog "Hello world"'` to trigger a dialog.

```
:Desktop # log stream --debug --predicate 'subsystem == "com.apple.TCC" AND eventMessage
BEGINSWITH "AttributionChain"
Filtering the log data using "subsystem == "com.apple.TCC" AND composedMessage BEGINSWITH
"AttributionChain"
Timestamp      Thread      Type      Activity      PID      TTL      t
2018-09-05 11:33:16.474912+1000 0x6c79      Info      0x763b      234      0      t
ccd: AttributionChain: ACC:{ID: com.apple.systemevents, PID, auid: 501, euid: 501, binar
y path: '/System/Library/CoreServices/System Events.app/Contents/MacOS/System Events'}, R
EQ:{ID: com.apple.WindowServer, PID, auid: 88, euid: 88, binary path: '/System/Library/Pr
ivateFrameworks/SkyLight.framework/Versions/A/Resources/WindowServer'}
2018-09-05 11:33:16.481370+1000 0x6c79      Info      0x763c      234      0      t
ccd: AttributionChain: RESP:{ID: com.apple.Terminal, PID, auid: 501, euid: 501, responsi
ble path: '/Applications/Utilities/Terminal.app/Contents/MacOS/Terminal', binary path: '/'
```

```
Applications/Utilities/Terminal.app/Contents/MacOS/Terminal'}, ACC:{ID: com.apple.osascript, PID, auid: 501, euid: 501, binary path: '/usr/bin/osascript'}, REQ:{ID: com.apple.WindowServer, PID, auid: 88, euid: 88, binary path: '/System/Library/PrivateFrameworks/SkyLight.framework/Versions/A/Resources/WindowServer'}
2018-09-05 11:33:16.493752+1000 0x6bf6      Info          0x0              234      0      t
ccd:  AttributionChain: RESP:{ID: com.apple.Terminal, PID, auid: 501, euid: 501, responsible path: '/Applications/Utilities/Terminal.app/Contents/MacOS/Terminal', binary path: '/Applications/Utilities/Terminal.app/Contents/MacOS/Terminal'}, ACC:{ID: com.apple.osascript, PID, auid: 501, euid: 501, binary path: '/usr/bin/osascript'}, REQ:{ID: com.apple.appleeventsd, PID, auid: 55, euid: 55, binary path: '/System/Library/CoreServices/appleeventsd'}
2018-09-05 11:33:16.494363+1000 0x6bf6      Info          0x0              234      0      t
ccd:  AttributionChain: RESP:{ID: com.apple.Terminal, PID, auid: 501, euid: 501, responsible path: '/Applications/Utilities/Terminal.app/Contents/MacOS/Terminal', binary path: '/Applications/Utilities/Terminal.app/Contents/MacOS/Terminal'}, ACC:{ID: com.apple.osascript, PID, auid: 501, euid: 501, binary path: '/usr/bin/osascript'}, REQ:{ID: com.apple.appleeventsd, PID, auid: 55, euid: 55, binary path: '/System/Library/CoreServices/appleeventsd'}
```

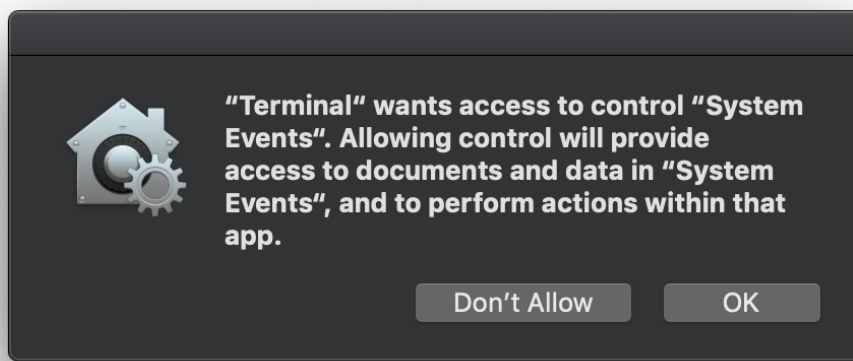
## Breaking it down

In the log output above, there are several significant keywords to help identify what is going on when the prompt is triggered.

Keyword	Definition	Description
ACC	Access/Accessing	Details the application or script that is attempting to access or control macOS that requires user consent.
RESP	Responsible	The application or script that is responsible for the application or script that is attempting to access or control macOS.
REQ	Request/Requesting	Action that is being requested.
ID	Identifier	The identifier of the application or script.
PID	Process ID	The process identifier.
AUID	Actual User ID	The real user identifier.
EUID	Effective User ID	The effective user identifier. This will be different to the AUID if a script has been run as a different user.
binary path	Path of binary	The full path to the binary or script.

In this example, what appears to be happening when the `osascript` is being executed, is that `System Events` is requesting access to `Window Server` in order to display a dialog. If there is no

existing Privacy Preference to allow this to happen, a user consent dialog is presented to the user.



User consent dialog presented after an `osascript` command has been executed from `Terminal`.

Stepping through the logs, the first entry accounts for the `system Event` requesting access to `Window Server`.

```
2018-09-05 11:33:16.474912+1000 0x6c79 Info 0x763b 234 0 t
ccd: AttributionChain: ACC:{ID: com.apple.systemevents, PID, auid: 501, euid: 501, binary
path: '/System/Library/CoreServices/System Events.app/Contents/MacOS/System Events'}, R
EQ:{ID: com.apple.WindowServer, PID, auid: 88, euid: 88, binary path: '/System/Library/PrivateFrameworks/SkyLight.framework/Versions/A/Resources/WindowServer'}
```

The next entry details the application or script that is relevant to what needs to be whitelisted.

```
2018-09-05 11:33:16.481370+1000 0x6c79 Info 0x763c 234 0 t
ccd: AttributionChain: RESP:{ID: com.apple.Terminal, PID, auid: 501, euid: 501, responsi
ble path: '/Applications/Utilities/Terminal.app/Contents/MacOS/Terminal', binary path: '/
Applications/Utilities/Terminal.app/Contents/MacOS/Terminal'}, ACC:{ID: com.apple.osascri
pt, PID, auid: 501, euid: 501, binary path: '/usr/bin/osascript'}, REQ:{ID: com.apple.Win
dowServer, PID, auid: 88, euid: 88, binary path: '/System/Library/PrivateFrameworks/SkyLi
ght.framework/Versions/A/Resources/WindowServer'}
```

In this `log` entry, the *responsible* application is the `Terminal` app. The path is captured in the `binary path`. The application/binary/script *accessing* is `osascript`, again, the path to which is captured in `binary path`, and lastly, the *request* is going to `window server`, the path, unsurprisingly, is found in the `binary path`.

The `window server` log entries are not of key importance in this particular scenario, rather, the first `log` entry where we see `system Events` attempting to *access* `window server` is important, coupled with the details in the subsequent entries relating to the `Terminal` application.

**Note:** In the case of scripts that are not code signed, and are being launched/run by a Launch Agent/Daemon, the script itself cannot be whitelisted, the path of the shell or interpreter must be used instead. My [previous post](#) covers this in a little more detail.

## Applying log info to a profile payload

In practice, the user consent dialog *should* be enough to provide insight as to what applications need to be used in creating an `AppleEvents` Privacy Preferences Policy Control Payload in a profile or other PPPCP payload types, but using a `log stream` may be required.

To create an `AppleEvents` profile using [tccprofile.py](#), the following would be used:

```
./tccprofile.py --appleevents /Applications/Utilities/Terminal.app,/System/Library/CoreServices/System\ Events.app --allow --payload-description="Whitelist Terminal to allow AppleEvents sent from commands run in Terminal" --payload-identifier="com.github.carlashley" --payload-name="Terminal App AppleEvents Whitelist" --payload-org="My Great Company" --payload-version=1 -o Terminal_AppleEvents.mobileconfig
```

## Post user consent approval

Here's the `log` event relating to the user approving control/access. Timestamps are different as this was captured in later tests.

```
2018-09-06 12:09:45.749330+1000 0x6002      Info          0x8392          245      0      t
ccd:  AttributionChain: ACC:{ID: com.apple.fseventsd, PID, auid: 501, euid: 501, binary path: '/System/Library/Frameworks/CoreServices.framework/Versions/A/Frameworks/FSEvents.framework/Versions/A/Support/fseventsd'}, REQ:{ID: com.apple.sandboxd, PID, auid: 0, euid: 0, binary path: '/usr/libexec/sandboxd'}
```

Here's the `log` event relating to the `osascript` being executed after consent is approved.

```
2018-09-06 12:10:36.165892+1000 0x63d4      Info          0x810e          245      0      t
ccd:  AttributionChain: RESP:{ID: com.apple.Terminal, PID, auid: 501, euid: 501, responsible path: '/Applications/Utilities/Terminal.app/Contents/MacOS/Terminal', binary path: '/Applications/Utilities/Terminal.app/Contents/MacOS/Terminal'}, ACC:{ID: com.apple.osascript, PID, auid: 501, euid: 501, binary path: '/usr/bin/osascript'}, REQ:{ID: com.apple.WindowServer, PID, auid: 88, euid: 88, binary path: '/System/Library/PrivateFrameworks/SkyLight.framework/Versions/A/Resources/WindowServer'}
2018-09-06 12:10:36.179368+1000 0x6230      Info          0x0            245      0      t
ccd:  AttributionChain: RESP:{ID: com.apple.Terminal, PID, auid: 501, euid: 501, responsible path: '/Applications/Utilities/Terminal.app/Contents/MacOS/Terminal', binary path: '/Applications/Utilities/Terminal.app/Contents/MacOS/Terminal'}, ACC:{ID: com.apple.osascript, PID, auid: 501, euid: 501, binary path: '/usr/bin/osascript'}, REQ:{ID: com.apple.appleeventsd, PID, auid: 55, euid: 55, binary path: '/System/Library/CoreServices/appleeventsd'}
```

```
2018-09-06 12:10:36.179915+1000 0x6230      Info          0x0                245      0      t
ccd: AttributionChain: RESP:{ID: com.apple.Terminal, PID, auid: 501, euid: 501, responsi
ble path: '/Applications/Utilities/Terminal.app/Contents/MacOS/Terminal', binary path: '/
Applications/Utilities/Terminal.app/Contents/MacOS/Terminal'}, ACC:{ID: com.apple.osascri
pt, PID, auid: 501, euid: 501, binary path: '/usr/bin/osascript'}, REQ:{ID: com.apple.app
leeventsd, PID, auid: 55, euid: 55, binary path: '/System/Library/CoreServices/appleevent
sd'}

2018-09-06 12:10:36.200470+1000 0x63c6      Info          0x85b2                294      0      t
ccd: AttributionChain: ACC:{ID: com.apple.systemevents, PID, auid: 501, euid: 501, binar
y path: '/System/Library/CoreServices/System Events.app/Contents/MacOS/System Events'}, R
EQ:{ID: com.apple.systemevents, PID, auid: 501, euid: 501, binary path: '/System/Library/
CoreServices/System Events.app/Contents/MacOS/System Events'}
```

# Errors/Corrections

Given this is based on observations of the `log stream`, and not any official documentation, any corrections to errors, etc, can be directed to **@carl** on the [macadmins](#) slack.

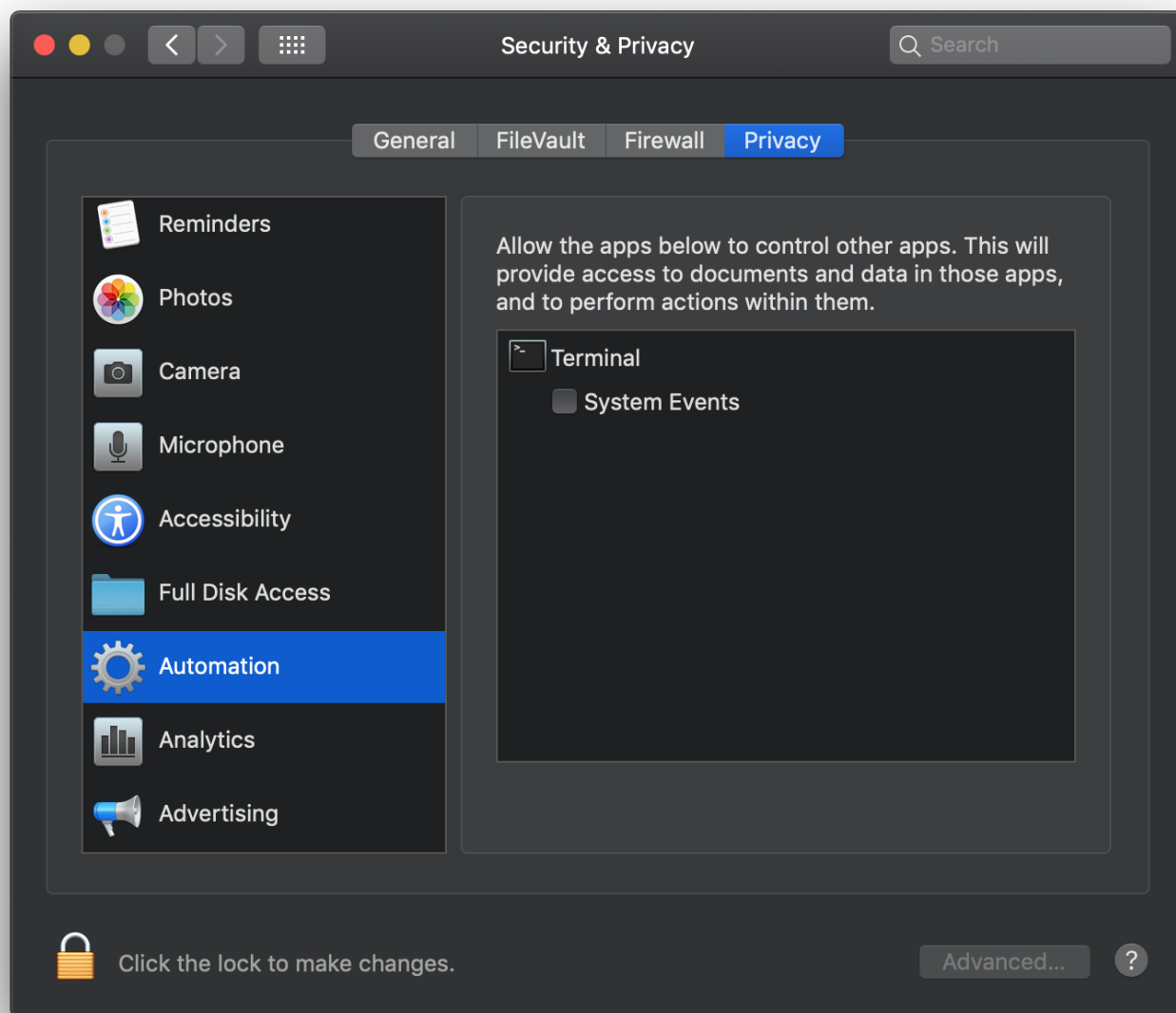
Details are current as at time of posting.

# PRIVACY PREFERENCES POLICY CONTROL

1 SEPTEMBER 2018

CATEGORIES: MAC OS

TAGS: LAUNCHAGENTS, LAUNCHD, LAUNCHDAEMONS, MOJAVE, PRIVACY PREFERENCES, TCC



With the release of macOS Mojave imminent, and a new [Privacy Preferences Policy Control Payload](#), it is important to properly launch scripts from `LaunchAgents` or `LaunchDaemons` to ensure the correct process is identified when the script runs, so that macOS uses the correct `codesign` requirements; this is especially true for `python` scripts as there are different requirements depending on how the `python` script is run from the `LaunchAgent` or `LaunchDaemon`.

If a `LaunchAgent` or `LaunchDaemon` does not explicitly use `/usr/bin/python` to execute the script, it uses the `codesign` details of `/System/Library/Frameworks/Python.framework/Resources/Python.app`.

2018-09-28 Update:

In the final release of mac OS Mojave 10.14, the TCC logs indicate that the System Framework `python` is being used, but depending on how the `python` script is executed from a `LaunchDaemon` Or `LaunchAgent`, the `codesign` details will vary.

This presents an issue as the `codesign` details for `/usr/bin/python` are different to those of `/System/Library/Frameworks/Python.framework/Resources/Python.app`.

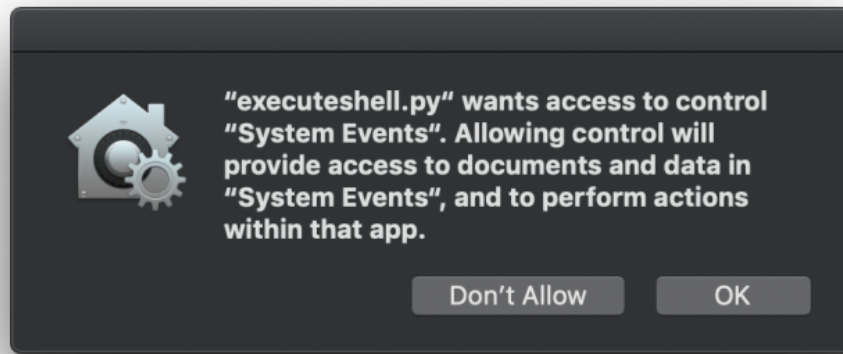
```
:Desktop # codesign -dr - /usr/bin/python
Executable=/usr/bin/python
designated => identifier "com.apple.python" and anchor apple

:Desktop # codesign -dr - /System/Library/Frameworks/Python.framework/Resources/Python.ap
p
Executable=/System/Library/Frameworks/Python.framework/Versions/2.7/Resources/Python.app/
Contents/MacOS/Python
designated => identifier "org.python.python" and anchor apple
```

An example of a `LaunchAgent` that will *not* work (as at time of writing), is:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/
PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>Label</key>
  <string>com.example.pythonScript</string>
  <key>ProgramArguments</key>
  <array>
    <string>/Users/testuser/Desktop/executeshell.py</string>
  </array>
  <key>RunAtLoad</key>
  <true/>
</dict>
</plist>
```

When this is launched by `launchd`, any privacy alerts triggered by the script will show `executeshell.py` as the parent process\*. It appears that in this example, macOS uses the `codesign` details of `/System/Library/Frameworks/Python.framework/Resources/Python.app`



### executeshell.py privacy alert

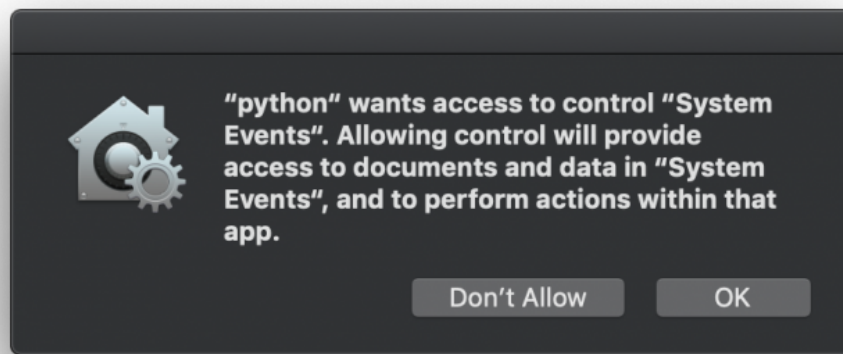
To correctly launch this script so that any privacy profiles work, the `ProgramArguments` array must include the path to the `python` interpreter.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
  <key>Label</key>
  <string>com.example.pythonScript</string>
  <key>ProgramArguments</key>
  <array>
    <string>/usr/bin/python</string>
    <string>/Users/testuser/Desktop/executeshell.py</string>
  </array>
  <key>RunAtLoad</key>
  <true/>
</dict>
</plist>
```

With the adjusted file, the script will have the correct parent process if it triggers any privacy alerts. It appears in this case that the `codesign` details of `/usr/bin/python` are used, which is what the privacy management system is expecting.

This is important, because the shebang in a script does not always appear to be evaluated correctly when `launchd` launches the script.





## python privacy alert

---

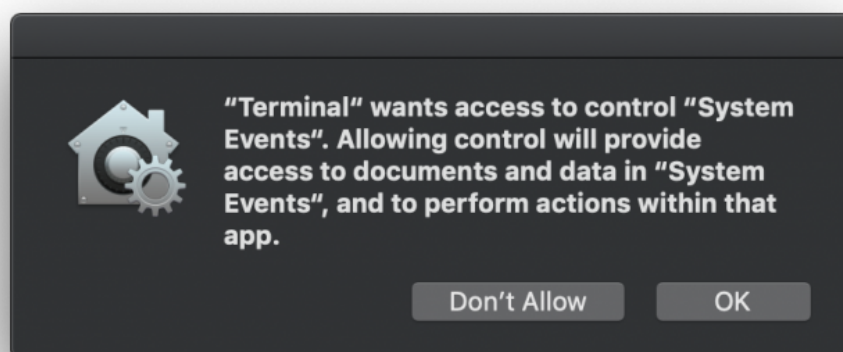
\*A note about parent processes in relation to whitelisting:

This article only discusses behaviour observed in testing profiles to whitelist apps/scripts.

Based on testing, to successfully whitelist scripts that interact with files/folders that are protected by the new privacy protections in macOS Mojave, or automation events that try to control macOS, the item being whitelisted must be the parent process.

For example, if you open up `Terminal.app` and run the below `osascript`, you will be prompted to allow Terminal to control System Events.

```
osascript -e 'tell app "System Events" to display dialog "Hello World"'
```



## Terminal controlling System Events

In this example, the parent process is the Terminal application, so to allow `Terminal` to control System Events, you would need to create a profile with an `AppleEvents` Privacy Preferences Policy Control Payload that whitelists the Terminal application.


Other examples include using outset to manage the execution of scripts, profile installs, etc during login or boot time. In this instance, `python` is the parent process of any scripts/profile installs that occur, so `/usr/bin/python` would need to be whitelisted in appropriate Privacy Preferences Policy Control Payloads in order for `outset` to be run any scripts without triggering privacy alerts.

# TCCPROFILE

17 AUGUST 2018

CATEGORIES: MAC OS

TAGS: MAC OS, MAC\_OS\_X, MACOS, MDM, PRIVACY PREFERENCES, TCC

 carlashley / **tccprofile**

Unwatch ▾ 4

★ Star 23

Fork 0

Code

Issues 0

Pull requests 0

Projects 0

Wiki

Insights

Settings

Creates a TCC profile for new Privacy Payloads in macOS Mojave

Edit

Add topics

34 commits

1 branch

0 releases

1 contributor

Apache-2.0

Branch: master ▾


New pull request

Create new file





Upload files

Find file

Clone or download ▾

 carlashley Update README.md

Latest commit 21b00d2 7 hours ago

 generated_profiles	Create Terminal_Whitelist.mobileconfig	a day ago
 LICENSE	Initial commit	16 days ago
 README.md	Update README.md	7 hours ago
 tccprofile.py	Handle multiline codesign output and other fixes	7 hours ago

<https://github.com/carlashley/tccprofile>

Quickly builds Accessibility profiles for macOS applications based on new Profile payloads for Privacy Preferences Policy Control Payload as outlined in Apple's [Configuration Profile Reference](#) documentation.

The utility can also sign a generated profile.

```
@aurora]:tccprofile # ./tccprofile.py -a /Applications/Adobe\ Photoshop\ CC\ 2018/Adobe\ Photoshop\ CC\ 2018.app --payload-description="TCC Whitelist for Adobe Photoshop"
--payload-name="TCC Whitelist Adobe Photoshop" --payload-name="TCC Whitelist" --payload-org="My Great Company"\n --payload-version=1 --payload-identifier="com.carlashley.github" -o
be_Photoshop_TCC.mobileconfig --allow --sign="cert_name"
@aurora]:tccprofile # ls
be_Photoshop_TCC.mobileconfig          LICENSE                                tccprofile.py
be_Photoshop_TCC_Signed.mobileconfig  README.md
```

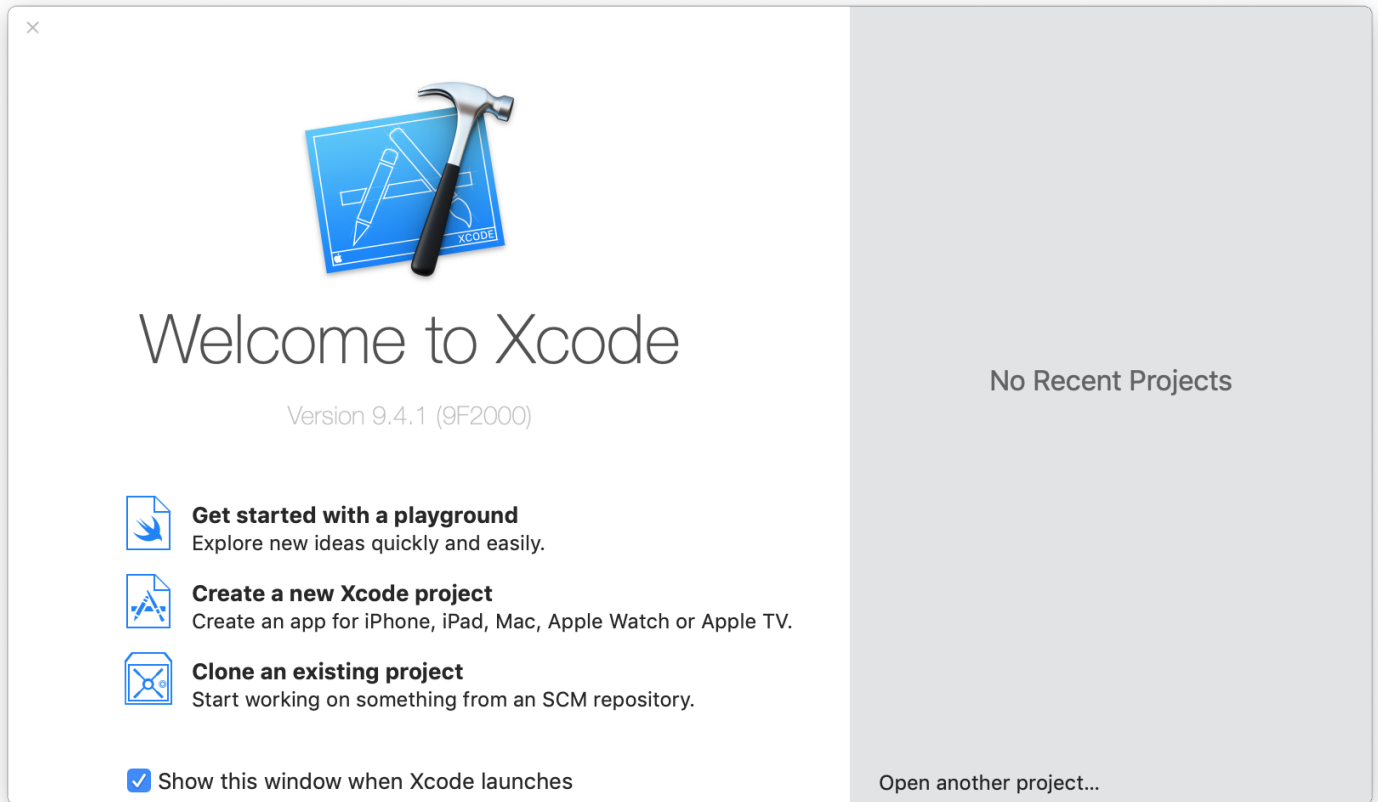
rofile run

# UPDATE TO XCODETOOLS.PY

5 AUGUST 2018

CATEGORIES: MAC OS

TAGS: COMMAND LINE TOOLS, XCODE



The [xcode\\_tools](#) script I wrote a while back has been updated to better handle newer releases of Xcode.

The new version can process most recent macOS releases (only tested 10.9+) to find the most recent Command Line Tools and SDK release for that version of macOS, as well as install found packages.

It requires python 2.7.10 and be run on a macOS release newer than 10.9.

To see available arguments, clone the repo and run `./xcodetools.py --help`. By default, running `./xcodetools.py` will download the latest version of the Command Line Tools and SDK (if available) to `/tmp/xcode`.

# REVISITING OBFUSCATING TEXT IN SCRIPTS

6 JULY 2018

CATEGORIES: MAC OS

TAGS: AES, CIPHER TEXT, ENCRYPTING, MAC OS, MAC OS X, OBFUSCATE, PLAIN TEXT, PYTHON, SCRIPTS



A little while ago I covered a technique that could be used to obfuscate sensitive text in scripts used to manage macOS; a recent issue with this particular technique left me needing to re-write this in `python`.

What you'll need:

- python 2.7.10 minimum
- [`pycrypto`](#)

Installing `pycrypto` using `pip` requires Xcode tools to be installed, which for some environments is not possible or desired, so you will need to work out a means of getting the `pycrypto` module onto target machines. In my instance, I used `logGen` to take a snapshot pre/post installation and built a package for distributing to clients.

To create a key for encrypting/decrypting, a file is created containing 32bytes of random data. An example of how this is generated is some simple `python` code:

```
#!/usr/bin/python

import os
import sys

# Create a random 32 byte keyfile for use with aescrypto.py

def createKey(output_file):
```

```

with open(output_file, 'wb') as output:
    output.write(os.urandom(32))

if len(sys.argv) is 2:
    createKey(sys.argv)
else:
    print 'Usage: {} '.format(sys.argv)
    sys.exit(1)

```

Next comes creating the encrypted text; turning that wonderful coder/scripter resource, Stack Overflow, in particular a discussion on how to encrypt text using AES256. Code has been borrowed heavily from [this comment in particular](#), I have a deep appreciation for people willing to share their knowledge with others.

[A sample script available here](#) covers how this can be used to encrypt/decrypt plain text. This file outputs the encrypted text into a `p1ist` file simply because in my use case I want the data in a machine readable format. Any other output method can be used.

Fundamentally, the key is read in, and a random initialisation vector is generated when the `encryptText()` function is called, this ensures that each time a string is encrypted, it doesn't result in the same cipher text. It does mean that the initialisation vector needs to be supplied with the cipher text in order to decrypt it.

The `decryptText()` reads the same key as used to encrypt, and the correct initialisation vector, then passes the cipher text back through the `aes.decrypt()` method to get the plain text.

I don't claim to know the complex ins and outs of which encryption methods work the best, so I don't plan on using this technique for mission critical/sensitive data, this is simply a means to ensuring simple text strings that don't need to be trivially accessible are obfuscated from prying eyes, especially where it is fairly trivial to expand a package file out, or trawl through web directories that aren't appropriately configured.

As this isn't true asymmetric encryption (using a private/public key pair), using this technique does have some risks.

- The key in this technique is the same key for encrypting and decrypting, so extra care needs to be taken to ensure it isn't easily accessible
- If storing the key on a target machine, store it in an area of the file system that only privileged users have access to, etc.
- If deploying the key in a package, limit access to the location the package is stored to ensure only authorised devices/users can access it

It's also advisable to assess whether this technique is appropriate for your needs and the scenario in which plain text must be obfuscated, and potentially any legislation governing how certain data

must be protect. My particular use case doesn't involve sensitive scenarios, and is sufficient enough to make it more complicated for any naughty little prying eyes to see what is going on.

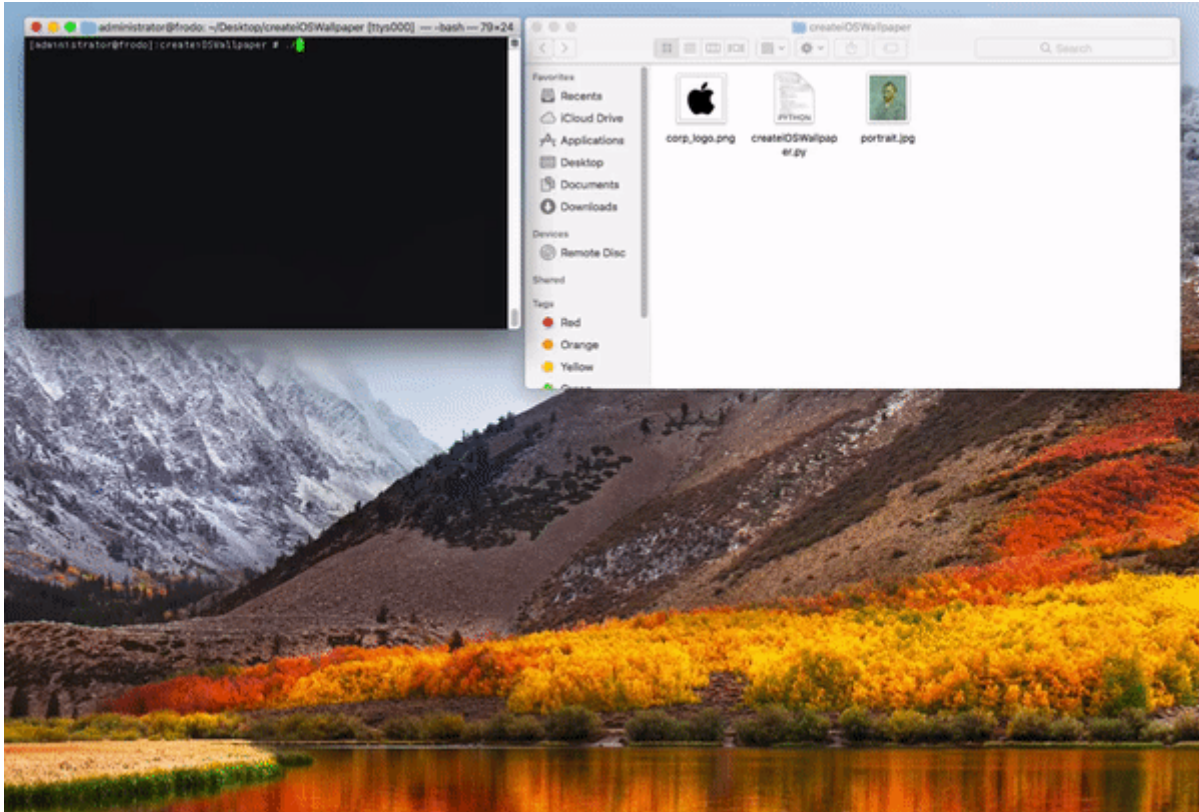


# CREATEIOSWALLPAPER.PY

25 JUNE 2018

CATEGORIES: MAC OS

TAGS: CREATEIOSWALLPAPER.PY, DEVICE MANAGEMENT, IOS, IPAD, MDM, WALLPAPER



A recent need for managing iPads within my workplace is creating a custom wallpaper for each device that includes a picture of the person the device is assigned to, as well as their name and a QR code for the asset details, plus the organisation logo. Given the large number of these devices, creating wallpapers by hand is out of the question.

Python to the rescue.

What you'll need:

- python 2.7.10 (minimum)
- pillow (`pip install pillow`)
- qrcode (`pip install qrcode`)

Both `pillow` and `qrcode` may have additional dependencies to install.

I've uploaded a sample bit of code to <https://github.com/carlashley/createiOSWallpaper/>

The code will build a wallpaper image based on the specified target device (check the `resolutions` dictionary for supported devices, can easily be extended to include iPhones), and

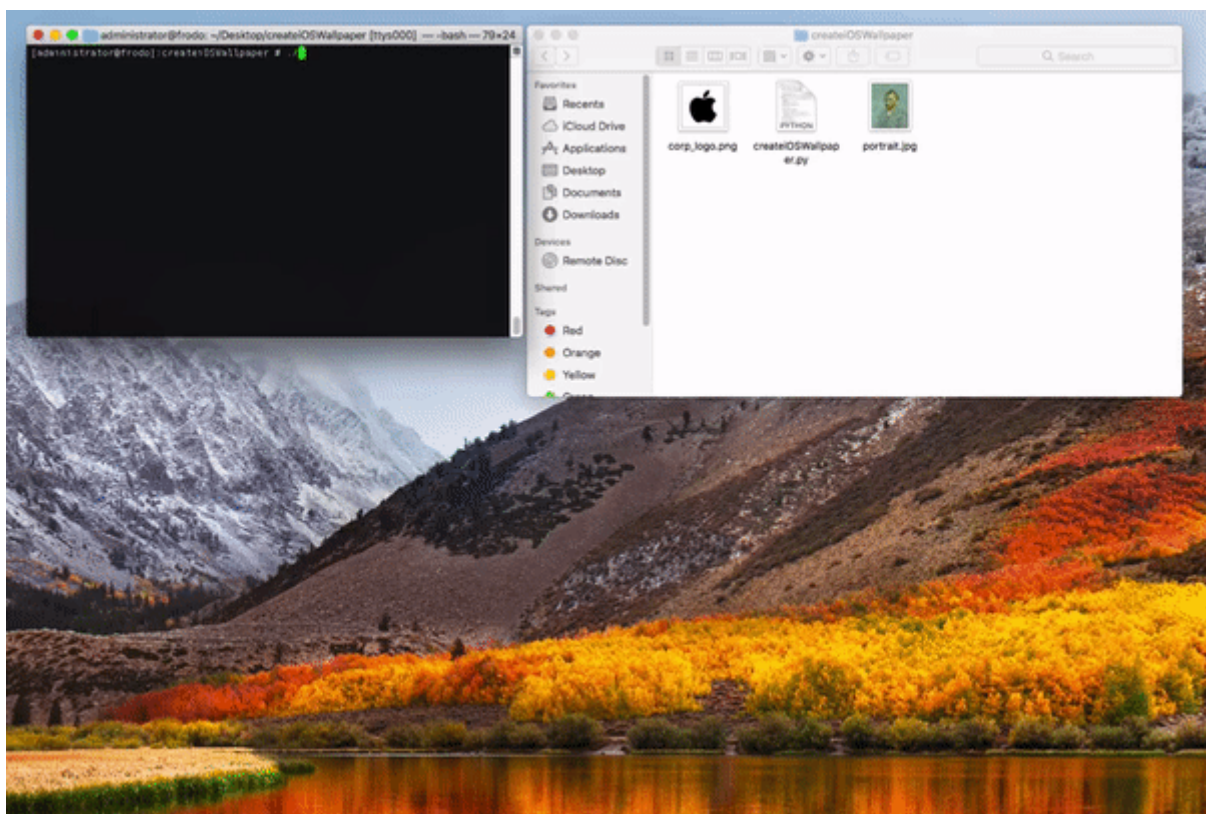
will also check if the picture of the user being inserted into the wallpaper exceeds the wallpapers width/height and resize it down.

The QR code is dynamically generated and dropped into the image, in a centre alignment, offset from the corporate logo. The user's name is dropped onto the bottom of the image with some vertical padding and a horizontal centre alignment.

The user's name text is single line, but could be made into a multiline text string, you would need to take into account the new height of the multiline text however, or use another method to wrap the multiline text and re-calculate the line heights.

This could be used in conjunction with your MDM to create a wallpaper for each iOS device managed by the MDM (if your MDM supports setting the device background via API).

Here's a simple video of the tool in action. In this sample, a wallpaper is created for each of the different iPad hardware versions (note, a number of iPad hardware models have the same resolution).



createiOSWallpaper.py in action

# REVISITING APPLELOOPS.PY AND MANAGING ADDITIONAL AUDIO CONTENT

18 OCTOBER 2017

CATEGORIES: MAC OS

TAGS: GARAGEBAND, LOGIC PRO X, MAC OS, MACOS, MAINSTAGE 3, MUNKI



A few months ago while planning for future macOS upgrades, I realised that the existing methodologies for managing the audio content for GarageBand (and other Apple audio apps) in munki and autopkg were not very well equipped to handle the frequent changes Apple make to the additional content for these apps.

While downloading and importing content for one version of GarageBand would work, this would only be valid for a specific release of GarageBand/Logic Pro X/MainStage, and any time those additional packages were changed by Apple, it would mean revisiting what audio content was imported into munki; while it is possible to simply keep importing packages into munki, and simply mark them as updates for X version of GarageBand, this wasn't a strategy that I felt was viable for long term deployments. Rather, I needed a solution that met the following criteria:

- Could be used as a `post_install` script for munki.

- Could be run as a script for other deployment tools, or used on its own.
- Required minimal effort to maintain.
- Could be used to mirror content locally.

So I decided to re-visit the `appleLoops.py` tool I had written and turn it into a means of managing the deployment of additional content for these Apple audio apps.

Below is the general outline of how the new `appleLoops.py` re-write can be used in munki. This assumes a familiarity with munki and related deployment techniques/tools; some minor programming experience in python is handy, but not essential.

The basic process that occurs with any of the three apps is that the first run will trigger a download of mandatory content, once this is installed, the user is given the option of downloading additional content that isn't essential to the app, but nice to have.

A more in depth explanation of what happens the first time these audio apps launch is available on [this Wiki page](#).

With this information covered, here is my deployment process for munki; note, this process can be adapted to use for deploying the additional content for Logic Pro X and MainStage 3.

1. Import a GarageBand/Logic Pro X/MainStage 3 app into munki. Typically I use an autopkg recipe that also handles modifying the `_MASReceipt/receipt` file so the app is not associated with a specific Apple ID.
2. Use `appleLoops.py` to download a mirror copy of the additional content for the specific versions being deployed, and place this mirror copy on the `munki_repo` web server. This can potentially be automated to keep the additional content up to date without continuing to manually update the mirror.
3. Add the `appleLoops.py` script as a `post_install` script to the imported app; I find using munkiadmin the easiest way to modify munki's `pkginfo` files.
4. Modify a small snippet of code at the end of the file.

Change:

```
if __name__ == '__main__':
    main()
```

To:

```
if __name__ == '__main__':
    al = AppleLoops(allow_insecure=True, deployment_mode=True, pkg_server='http://munkiserver/munki_repo/pkgs/apple_audio/', dry_run=False, log_path='/var/log', mandatory_loops=True, optional_loops=True)
    al.main_processor()
```

The benefit of this is that it is simple to make a change to the `post_install` script in the relevant munki `pkginfo` without having to install the script on each client, then maintain it on each client.

The arguments that are used in the example above are applicable to my environment as there is a local mirror of the additional content that is maintained; you would need to adjust to suit your environment.

While the whole script could be shrunk down to do just the essential parts of deploying the audio content and inserted into deployment tools as either `post_install` scripts, or other workflow processes in (for example, in JAMF), there are many other features that this updated release includes:

- Downloading a mirror of the Apple audio content for local hosting.
- Skips existing downloaded packages to reduce bandwidth usage.
- Can be run as a scheduled job (either through cron or launchd or other scheduling systems) on the local mirror to download new content.
- Utilise a Caching Server on the network to dynamically cache loops for deployments.
- Create DMG's of downloaded content.
- Download content for specific releases of Apple's audio apps.
- Dry run deployment mode to determine what would be installed or upgraded.
- Set a free space threshold to avoid filling up local storage when in deployment mode.
- Silent output during deployment mode.
- Re-install all previously installed loops.

There are some things to be aware of that currently are limitations or:

- Each time the script is run in deployment mode, it will look for *any* of the three apps installed; if one or more is found, it will deploy the content for all the apps found. This means if you're deploying GarageBand and Logic Pro X together, the first time the script is run, it will deploy the loops for *both* GarageBand and Logic Pro X; the second time it is run, it will skip any packages already installed.
- Currently there is no means of providing progress of a `post_install` script to munki, so any progress UI elements in user facing components of munki will only show that scripts are being run, which for an end user could look as though the progress of the installation has stalled. If you deploy these audio apps as a "self service" deployment with munki, you may wish to take advantage of `preinstall_alert` to alert users.

Further information can be found on the [appleLoops.py.wiki](#).

The [latest release is available here](#).

Support can be found by joining the #musicsupport room in the [macadmins Slack](#).

# OBFUSCATING SENSITIVE TEXT IN SHELL SCRIPTS

9 OCTOBER 2017

CATEGORIES: MAC OS

TAGS: MACOS, OBFUSCATE, OPENSSEL, SCRIPT, SHELL, SSH-KEYGEN

For many years I've been using a simple method of obfuscating sensitive information in shell scripts. I can't take the credit for this, as it was something inherited in my work environment.

This technique uses `openssl` to encrypt and decrypt a string of text without having to embed the sensitive data in plain text. Note, this isn't proper private/public key-pair encryption, so treat this as simply obfuscating text.

First step in the process is to generate what can be referred to as a *key* file. The key file must contain a random string. The easiest way I find to do this is to use `ssh-keygen` and copy a large chunk of text either from the private key, or the public key, and save it in a file.

For example, `key.txt` contains the following:

```
AAAAB3NzaC1yc2EAAAADAQABAAQACpppktHMeS0D2wgxd0NdGAeNHlqcpNUoQ7LdYZLkDA4Y6Mq25wrkVh4iheK
NhwiEyz+cdmhkpf4oMXu8ccglvxRASBqq2GIJuWHPkFVKmbxCq6+G5uQz8shvOLE5Egy6rWgltnkUSJpCJ9LJO2t
I8Jvlyr34lrJvYTitI9E+4bGGXcmSxrG236RJKto6g4bV+IYszjAM6EHaiJwzILplhRApAETq23hEE9TVOW1POa6D
bGhCSz+jwh2ZCSiod7yTeZy9DtPJ5rNm8FLJMH65wt48rRqgfy4UuUy8NYw79LS4S8XJ3PiklBhpkPAPTWrGRCa91
DlPownCjTMizlc5r
```

The shell script below is an example of this obfuscation/de-obfuscation in process.

```
#!/bin/sh
password="test"
secret="cat /var/root/key.txt"
generated_secret=$(echo ${password} | openssl enc -aes-256-cbc -pass "pass:${secret}" -a
-e)
cleartext_secret=$(echo ${generated_secret} | openssl enc -aes-256-cbc -pass "pass:${secre
t}" -a -d)
echo "Obfuscated password: ${generated_secret}"
echo "De-obfuscated password: ${cleartext_secret}"
```

When this is run:

```
: # sudo ./foo
Obfuscated password: U2FsdGVkX18FWNJ0K3nFa0QPyb9YksGxavWE0p4Km7g=
De-obfuscated password: test
```

To make this a little more obscure, the `key.txt` file should exist somewhere that normal users cannot access, but still readable by root.

For example, `/var/root/key.txt` with read-only permission for root. Any script that needs to de-obfuscate sensitive information therefore needs to be run as root, or have a `suid` set.

Additionally, any script that uses this technique should be kept separate from the key file, and when used in any shell script, make sure any variable containing the de-obfuscated text is used only where required, and unset/destroyed after use.

I typically only use this in single use scripts – for example, creating a local user account when a machine is imaged/deployed.

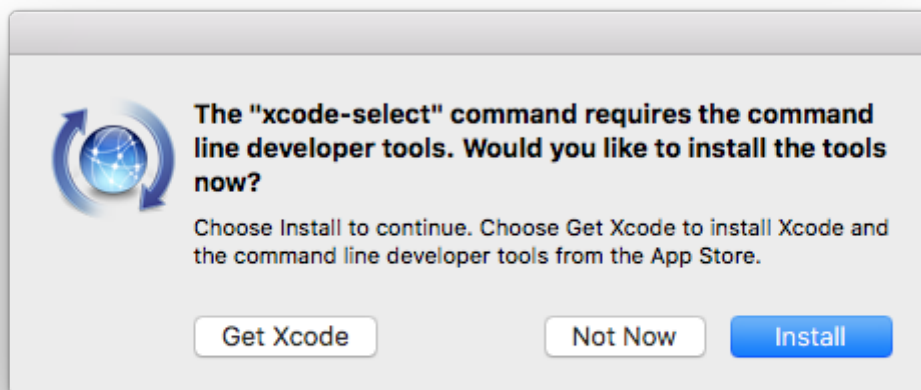
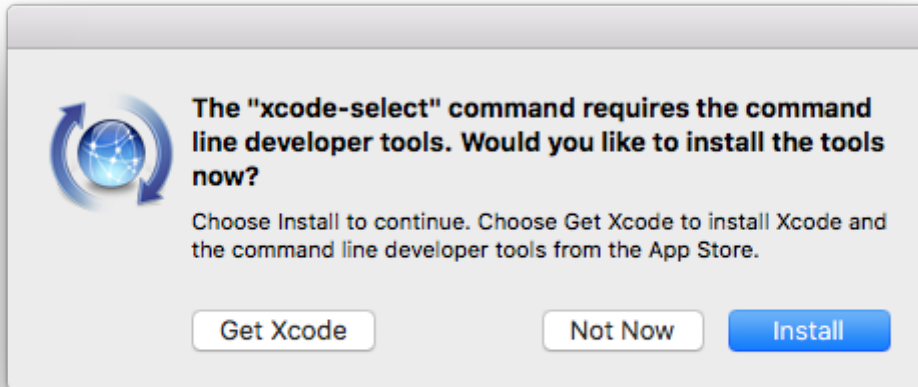


# XCODE COMMAND LINE TOOLS

13 APRIL 2017

CATEGORIES: MAC OS

TAGS: CLI, CLI TOOLS, COMMAND LINE TOOLS, MAC OS X, XCODE



Installing the Xcode command line tools is generally a pretty easy affair, with either a visit to the Apple Developer portal to download the single DMG file, or using the `xcode-select --install` command.

But.

Both of these methods require interacting with the GUI, and frankly, I'd rather not have dialog boxes pop up when I've got an automated process to configure my machine, or have to log into the Developer portal just to download the DMG. So while trolling through the macOS Software Update catalog for Sierra, I stumbled on three packages that appeared to contain all the CLI tools.



A quick capture of HTTP events in a Squid log revealed these three packages are the same as those downloaded when the `xcode-select --install` command is run.

So, here is [xcode\\_tools.py](#). It's really basic. Like, *really*, really basic. It just downloads the tools to your `~/Desktop` folder. Done. Nothing else to do except install at your leisure.

I can't make any guarantee that this will work forever, and I'll have to wait for the next Xcode release to see what happens with the software update catalog, but at least for the time being, you can download the Xcode 8.3 CLI tools without those annoying GUI pop ups.

# GARAGEBAND, LOGIC PRO X, MAINSTAGE 3, OH MY!

27 FEBRUARY 2017

CATEGORIES: MAC OS

TAGS: APPLELOOPS, GARAGEBAND, LOGIC\_PRO\_X, MAINSTAGE

Apple have a great EDU deal going for their 'Pro Apps', consisting of Final Cut Pro X, Logic Pro X, Motion 5, Compressor 4, and MainStage 3, all up, this deal in [dollydoos](#) is \$299.99. That's a nice saving of \$727 (and any unused redemption codes can be [converted to codes for managed devices](#) for use in VPP).

But, that's not the point of this post...

I recently re-wrote the `get_audio_content.py` script that downloaded the loop packages that GarageBand and Logic Pro X come with, the new tool is now called `appleLoops` and [has moved](#). The new tool will now download loop packages for GarageBand, LogicPro X, and MainStage 3. More information can be found in the [README](#).

Some observations about these loops based on my experiences:

- GarageBand, Logic Pro X, and MainStage all have a plist file contained within the app that lists all the packages required for that particular release.
- There is *no* specific order that the loop/content packages need to be installed in.
- This same file is also hosted on Apple servers, and is fetched by each app.
- After upgrading from one version to another, where loop packages have changed, the apps will only download those that are required for the newest version.
- Loop packages don't change with each release, so there are some app updates that won't have new loop contents, in that instance, deploy the latest content that you have. For example, content in GarageBand versions 10.1.2 to 10.1.4 didn't change until Apple released GarageBand 10.1.5.
- There are a lot of loop packages that are common to all three apps; this content is Caching Server aware, and will be cached if you have a Caching Server on your network.
- Additional content in GarageBand, Logic Pro X, and MainStage is now a free "in app purchase"; although this purchase is not handled via in app purchase mechanisms, it's just a straight up download. You are prompted on first use to download the additional content after downloading and installing the essential content packages.
- Loops are indexed as required. You can copy `~/Music/Audio\ Music\ Apps/Databases/LoopsDatabaseV09.db` from an existing install to skip this step. Loops are re-indexed when they are updated. Thanks to neilmartin83 in the macadmins Slack for pointing this out.
- You can disable the 'Whats new in Logic Pro X' on a fresh install by running

```
touch ~/Library/Preferences/com.apple.logic10.plist
```

- The GarageBand welcome screen can be disabled by running

```
defaults write ~/Library/Containers/com.apple.garageband10/Data/Library/Preferences
com.apple.garageband10 welcomeScreenShown -bool true
```

- The GarageBand prompt to download additional optional content can be disabled by running

```
defaults write ~/Library/Containers/com.apple.garageband10/Data/Library/Preferences
com.apple.garageband10 ShowMoreDownloadsDialogGB -bool false
```

Deploying these loops can be a pain in the neck, so to make it a bit easier, the `appleLoops` tool will download content and store it in a fairly straight forward set of folders; it will also attempt to reduce the amount of data downloaded by copying a file from another folder within the specified download directory if it already exists. The tool will also skip over already downloaded content and resume from the last partial file downloaded.

An example of the folder structure is as below.

```
:loops # ls -lha
drwxr-xr-x  4 foo  staff   136B 16 Feb 19:39 logicpro1030
:loops # ls -lha logicpro1030/
drwxr-xr-x  3 foo  staff   102B 16 Feb 19:39 2013
drwxr-xr-x  4 foo  staff   136B 16 Feb 19:35 2016
:loops # ls -lha logicpro1030/2016/
drwxr-xr-x  31 foo  staff   1.0K 16 Feb 19:29 mandatory
drwxr-xr-x 544 foo  staff   18K 16 Feb 21:37 optional
:loops # ls -lha logicpro1030/2016/optional/
-rw-r--r--  1 foo  staff   20M 16 Feb 20:03 MAContent10_AssetPack_0002_AlchemyOrgans.p
kg
-rw-r--r--  1 foo  staff   99M 16 Feb 19:54 MAContent10_AssetPack_0003_AlchemyBrass.pk
g
...
-rw-r--r--  1 foo  staff   912K 16 Feb 21:02 MAContent10_AssetPack_0625_AlchemySettings
TexturesNEffects.pkg
-rw-r--r--  1 foo  staff   908K 16 Feb 19:46 MAContent10_AssetPack_0626_AlchemySettings
VintageSynth.pkg
```

If you're a munki user, the easiest way to import these into munki is with a `for` loop in a shell script that does this for you. A simple example to import loops for Logic Pro X is below. In this instance, you'd need to run it in all the folders that the loops are found in.

```
#!/bin/sh
# Bulk import loop/content packages into munki. Modify to suit your munki setup, etc.

pkg_category="Audio" # Category you want to have the loops available in.
pkg_developer="Apple" # Developer of the loops.
import_path="logic_pro" # Folder within the munki repo pkgs folder you want these to go i
```

```
n. I use separate folders as this helps me keep the loops manageable, even if disk space is used up.  
update_for="LogicProX" # The munki pkginfo name the loop package is for.  
requires="LogicProX" # This ensures that the app is installed before the loop package is installed.
```

```
# Note, the --update_for="foo" flag must be used multiple times if you need to specify the package as an update for more than one application. In this instance, I suggest manually specifying the apps rather than using the variable update_for
```

```
for i in *.pkg; do  
    /usr/local/munki/munkiimport ${i} --nointeractive --unattended_install --category  
    ="${pkg_category}" --displayname=$(basename -s .pkg ${i}) --developer="${pkg_developer}"  
    --subdirectory="${import_path}" -c "testing" --update_for="${update_for}" --requires="${requires}"  
done
```

Another thing I do as part of managing these loops in munki, is to make a manifest that contains *all* the loops for a given application, as a managed install, so that if I ever needed to use it, the manifest is available. So far I haven't needed it, as setting the package as an update for a given application has worked for me.

Keeping these packages managed in a mixed version environment can get tricky when there are loop/content changes, so try and keep your clients on the same app version; if you do have to upgrade, try and upgrade all the clients simultaneously.

One of the great resources any Mac admin should have in their arsenal is the Mac Admin's Slack – if you haven't already, [sign up](#) (it's free!) and check out #garageband or #logicpro – they're quiet, but any questions can be asked in there. If you're an Aussie or Kiwi Mac admin, join the #anzmac channel for shenanigans.

# PROMETHEAN ACTIV DRIVERS

28 NOVEMBER 2016

CATEGORIES: MAC OS

TAGS: ACTIVDRIVER, MACOS, PROMETHEAN, SIERRA

If you're installing the Prometheus Activ Driver "app" on Mac OS El Capitan and macOS Sierra, as well as running into issues with the software not launching or not detecting connected interactive whiteboards, the issue appears directly related to bad permissions set by the installer application.

There is a [community support thread](#) that has a fix, however, I've slightly tweaked the fix on the basis that `root` ownership should have `rxw`, while `group` and `others` can have `r-x`.

I'm also nervous when LaunchAgents and LaunchDaemons installed by apps/packages get execute permissions – they're entirely not needed; macOS only requires read permissions, so I also change permissions for the Prometheus LaunchAgents.

To fix, you can simply add a `postinstall` script to a package, or if you use munki, simply add this as a post-install script in the `pkginfo` file for the imported package.

```
#!/bin/sh
/bin/chmod -R 755 /usr/local/share/promethean
/bin/chmod -R 644 /Library/LaunchAgents/com.promethean.*
/bin/chmod -R 755 /usr/local/lib
```

The current driver version (5.14.21) appears to work correctly in macOS Sierra, even though Prometheus indicates the driver set is only compatible for Mac OS X 10.8 to 10.11 – use these drivers in Sierra at your own discretion/judgement.

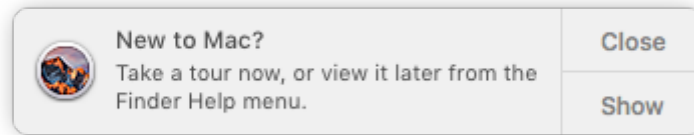
On the subject of Prometheus software, I also suggest you hit their support team up and ask them to return to using the standardised Apple `pkg` format rather than the awful app based installer they've switched to.

# COM.APPLE.TOURISTD

19 OCTOBER 2016

CATEGORIES: MAC OS

TAGS: MACOS, TOURISTD



New to Mac notification

This notification bugs me, but only because I don't particular care for operating systems taking me on a tour of all the "magical" features that I end up ignoring, and also because it's one more thing I have to dismiss after setting up a fresh macOS install or upgrade.

I don't have OCD (and I feel for those that suffer from it), but I am *particular* about setting up my macOS installs. So I went down a little alleyway to see if there was some way to never be bothered with the notification again.

There is a preference file that gets created/modified after dismissing or activating the notification – `~/Library/Preferences/com.apple.touristd.plist`.

Lets do a quick look at what is in the file:

```
:~ # defaults read com.apple.touristd
{
    "seed-https://help.apple.com/osx/mac/10.12/whats-new" = "2016-10-19 11:04:39 +0000";
}
```

A quick `defaults read-type` on that key indicates it's a date value (which is obvious enough from the output, but matters when it comes to applying the preference with `defaults write`).

Sure enough, deleting this file and logging back in causes the notification to appear again. If you want to make sure that the notification isn't presented to the user, you can create the preference with the correct\* key in it.

```
defaults write com.apple.touristd seed-https://help.apple.com/osx/mac/10.12/whats-new -date "$(date)"
```

\*This *may* be different based on your machine type & whether the OS is upgraded or a fresh install.

Looking for the binary that is checking and issuing the notification, we get the following results (truncated output is shown):

```
# find . -type f -iname "tourist*"
...
./System/Library/PrivateFrameworks/Tourist.framework/Versions/A/Resources/touristd
```

A cursory glance at `strings` contained in the binary indicates it can be used at the command line.

```
:Resources # ./touristd --help
Usage: touristd
Options:
  --help                Show this help and exit
  --board-id=ID         Override detected board ID
  --scaling-factor=NUM  Override detected scaling factor (main screen)
  --previous-system=VERSION Override detected previous system version
  --locale=CODE         Override detected locale setting
  --tours=FILE          Path to an alternate list of tour definitions (.plist)
  --menu=NUM            (menu mode) trigger menu select action for item NUM
  --hours              Show hours elapsed since last OS install
  --reset              Clear setting of whether notification was shown for all tours

Commands:
  With no command argument, touristd will run as a daemon.
  profile              Show profile
  status              Show all known tours
  match               Show ranked list of profile matches against known tours
  notify              Show notification for matched tour
  activate            Directly trigger 'action' of notification
  dismiss             Directly trigger 'cancel' of notification
  menu                Show exported Finder menu items
```

It appears that there are a number of different “tours” that can be sent to the user as a notification, based on hardware types, and whether the OS is a fresh install or upgrade. It is also possible to dismiss or notify the user (although if the user has already dismissed or activated the tour, it won't appear again unless the `--reset` flag is called first).

You can determine which tours the system matches against by using the `match` flag; the `activate` flag is used to launch the first “activate” tour; in my testing this launched the “What's new” tour in Safari.



macOS Sierra What's New tour in Safari

Based on further testing, it is not currently possible to use the `touristd` binary to launch other tours (such as the "New to Mac", or a tour specific to the Mac hardware), nor can you just modify the preference domain to force the `activate` flag to launch a different tour, *but* you can launch specific tours through Safari by using `open -a safari` – for example, to launch the tour relating to the 27" iMac, you would use `open -a safari https://help.apple.com/osx/mac/10.12/imac-27`

This is some pretty handy stuff, especially if you're in an environment where you may want to have the user automatically get a tour of new features in macOS, or their Mac device.

As for me, I'm happy that I can finally automate dismissing the notification 😊

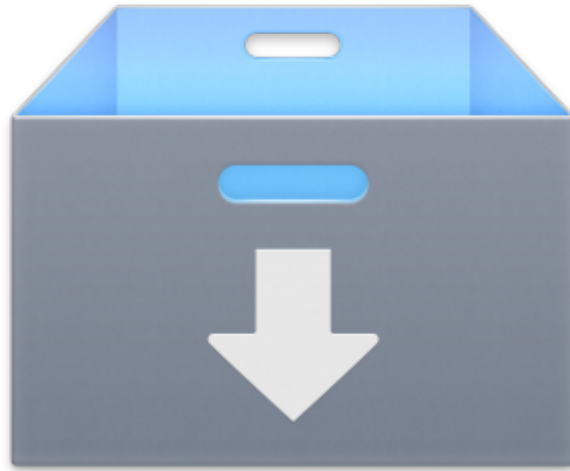


# PRE CACHING MACOS SOFTWARE

10 OCTOBER 2016

CATEGORIES: MAC OS

TAGS: CACHING SERVER, KRYPTED, MAC OS X, MACOS, PRECACHE



Caching Server

I often find myself in a spot where I'm always re-downloading and re-caching iOS IPSW files because they're infrequently being downloaded from Apple. This is pretty frustrating when it comes time to downloading macOS installers, iWork/iLife apps, or IPSW's for the infrequent restoring of an iOS device in Apple Configurator. This is a feature sorely lacking in Apple's macOS Caching Server; so until that feature is implemented, we make do...

Charles from [krypted.com](http://krypted.com) has a great post about how to pull cacheable assets through the Caching Server, so using this method, we can figure out where those Mac App Store apps are coming from, and pre cache those!

Lets take a look at the Caching Server logs found at `/Library/Server/Caching/Logs/` – specifically `Service.log`

```
2016-10-10 10:14:45.829 #/lsyeQGL0tvE Received GET request by "MacAppStore/2.2" for /apple-assets-us-std-000001/Purple62/v4/ed/3d/8e/ed3d8e87-09da-2272-fc3a-b1678d8067a0/iyp5743666419479406275.pkg
2016-10-10 10:15:45.627 #E3oGY2esLpZE Received GET request by "itunesstored/1.0" for /apple-assets-us-std-000001/Purple20/v4/57/44/6b/57446bcb-a3b7-46ae-3307-788a3b5ef280/pre-thinned2492386531421197760.thinned.signed.dpkg.ipa
2016-10-10 10:15:50.174 #E3oGY2esLpZE Served all 2.9 MB of 2.9 MB; 0 bytes from cache, 2.9 MB stored from Internet, 0 bytes from peers
2016-10-10 10:17:35.870 #ziqCwn3ZEdaC Received GET request by "itunesstored/1.0" for /iOS7.1/031-0753.20140310.D1cKf/com_apple_MobileAsset_DictionaryServices_dictionary2/274d8f957d6bd64df440645d851c0dbd4deea358.zip
```

```
2016-10-10 10:17:42.328 #ziqCwn3ZEaC Served all 28.0 MB of 28.0 MB; 28.0 MB from cache,  
0 bytes stored from Internet, 0 bytes from peers
```

In the example above, we see that the Caching server has received a `GET` request from a Mac App Store client, and there is a package file being requested; the URL however is incomplete.

Looking at active requests on our proxy server at the time the request went through, the complete requested URL is found:

```
by kid3 {  
...  
uri http://osxapps.itunes.apple.com/apple-assets-us-std-000001/Purple62/v4/ed/3d/8e/ed3d8  
e87-09da-2272-fc3a-b1678d8067a0/iyp5743666419479406275.pkg  
...  
}
```

Bingo!

Now we can pull Mac App Store apps through the Caching Server, all we need to do is watch the Caching Server `service.log` file for any requests for items that aren't cached. The package URL's are not human readable, so basically this involves watching the logs with something like `tail -f /Library/Server/Caching/Logs/Service.log` or watching the Caching Server service log in the Server app.

Here's some URL's I've grabbed so far (these have all been downloaded on macOS Sierra 10.12.0\* and are "correct" as at 2016-10-10) – you can use `curl` or `wget` to grab these through your Caching Server:

\*The iWork apps are macOS Sierra specific.

Pages 6.0

```
http://cache_server_url:PORTNUM/apple-assets-us-std-000001/Purple62/v4/8a/ee/6e/8aee6e8b-  
e8cb-2434-b050-31dbbcc01974/daf974703926683564923.pkg?source=osxapps.itunes.apple.com
```

Keynote 7.0

```
http://cache_server_url:PORTNUM/apple-assets-us-std-000001/Purple71/v4/a6/96/42/a696423f-  
181c-fc2b-b572-3d3697146d47/h1z1727390940373748952.pkg?source=osxapps.itunes.apple.com
```

Numbers 4.0

```
http://cache_server_url:PORTNUM/apple-assets-us-std-000001/Purple71/v4/69/02/32/69023287-  
bd7a-ef14-0424-234d8fc589e4/mto6541029270492763328.pkg?source=osxapps.itunes.apple.com
```

GarageBand 10.1.2

```
http://cache_server_url:PORTNUM/apple-assets-us-std-000001/Purple30/v4/19/78/8b/19788bde-  
3172-3b98-8300-b8c4a9458bae/iat2506504784673372233.pkg?source=osxapps.itunes.apple.com
```

iMovie 10.1.2

```
http://cache_server_url:PORTNUM/apple-assets-us-std-000001/Purple20/v4/80/6d/9b/806d9b4e-
```

```
776c-baae-574c-ed8afbc70acb/gyj6237528809531298180.pkg?source=osxapps.itunes.apple.com
```

Xcode 8.0

```
http://cache_server_url:PORTNUM/apple-assets-us-std-000001/Purple62/v4/ed/3d/8e/ed3d8e87-09da-2272-fc3a-b1678d8067a0/iyp5743666419479406275.pkg?source=osxapps.itunes.apple.com
```

macOS Server 5.2

```
http://cache_server_url:PORTNUM/apple-assets-us-std-000001/Purple62/v4/44/71/01/44710118-b2c9-1e31-73f6-fa7a0a26e594/wjs7031774084062486733.pkg?source=osxapps.itunes.apple.com
```

Substitute the `cache_server_url` for your Caching Server address, and the `PORTNUM` for the Caching Server port number, which can be found by running `sudo serveradmin fullstatus caching`. This process can be done for any of the Mac App Store apps, however there are a couple of *small gotcha's*:

- iOS 9 introduced App Thinning, this makes it difficult to pre cache iOS apps, as there are now many different downloadable assets for the one app.
- These URL's may change any time there is an app update.
- Downloaded packages are not installable from the GUI or the CLI as they are encrypted.

You can grab Charles' [precache.py](#) from [here](#). This utility will pre cache iOS, watchOS, and tvOS Over the Air (OTA) updates, as well as the IPSW files for the same OS's, it will also pre cache macOS installers for Mountain Lion through to current macOS Sierra release.

There is also the [CacheWarmer](#) utility available.

**\*\* Note \*\***

So, for all those thinking to themselves, *"Hey Carl, that's cool, but why don't I just download them from the Mac App Store when they come out?"*

Well, that's an entirely reasonable approach to take, until you're in a situation where you need to make sure these apps *stay* in your Caching Server. By creating a script to download these apps, you can guarantee that they'll be there when you need them. It also makes it pretty handy to get a Caching Server ready again after having to reset the cache after an asset becomes corrupted (which does happen), or in instances where Caching Server decides to [clear the cache](#) when it can't contact Apple for registration.

# GARAGEBAND AUDIO

1 JUNE 2016

CATEGORIES: MAC OS

TAGS: GARAGEBAND, LOOPS, MAC OS X, PACKAGING



Deploying GarageBand audio content

On the 16th of May, Apple released an updated version of GarageBand, and shortly after, updated the “core content” that is downloaded the first time GarageBand is launched.

If you run a managed Mac environment, such as classrooms/labs, you usually want to avoid end users going through the process of downloading this content, typically you build packages using a tool like [Iceberg](#), or by using tools like [AutoPKG](#) and then deploy them to the Mac/s. In this post, I'll walk through how I've packaged the “core audio content” for deployment.

First, you'll need to get the loops. There are *many* different ways to get this content, such as using [Charles Proxy](#), or you can use a nifty command line tool such as the one I created, [available here](#).

Previous versions of GarageBand 10 required content from the 2013 and 2015 “release” years. GarageBand 10.1.2 doesn't; it only requires the content released in 2016. All testing has been done with Mac OS X 10.11.4, 10.11.5, and fresh installs of GarageBand 10.1.2.

To grab the 2016 content, I run :

```
./get_audio_content.py -y 2016 -p garageband -o ~/Desktop/loops/
```

This grabs the content for GarageBand, and saves it within the loops folder on my desktop. At time of writing, there are 91 packages to download, totalling approximately 9.2GB, so if you've got a slow Internet connection, you might want to do this overnight.

Once the content is downloaded, you can then use your favourite packaging tool to bundle the packages for deployment. This may be [munki](#), [Casper](#), or other similar deployment tool.

**\*\* Update June 2, 2016 \*\***

It appears that in some circumstances, if you package the loops by wrapping them in any type of package, and use a postinstall script, some or all packages do not install properly.

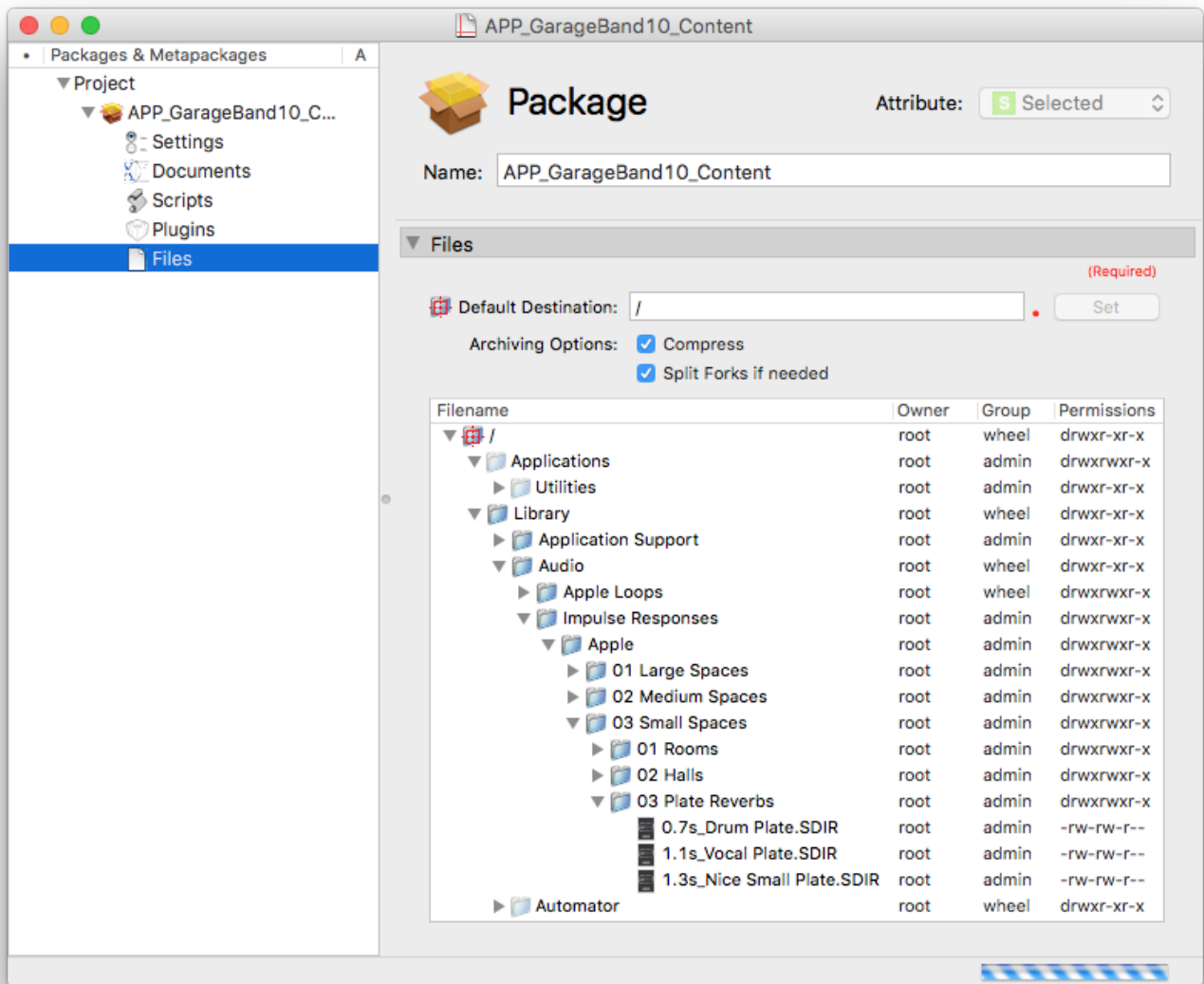
`/var/log/install.log` indicates this with the following error:

```
Jun  2 10:19:45 mithrandir installer: PackageKit: Install Failed: (null) (null)
```

These packages install correctly when a simple `for` loop is called directly from the command line.

I've fallen back to the snapshot method, using [logGen](#) to take a pre install and post install snapshot of the OS, and then used [pkgGen](#) to gather all the files that it finds have been added or changed; then building a package that drops those files directly in place, with the correct permissions.

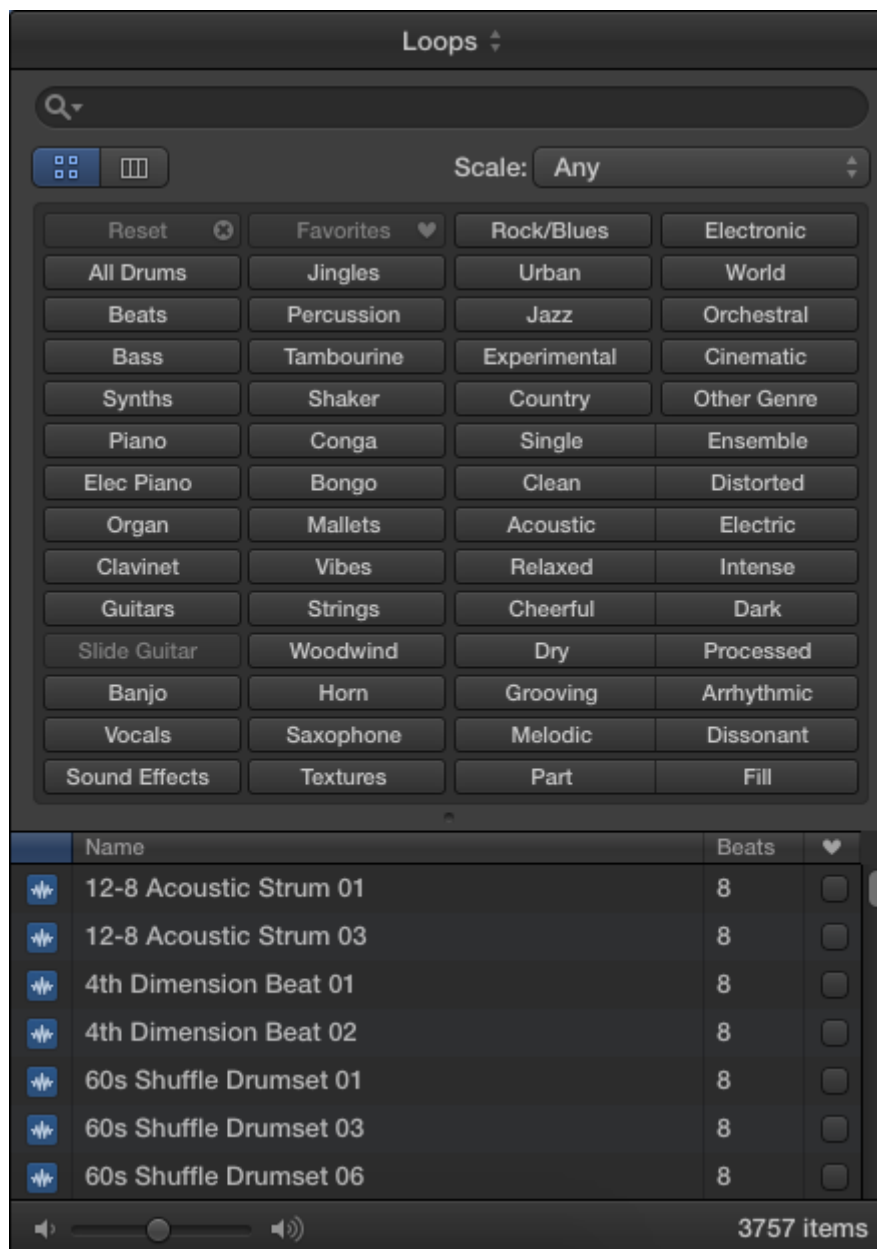
You can see in the screenshot below an example of building the package – this includes all the 2013-2016 content.



GarageBand 10.x content package build

Test the package on a virtual machine, or other test setup, to verify the install process works correctly. Any errors from the install run are found in `/var/log/install.log`.

When the package has passed deployment tests, move the package into your deployment workflows, and you're ready to rock on; GarageBand won't prompt to download the content, and the loops will be indexed the first time the app is launched, this index is created *per user* and will re-index anytime new loops are added to GarageBand.



GarageBand 10.1.2 with 2016 audio content installed

\*\* Update September 2, 2016\*\*

Morgs from #anzmac in the [macadmins](#) Slack group has recently pointed me towards this [excellent post](#) by Alan Siu; it covers how to import these packages into [munki](#) without having to create a single installer as I've outlined here. I'm in the process of moving the deployment system in my workplace to munki, so I'll be sure to use this process in my deployment, rather than building a single monolithic package.

Incidentally, if you haven't already, I highly recommend joining the 6600+ strong macadmin community on Slack. You'll find plenty of help from fellow Mac admin's around the world, as well as make new friends!

I also recommend subscribing to the [macadmins podcast](#).

