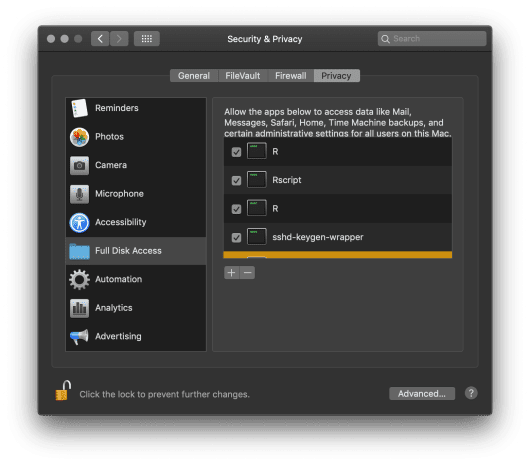
So, we’ll cover just how to do that (find things in iOS backups) in this post along with how to deal with some “gotchas” if you’re doing this from macOS.

**macOS (the Knife)**

I bring that up to talk about the cutting, biting, dangerous edge of macOS that is Apple’s somewhat mixed attempt at protecting your privacy and keeping out of sight sensitive files and directories from the the sharp teeth of malware (and to re-pimp my {mactheknife} package) . You can read up on Apple’s new protections more thoroughly over at [The Eclectic Light Company](https://eclecticlight.co/?s=sandbox). For the purposes of this blog post, Apple’s macOS Sandbox policies means you have to do some extra steps to gain access to the folder and files associated with iOS backups (which is ~/Library/Application Support/MobileSync/Backup/).

If you want RStudio, R, and anything run with Rscript to access these sandboxed areas you’ll need to enable “Full Disk Access” those apps and executables. First you’ll need to open System Preferences > Security & Privacy and then make the Privacy tab active. Keep that window open and tap the lock to unlock the settings.

[](https://i1.wp.com/rud.is/b/wp-content/uploads/2019/06/full-disk-access.png?ssl=1)

Full Disk Access settings panel in macOS

Adding RStudio is easy. Just make Finder active and hit Cmd Shift A and then find and drag the “RStudio” application into the pane+tab you opened in the previous step. Back in the Finder, hit Cmd Shift G and paste in: /Library/Frameworks/R.framework/Resources/bin and go to that folder. Drag in R and Rscript each into the pane+tab from the aforementioned step. Finally (and this got me for a minute) you *also* need to (again, in Finder) hit Cmd Shift G and paste in /Library/Frameworks/R.framework/Versions/3.6/Resources/bin/exec and drag *that R executable* into the Security & Privacy Privacy/Full Disk Access pane+tab as well. When you’ve done all that, lock the System Preferences pane and close it.

It is important to note that you just gave “R” and anything that calls R from your user space complete (well, *almost*) access to every sandboxed area on your system. R is a general purpose programming and scripting language which means any bit of malicious code that knows you have added those executables can use R to read from and write to any area on your system.

It is also important to note that I had to use 3.6 vs the Current symlink for the last entry so that means you need to do this for each new R version you install.

I hope folks on legacy Windows OS installs didn’t skip over this part as you’ll need to go [here](http://bfy.tw/Nvfc) to figure out where your iOS backups folder is to go through the rest of the post.

**Sneakin’ Round The Corner**

Windows folks hopefully read at least the last bit of the previous section to figure out where their iOS backups are. On macOS that’s ~/Library/Application Support/MobileSync/Backup/. You need a local backup there (most folks just use iCloud backups these days) and [Apple tells you how to do this](https://support.apple.com/en-us/HT203977).

Once you know you’ve got an (**unencrypted**) backup just go to your iOS backups directory and list the files by date and note the name/path of the most recent backup. Now we can have some fun.

library(XML) # to read plist (property list) files

library(tidyverse) # for printing and access to sqlite dbs

# replace this with the relative path to your most recent backup dir

mb <- "~/Library/Application Support/MobileSync/Backup/28500cd31b9580aaf5815c695ebd3ea5f7455628-20190601-165737"

list.files(mb, pattern = ".\*\\.(db|plist)$")

## [1] "Info.plist" "Manifest.db" "Manifest.plist" "Status.plist"

The above code looks for some key metadata files for iOS backups.

* Info.plist has info on your device
* Manifest.db has tons of info on all the files in the backup in a SQLite database
* Manifest.plist has some additional metadata on the backup including applications included in the backup
* Status.plist contains info on the status of the backup

Let’s take a look at the plists:

info\_p <- file.path(mb, "Info.plist")

file.copy(info\_p, "/tmp", overwrite = TRUE)

system2("plutil", args=c("-convert", "xml1", "-o", "/tmp/Info.plist", "/tmp/Info.plist"))

info <- XML::readKeyValueDB("/tmp/Info.plist")

str(info)

## List of 11

## $ Device Name : chr REDACTED

## $ Display Name : chr REDACTED

## $ ICCID : chr REDACTED

## $ IMEI : chr REDACTED

## $ IPBE Backup Version: int 1

## $ Last Backup Date : POSIXct[1:1], format: "2019-06-01 21:23:02"

## $ Phone Number : chr REDACTED

## $ Product Type : chr REDACTED

## $ Product Version : chr REDACTED

## $ Serial Number : chr REDACTED

## $ Unique Identifier : chr REDACTED

status\_p <- file.path(mb, "Status.plist")

file.copy(status\_p, "/tmp", overwrite = TRUE)

system2("plutil", args=c("-convert", "xml1", "-o", "/tmp/Status.plist", "/tmp/Status.plist"))

status <- XML::readKeyValueDB("/tmp/Status.plist")

str(status)

## List of 6

## $ BackupState : chr "new"

## $ Date : POSIXct[1:1], format: "2019-06-01 21:22:53"

## $ IsFullBackup : logi FALSE

## $ SnapshotState: chr "finished"

## $ UUID : chr REDACTED

## $ Version : chr "3.3"

mainf\_p <- file.path(mb, "Manifest.plist")

file.copy(mainf\_p, "/tmp", overwrite = TRUE)

system2("plutil", args=c("-convert", "xml1", "-o", "/tmp/Manifest.plist", "/tmp/Manifest.plist"))

manifest <- XML::readKeyValueDB("/tmp/Manifest.plist")

str(manifest, 1)

## List of 8

## $ Applications :List of 745

## $ BackupKeyBag : chr \_\_truncated\_\_

## $ Date : POSIXct[1:1], format: "2019-06-01 20:57:40"

## $ IsEncrypted : logi FALSE

## $ Lockdown :List of 12

## $ SystemDomainsVersion: chr "24.0"

## $ Version : chr "10.0"

## $ WasPasscodeSet : logi TRUE

You’ll note we’re making copies of these files (never play with system-managed files directly unless you know what you’re doing) and turning binary property lists into plain text XML property lists as well so we can read them with the XML::readKeyValueDB() function.

Most of that information is fairly useless for this blog post but I figured you might like to see the hidden things the system knows about your devices. What we do want to check is to see if the John Deere application and data made it into the backup. The Applications slot is a names list of application metadata. Let’s see if there’s anything Deere-ish in it:

grep("deere", names(manifest$Applications), ignore.case = TRUE, value = TRUE)

## key

## "com.deere.mowerplus"

str(manifest$Applications$com.deere.mowerplus, 1)

## List of 4

## $ CFBundleIdentifier : chr "com.deere.mowerplus"

## $ CFBundleVersion : chr "180"

## $ ContainerContentClass: chr "Data/Application"

## $ Path : chr "/var/containers/Bundle/Application/30DF2640-A9AA-43A0-AD87-932CA513D75A/MowerPlus.app"

Aye! This means we should have some luck finding “mower” data in the Manifest SQLite database.

Now, we *could* try to follow UUIDs around but we can also take a stab at a less cumbersome approach. Let’s make a copy of the Manifest database and see what it holds:

mainf\_d <- file.path(mb, "Manifest.db")

file.copy(mainf\_d, "/tmp", overwrite = TRUE)

## [1] TRUE

(manifest\_db <- src\_sqlite("/tmp/Manifest.db"))

## src: sqlite 3.22.0 [/private/tmp/Manifest.db]

## tbls: Files, Properties

We want to get to (hopefully) a SQLite file with the mowing data so we likely care about the Files table. Let’s take a look at the structure of that table:

(fils <- tbl(manifest\_db, "Files"))

## # Source: table [?? x 5]

## # Database: sqlite 3.22.0 [/private/tmp/Manifest.db]

## fileID domain relativePath flags file

##

## 1 c1da4199a18d0b5… AppDomain-com… "" 2

## 2 7426ac0386e2887… AppDomain-com… Library 2

## 3 a6393e739e1ad37… AppDomain-com… Library/WebKit 2

## 4 c54f5c77a5e970b… AppDomain-com… Library/WebKit/Websit… 2

## 5 578f2c96f219e95… AppDomain-com… Library/WebKit/Websit… 2

## 6 c8833032ce7c9e9… AppDomain-com… Library/WebKit/Websit… 2

## 7 6af21902e595f7c… AppDomain-com… Library/WebKit/Websit… 2

## 8 4c1c49324646af0… AppDomain-com… Library/WebKit/Websit… 2

## 9 d0636bf9b5ba2ae… AppDomain-com… Library/WebKit/Websit… 2

## 10 0b6bb30c8abaa4e… AppDomain-com… Library/Preferences 2

## # … with more rows

If you have a ton of apps, this is a pretty big haystack to comb through. We may be able to narrow things down a bit, though, and we’ll start by seeing what that domain column holds:

distinct(fils, domain)

## # Source: lazy query [?? x 1]

## # Database: sqlite 3.22.0 [/private/tmp/Manifest.db]

## domain

##

## 1 AppDomain-Outils-OBD-Facile.EOBD-Facile

## 2 AppDomain-ch.threema.iapp

## 3 AppDomain-co.humanco.Human

## 4 AppDomain-co.ortatech.colr-app

## 5 AppDomain-co.vero.app

## 6 AppDomain-com.7thg.Tides

## 7 AppDomain-com.AerLingus

## 8 AppDomain-com.BloomSky.BloomSky

## 9 AppDomain-com.PunchThrough.LightBlue

## 10 AppDomain-com.agilebits.onepassword-ios

## # … with more rows

So, these are app-specific and the bits after the - in each one look like the CFBundleIdentifiers from above. Let’s make sure:

filter(fils, lower(domain) %like% "%com.deere.mowerplus%") %>%

distinct(domain)

## # Source: lazy query [?? x 1]

## # Database: sqlite 3.22.0 [/private/tmp/Manifest.db]

## domain

##

## 1 AppDomain-com.deere.mowerplus

Aye! Let’s check to see what files are in there (and hope for a nice SQLite database):

filter(fils, domain == "AppDomain-com.deere.mowerplus") %>%

select(relativePath) %>%

collect()

## # A tibble: 14 x 1

## relativePath

##

## 1 ""

## 2 Library

## 3 Library/Preferences

## 4 Library/Application Support

## 5 Library/Application Support/com.crashlytics

## 6 Documents

## 7 Library/GoAppPersistentStore-GoMow.sqlite

## 8 Library/googleanalytics-aux-v4.sql

## 9 Library/googleanalytics-v3.sql

## 10 Library/Preferences/com.deere.mowerplus.plist

## 11 Library/Application Support/ActivityCards.sqlite

## 12 Library/Application Support/com.crashlytics/CLSUserDefaults.plist

## 13 Library/googleanalytics-v2.sql

## 14 Library/Application Support/MowTracking.sqlite

It turns out that last one is what we’re looking for. Now we just need a bit of crucial metadata to get to it:

filter(fils, relativePath == "Library/Application Support/MowTracking.sqlite") %>%

select(fileID, relativePath)

## # Source: lazy query [?? x 2]

## # Database: sqlite 3.22.0 [/private/tmp/Manifest.db]

## fileID relativePath

##

## 1 ad0009ec04c44b544d37bfc7ab343869… Library/Application Support/MowTrackin…

That fileID maps to the seriously ugly directory tree that is the rest of the iOS backups folder (you likely looked into it and wondered “What the heck?!”). The top level is a 2-digit hex prefix with files underneath it (likely for performance reasons but a bit of obfuscation never hurts, too). We’ll get the whole string:

filter(fils, relativePath == "Library/Application Support/MowTracking.sqlite") %>%

select(fileID)

## # Source: lazy query [?? x 1]

## # Database: sqlite 3.22.0 [/private/tmp/Manifest.db]

## fileID

##

## 1 ad0009ec04c44b544d37bfc7ab3438697d23d618

and, then copy over the mowing database somewhere safe to work on:

file.copy(

file.path(mb, "ad/ad0009ec04c44b544d37bfc7ab3438697d23d618"),

"/tmp/mowtrack.sqlite",

overwrite = TRUE

)

## [1] TRUE

mow <- src\_sqlite("/tmp/mowtrack.sqlite")

mow

## src: sqlite 3.22.0 [/private/tmp/mowtrack.sqlite]

## tbls: Z\_METADATA, Z\_MODELCACHE, Z\_PRIMARYKEY, ZACTIVITY, ZDEALER,

## ZMOWALERT, ZMOWER, ZMOWLOCATION, ZSMARTCONNECTOR, ZUSER

tbl(mow, "ZMOWLOCATION") %>%

glimpse()

## Observations: ??

## Variables: 16

## Database: sqlite 3.22.0 [/private/tmp/mowtrack.sqlite]

## $ Z\_PK 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 1…

## $ Z\_ENT 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8,…

## $ Z\_OPT 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,…

## $ ZISPAUSEDPOINT NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, …

## $ ZORDER 1, 2, 0, 11, 20, 58, 38, 43, 30, 25, 21, 10,…

## $ ZSESSION 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,…

## $ ZSESSION2 NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, …

## $ ZALTITUDE 42.64804, 42.70590, 40.99661, 39.54770, 38.2…

## $ ZCOURSE 358.242188, 332.226562, 18.281250, 260.85937…

## $ ZHORIZONTALACCURACY 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,…

## $ ZLATITUDE 43.25913, 43.25914, 43.25913, 43.25915, 43.2…

## $ ZLONGITUDE -70.80069, -70.80069, -70.80069, -70.80067, …

## $ ZSPEED 0.0000000, 0.4250179, 0.5592341, 0.3802792, …

## $ ZTIMESTAMP 581100271, 581100272, 581100270, 581100281, …

## $ ZVERTICALACCURACY 6, 6, 8, 6, 4, 4, 4, 3, 4, 4, 4, 6, 4, 4, 4,…

## $ ZKLVDATA , , , , , , , <…

**(Shark) FIN**

Even if you don’t have this mower app that I’m currently obsessed with, you now have a primer on how to get to SQLite databases stored by any application on your iOS device. That alone may unearth some fun projects for you to hack on. Plus, you also learned a bit on how to do some light forensics on iOS backups with R/RStudio.

If you did your own trawling and found something interesting definitely blog or tweet about it and drop a link in the comments.