

# macOS Catalina Artifact Exporter

Stable Build: <https://github.com/andrewbluepiano/macOS-CatalinaForensicsTool/releases>

## Overview

### The simple description:

The tool is an Objective-C (Cocoa) interface that manages calls to AppleScript functions.

### The advanced description:

By default, macOS is designed to prevent applications from gaining privileged access to the filesystem. Gatekeeper and sandboxing are requirements for any application being submitted to Apple for distribution on the app store. Because of this, there are few forensic GUI applications using native Apple's application format, .app, which would either be a Swift or an Objective-C application. Due to this, the few forensic applications ever published, or applications designed to perform actions with root privileges in general, are sparse, generally outdated and no longer maintained, and deprecated.

The NIST Tool catalog only lists 4 Suites for Mac, the most recent of which is from 2018, and only supports up to macOS Mojave.<sup>1</sup> Seemingly with no examples to turn to as to how to provide a native GUI for managing the collection of forensic artifacts much less one that can use escalated privileges, luckily an Apple engineering application for collecting system information to diagnose problems shed some light on how our goal could be accomplished;

"CaptureData.app". As shown in our presentation, some basic forensic analysis on the app shows that its executable is incredibly small for an app that collects over 10 types of detailed information. Continued inspection revealed that the actual data collection was being done by an AppleScript file. The executable was simply a GUI managing the calls to the AppleScript.

A bit of digging on the web will show that Apple has created ways to incorporate AppleScript into many modern languages, including Objective-C via AppleScriptObjC. Although the functions in the AppleScript file for CaptureData.app were purely diagnosis focused, not particularly helpful for forensics, it did provide a rough outline of how things could be implemented. It also shows how one can run commands with administrative privileges via an Objective-C GUI, albeit taking a step through AppleScript first.

First, by removing all Gatekeeper and Sandboxing settings from our app, we are able to bypass those two settings that prevent privilege escalation. Although this means the app can never be submitted to the app store, and that users must first run a command on the .app file to allow macOS to let it run, this is not a problem for an open source app where we can assume the end user will be ready to do such things, and check what is being done before running the program.

Second, by having the user enter their password into a field that sends it to a property (Variable) in the AppleScript, we are able to keep a nice automated run of multiple administrative actions, without having to prompt the user for permission each time.

---

<sup>1</sup> [https://toolcatalog.nist.gov/search/index.php?all\\_tools=refine&ff\\_id=16&1%5B%5D=3](https://toolcatalog.nist.gov/search/index.php?all_tools=refine&ff_id=16&1%5B%5D=3)

Third, by using AppleScript, we can make calls to other macOS native applications, such as “System Events” and “Finder”, which after prompting for the users permission and password to do so, can run functions of their own with their native system privileges. Combining all of these elements gives us the base functionality we need for a forensic application.

### **Significance in the Forensics Field**

There is currently a lack of forensics suites that are designed specifically for the macOS operating system. Some full professional suites offer some specific functionalities for analyzing and extracting information from macOS images, but many have not been updated with each macOS release, and with each release more of their features are unable to operate. In October 2019 Apple released macOS 10.15 “Catalina”, that made significant changes to macOS. This update moved all protected system information to a read only /data partition, protected by macOS’ SIP. Given that there are few Apple forensics tools already, it is not surprising that there are none that offer targeted functionality to macOS Catalina. By creating an open source forensic toolkit designed specifically for Apple’s Catalina operating system, hopefully we can create a basis for others to add on functionality relevant to the modern OS.

By designing an application in Apple’s native .app format, using technologies that are included and consistent across all macOS Catalina systems, we ensure maximum compatibility and functionality of the tool, and hopefully future-proof it as well.

## **Description of Functions / Demonstrations**

### **System Profile**

~Method Name:

systemProfile

~Overview:

This function provides general configuration information for both the hardware and software of the system by generating a system profiler report. This is the equivalent of clicking the [Apple Logo in Menu Bar] -> About This Mac -> System Report...

~Method Implementation:

Collects the most detailed level of information, outputs to a .spx file so that SystemInformation.app can display it in a nice and easily readable format.

~Method Operation:

Runs command: `system_profiler -detailLevel full -xml > SystemProfile.spx`

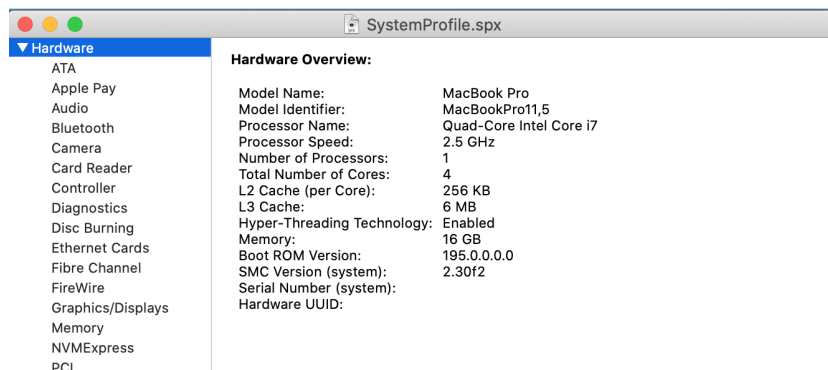
~Relevant Information Gathered in this Report:

- System Hardware Information
  - Serial Number
  - Hardware UUID

- Bluetooth Information
  - System bluetooth information
    - Name, MAC Address, Firmware Version, Discoverable (y/n), Connectable (y/n), Auto Seek Pointing (y/n), Remote wake (y/n), Auto Seek Keyboard (y/n)
  - Bluetooth device information:
    - MAC Address, Services, Paired (y/n), Connected (y/n), Manufacturer,
  - Bluetooth Service's Information
    - Bluetooth File Transfer
    - Bluetooth File Exchange
    - Bluetooth Internet Sharing
- Power Information
  - System Power Settings:
    - Wake on LAN (y/n), PrioritizeNetworkReachabilityOverSleep (y/n), TCPKeepAlivePref (y/n)
  - Scheduled events triggered by system power status changes
    - appPID, service scheduled by
- Mounted/Attached Volumes (local & network) & Physical Data Storage Devices
  - Mount point, filesystem format, ignore Ownership (y/n), BSD Name, Volume UUID, Protocol,
- Network adapters & information
  - Addressing, Proxies, BSD Device Name, Routes, Priority
- Firewall settings
  - Operation Mode, Manually configured settings per application, Firewall Logging (y/n),
- Currently detected WiFi networks & detailed information about their signals
- Currently 'installed' applications & their details
  - Highlighted details: Version, Obtained from (publisher), Last Modified Date, Signing Authority, Location on disk
  - 'Installed' seems to mean any .app on the disk.
- Currently installed kernel extensions (.kexts) & their details
  - Highlighted details: Version, Last Modified Date, Bundle ID, Notarized (y/n), Loaded (y/n), Get Info String, Obtained from (publisher), Location on disk, Signing Authority
- Currently installed Frameworks & their details (Frameworks are shared libraries)
  - Highlighted details: Version, Obtained from (publisher), Last Modified Date, Signing Authority, Location on disk, Private (y/n)
- "Installations" & some details
  - Unclear what this category encompasses, besides obviously being installed software. Perhaps a log from "installer.app".
  - Only information is title, install date, and publisher
- Legacy Software (mainly 32-bit applications or executables) & details
  - Highlighted details: Reason for being legacy, Last used date, Path on disk, Parent process, Publisher and Signing information where applicable.

- Logs\* & details
  - \*Only a certain amount of recent data from each log is included in SystemProfile.spx
  - Log path
  - Log size
  - Log last modified date
  - Included Logs (From tests on our machines)
    - Apple System Log (ASL) Messages
    - Application firewall log
    - Disk recording log
    - Filesystem repair log
    - Installer log
    - IORegistry Content
    - Kernel log
    - nvram content
    - Panic (system crashes) log(s)
    - Power management log
    - Printer access log
    - User filesystem repair log
    - VPN, PPPoE and PPP (modem) log
    - WiFi log
- Device management enrollment information
- SmartCard information
- Contents of /Library/StartupItems/
  - Not appropriate to call this section startup items, as this folder is far from an accurate representation of what is set to run at start.
- Sync Services (logs) & details
  - Similar to Log section above

Figure 1. Sample output of systemProfile



## Unified Logs

~Method Name:

getUnifLogs

~Overview:

Starting with macOS 10.12, the main format of logging used by the system is the unified log. Unified logs provide information on the log data in both memory and on the disk.

~Method Implementation:

Runs a shell command to get the entirety of the unified log, outputs to a .logarchive file viewable by Console.app

~Method Operation:

Runs command: `sudo log collect --output unifLogs.logarchive`

~Information Gathered in this Report:

The entirety of the unified log from when it was implemented in 2016

~Links:

<https://developer.apple.com/documentation/os/logging>

Figure 2. Sample output of getUnifLogs

All Messages			
Errors and Faults			
Type	Date & Time	Process	Message
	2019-11-23 17:12:04.01606...	com.ap...	CDN - initialize client context retry
	2019-11-23 17:12:04.01607...	com.ap...	CDN - client setup_remote_port
	2019-11-23 17:12:04.016061-0500	CDN	CDN - Bootstrap Port: 5123
	2019-11-23 17:12:04.01638...	kernel	Sandbox: com.apple.WebKit(2819) deny(1) mach-lookup com.apple.CoreDisplay.Notification
	2019-11-23 17:12:04.01645...	com.ap...	CDN - setup_remote_port: bootstrap_look_up() => unknown error code (1100)
	2019-11-23 17:12:04.02588...	hidd	[HID] [MT] dispatchEvent Dispatching event with 2 children, _eventMask=0x42 _childEventMask=0x42 Cancel=0 Touching=0 inRange=1
	2019-11-23 17:12:04.05171...	diskar...	<private>
	2019-11-23 17:12:04.05171...	diskar...	<private>
	2019-11-23 17:12:04.05172...	diskar...	<private>
	2019-11-23 17:12:04.05252...	diskar...	<private>
	2019-11-23 17:12:04.06409...	diskar...	<private>
	2019-11-23 17:12:04.06411...	diskar...	<private>
	2019-11-23 17:12:04.06411...	diskar...	<private>
	2019-11-23 17:12:04.06486...	diskar...	<private>
	2019-11-23 17:12:04.62765...	hidd	[HID] [MT] dispatchEvent Dispatching event with 2 children, _eventMask=0x63 _childEventMask=0x62 Cancel=0 Touching=1 inRange=1
	2019-11-23 17:12:04.85766...	dasd	Attempting to suspend based on triggers: ( "com.apple.duetactivityscheduler.photospolicy.appchanged" )
	2019-11-23 17:12:04.85778...	dasd	Ignoring trigger com.apple.duetactivityscheduler.photospolicy.appchanged because conditions are deteriorating
	2019-11-23 17:12:04.85778...	dasd	Evaluating 0 activities based on triggers
	2019-11-23 17:12:05.10685...	com.ap...	Current memory footprint: 123 MB
	2019-11-23 17:12:05.11774...	hidd	[HID] [MT] dispatchEvent Dispatching event with 2 children, _eventMask=0x42 _childEventMask=0x42 Cancel=0 Touching=0 inRange=1
	2019-11-23 17:12:05.39569...	hidd	[HID] [MT] dispatchEvent Dispatching event with 2 children, _eventMask=0x63 _childEventMask=0x62 Cancel=0 Touching=1 inRange=1
	2019-11-23 17:12:05.71129...	hidd	[HID] [MT] dispatchEvent Dispatching event with 2 children, _eventMask=0x23 _childEventMask=0x3 Cancel=0 Touching=0 inRange=0
	2019-11-23 17:12:05.83811...	hidd	[HID] [MT] dispatchEvent Dispatching event with 2 children, _eventMask=0x61 _childEventMask=0x62 Cancel=0 Touching=1 inRange=1
	2019-11-23 17:12:06.02155...	hidd	[HID] [MT] dispatchEvent Dispatching event with 4 children, _eventMask=0x63 _childEventMask=0x262 Cancel=0 Touching=1 inRange=1
Showing: Last 5 Minutes			

--

## **Installation History:**

~Method Name:

getInstallHist

~Overview:

Backs up /Library/Receipts/InstallHistory.plist, which is a log of all<sup>2</sup> software & update installations to the system.

~Method Implementation:

When making copies of information for forensic analysis on macOS using the 'cp' command, you must use the -p flag to preserve the metadata of the file<sup>3</sup>:

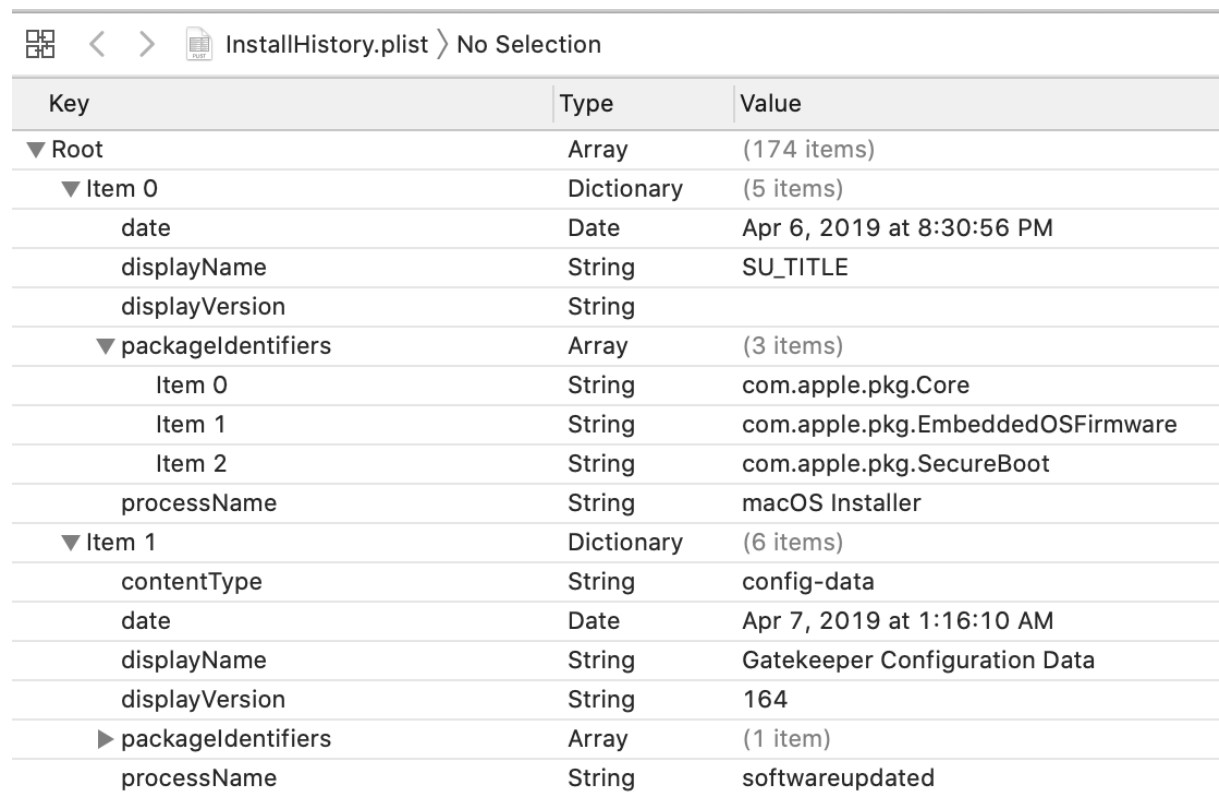
~Method Operation:

Runs command: `cp -p /Library/Receipts/InstallHistory.plist /OutputLocation`

~Relevant Information Contained:

Installation date, name displayed to user during installation, package identifier strings, name of process that called the installation.

Figure 3. Sample output of getInstallHist



Key	Type	Value
▼ Root	Array	(174 items)
▼ Item 0	Dictionary	(5 items)
date	Date	Apr 6, 2019 at 8:30:56 PM
displayName	String	SU_TITLE
displayVersion	String	
▼ packageIdentifiers	Array	(3 items)
Item 0	String	com.apple.pkg.Core
Item 1	String	com.apple.pkg.EmbeddedOSFirmware
Item 2	String	com.apple.pkg.SecureBoot
processName	String	macOS Installer
▼ Item 1	Dictionary	(6 items)
contentType	String	config-data
date	Date	Apr 7, 2019 at 1:16:10 AM
displayName	String	Gatekeeper Configuration Data
displayVersion	String	164
▶ packageIdentifiers	Array	(1 item)
processName	String	softwareupdated

<sup>2</sup> Rarely, there will be an installation missing from this file which can be seen using the 'pkgutil' command, but the 'pkgutil' command does not contain as much information about installations as this file, so they are not replacements for one another.

<sup>3</sup> See / run command 'man cp'

## **FSEvents Parsing**

~Method Name:

fsEventsParse

~Dependencies:

FSEventsParser/FSEParser\_V4

FSEventsParser/report\_queries.json



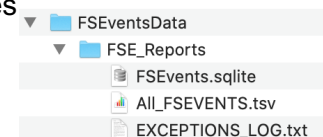
~Overview:

FSEvents is an API in macOS that monitors and records all changes to the filesystem. The information is written to a FSEvents database file named .fseventsd that is stored at the root of the drive it contains logs from. This helps with understanding records that have been written to a specific volume in memory. There is potential for a user to be able to disable the FSEvents api in theory, but it is extremely unlikely, as in addition to most users being unaware of the service, it would be disruptive to the operation of macOS and its recovery operations.

~Method Implementation:

The implementation of this function is unique in a few ways.

- As there is no native way on macOS to parse this database as a user, the program relies on using David Cowen's FSEventsParser application to parse the .fseventsd database to a sqlite database, and tsv file.
  - This requires including a unix executable static binary in the Cocoa application, then calling it using the shell, from AppleScript.
  - That means that this function offers an example for others who fork the application that they can use to take their existing binaries used for forensics, and implement them easily into the GUI.



~Method Operation:

Runs Command: `sudo ./FSEParser_V4 -s /.fseventsd/ -o /outputfolder -t folder`

~Relevant Information Contained:

- Full path of file
- Filename
- Whether a file or folder event
- List of specific event(s) that occurred
  - Created, Modified, Renamed, Removed, InodeMetaMod, ExtendedAttrModified, FolderCreated, PermissionChange, ExtendedAttrRemoved, FinderInfoMod, DocumentRevisioning, Exchange, ItemCloned, LastHardLinkRemoved, Mount, Unmount, EndOfTransaction
- File mask
- Node id
- record\_end\_offset
- time of change (microsecond)

~Links:

<https://github.com/dlcowen/FSEventsParser>

<https://www.crowdstrike.com/blog/using-os-x-fsevents-discover-deleted-malicious-artifact/>

Figure 4. Sample output of the fsEventsParse.

	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
180 9	0000000000f9...	.OSInstallSandboxPath/Root/System/Lib...	HelveticaNeue.strings	FileEvent;	Renamed;	Unknown	0x08008000	115292150031...	143860	/.fseve...	2019.10.29 19:...
181 3	0000000000f9...	System/Library/Templates/Data/Library...	PreferredFonts.strings	FileEvent;	Renamed;	Unknown	0x08008000	115292150031...	427901	/.fseve...	2019.10.29 19:...
182 2	0000000000f9...	System/Library/Templates/Data/Library...	PreferredFonts.strings	FileEvent;	Renamed;	Unknown	0x08008000	115292150031...	427694	/.fseve...	2019.10.29 19:...
183 1	0000000000f9...	.OSInstallSandboxPath/Root/System/Lib...	PreferredFonts.strings	FileEvent;	Renamed;	Unknown	0x08008000	115292150031...	144094	/.fseve...	2019.10.29 19:...
184 3	0000000000f9...	.OSInstallSandboxPath/Root/System/Lib...	no.lproj	FolderEvent;	Removed;	Unknown	0x02000001	115292150031...	151088	/.fseve...	2019.10.29 19:...
185 5	0000000000f9...	System/Library/Templates/Data/Library...	Localizable.strings	FileEvent;	Renamed;	Unknown	0x08008000	115292150031...	435917	/.fseve...	2019.10.29 19:...
186 4	0000000000f9...	System/Library/Templates/Data/Library...	Localizable.strings	FileEvent;	Renamed;	Unknown	0x08008000	115292150031...	435724	/.fseve...	2019.10.29 19:...
187 3	0000000000f9...	.OSInstallSandboxPath/Root/System/Lib...	Localizable.strings	FileEvent;	Renamed;	Unknown	0x08008000	115292150031...	151308	/.fseve...	2019.10.29 19:...
188 0	0000000000f9...	.OSInstallSandboxPath/Root/System/Lib...	TextStyles	FolderEvent;	Removed;	Unknown	0x02000001	115292150031...	151519	/.fseve...	2019.10.29 19:...
189 7	0000000000f9...	System/Library/Templates/Data/Library...	HelveticaNeue.strings	FileEvent;	Renamed;	Unknown	0x08008000	115292150031...	436329	/.fseve...	2019.10.29 19:...
190 5	0000000000f9...	System/Library/Templates/Data/Library...	HelveticaNeue.strings	FileEvent;	Renamed;	Unknown	0x08008000	115292150031...	436123	/.fseve...	2019.10.29 19:...
191 5	0000000000f9...	.OSInstallSandboxPath/Root/System/Lib...	HelveticaNeue.strings	FileEvent;	Renamed;	Unknown	0x08008000	115292150031...	151752	/.fseve...	2019.10.29 19:...

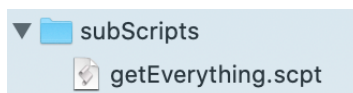
## MetaData Backup

~Method Name:

getMetaData

~Dependencies:

subScripts/getEverything.scpt



~Overview:

Spotlight is the indexing system on macOS that indexes the metadata of the contents of all disks on the system (excluding file formats not native to macOS, and SMB volumes). Metadata on macOS is generated, stored, and maintained by the Metadata Server, which records it to the spotlight indices. The spotlight index database would be an incredibly useful piece of information in a forensic investigation, unfortunately there is no verified instance of anyone ever being able to reverse engineer the databases encryption (Link 1).

~Method Implementation:

Fortunately, there is a command line tool that allows you to view the metadata information for an individual file or folder, the 'mdls' command. The result is a set of keys and values containing the metadata information. This command is the forensic communities accepted method of retrieving metadata from items in lieu of being able to access spotlights database. But implementation of this command to create a semi-organized output for a given folder or file is extremely difficult, due to several roadblocks in the logical implementation of the tool, which is to recursively apply it to a given folder, and store the keys and values for the metadata to a single relational file.



1. First, getting a list of the paths of all items in a folder, and its subdirectories.

- Using 'ls' would require additional text parsing, and lots of iteration, and is simply not a reasonable approach
- AppleScript can call methods from specific applications using terminology such as: **tell application "Finder" to get the entire contents of myFolder** unfortunately, while this does accomplish an ideal result of getting a list of folder and file items (objects), this method is extremely inefficient time wise, and from the comparative tests in link 2, we see that that method is also not feasible.
- The same link (2) shows that it is possible to accomplish the same task in around 0.05th the time by telling the application "System Events" to do a series of recursive calls. This is where another problem comes in,

AppleScript defines one application under its "current application" value, which comes into play during the parsing. Without getting too in depth, this means that the directory mapping must be loaded as an external AppleScript module, which can operate with a separate "current application" to achieve a quick recursive directory mapping.

- This is also valuable to anyone forking the project, as recursive folder mapping allows for easy application of additional operations on files and folders.

2. The next difficulty is getting the mdls output, which is clearly a dictionary, into any manageable dictionary object. For those familiar with the macOS filesystem, the .plist file format is standard for recording key value pairs of information. The man pages for mdls don't reveal it, but mdls has the option to add the -plist flag to output the information to a .plist file. Seems like problem solved, but for a small folder with 100 text files, that would generate 100 different .plist files, and would require either using a specialty GUI to save the relation between the original filename and the output metadata plist file, or some other method.

- The ideal solution is to have a single plist file, that contains separate entries with the item name as a key, and the value being the metadata dictionary. Unfortunately, there is seemingly no documented method online for how to import a plist file into another plist file as a value for a key.
- But a bit of digging through the man page indices reveals that there is a plist manipulation command line tool in macOS that is not well advertised, PlistBuddy (/usr/libexec/PlistBuddy). Its man page shows two options that seem like they may be exactly what we are looking for:
  - Merge file [entry]: Adds the contents of plist file to entry.
  - Import entry file: Creates or sets entry to the contents of file.
- Unfortunately beyond the option description, the tools usage description is extremely vague. Designed to run in an interactive shell, it does contain

```
kMDItemDisplayNameWithExtensions = "SimpleTest"
kMDItemContentCreationDate        = 2019-11-22 09:13:07 +0000
kMDItemContentCreationDate_Ranking = 2019-11-22 00:00:00 +0000
kMDItemContentModificationDate    = 2019-11-23 23:54:27 +0000
kMDItemContentModificationDate_Ranking = 2019-11-23 00:00:00 +0000
kMDItemContentType                = "public.folder"
kMDItemContentTypeTree            = (
    "public.folder",
    "public.directory",
    "public.item"
)
kMDItemDateAdded                  = 2019-11-22 09:13:07 +0000
kMDItemDateAdded_Ranking          = 2019-11-22 00:00:00 +0000
kMDItemDisplayName                = "SimpleTest"
kMDItemDocumentIdentifier         = 0
kMDItemFSContentChangeDate        = 2019-11-23 23:54:27 +0000
kMDItemFSCreationDate             = 2019-11-22 09:13:07 +0000
kMDItemFSCreatorCode              = ""
kMDItemFSFinderFlags              = 0
kMDItemFSHasCustomIcon            = (null)
kMDItemFSInvisible                = 0
kMDItemFSIsExtensionHidden        = 0
kMDItemFSIsStationary             = (null)
kMDItemFSLabel                    = 0
kMDItemFSName                     = "SimpleTest"
kMDItemFSNodeCount                = 6
kMDItemFSOwnerGroupID             = 20
kMDItemFSOwnerUserID              = 501
kMDItemFSSize                     = (null)
kMDItemFSFileTypeCode             = ""
kMDItemInterestingDate_Ranking    = 2019-11-24 00:00:00 +0000
kMDItemKind                       = "Folder"
kMDItemLastUsedDate               = 2019-11-24 01:45:54 +0000
kMDItemLastUsedDate_Ranking       = 2019-11-24 00:00:00 +0000
kMDItemUseCount                   = 6
kMDItemUsedDates                  = (
    "2019-11-22 05:00:00 +0000",
    "2019-11-23 05:00:00 +0000"
)
```

a strictly command mode using the -c flag, but there are only 6 usage examples of the tool in the man page, none of which use the -c flag. Digging online will return similarly sparse examples for using the flag.

- Through trial and error, it turns out it is possible to use PlistBuddy to import a plist file as a value for a key, through a series of multiple calls. This is the final part of the puzzle of how to replicate the spotlight database for a given folder or file.

#### ~Method Operation:

1. Get list of paths of all objects in a folder recursively.
2. Create a plist for storage of all metadata. metadata.plist
3. Iterate over the list of paths. Set a equal to index of path in list. With each path:
  - a. Create a temporary plist file that contains the contents of the items metadata using the command:
    - `mdls <path to file> -plist /tmp.plist`
  - b. Use PlistBuddy to import /tmp.plist as a value in metadata.plist
    - i. Create a key in metadata.plist with the value of a, as to avoid issues from duplicate filenames.
      - `/usr/libexec/PlistBuddy -c "Add :<value of a> dict" metadata.plist`
    - ii. As the metadata does not include the full disk path of each item, add that manually as a value in the dictionary at 'a' for a key named "FilePath".
      - `/usr/libexec/PlistBuddy -c "Add :<value of a>:FilePath string <full path>" metadata.plist`
    - iii. Finally, merge the metadata dictionary contained in tmp.plist into key 'a' of metadata.plist
      - `/usr/libexec/PlistBuddy -c "Merge tmp.plist :<value of a> " metadata.plist`

#### ~Information Contained in this Report:

All parsable metadata via the 'mdls' command for all accessible items in the selected directory, or for the selected file.

#### ~Links:

1. <https://arxiv.org/pdf/1903.07053.pdf> - Shining a light on Spotlight: Leveraging Apple's desktop search utility to recover deleted file metadata on macOS
2. <https://macscripter.net/viewtopic.php?pid=195609>

Key	Type	Value
Root	Dictionary	(8 items)
0	Dictionary	(35 items)
FilePath	String	/J Desktop/
_kMDItemDisplayNameWithExtensi...	String	plistbuddy tester.scp
com.apple.metadata_modtime	Number	596,263,248.317879
kMDItemContentCreationDate_Ran...	Date	Nov 23, 2019 at 11:40:48 PM
kMDItemContentModificationDate	Date	Nov 23, 2019 at 11:40:48 PM
kMDItemContentModificationDate_...	Date	Nov 23, 2019 at 7:00:00 PM
kMDItemContentType	String	com.apple.applescript.script
kMDItemContentTypeTree	Array	(8 items)
kMDItemDateAdded	Date	Nov 23, 2019 at 11:40:48 PM
kMDItemDateAdded_Ranking	Date	Nov 23, 2019 at 7:00:00 PM
kMDItemDisplayName	String	plistbuddy tester.scp
kMDItemDocumentIdentifier	Number	1,529
kMDItemFSContentChangeDate	Date	Nov 23, 2019 at 11:40:48 PM
kMDItemFSCreationDate	Date	Nov 23, 2019 at 11:40:48 PM
kMDItemFSCreatorCode	Number	1,416,591,699
kMDItemFSFinderFlags	Number	0
kMDItemFSInvisible	Boolean	NO
kMDItemFSIsExtensionHidden	Boolean	NO
kMDItemFSLabel	Number	0
kMDItemFSName	String	plistbuddy tester.scp
kMDItemFSOwnerGroupID	Number	20
kMDItemFSOwnerUserID	Number	501
kMDItemFSSize	Number	2,581
kMDItemFSTypeCode	Number	1,869,832,563
kMDItemInterestingDate_Ranking	Date	Nov 23, 2019 at 7:00:00 PM
kMDItemKind	String	Script
kMDItemLastUsedDate	Date	Nov 23, 2019 at 11:40:48 PM
kMDItemLastUsedDate_Ranking	Date	Nov 23, 2019 at 7:00:00 PM
kMDItemLogicalSize	Number	2,581
kMDItemPhysicalSize	Number	8,192
kMDItemUseCount	Number	3
kMDItemUsedDates	Array	(1 item)
kMDItemUserCreatedDate	Array	(1 item)
kMDItemUserCreatedUserHandle	Array	(1 item)

## **Time Stamps**

~Method Name:

timeStamp

~Overview:

This function simply takes a time, and a string, and appends them to a timestamp.plist file that we create at the beginning of the programs operation. Then at the start of each function, the time is recorded, then upon completion, the start time is recorded to the file. This allows for accurate documentation of the forensic collection process.

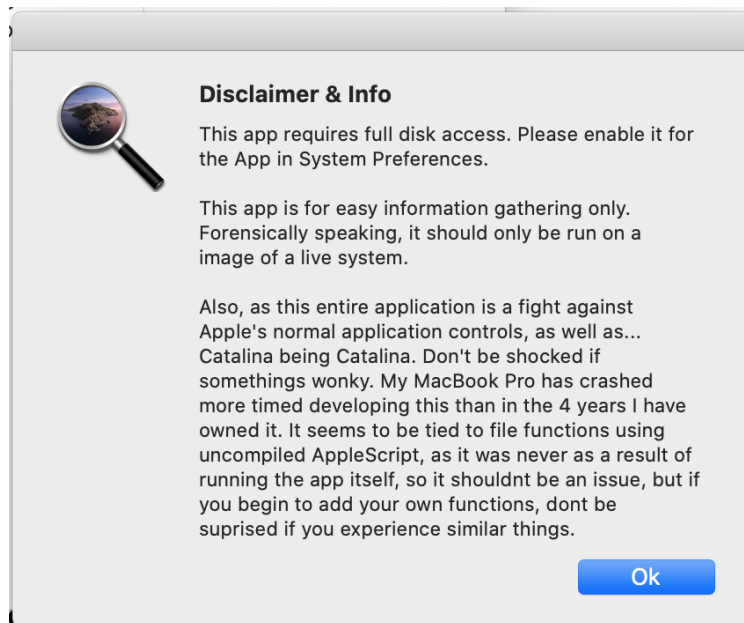
~Relevant information in this Report:

Timestamps marking the beginning of collection of each item.

## **Tool Interface**

### **Warning Window:**

Appears at runtime with important information about running the app.



## Main Window:

The screenshot shows the 'CatalinaExporter' application window. The title bar includes an Apple logo, the name 'CatalinaExporter', and menu items: File, Edit, Format, View, Window, and Help. The window has standard macOS window controls (red, yellow, green buttons). The main content area is titled 'Setup Functions:' and contains two steps. Step 1: 'Please enter your administrator password below to allow the AppleScript to run correctly.' It features a password input field with four dots and an 'Enter' button. Step 2: 'Choose Output Folder' with a text input field. Below these steps is a section titled 'What to Grab:' with the instruction 'hover over item for description'. It contains four checked checkboxes: 'System Report', 'Unified Logs', 'Installation History', and 'FSEvents', along with an unchecked 'Directory Metadata' checkbox. At the bottom, there are 'Testing Function' and 'Run' buttons, and a 'Current Output Location:' field showing '/Users/[redacted] Desktop/CatalinaArtifacts/'.

**Setup Functions:**

1: Please enter your administrator password below to allow the AppleScript to run correctly.

2: Choose Output Folder

**What to Grab:**  
hover over item for description

- ☒ System Report
- ☒ Unified Logs
- ☒ Installation History
- ☒ FSEvents
- ☒ Directory Metadata

Testing Function Run

Current Output Location: /Users/[redacted] Desktop/CatalinaArtifacts/

## Tool Tip:

Give the users a brief rundown of what each option does.

This screenshot focuses on the 'What to Grab:' section of the application. The checkboxes for 'System Report', 'Unified Logs', 'Installation History', and 'FSEvents' are all unchecked. A mouse cursor is hovering over the 'FSEvents' checkbox, which has triggered a tooltip. The tooltip text reads: 'Exports the FSEvents data as a sqlite db from /.fsevents/ using David Cowen's FSEventsParser https://github.com/dlcowen/FSEventsParser WARNING: This one takes a while. It uses python.' Below the 'What to Grab' section, the 'Testing I' button and the 'Current Outp' field are partially visible.

**What to Grab:**  
hover over item for description

- ☐ System Report
- ☐ Unified Logs
- ☐ Installation History
- ☐ FSEvents
- ☐ Directory Metadata

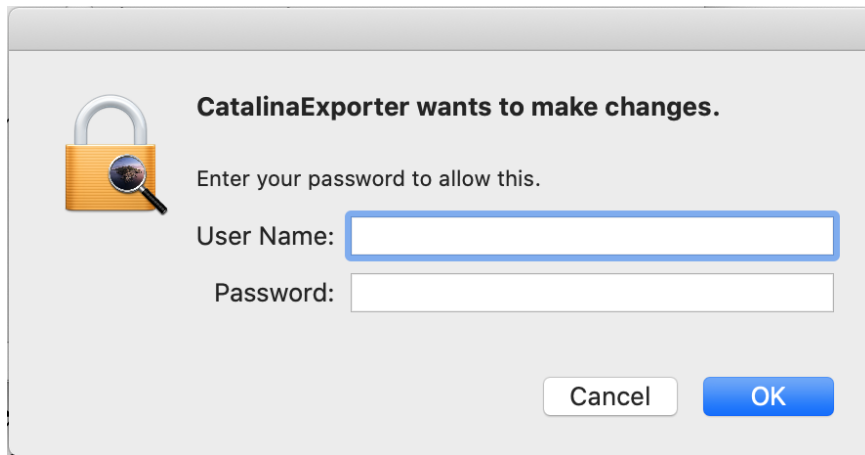
Testing I

Current Outp

Exports the FSEvents data as a sqlite db from /.fsevents/ using David Cowen's FSEventsParser  
<https://github.com/dlcowen/FSEventsParser>  
WARNING: This one takes a while. It uses python.

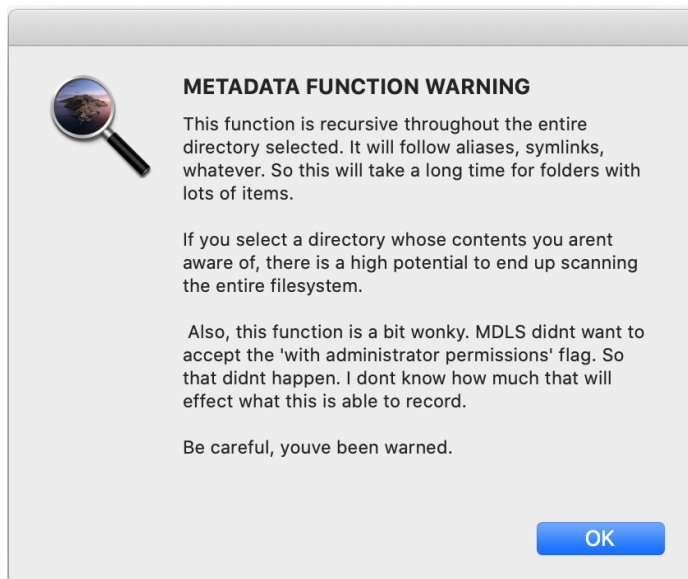
### Privilege Escalation Window:

One of the core components. This enables administrative calls from within the application. Unfortunately it is unclear where each of the two authentication mechanisms apply, in addition to the permissions gained by enabling full disk access to the application as well as access to “System Events”, but this is the methodology used by Apple in “CaptureData.app”, and ultimately it works here.



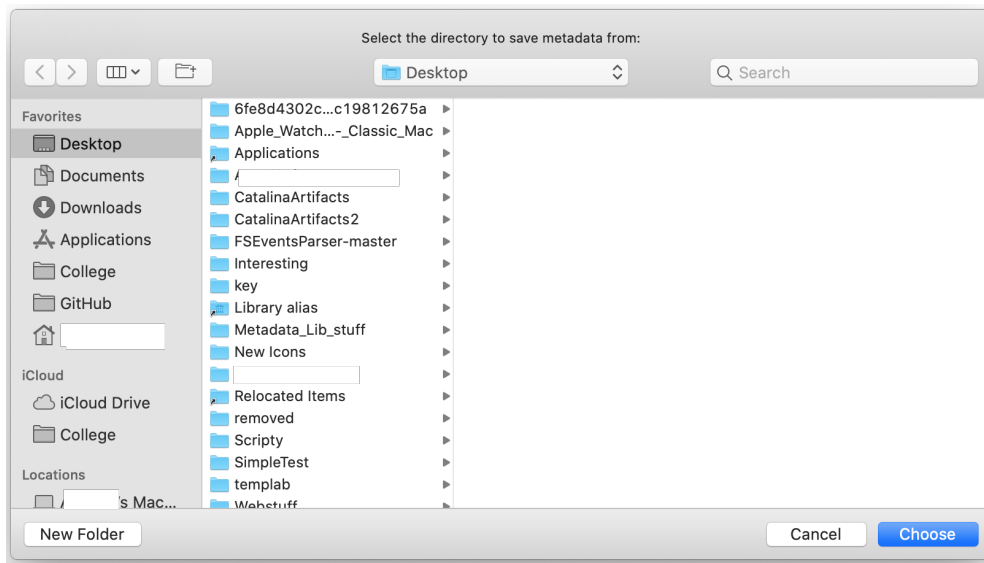
### Metadata function warning:

This function uses a recursive call to parse the entire contents of a directory, and its subdirectory. It follows all aliases. Meaning there is a high potential for it to end up scanning the entire HD, which if not intended will be a huge resource drain. That requires a warning.



## Metadata target folder / file selection window:

Prompts the user to select a target directory or file to build a simulated spotlight.db for.



## Output:

Name
▼ UnifiedLogs
unifLogs.logarchive
▼ FSEventsData
▼ FSE_Reports
FSEvents.sqlite
All_FSEVENTS.tsv
EXCEPTIONS_LOG.txt
▼ SystemInformation
SystemProfile.spx
▼ InstallationHistory
InstallHistory.plist
▼ <b>MetaData</b>
MetaData.plist
Timestamps.plist

## Application Structure:

▼ CatalinaExporter
▼ CatalinaExporter
▼ subScripts
getEverything.scpt
▼ FSEventsParser
FSEParser_V4
report_queries.json
ArtifactFinder.applescript
ArtifactFinder.h
AppDelegate.h
AppDelegate.m
Assets.xcassets
MainMenu.xib
Info.plist
<b>main.m</b>
CatalinaExp...entitlements
CatalinaExp...cdatamodeld