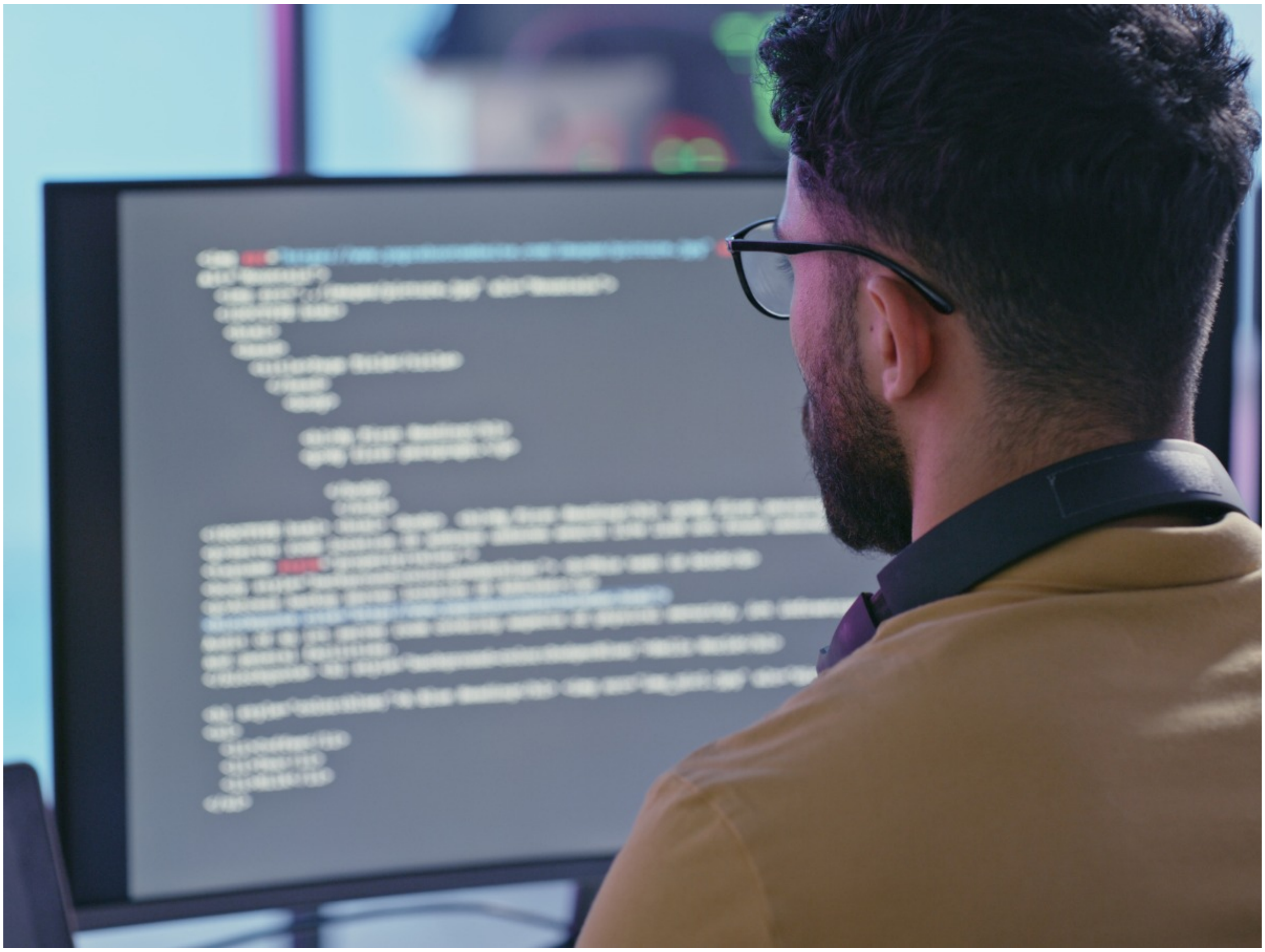
Fortinet Reverses Flutter-based Android Malware “Fluhorse”

fortinet.com/blog/threat-research/fortinet-reverses-flutter-based-android-malware-fluhorse

June 21, 2023



Android/Fluhorse is a recently discovered malware family that emerged in May 2023. What sets this malware apart is its utilization of Flutter, an open-source SDK (software development kit) renowned among developers for its ability to build applications compatible with Android, iOS, Linux, and Windows platforms using a single codebase. While previous instances of threat actors using Flutter for malware exist, such as MoneyMonger, they actually used Flutter for its cross-platform UI elements without carrying the actual malicious payload. So, despite Flutter application reversing being notoriously difficult, MoneyMonger can actually be quite easily reversed with the usual Android reversing techniques.

The Android/Fluhorse family represents a significant shift as **it incorporates the malicious components directly within the Flutter code**.

This blog post covers insights on two notable aspects:

1. The Fluhorse campaign of May had basic obfuscation and no packing. **In June, however, the sample we analyzed was packed. This is evidence that malware authors are moving to the next step of maturity**.

Figure 1: Android/Fluhorse sample of May 2023 isn't packed, and directly uses Flutter

Figure 2: Android/Fluhorse sample of June 2023 is packed and conceals its

2. **Reverse engineering technique**. Reversing Flutter applications is widely considered a challenging endeavor. Many researchers treat it as a black box, only analyzing those components that can be observed from the outside. Some employ dynamic instrumentation tools, like re-Flutter and re-flutter-demo, which come with their own complexities and risks associated with running malware in controlled environments. But for our analysis, we successfully managed to **fully reverse-engineer the Fluhorse malware in a static manner**, without the need for dynamic execution.

Note that Fortinet customers are fully protected against this family of malware. These samples are detected as Android/Fluhorse.A!tr.spy or Android/Packed.57103!tr, and all related URLs mentioned in this blog are detected and blocked.

Quick description of Android/Fluhorse

The malware we analyzed posed as a legitimate app for an **electronic toll system** used in Southern Asia. It lures the victim into entering his/her credentials, steals them, and also steals 2FA (two-factor authentication) codes by listening to incoming SMS and forwarding them to a website controlled by the attackers.

Figure 3: Android/Fluhorse poses as an electronic toll collection application

Victims typically install the malware via e-mail phishing campaigns, with previous research claiming that Fluhorse has been downloaded over 100,000 times.

The current version we analyzed (SHA256: 2c05efa757744cb01346fe6b39e9ef8ea2582d27481a441eb885c5c4dcd2b65b) was first seen on June 11, 2023. It is distributed from the malicious URL hxxps://fasd1[.]oss-ap-southeast-1.aliyuncs.com/*ETC*.apk. FortiGuard’s telemetry shows it has been being accessed from Asia since June 12.

Packer

The sample is packed, concealing the encrypted payload within the wrapping application's Dalvik executable, classes.dex. The packer adds a 4-byte integer at the end of the DEX that includes (1) the size of the encrypted payload and, just before that, (2) the encrypted payload.

Figure 4: The encrypted payload is hidden with classes.dex itself

Decryption is performed at the native level (to harden reverse engineering) using OpenSSL's EVP cryptographic API. The encryption algorithm is AES-128-CBC, and its implementation uses the same hard-coded string for the key and initialization vector (IV). From a cryptographic point of view, this is a bad idea as the IV is not meant to be confidential.

Figure 5: Native library libapksadfsalkwes.so decrypts the malicious payload using AES 128

The decrypted payload is a ZIP file that contains another DEX. The payload is then installed (the packer's implementation includes Android sources for MultiDex support). The "main" of the malware is found in the payload (com.dsfdgfd.sdfsdf.MainActivity). It simply loads the Flutter application, as in the unpacked version of May 2023.

Figure 6: Contents of the decrypted malicious payload

Our in-depth static analysis of libapp.so revealed everything we needed to know, including the supposed directory structure of the malicious source code. The malicious Dart package is named sms_fluter and has the following files:

- sms_flutter/main.dart. This file implements the main application. We can see below that the functionalities are basic.
- sms_flutter/api/login.dart. This file implements communications with the remote attacker's website. Class LoginApi implements a method postSms.
- sms_flutter/views/webifrview.dart. This file implements several classes: WebIfrViewState and WebIfrView, which basically handle the UI, but also request necessary permissions to listen to incoming SMS messages.

The malware uses the Dart **Telephony** package. This package is non-malicious and its code is open-source and documented. In particular, the package offers the possibility of **listening to incoming SMS in the background**. The malware author closely followed the documentation and created the following code (code is my manual re-construction from assembly).

```
onBackgroundMessage(SmsMessage message) async {
  // malicious tasks
}
```

```
void main() {
  runApp(MyApp());
}
```

The corresponding assembly lines are shown below. For the reverse engineer, finding the address of a given Dart function is difficult. Several researchers use reFlutter, or similar techniques where the application is instrumented and then run, causing each address to be dumped. Unfortunately, for malware analysts, this technique has two major drawbacks besides its complex setup: first, you only get the addresses of the functions you visit. Second, **you instrument and run a malware**, which should be avoided.

Figure 7: Fluhorse's main

To obtain the addresses I was looking for, I used JEB, a reverse-engineering tool that offers some limited (but at least existing) support for Flutter. In particular, it finds the addresses of each referenced method and is usually able to correctly map the name in the assembly code. When it fails (more frequently on ARM32 and ARM64), I fall back to reversing the x86_64 library. Fortunately, the x86_64 version was provided in this Fluhorse sample, probably by error, because the rest of the implementation (for example, the packing) is not implemented for x86_64. If that doesn't work, then I manually get the address from JEB's code information and re-base it with the relocation base for zero-based objects.

We can see below that the malware listens to incoming SMS. First, it registers for change notifications on SMS and then calls listenIncomingSms in the Telephony package.

Figure 8: Dart Assembly code to listen to incoming SMS

The malware then posts the incoming SMS message to a remote website in an asynchronous task.

Figure 9: The malware builds the URL of the remote website to contact: hxxp://pmm122.com/addcontent3… Later, the assembly adds the intercepted SMS body in argument "c4" of the URL.

Reverse engineering the code reveals several specificities to Dart assembly. In Dart, all constants, literals, etc, are stored in an Object Pool, which is like an index table. Later, all access to strings is done indirectly: the code asks for access to a given index. Dart assembly dedicates a custom register R15 for x86_64 to access the Object Pool. The illustrated example above accesses R15+8FB7, which corresponds to index 0x8FB7 // 8 = **4598**. JEB shows the Object Pool indexes for us and lets us map the assembly line by fetching the domain name string.

Figure 10: JEB shows Object Pool indexes

The SMS body is inserted in the URL as argument “c4” (e.g., hxxp://pmm122[.]com/addcontent?c4=hello).

The *incoming phone number is not provided* because the Telephony package does not provide this feature. In the following code of the (non-malicious) Telephony package, see that only the SMS body is retrieved and provided to the callback.

```
try {
  await handlerFunction(SmsMessage.fromMap(
      call.arguments['message'], INCOMING_SMS_COLUMNS));
} catch (e) {
```

Later, Fluhorse’s code performs the HTTP POST of the SMS message and reads the response. This is handled by Dart’s standard HTTP package.

Figure 11: Assembly lines where the malware actually posts the HTTP request and reads the response. This is how Android/Fluhorse steals 2FA codes sent by SMS.

Conclusion

Reversing Flutter applications statically is a breakthrough for anti-virus researchers, as, unfortunately, more malicious Flutter apps are expected to be released in the future.

Building expertise in dealing with these samples and improving tools is necessary. This blog post provides results for Android/Fluhorse. More detailed explanations have been submitted to conferences in Q4 2023, where I hope to be selected to explain this live. ;) Stay tuned!

Fortinet Protections

Fortinet customers are fully protected against this family of malware.

Samples in this blog are detected by the FortiGuard AV engine as:

Android/Fluhorse.A!tr.spy

Android/Packed.57103!tr.

The FortiGuard AntiVirus service is supported by FortiGate, FortiMail, FortiClient, and FortiEDR. Fortinet EPP customers running current AntiVirus updates are also protected.

Fortinet Webfiltering blocks all URLs identified in this blog.

If you believe this or any other cybersecurity threat has impacted your organization, please contact our Global FortiGuard Incident Response Team.

IOCs

The samples mentioned in this blog post are (sha256 hash):

2c05efa757744cb01346fe6b39e9ef8ea2582d27481a441eb885c5c4dcd2b65b

Other similar samples:

e8cdf809a5655124fa9347e7a90f071bed74907d3098737fd1184148ab475e39
6e7293564e7e2e051d42168b068535a7963974cdd6437a3242230b9593dc7f04
7cee6677790c493dddf16ff610a174e6536208f8853816cd0d71fc6bde56e93c
91f0a27ae5ca77930c21b19f33479e7abcb10dfcf2a92b690ccddea01434fe84
7dff0f7987f956c948847ea3659730408e35e4513b6adcd92d60ba48a93f62f1
6f0b3733f91a6af56bf5bc789b808475cb556f2d360131ef6a9082b98dfd0139
0a577ee60ca676e49add6f266a1ee8ba5434290fa8954cc35f87546046008388
25fee29a8cb3e6b71771897e34a58cf9c7c0be4805acabb36be886e93de03f62
663033dce1688186d6111c8637dd3bc79483bdb8fc1b2ad4d5ead030f79f84b7
6dbde61a3aa372e8af7aa049dd466a2892bbe0d1229866cb2ba46c8f61648a57
94bb98d9955947f9e7c502961e4b2a7724289e80b566035c14ac9fa6cf36df1c
852314984cbea056a782520654f84c828588b6a0163bdeb8f8d5016b05c205f9
c55feac16e7ca084f47a899281b566faf41b5666376353efeb9010fe5d23b526
0a106c851a267fb8590be1f033e995bfc559ffaf2be050b3f12f599e0c8c021c
32c427581a0368b66dd50b381772fb0d6dab30d8316f4e4f0d0373d453091cd0
7909593a310b245a1a92a78469be341b0849e6f1076af30f8266b1c5a861ead1
c25d533487499204771fac87787d38df91f0971b693dffa9b17fa0d92c80bfac
5af2ec81d09ecbf8c26a8887d96c948b3c61667b7ffc488fbd67239ea9ac2cd6
5481348e4751a494bc76ab4908071124f12624369401b717c7766e7d0645754d