

# Intercepting Flutter iOS Application

Posted on June 23, 2021

## TL;DR

Hi, this is Debugger (<https://github.com/bhattsameer>) ready to debug Mobile Application.

In this blog I will share how I have intercepted the traffic of Flutter based iOS application for dynamic analysis, Also we will see the root detection and SSL verification bypass method I have used.

TL;DR Quote

## Introduction

As Flutter uses dart, Dart is not proxy aware and uses its own certificate store. Hence, The application doesn't take any proxy settings from the system and sends data directly to server, because of this we cannot intercept the request using Burpsuite (<https://portswigger.net/burp>) or any MITM tool, so changing the proxy settings in wifi or trusting any certificate won't help here.

### Questions:

So what we can do in this situation?

There are multiple approaches here to intercept the traffic:

#### **Thought 1:**

SSH into your iOS device and use iptables to redirect the traffic.

**Note: iptables won't work in iOS device as it requires kernel support.**

#### **Approche 1:**

Create a WIFI hotspot using a second WIFI adapter and use iptables to redirect all traffic to Burp.

## Approche 2:

Using OpenVPN to create a tunnel and use iptables to redirect all the traffic to Burp.

You can read about this all approaches in details using nviso blog (<https://blog.nviso.eu/2020/06/12/intercepting-flutter-traffic-on-ios/>).

I have used **Approche 2** to intercept the traffic on the application I am working on.

## Pre-Requisites:

1. Jailbroken (<https://canijailbreak.com/>) iOS device.
2. Burp Suite (<https://portswigger.net/burp>) up and running.
3. OpenVPN (<https://apps.apple.com/us/app/openvpn-connect/id590379981>) application installed in your iOS device.
4. Liberty - Root detection bypass Cydia Repo (<https://rileyangus.com/repo>).
5. Filza - To extract ipa from iOS device Cydia Repo (<https://apt.thebigboss.org/repofiles/cydia/>)
6. Your system and mobile device must be connected to same wifi network.
7. Must have Frida (<https://frida.re/>) installed in your system as well as in iOS device.
8. Must have Ghidra (<https://ghidra-sre.org/>) or any binary analyzer tool pre installed in your system.

## Let's Get Started:

A. First install OpenVPN application to the iOS device from app store:

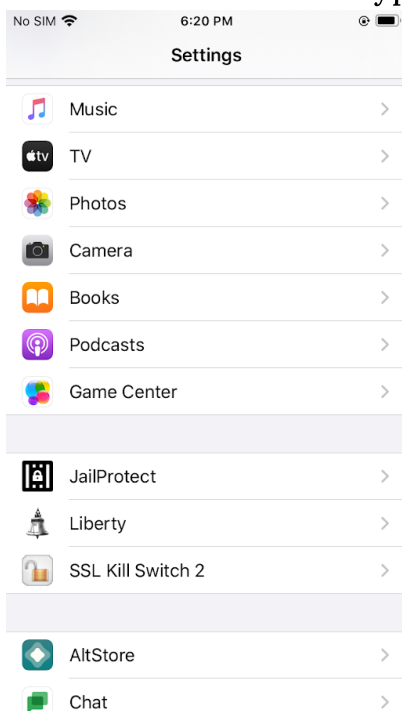


## B. Install Your Flutter application to iOS device:

As the application has jailbreak detection implemented, Hence when we start application it will crash on splash screen.

To bypass the jailbreak detection we can use **Liberty**. You can install Liberty from **Cydia** (<https://en.wikipedia.org/wiki/Cydia>) using Source repo (<https://rileyangus.com/repo>)

- i. Once you have installed Liberty, Go to your device Settings and scroll down a little and you can see the Liberty app in it.
- ii. Click on Liberty -> Block Jailbreak Detection -> select the desired application of which we wanted to bypass the jailbreak detection.



C. Now we can run the app in jailbroken device, let's move ahead.

### Brief Process:

1. We are going to create one OpenVPN connection file, Configure it in our iOS device using OpenVPN application and establish the connection.
2. Using Iptable command we will route the device traffic through our system.
3. find and analyze the binary which contains the SSL verification code.
4. Using frida we will bypass the SSL verification implementation.

### Let's go step by step with the process:

1. Create OpenVPN file to connect:

Use below command to download one script which helps us in creating OpenVPN file as per our need.

Script: <https://github.com/Nyr/openvpn-install>  
(<https://github.com/Nyr/openvpn-install>)

```
> sudo wget https://git.io/vpn -O openvpn-install.sh
> sudo sed -i "$((($(grep -ni "debian is too old" openvpn-install.sh | cut
> sudo chmod +x openvpn-install.sh
> sudo ./openvpn-install.sh
```

After running Scripts Select Below options:

```
Which IPv4 address should be used?
> Select option of your system IP. i.e. 192.168.1.118

Public IPv4 address / hostname []:
> Provide your system IP i.e. 192.168.1.118

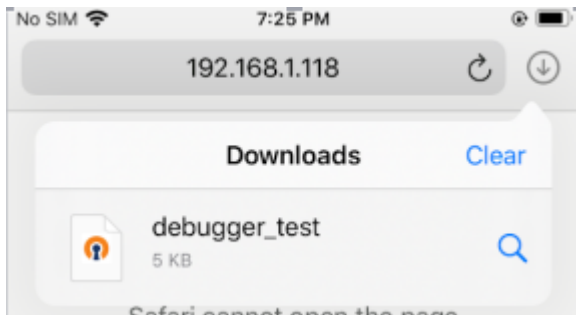
Protocol [1]: 1
Port [1194]: 1194
DNS server [1]: 1
Name [client]: debugger_test
```

And Press enter. OpenVPN file will be created at **/root/debugger\_test.ovpn**

## 2. Adding OpenVPN file to device:

Run python http server using below command in your system and download the ovpn file to your device.

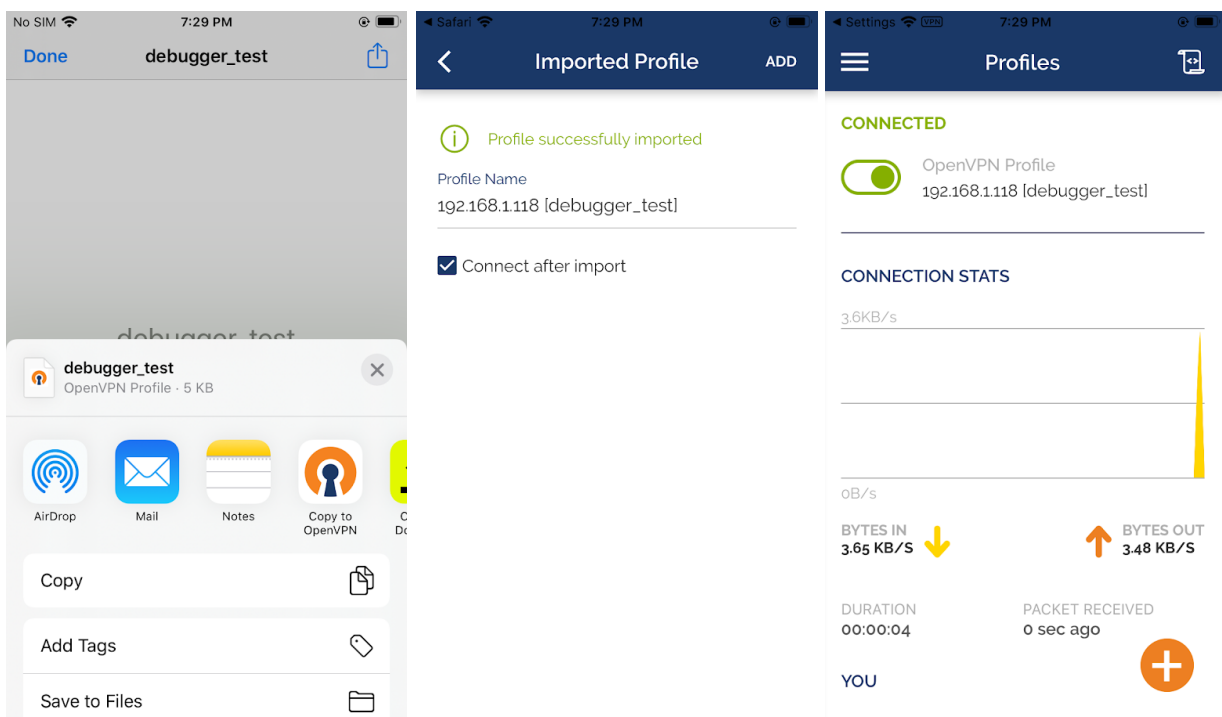
```
> python3 -m http.server 8081 --directory /root/
```



Open the downloaded ovpn file using OpenVPN, configure and connect to it.

**Note:** You can start openvpn in your system through below command:

```
> sudo service openvpn start
```



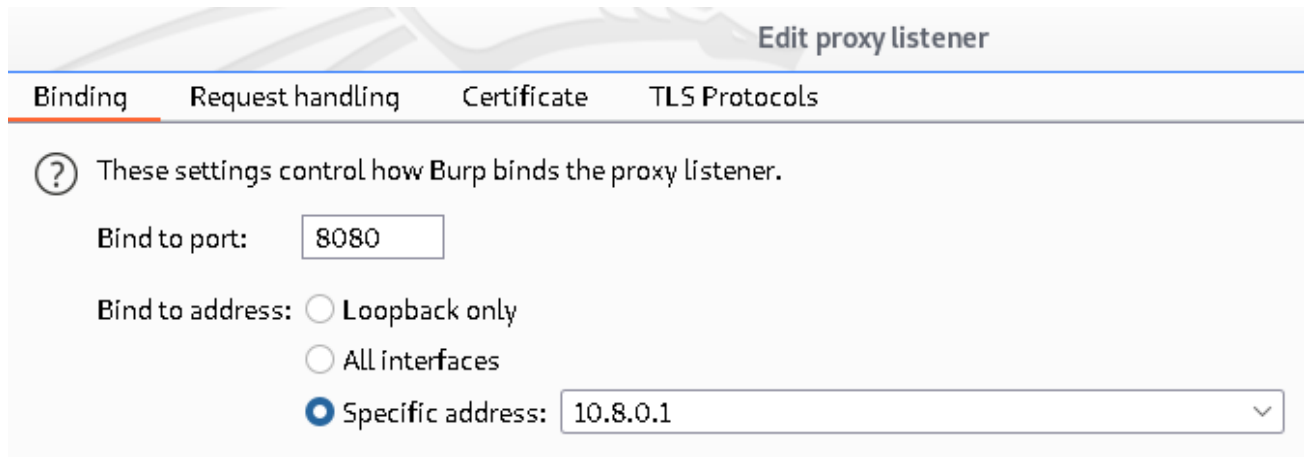
## 3. Route the traffic and burp proxy configuration:

Run below commands to route the traffic from your iOS device through your system.

**Note:** In the last command the provided IP is your iOS device IP i.e. 192.168.1.101.

```
> sudo iptables -t nat -A PREROUTING -i tun0 -p tcp --dport 80 -j REDIRECT --to-destination 10.8.0.1:8080
> sudo iptables -t nat -A PREROUTING -i tun0 -p tcp --dport 443 -j REDIRECT --to-destination 10.8.0.1:8080
> sudo iptables -t nat -A POSTROUTING -s 192.168.1.101/24 -o eth0 -j MASQUERADE
```

Once all done open burpsuite proxy tab and configure it as below:



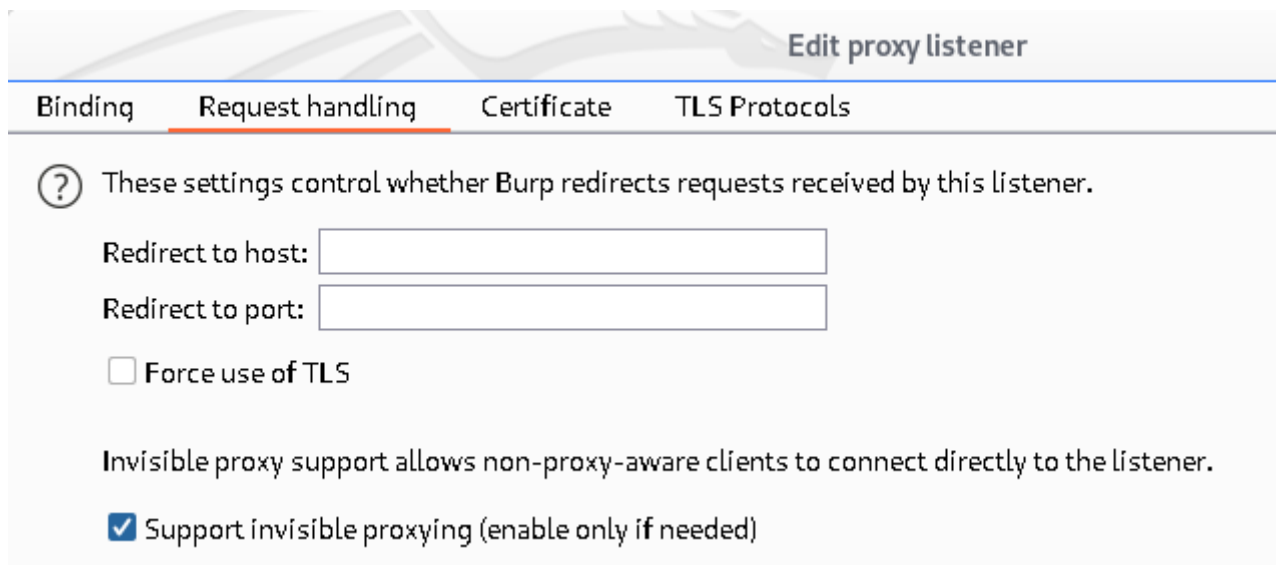
**Edit proxy listener**

**Binding** Request handling Certificate TLS Protocols

? These settings control how Burp binds the proxy listener.

Bind to port:

Bind to address: ☐ Loopback only  
☐ All interfaces  
☒ Specific address:



**Edit proxy listener**

Binding **Request handling** Certificate TLS Protocols

? These settings control whether Burp redirects requests received by this listener.

Redirect to host:

Redirect to port:

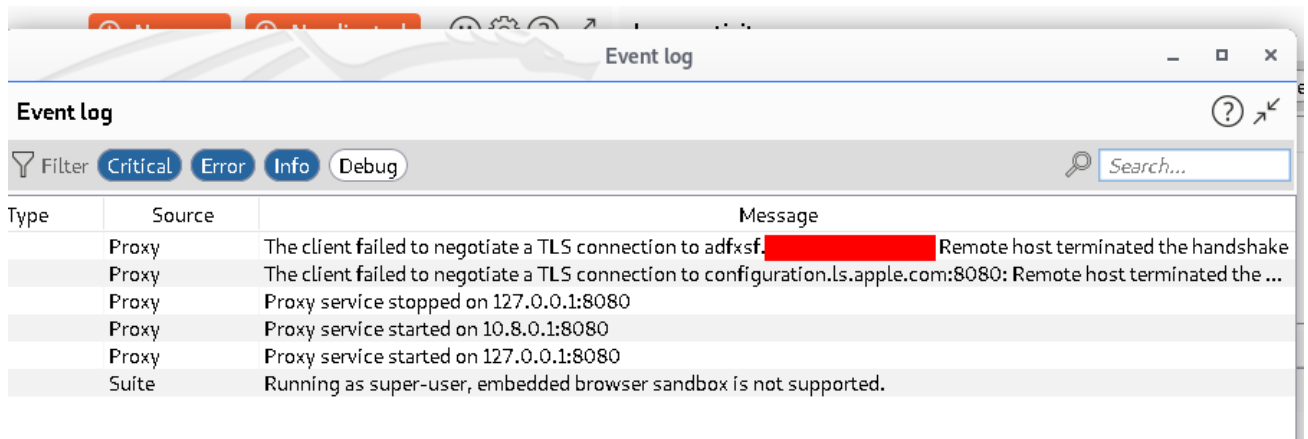
☐ Force use of TLS

Invisible proxy support allows non-proxy-aware clients to connect directly to the listener.

☒ Support invisible proxying (enable only if needed)

#### 4. Identified SSL verification implemented using x509.cc.

Run the application and observe the burpsuite dashboard -> Event logs.  
We got TLS handshake error, Hence we can not intercept https traffic.



5. Use the **Filza** to download the application folder from iOS device.

You can install Filza from **Cydia** using Source repo  
(<https://apt.thebigboss.org/repofiles/cydia/>)

Start the WebDAV server of Filza from settings and Access it through your system browser by accessing [http://<Your\\_mobile\\_IP>:11111](http://<Your_mobile_IP>:11111).

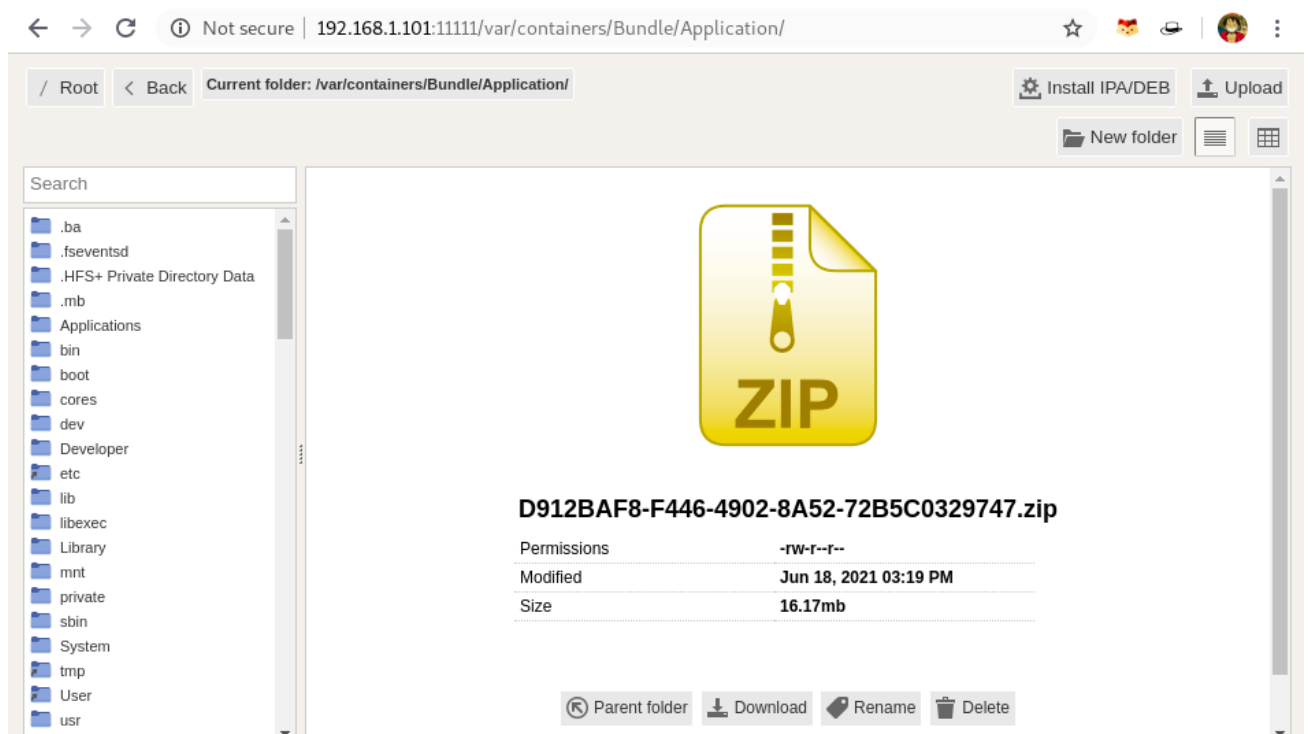
**Note:** Make sure your phone and system are connected to same wifi network.

In your phone open Filza and navigate to:

**/var/containers/Bundle/Application/**

Observe the screen and create a zip of your application folder, you can do it by long press on the folder.

In browser Filza, access to same path and download the zip file.



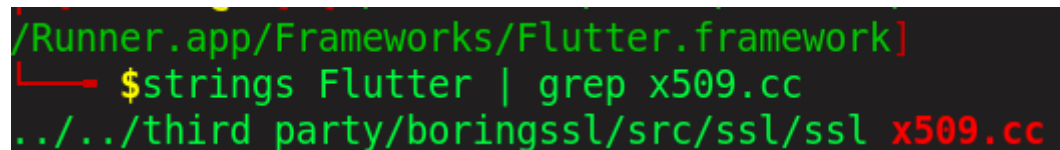
**Note:** If we install the application directly from appstore, the application has encryption flag enabled. Hence to proceed with the binary analysis we first need to decrypt the application binary, For that you can use Clutch (<https://github.com/KJCracks/Clutch>) Or Frida-ios-dump (<https://github.com/AloneMonkey/frida-ios-dump>).

6. Extract Zip in your system and open it up through Terminal:

Binary: /Runner.app/Frameworks/App.framework/App  
This contains application flutter code.

Binary: /Runner.app/Frameworks/Flutter.framework/Flutter  
This contains the x509.cc SSL verification code.

And to bypass the SSL verification we have to analyze the Flutter binary.



```
/Runner.app/Frameworks/Flutter.framework]  
└─ $strings Flutter | grep x509.cc  
../../../../third_party/boringssl/src/ssl/ssl_x509.cc
```

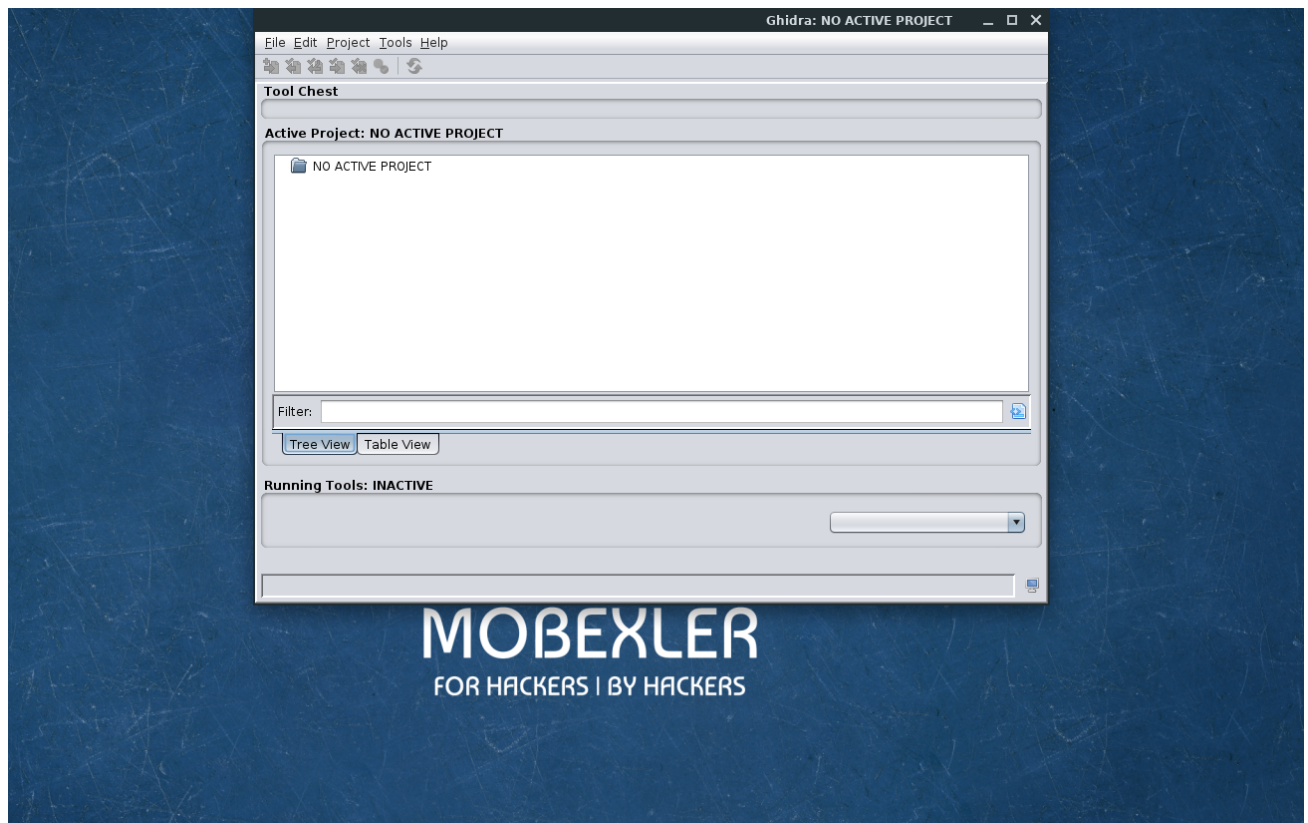
7. Install Ghidra (<https://ghidra-sre.org/>) and disassemble the Flutter binary.

Binary path Runner.app/Framworks/Flutter.framework/Flutter

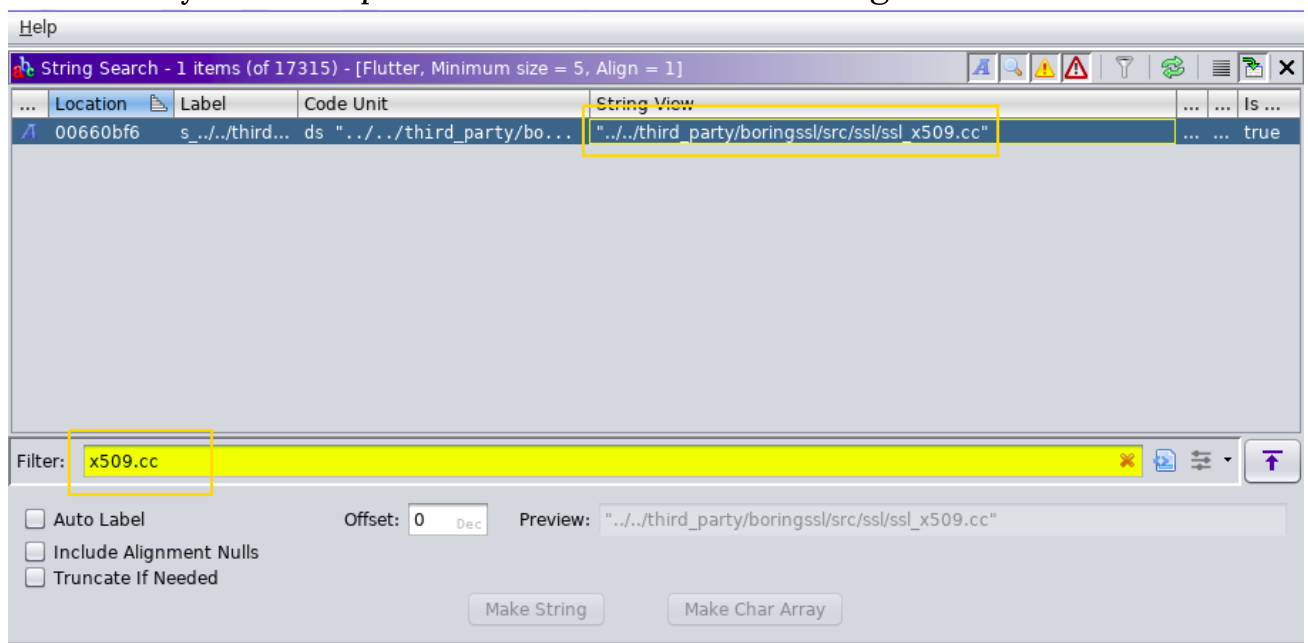
Start Ghidra and drag and drop the Flutter binary to it, select all default details.

Wait for ghidra to complete the analysis process.





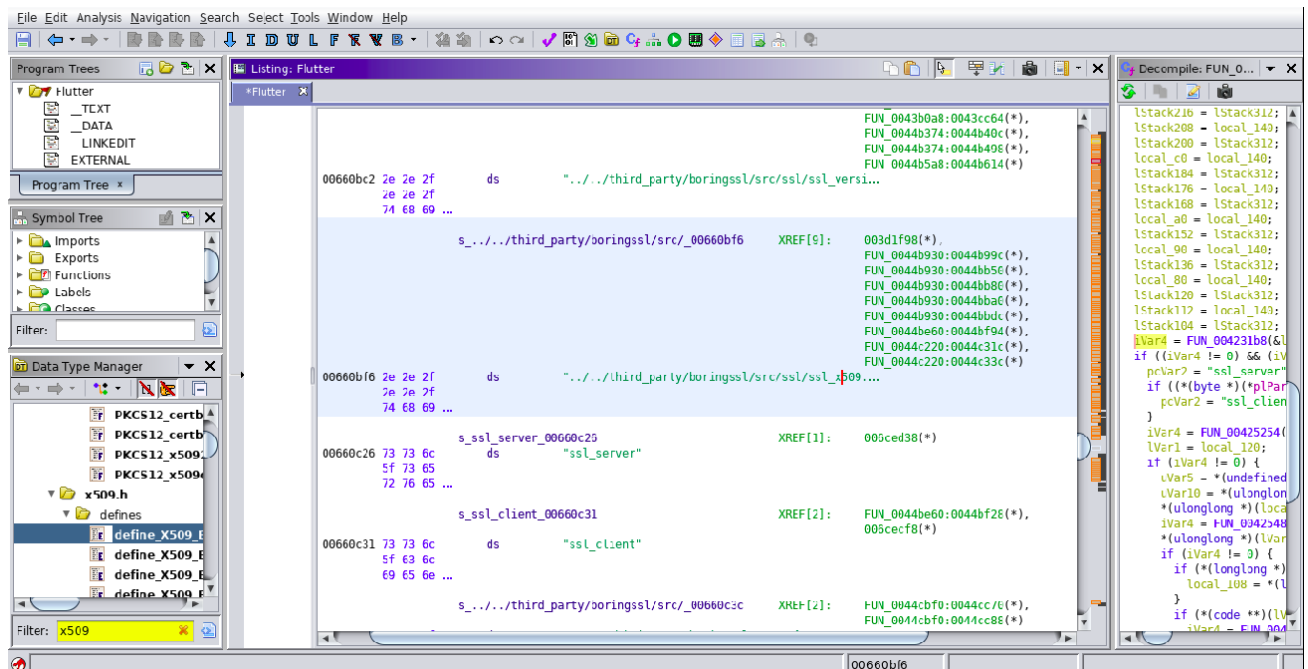
Once Analysis is completed search for x509.cc string:



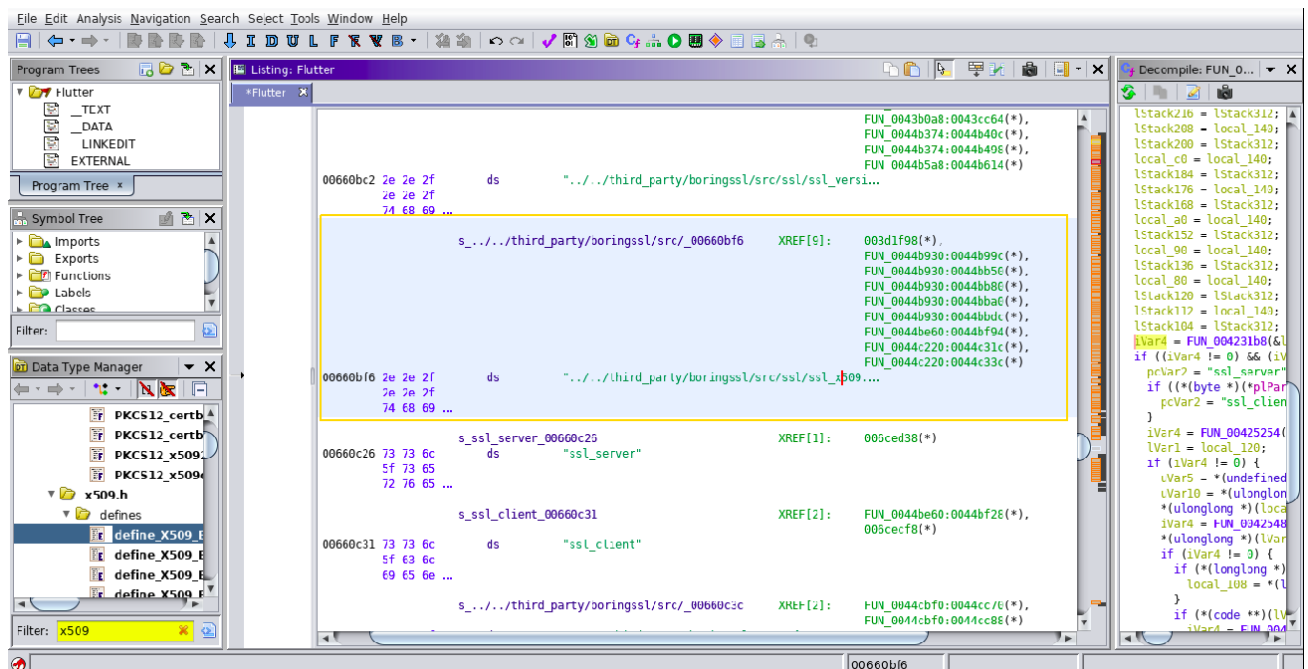
android-x64/) Team.

Help				
Scalar Search [filter: 0x186] (Flutter) 4 items				
Location	Preview	Hex	Decimal (S...	Function Name
005a2928	ushort 186h	186	390	
0044bfa0	mov w3,#0x186	186	390	FUN_0044be60
00362f80	cmp w0,#0x186	186	390	FUN_00362ef0
00362a08	cmp w0,#0x186	186	390	FUN_00362970

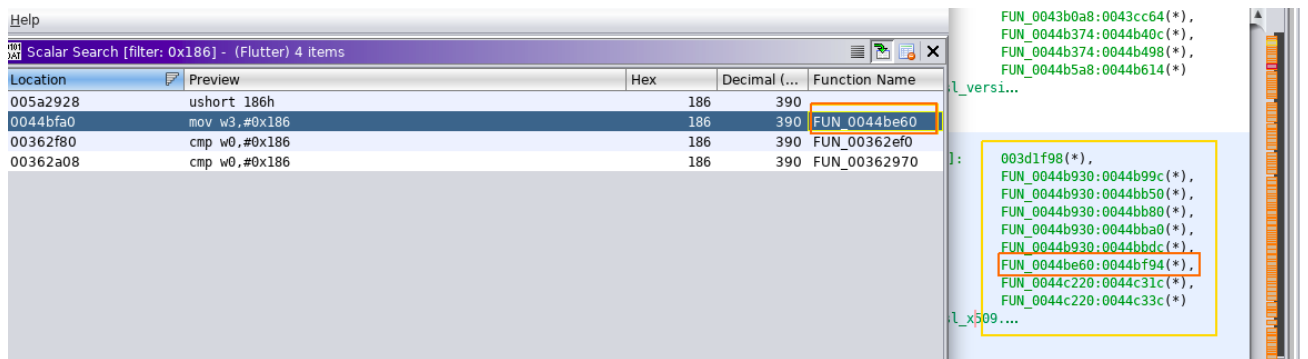
Click on the output you got from string search of x509.cc and you will moved to the address directly.



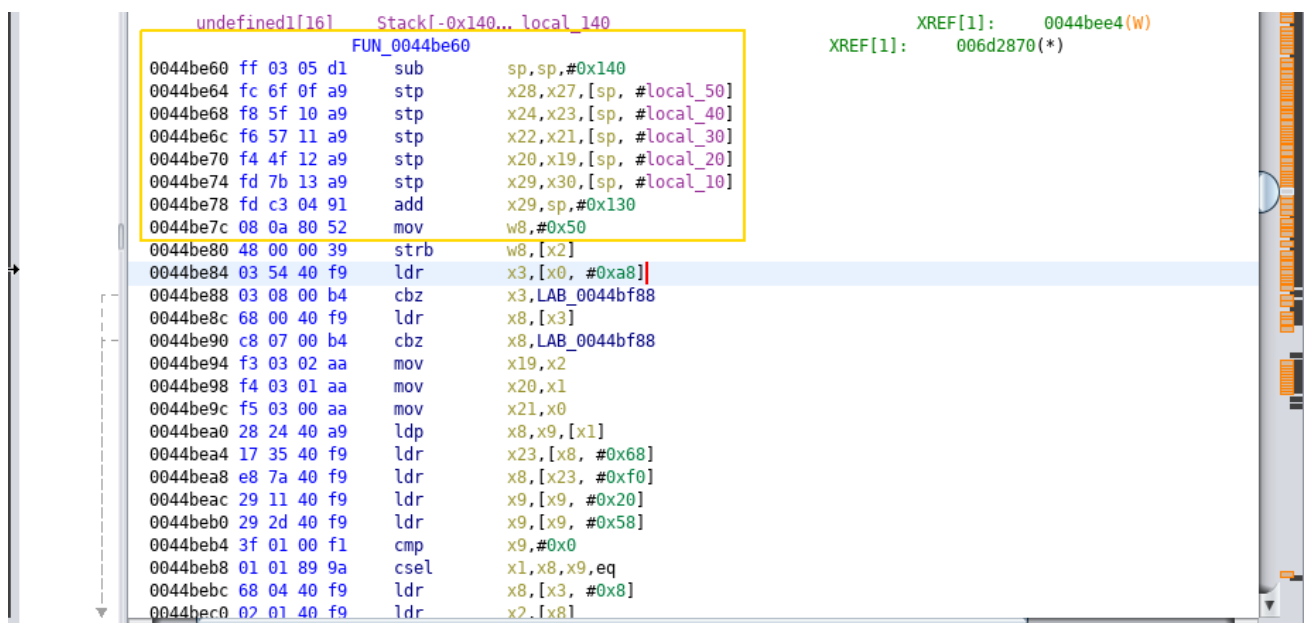
As we can see, we have multiple XREF[0] against the x509.cc line. We have to identify the valid function address, For that we will take help of our scalar search output which we have done previously.



Map scalar search near to the x509.cc output and observe that one address we found in scalar search is in the range of the XREF[0] we got for x509.cc.



Click on that address range FUN\_0044be60:0044bf94(\*) and Observe the initial bytes value of FUN\_0044be60. Copy those bytes:



Copy the initial bytes as below and send to binwalk for offset count if needed.

```
ff 03 05 d1 fc 6f 0f a9 f8 5f 10 a9 f6 57 11 a9 f4 4f 12 a9 fd 7b 13 a9 fd  
c3 04 91 08 0a 80 52
```

Replace the address value in the below frida script as a pattern variable.

```
function hook_ssl_verify_result(address)
{
  Interceptor.attach(address, {
    onEnter: function(args) {
      console.log("Disabling SSL validation")
    },
    onLeave: function(retval)
    {
      retval.replace(0x1);
    }
  });
}

function disablePinning()
{
  var pattern = "ff 03 05 d1 fc 6f 0f a9 f8 5f 10 a9 f6 57 11 a9 f4 4f 12 a9 fd 7b 13 a9 fd c3 04 91 08 0a 80 52";
  Process.enumerateRangesSync('r-x').filter(function (m)
  {
    if (m.file) return m.file.path.indexOf('Flutter') > -1;
    return false;
  }).forEach(function (r)
  {
    Memory.scanSync(r.base, r.size, pattern).forEach(function (match) {
      console.log('[+] ssl_verify_result found at: ' + match.address.toString(16));
      hook_ssl_verify_result(match.address);
    });
  });
}

setTimeout(disablePinning, 1000)
```

Connect mobile using USB with your system and run ***frida-ps -Uai***

```

/iOS application/test# frida-ps -Uai
  PID  Name                Identifier
-----
24807  Camera                 com.apple.camera
25034  [REDACTED]             com.[REDACTED]
24992  Filza                  com.tigissoftware.Filza
25018  OpenVPN                net.openvpn.connect.app
25013  Safari                 com.apple.mobilesafari
25019  Settings               com.apple.Preferences

```

Run frida script to your application package Name using below command:

```
> frida -Uf `package` -l `Bypass script` --no-pause
```

```

/test# frida -Uf com.[REDACTED] -l /home/sameer/Downloads/test4/flutter/disable.js --no-pause

  /_/_|  Frida 14.2.6 - A world-class dynamic instrumentation toolkit
 |(_|_|  Commands:
  >_|_|  help      -> Displays the help system
  /_/_|_| object?   -> Display information about 'object'
  . . . . exit/quit -> Exit
  . . . .
  . . . . More info at https://www.frida.re/docs/home/
Spawning `com.[REDACTED]`...                               Spawned `com.[REDACTED]`
[REDACTED]. Resuming main thread!
[iOS Device::com.[REDACTED]-> [+] ssl_verify_result found at: 0x1028ffe60
[iOS Device::com.[REDACTED]-> █

```

Perform some activity in the application and check back to burp to see the requests from app being intercepted now:

The screenshot displays a network traffic analysis tool interface. The top section shows an HTTP request details:
 

- Method: GET
- URL: /api/v4/app/auth/login\_mod/?token=[REDACTED]
- User-Agent: Dart/2.10 (dart:io)
- Accept-Encoding: gzip, deflate
- api-key: [REDACTED]
- content-length: 0
- authorization: [REDACTED]
- host: getes[REDACTED]
- Connection: close

The bottom section shows an 'Event log' with a filter set to 'Debug'. The log contains several messages from the 'Proxy' source:
 

- getes[REDACTED] is using HTTP/2
- Connection reset
- The client failed to negotiate a TLS connection to [REDACTED] Rem
- The client failed to negotiate a TLS connection to p51-buy.itunes.apple.com:80
- [4] Communication error: bag.itunes.apple.com
- Could not create new HTTP/2 connection
- Timeout in communication with remote server

## Conclusion:

This blog was to share how I have bypassed the security implementation of an iOS application, and how I have intercepted the traffic of flutter iOS application. As the method for the same is different compare to what we actually do in mobile application testing to intercept the traffic.

I enjoyed writing this article and hope you find this useful.  
Thank you for your time and stay tuned for more!

You can find me here:

Twitter : @sameer\_bhatt5 ([https://twitter.com/sameer\\_bhatt5](https://twitter.com/sameer_bhatt5))

Github : bhattsameer (<https://github.com/bhattsameer>)

LinkedIn: bhatt-sameer (<https://linkedin.com/in/bhatt-sameer>)

 *Buy me a coffee* (<https://www.buymeacoffee.com/bhattsameer>)

## Reference:

Huge Thanks to Nviso Team.

<https://blog.nviso.eu/2020/06/12/intercepting-flutter-traffic-on-ios/>  
(<https://blog.nviso.eu/2020/06/12/intercepting-flutter-traffic-on-ios/>)

Visitors 1 / 2049 (<https://hits.seeyoufarm.com>)



← **PREVIOUS POST** (</2021/02/21/CLIENT-SIDE-ENCRYPTION-BYPASS-PART-3.HTML>)



(<https://github.com/bhattsameer>)



([https://twitter.com/sameer\\_bhatt5](https://twitter.com/sameer_bhatt5))



(<https://linkedin.com/in/bhatt-sameer>)

Sameer Bhatt (Debugger) • 2022 • [bhattsameer.github.io](https://bhattsameer.github.io) (<https://bhattsameer.github.io>)

Theme by beautiful-jekyll (<http://deanattali.com/beautiful-jekyll/>)