

Mobile Application Security Analysis

Phase 1: Environment Setup

First, I installed the Java Development Kit (JDK) and Android Studio. I used the built-in Android Virtual Device (AVD) emulator from Android Studio.

Then I installed necessary tools using Homebrew:

brew install android-platform-tools

adb version

brew install apktool jadx

```
openjdk version "23.0.2" 2025-01-21
ajzankulkibaeva@MacBook-Air-Ajzan ~ % brew install android-platform-tools
```

```
##### 100.0
==> Installing Cask android-platform-tools
==> Linking Binary 'adb' to '/opt/homebrew/bin/adb'
==> Linking Binary 'etc1tool' to '/opt/homebrew/bin/etc1tool'
==> Linking Binary 'fastboot' to '/opt/homebrew/bin/fastboot'
==> Linking Binary 'hprof-conv' to '/opt/homebrew/bin/hprof-conv'
==> Linking Binary 'make_f2fs' to '/opt/homebrew/bin/make_f2fs'
==> Linking Binary 'make_f2fs_casefold' to '/opt/homebrew/bin/make_f2fs_casefold'
==> Linking Binary 'mke2fs' to '/opt/homebrew/bin/mke2fs'
🍺 android-platform-tools was successfully installed!
ajzankulkibaeva@MacBook-Air-Ajzan ~ %
```

```
Android Debug Bridge version 1.0.41
Version 36.0.0-13206524
Installed as /opt/homebrew/bin/adb
Running on Darwin 23.6.0 (arm64)
ajzankulkibaeva@MacBook-Air-Ajzan ~ %
```

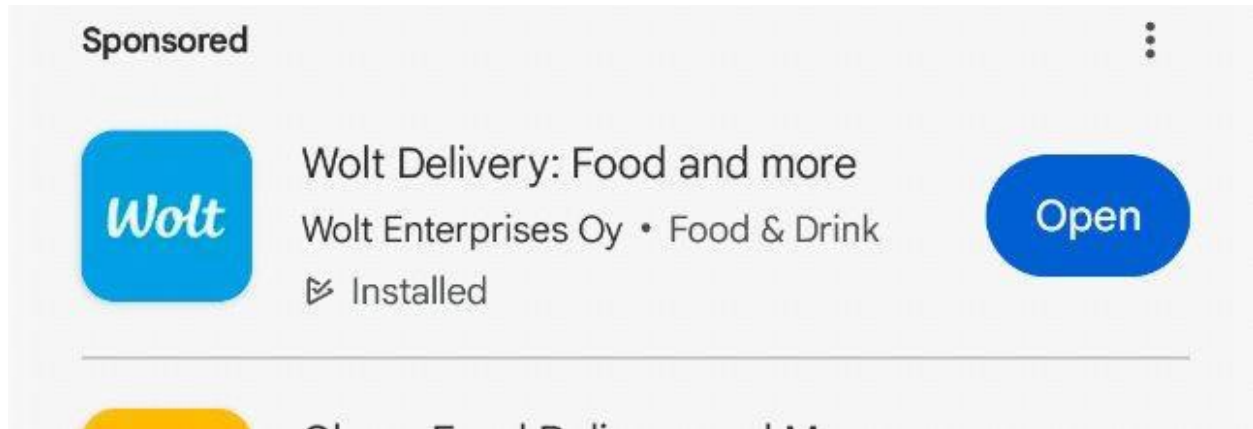
```
#####
==> Pouring jadx--1.5.1.all.bottle.tar.gz
🍺 /opt/homebrew/Cellar/jadx/1.5.1: 12 files, 121.2MB
==> Running `brew cleanup jadx`...
Disable this behaviour by setting HOMEBREW_NO_INSTALL_CLEANUP.
Hide these hints with HOMEBREW_NO_ENV_HINTS (see `man brew`).
```

Phase 2: Application Selection

Next, I selected the Wolt application (a food delivery service) because it clearly relies on network API communication.

I launched the AVD emulator, opened Google Play Store, signed in, installed Wolt, and confirmed the app was functioning properly.

wolt installed



Phase 3: APK Extraction

Then I extracted the APK using ADB:

adb devices

List of devices attached

emulator-5554 device

adb shell pm list packages | grep wolt

adb shell pm path com.wolt.android

adb pull /data/app/com.wolt.android-1/base.apk extracted_app.apk

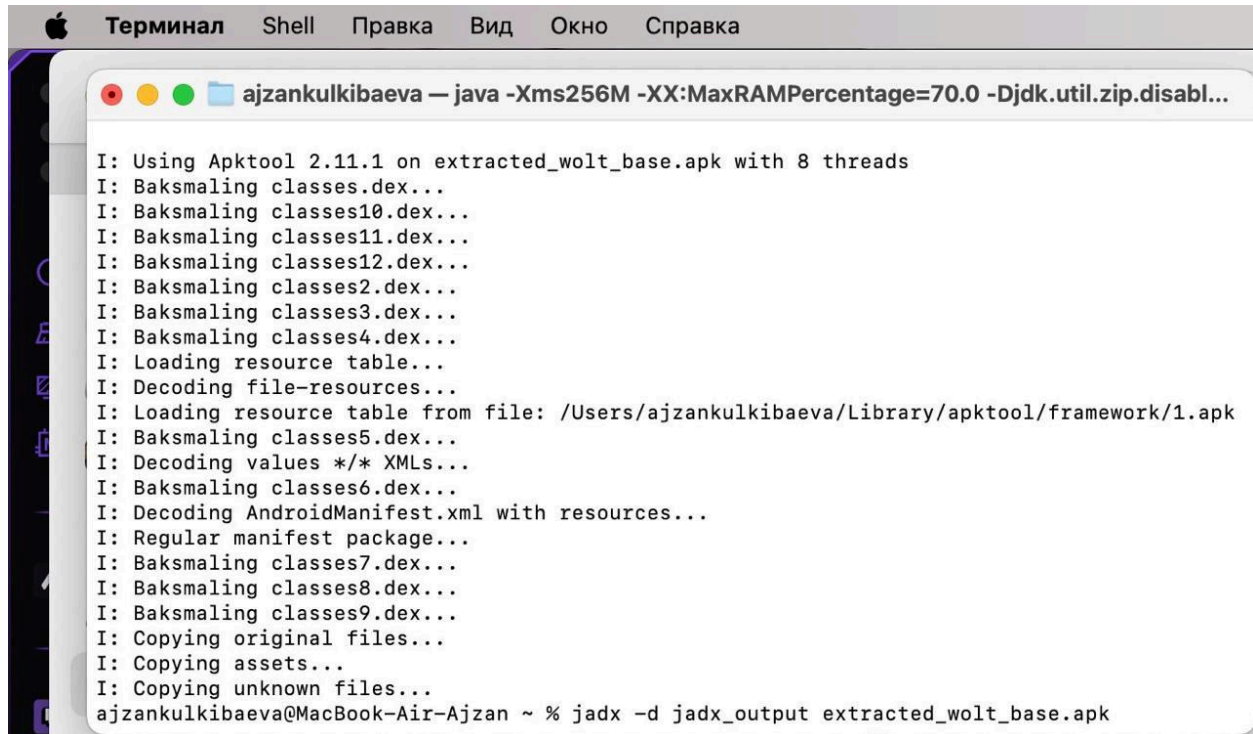
Phase 4: Decompiling the Application

I decompiled the APK using:

apktool d extracted_app.apk -o decompiled_app

jadx -d jadx_output extracted_app.apk

In the decompiled source, I found references to Retrofit and OkHttp, as well as obfuscated or encoded API endpoints.



```
Терминал  Shell  Правка  Вид  Окно  Справка
ajzankulkibaeva — java -Xms256M -XX:MaxRAMPercentage=70.0 -Djdk.util.zip.disabl...

I: Using Apktool 2.11.1 on extracted_wolt_base.apk with 8 threads
I: Baksmaling classes.dex...
I: Baksmaling classes10.dex...
I: Baksmaling classes11.dex...
I: Baksmaling classes12.dex...
I: Baksmaling classes2.dex...
I: Baksmaling classes3.dex...
I: Baksmaling classes4.dex...
I: Loading resource table...
I: Decoding file-resources...
I: Loading resource table from file: /Users/ajzankulkibaeva/Library/apktool/framework/1.apk
I: Baksmaling classes5.dex...
I: Decoding values */* XMLs...
I: Baksmaling classes6.dex...
I: Decoding AndroidManifest.xml with resources...
I: Regular manifest package...
I: Baksmaling classes7.dex...
I: Baksmaling classes8.dex...
I: Baksmaling classes9.dex...
I: Copying original files...
I: Copying assets...
I: Copying unknown files...
ajzankulkibaeva@MacBook-Air-Ajzan ~ % jadx -d jadx_output extracted_wolt_base.apk
```

Phase 4.5: SSL Certificate Pinning Bypass

The Wolt app uses SSL certificate pinning, which blocked HTTPS traffic interception.

To bypass it, I used Frida. I attached to the Wolt process and injected the following Frida script to disable certificate validation:

, I used two main tools: Frida and Burp Suite. To get everything working, I needed:

- A rooted Android device — I used an Android Studio emulator and rooted it.
- Burp Suite installed on my laptop.
- Frida installed in my Python environment.

First, I opened Burp Suite and exported its CA certificate in DER format — this is what helps the app trust my traffic proxy.

Then, I configured the Android Emulator's Wi-Fi proxy to use my laptop's IP (**10.0.2.2**) on port **8080**. That routes traffic through Burp Suite.

I wrote and loaded a Frida script — the one that bypasses SSL pinning. It works by hijacking the SSL context initialization and injecting my own certificate as trusted.

The script loads my **caacer-der.crt** file from the device's **/data/local/tmp/** path and inserts it into a new **TrustManager**.

Then I ran: **frida -U -f com.target.app -l fridascript.js --no-pause**
then,

```
adb push caacerder.crt /data/local/tmp
```

```
adb push frida-server /data/local/tmp
```

```
adb shell chmod 777 /data/local/tmp/frida-server
```

code :**fridascript.js**

```
setTimeout(function(){  
    Java.perform(function (){  
        console.log("");  
        console.log("[.] Cert Pinning Bypass/Re-Pinning");  
    });  
});
```

```
var CertificateFactory =
Java.use("java.security.cert.CertificateFactory");
var FileInputStream = Java.use("java.io.FileInputStream");
var BufferedInputStream =
Java.use("java.io.BufferedInputStream");
var X509Certificate = Java.use("java.security.cert.X509Certificate");
var KeyStore = Java.use("java.security.KeyStore");
var TrustManagerFactory =
Java.use("javax.net.ssl.TrustManagerFactory");
var SSLContext = Java.use("javax.net.ssl.SSLContext");

// Load CAs from an InputStream
console.log("[+] Loading our CA...")
var cf = CertificateFactory.getInstance("X.509");

try {
    var fileInputStream =
FileInputStream.$new("/data/local/tmp/cert-der.crt");
}
catch(err) {
    console.log("[o] " + err);
}

var bufferedInputStream =
BufferedInputStream.$new(fileInputStream);
var ca = cf.generateCertificate(bufferedInputStream);
bufferedInputStream.close();

var certInfo = Java.cast(ca, X509Certificate);
console.log("[o] Our CA Info: " + certInfo.getSubjectDN());

// Create a KeyStore containing our trusted CAs
console.log("[+] Creating a KeyStore for our CA...");
var keyStoreType = KeyStore.getDefaultType();
var keyStore = KeyStore.getInstance(keyStoreType);
```

```

keyStore.load(null, null);
keyStore.setCertificateEntry("ca", ca);

// Create a TrustManager that trusts the CAs in our KeyStore
console.log("[+] Creating a TrustManager that trusts the CA in our
KeyStore...");
var tmfAlgorithm = TrustManagerFactory.getDefaultAlgorithm();
var tmf = TrustManagerFactory.getInstance(tmfAlgorithm);
tmf.init(keyStore);
console.log("[+] Our TrustManager is ready...");

console.log("[+] Hijacking SSLContext methods now...")
console.log("[-] Waiting for the app to invoke SSLContext.init()...")

SSLContext.init.overload("[Ljavax.net.ssl.KeyManager;",
"[Ljavax.net.ssl.TrustManager;",
"java.security.SecureRandom").implementation = function(a,b,c) {
    console.log("[o] App invoked javax.net.ssl.SSLContext.init...");
    SSLContext.init.overload("[Ljavax.net.ssl.KeyManager;",
"[Ljavax.net.ssl.TrustManager;",
"java.security.SecureRandom").call(this, a, tmf.getTrustManagers(),
c);
    console.log("[+] SSLContext initialized with our custom
TrustManager!");
}
});
},0);

```

This allowed me to see HTTPS traffic in Burp Suite.

```

find ~/Downloads -name "caacer.crt"
/Users/ajzankulkibaeva/Downloads/caacer.crt

```

```
ajzankulkibaeva@MacBook-Air-Ajzan Desktop % adb push
/Users/ajzankulkibaeva/Downloads/caacer.crt
/data/local/tmp/cert-der.crt
/Users/ajzankulkibaeva/Downloads/caace...ipped. 1.6 MB/s (1326
bytes in 0.001s)
ajzankulkibaeva@MacBook-Air-Ajzan Desktop % frida -U -f
com.wolt.android -l /Users/ajzankulkibaeva/Desktop/fridascript.js
```

```

┌───┐
/_ |  Frida 16.7.14 - A world-class dynamic instrumentation toolkit
|(_)|
>_ |  Commands:
/_/|_|  help    -> Displays the help system
....   object? -> Display information about 'object'
....   exit/quit -> Exit
....
....   More info at https://frida.re/docs/home/
....
....   Connected to Android Emulator 5554 (id=emulator-5554)
Spawned `com.wolt.android`. Resuming main thread!
[Android Emulator 5554::com.wolt.android ]->
[.] Cert Pinning Bypass/Re-Pinning
[+] Loading our CA...
[o] Our CA Info: CN=PortSwigger CA, OU=PortSwigger CA,
O=PortSwigger, L=PortSwigger, ST=PortSwigger, C=PortSwigger
[+] Creating a KeyStore for our CA...
[+] Creating a TrustManager that trusts the CA in our KeyStore...
[+] Our TrustManager is ready...
[+] Hijacking SSLContext methods now...
[-] Waiting for the app to invoke SSLContext.init()...
[o] App invoked javax.net.ssl.SSLContext.init...
[+] SSLContext initialized with our custom TrustManager!
[o] App invoked javax.net.ssl.SSLContext.init...
[+] SSLContext initialized with our custom TrustManager!
[o] App invoked javax.net.ssl.SSLContext.init...
[+] SSLContext initialized with our custom TrustManager!
```

[o] App invoked javax.net.ssl.SSLContext.init...
[+] SSLContext initialized with our custom TrustManager!
[o] App invoked javax.net.ssl.SSLContext.init...
[+] SSLContext initialized with our custom TrustManager!
[o] App invoked javax.net.ssl.SSLContext.init...
[+] SSLContext initialized with our custom TrustManager!
[Android Emulator 5554::com.wolt.android]->
[Android Emulator 5554::com.wolt.android]-> **Process terminated**
[Android Emulator 5554::com.wolt.android]->
Thank you for using Frida!

Phase 5: Burp Suite Configuration



qemu-system-aarch64

ajzankulkibaeva — adb shell — 80x24

Android Emulator - Medium_Phone_...

2:06 ⓘ 🔒



⌂ ⓘ burp



This site can't be reached

Check if there is a typo in burp.

DNS_PROBE_FINISHED_NXDOMAIN





qemu-system-aarch64



~ — adb shell

ajzanku@MacBook-Air:~\$

arm64-v8a

ajzanku@MacBook-Air:~\$

List of

emulators

[ajzanku@MacBook-Air:~\$

emu64a:~\$



Dashboard

Target

Pr

Intercept

HTTP history

Android Emulator - Medium_Phone_...

2:17 ⓘ 🔒 👤



Network details

AndroidWifi

Proxy

Manual



The HTTP proxy is used by the browser but may not be used by the other apps.

Proxy hostname (optional)

10.48.147.240

Proxy port (optional)

8080

Bypass proxy for

example.com,mycomp.test.com,localhc

IP settings

DHCP



tem-aarch64

Android Emulator - Medium_Phone_...

2:18 ⓘ 🔒 🔑



burp



Burp Suite Community Edition

CA Certificate

Welcome to Burp Suite Community Edition.



[ajzankulki
emu64a:/ \$

Download file again?

File name already exists



cacert (1).der



Downloads



Don't show again

Cancel





qemu-system-aarch64

Android Emulator - Medium_Phone_...

3:18 ⓘ 🔒 ⚠️ •



wolt.com/en/kaz/almaty/



4



Wolt

Wolt App

The easiest way to order



Open

Okadzaki

Wolt

Q okadzaki



Restaurants and stores

See all



Okadzaki Seifullin

pizza, sushi, udon



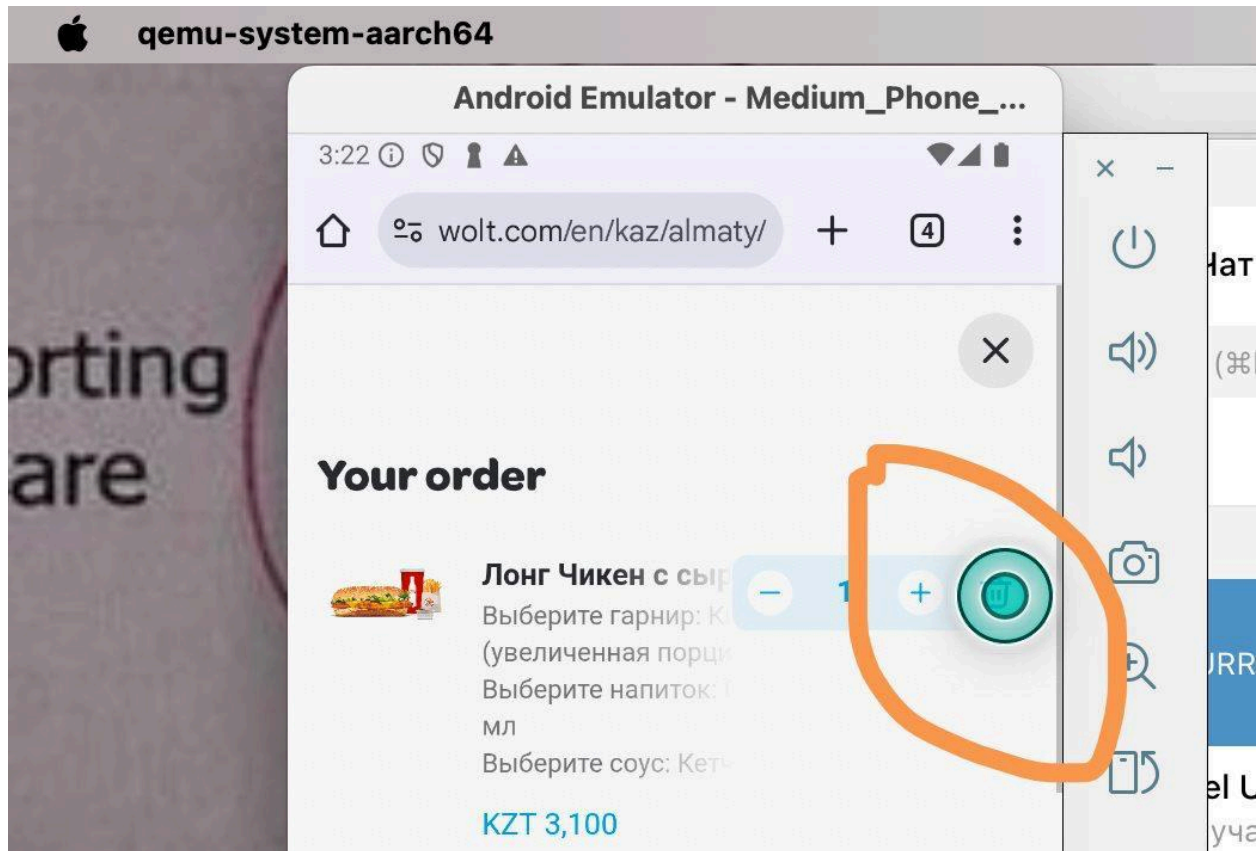
Okadzaki Meridian

pizza, sushi, poke



Ok

piz:



Phase 6,7: Intercepting API Requests

1. OPTIONS Request Overview:

- **Method: OPTIONS** — This pre-flight request checks if a cross-origin request from a different domain is allowed by the server.
- **Endpoint: </v1/pages/venue-list/category-burgers>**
- **Query Parameters:**
 - **lat=43.245132:** Latitude of the location.
 - **lon=76.954158:** Longitude of the location.

Purpose of the OPTIONS Request:

- This request is sent by browsers to check if a cross-origin request (such as from wolt.com) is allowed by the server. The pre-flight check asks the server if the actual GET request can be made.

- The server's response should specify which methods and headers are permitted for the actual request.

HTTP/2 200 OK

Access-Control-Allow-Origin: https://wolt.com

Access-Control-Allow-Methods: GET

Access-Control-Allow-Headers: app-currency-format, app-language, authorization, client-version, clientversionnumber, platform, w-wolt-session-id, x-wolt-web-clientid

Second Request: **/v1/wauth2/access_token**

- **Endpoint:** **/v1/wauth2/access_token**
- **Request Body:** Contains the Google token for token exchange (OAuth2 flow).
- **Error:** No error message in the logs yet.

This second request is attempting to exchange the Google OAuth token for an access token. If everything is configured correctly, you should get an access token in return.

How It Works:

- A valid Google token is sent in the body of the request (under **google_token**) and the grant type is set to **google_token**.
- If valid, the response will contain an access token (typically in JSON format).
- The access token will be used for further API requests to authenticate the user.

Phase 8: Demo Application Development

```
<!DOCTYPE html>
```

```
<html lang="kk">
```

```
<head>
```

```
  <meta charset="UTF-8">
```

```
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
  <title>Wolt Мейрамханалар</title>
```

```
  <style>
```

```
    body {
```

```
      font-family: Arial, sans-serif;
```

```
      margin: 0;
```

```
      padding: 0;
```

```
      background-color: #f4f4f4;
```

```
    }
```

```
    .container {
```

```
      width: 80%;
```

```
      margin: 0 auto;
```

```
      padding: 20px;
```

```
    }
```

```
    h1 {
```

```
      text-align: center;
```

```
    }
```

```
.restaurant-list {  
    display: flex;  
    flex-wrap: wrap;  
    justify-content: space-around;  
}  
  
.restaurant {  
    background-color: white;  
    margin: 10px;  
    padding: 15px;  
    width: 30%;  
    border-radius: 8px;  
    box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);  
}  
  
.restaurant img {  
    width: 100%;  
    border-radius: 8px;  
}  
  
.restaurant h2 {  
    margin-top: 10px;  
    font-size: 18px;  
}
```

```
.restaurant p {
    font-size: 14px;
    color: #555;
}

</style>

</head>

<body>

<div class="container">

    <h1>Wolt Мейрамханалар</h1>

    <div class="restaurant-list" id="restaurant-list">

        <!-- Мейрамханалар осында көрсетіледі -->

    </div>

</div>

<script>

    const apiUrl =
"https://consumer-api.wolt.com/v1/pages/restaurants?lat=43.245132&lon=7
6.954158";

    const options = {
        method: 'GET',
        headers: {
```

'Authorization': 'Bearer YOUR_API_KEY' // Егер API кілті қажет болса, оны қосыңыз

```
}  
};
```

```
fetch(apiUrl, options)
```

```
.then(response => response.json())
```

```
.then(data => {
```

```
    const restaurants = data.data.restaurants; // API-дің деректері
```

```
    const restaurantList = document.getElementById('restaurant-list');
```

```
    restaurants.forEach(restaurant => {
```

```
        const restaurantElement = document.createElement('div');
```

```
        restaurantElement.classList.add('restaurant');
```

```
        restaurantElement.innerHTML = `
```

```
            
```

```
            <h2>${restaurant.name}</h2>
```

```
            <p>${restaurant.description || 'Сипаттама жоқ.'}</p>
```

```
        `;  
    }
```

```
        restaurantList.appendChild(restaurantElement);

    });

})

.catch(error => {

    console.error('Қате орын алды:', error);

    const restaurantList = document.getElementById('restaurant-list');

    restaurantList.innerHTML = '<p>Қателік орын алды. Деректер
алынбады.</p>';

    });

</script>

</body>

</html>
```

