

Mastering React Native Application Pentesting: A Practical Guide

 **Payatu**
E B O O K



About the Author and Contributor



Vedant Wayal

**Senior Security Consultant
Mobile Tower - Payatu**

Vedant is an infosec enthusiast with over four years of experience in Mobile & Web application pentesting. He enjoys diving into new areas of research and creating CTF challenges, particularly in the mobile application security domain.

He has experience working on various mobile application assessments including native, hybrid, and cross-platform applications, and has performed assessments on various Android and iOS mobile applications for vulnerabilities and security flaws.



Tanvi Tirthani

**Content and Media Strategist
- Payatu**

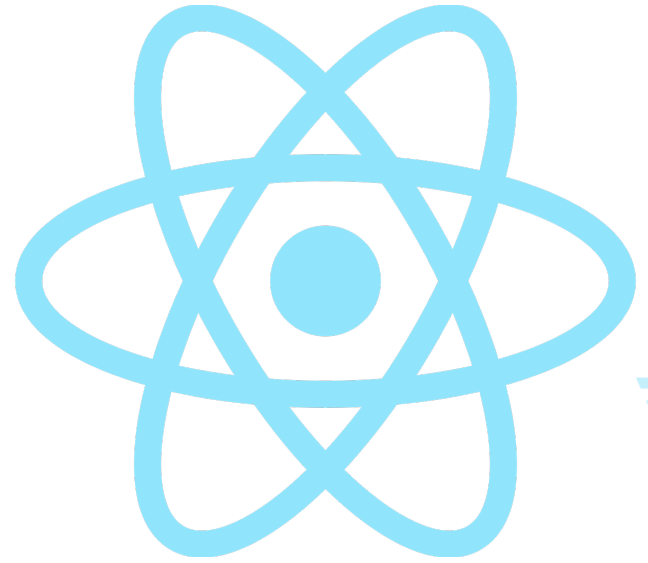
Tanvi is a Content and Media Strategist with a special foray into technology. With an MBA in Marketing, Tanvi is well equipped to develop memorable content collaterals, where technology comes easy to her!

At Payatu, you will find her working with the tech team to help them enrich their copies and assets, before they are rolled out to the general public. A lot of her time here is spent understanding the cybersecurity arena and penning things down in a distinct reflective manner.



Table of Contents

| | |
|---|-----------|
| Introduction | 1 |
| Chapter 1. What is React Native? | 4 |
| Chapter 2. The Bridge Concept | 10 |
| Chapter 3. Reverse Engineering React Native Apps | 13 |
| Chapter 4. How to Find out if the Application is Built on React Native? | 18 |
| Chapter 5. The Fun Part - Attack Surfaces Static Analysis | 25 |
| Chapter 6. Editing and Patching React Native Application | 41 |
| Chapter 7. Modifying Hermes bytecode | 49 |
| Chapter 8. SSL Certificate Pinning Bypass | 71 |
| Chapter 9. Identify Manually Installed npm Packages | 76 |
| Chapter 10. React Native npm Package CVEs Walkthrough | 83 |
| Final Thoughts | 88 |
| About Payatu | 90 |



Introduction





Nowadays, there is an emergence of cross-platform hybrid applications on a large scale. Many top organizations are adapting different frameworks to develop or even entirely rewrite their mobile applications.

In this wave, React Native framework is gaining popularity for building cross-platform mobile applications. Began as a hackathon project, React Native is designed on Facebook's React JavaScript toolkit, which extends the capabilities of the platform to native mobile app development.

What to expect from this ebook?

Apart from usual Android application pen-test cases, we have curated some out-of-the-box test cases and attack surfaces that you can use while specifically pentesting React Native applications.

This book covers React Native Android application's pentesting. However, most of the techniques can be used in iOS React Native applications as well. You will walk through:

- Introduction to React Native Framework
- React Native JS code to Java Native Code Translation
- React Native Application Architecture
- Reverse Engineering of React Native Application
- Static Analysis of React Native Android Application



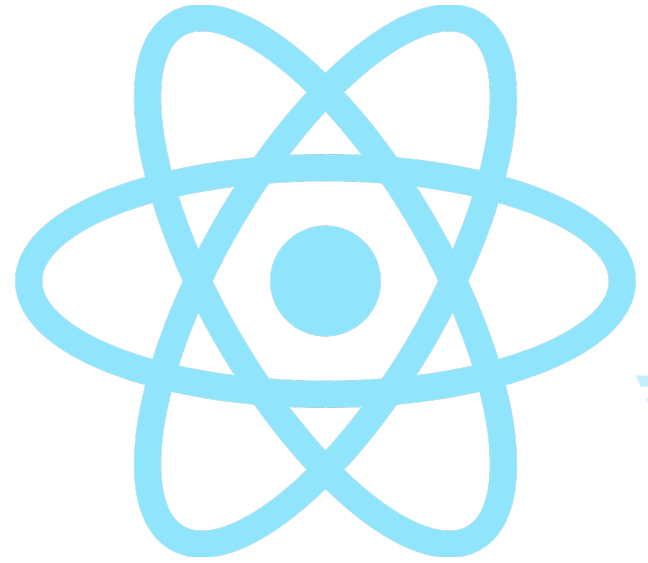


Prerequisites

It is assumed that the reader has prior knowledge about the following:

- >> Basic knowledge about Android applications. Below are some references to get started:
 - [Android pentesting lab](#)
 - [Android pentesting tools](#)
 - [Oversecured vulnerable app overview](#)
 - [Getting started with Frida on Android App](#)
 - [Android Security Part-1](#)
- >> Basic knowledge about JavaScript and webpack bundler
- >> Introductory knowledge about React Native language (Core react native, JSX, Babel)





Chapter 1

What is React Native?



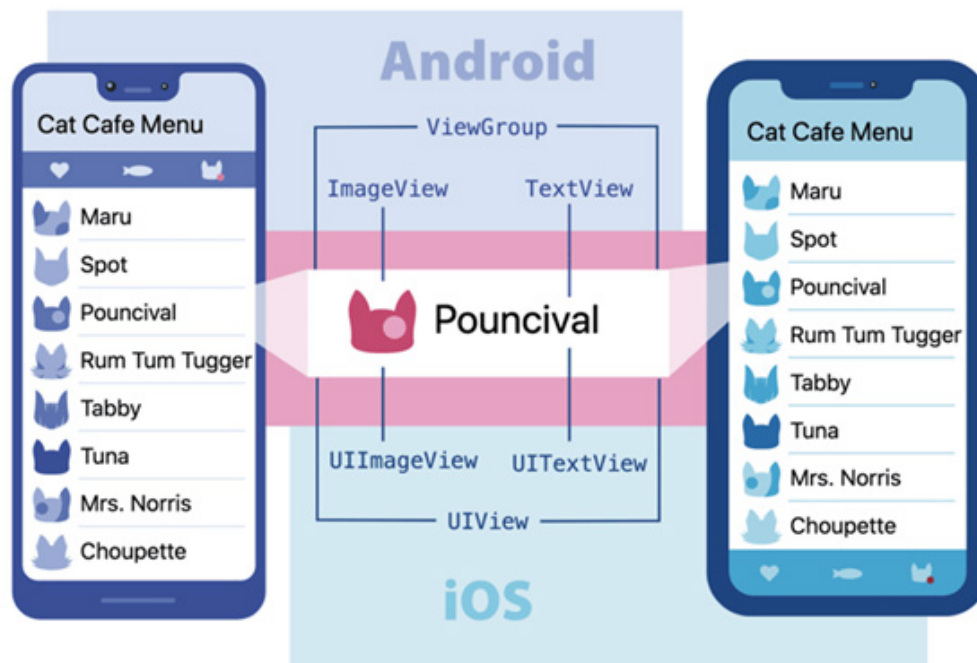


React Native is a JavaScript-based framework curated for developing native applications on platforms like Android and iOS. Facebook initially made React Native available as an open-source project in 2015. It quickly rose to the top of the list of tools used for mobile application development.

Why is there a lot of buzz around the React Native framework?

The tagline of React Native itself is **“Learn once, write anywhere.”** Thanks to the feature of re-using a large chunk of code of application across different platforms, React Native framework makes it easier to develop applications that provide a better user experience by utilizing the platform's features along with building apps that are easier to develop and operate on a wider range of platforms and devices.

We can write applications for different platforms such as iOS, and Android with minor tweaks in code as per the platform which translates into saving great time and resources. React Native combines the best parts of native development with React, a best-in-class JavaScript library for building user interfaces.



Cross-platform compatibility of React Native applications

Image Source: <https://dev.to/goodpic/understanding-react-native-architecture-22hh>





Examples of React Native applications:

Some of the prominent examples of React Native apps are:



Call of Duty Companion App



DonaldDaters



NerdWallet



Uber Eats



Wix

You can download the APKs of these apps and play around.

Why not use a hybrid application which displays data over WebView instead of React Native?

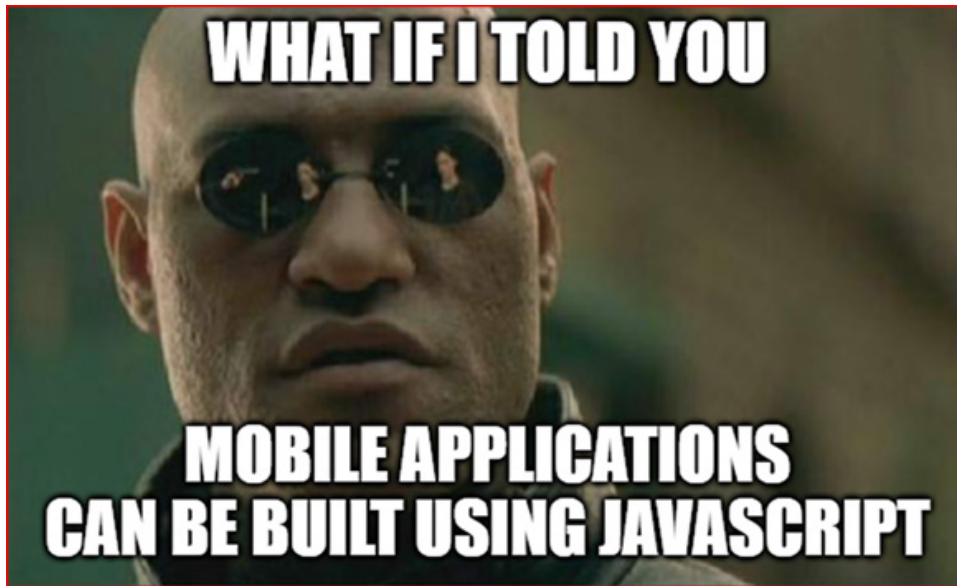
Web+Mobile hybrid applications are capable of displaying web content using WebView in native Android applications. Users can interact with the web content loaded inside the WebView. However, there are challenges to this type of architecture when the application wants to access the user's device resources such as camera, storage, various sensors, basic device information, etc.

React Native has made it possible to access these native features of the device along with JavaScript besides deploying on the web. For this, utilized is a "JavaScript Bridge" concept, which we will discuss in the upcoming segment of this ebook.

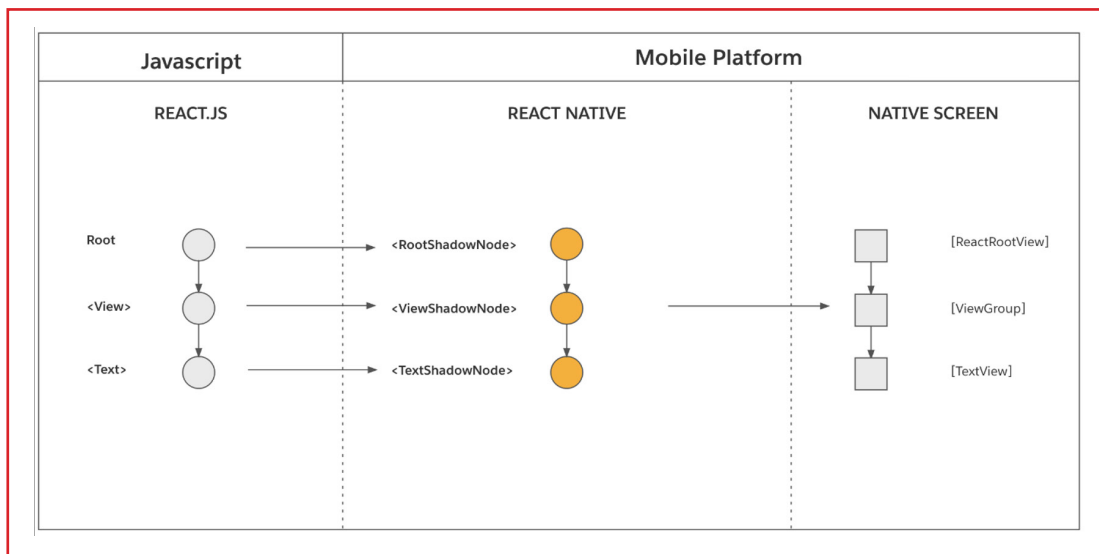




React Native Application Architecture



React Native applications are written in a combination of JavaScript and JSX. JSX is a special syntax extension to JavaScript. A key concept in React Native is "Component". A component is a piece of a user interface similar to the "Activities" in JAVA-based android applications. A React Native application can be made of multiple components which are interconnected. These components are composable and reusable throughout the application.



JavaScript to Native code translation

Image Source: <https://reactnative.dev/architecture/render-pipeline>



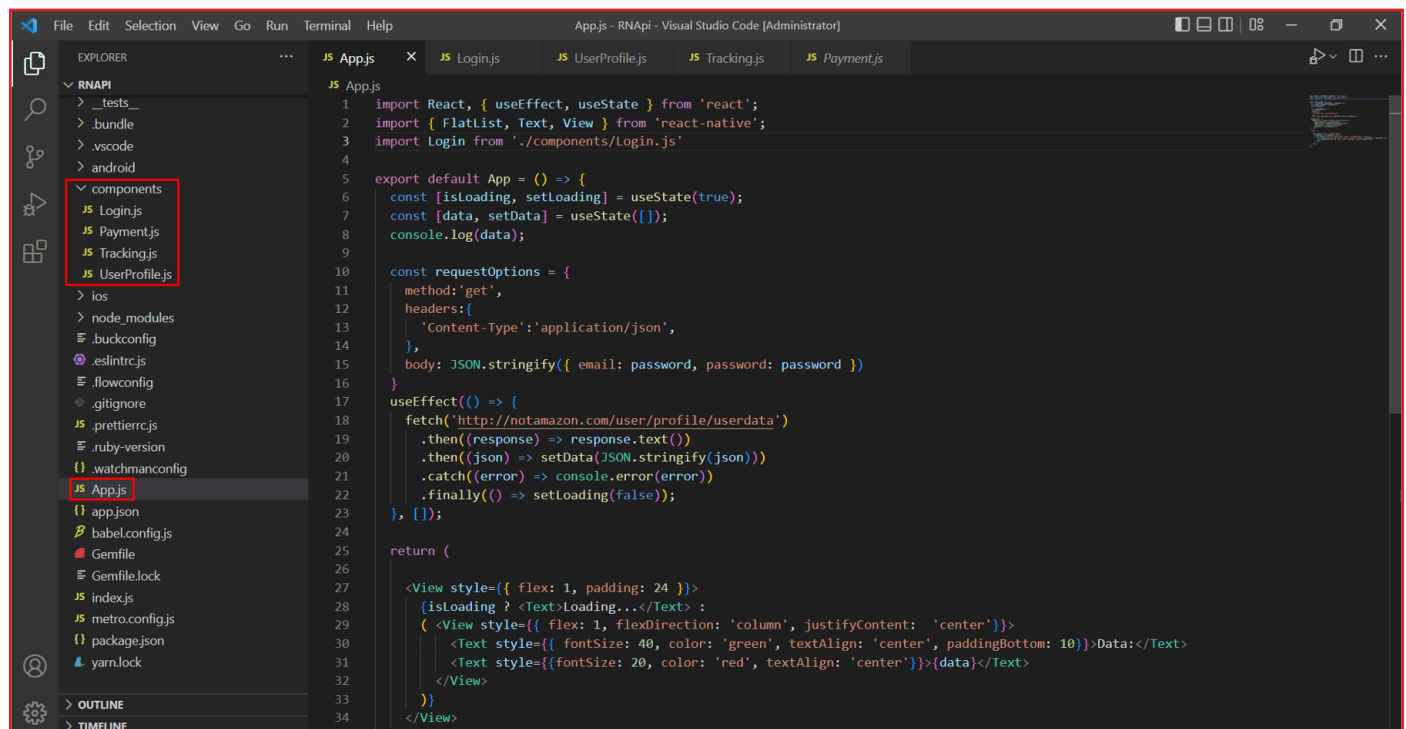


How does react native JavaScript code gets translated into the native code of the platform (Android/iOS)?

React Native brings React's declarative UI framework to iOS and Android platforms. With React Native, you use native UI controls and have full access to the native platform features.

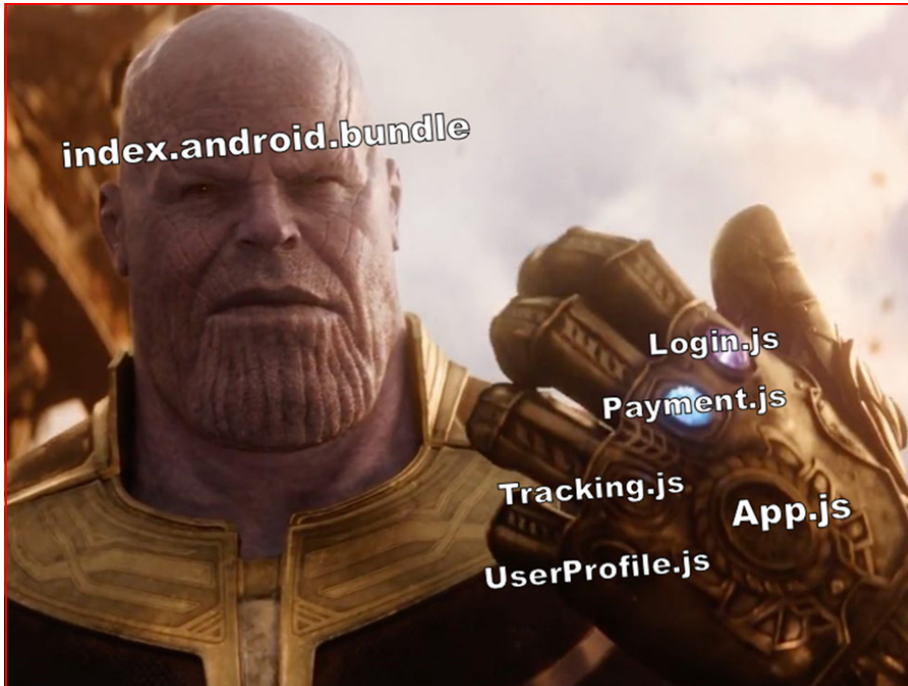
As we discussed above, React Native app can have multiple components. During the compilation, all of the components get compiled into one single file as demonstrated below:

Sample React Native project structure:



As you can see above; the application consists of multiple JS component files during development. However, during compilation into APK, all the code in these multiple JS components gets bundled into one single file i.e., "index.android.bundle".



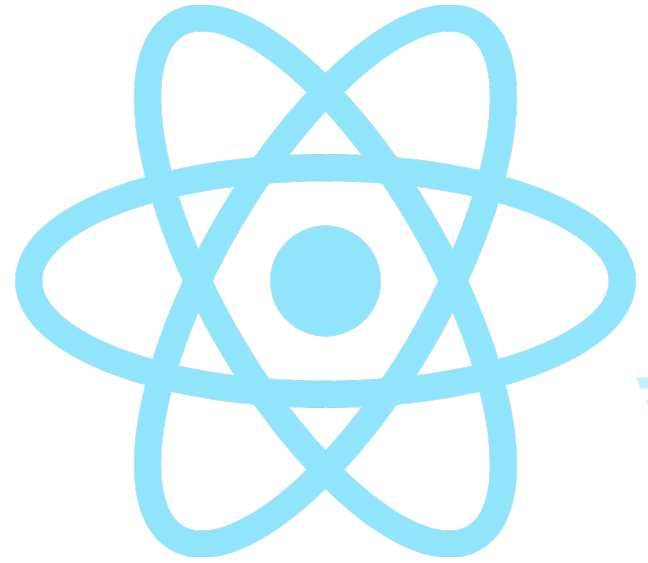


Before understanding "How things work?", we need to first understand "What are those things?"

A very brief overview of the React Native application's workflow:

- 🌀 We write code in JavaScript
- 🌀 This JavaScript code gets converted into Native code





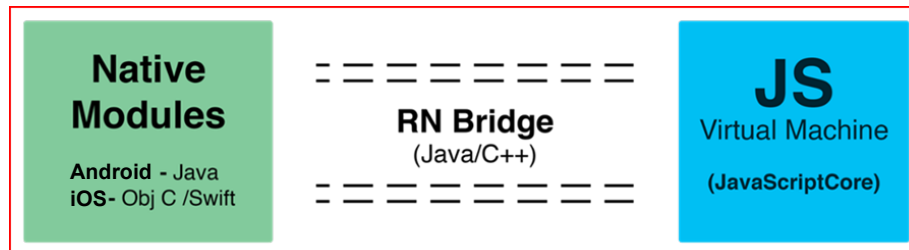
Chapter 2

The Bridge Concept





React Native deals with two realms, JavaScript and Native. The communication happens between these two realms over a communication channel called the “Bridge”. As the name suggests, it provides a literal bridge for these two realms to communicate. Bridge provides a way for bidirectional and asynchronous communication. In short, it provides a way of communication for completely two different technologies i.e., JavaScript and Native.



React Native bridge concept

Image Source: <https://approov.io/blog/react-native-bridging-an-Android-native-module-for-app-authentication>

Now coming back to “How things work?”, below is how the JS code gets translated into a mobile application.

- React Native app is written in JavaScript + JSX.
- The Bridge sends this JavaScript code to the JavaScript core Runtime to further communicate with native components.
- The communication happens in multiple threads. As React Native is asynchronous, each code and process run in different threads. For example, layout calculations happen in one thread while native code rendering happens in another. These two threads never communicate directly and never block each other.
- JavaScript threads communicate with Native threads via the Bridge.
- Finally, the native components of the platform communicate with iOS/Android SDK and execute the operation instructed in the initial JS code.

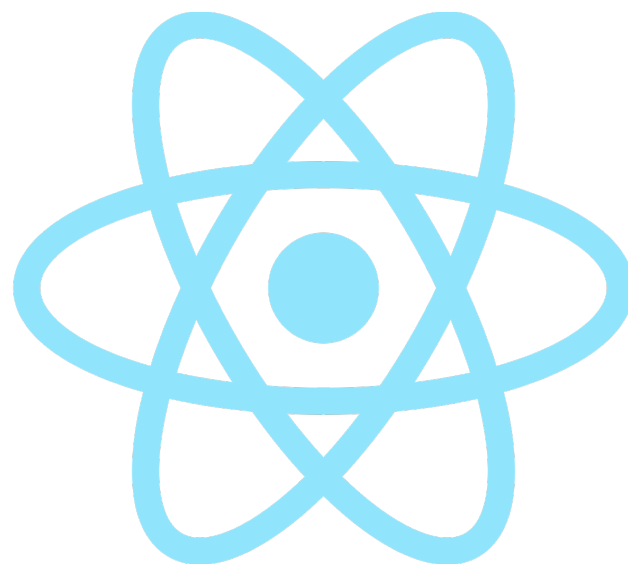




TL;DR

- **React Native code** ⇒ JS+JSX code written during development.
- **JS core engine** ⇒ Converts JS code into respective native code with the help of a bridge between React Native and Native code.
- **Bridge** ⇒ Facilitates communication between JavaScript code and Native components.
- **Android/iOS native code** ⇒ Runs the converted Native code on the platform and also provides support for all native features such as camera access, sensor access, device information, etc.





Chapter 3

Reverse Engineering React Native Apps





Enough with the theory, it's time to get to real business.

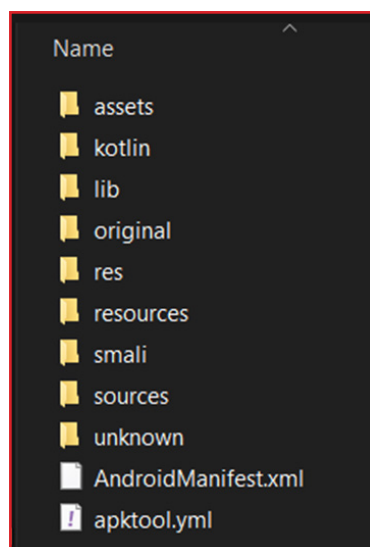


Method 1: Using the APKTool:

This method will convert files such as “AndroidManifest.xml”, “classes.dex” etc files into human-readable format. (Unlike Method 2)

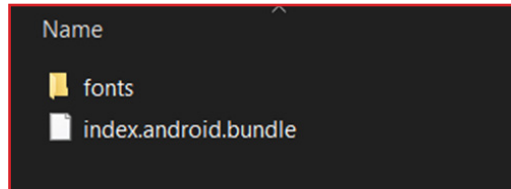
Steps:

1. Install **APKTOOL**: <https://ibotpeaches.github.io/APKtool/>
2. Open cmd and type
`apktool d app.APK`
3. The application will be decompiled.





4. Go to “/assets/” folder. It should contain the “index.android.bundle” file.



5. If you open this file, you will find all React Native JS code in minified format.

```
index.android.bundle [3]
241255 var _NotificationRedux2 = babelHelpers.interopRequireDefault(_NotificationRedux);
241256
241257 var _marked = regeneratorRuntime.mark(addNotification);
241258
241259 function addNotification(api, _ref) {
241260   var notification = _ref.notification;
241261   var custom_notification, id;
241262   return regeneratorRuntime.wrap(function addNotification$(_context) {
241263     while (1) {
241264       switch (_context.prev = _context.next) {
241265         case 0:
241266           _context.prev = 0;
241267           if (!notification) {
241268             _context.next = 8;
241269             break;
241270           }
241271         }
241272         custom_notification = notification.custom_notification ? JSON.parse(notification.custom_notification) : {}, id = notification.id || notification.userid ||
241273         custom_notification.id;
241274
241275         if (!id) {
241276           _context.next = 6;
241277           break;
241278         }
241279         _context.next = 6;
241280         return (0, _effects.put)(_NotificationRedux2.default.storeNotifications(id));
241281
241282         case 6:
241283           _context.next = 10;
241284           break;
241285
241286         case 8:
241287           _context.next = 10;
241288           return (0, _effects.put)(_NotificationRedux2.default.storeNotifications('interactions'));
241289
241290         case 10:
241291           _context.next = 15;
241292           break;
241293
241294         case 12:
241295           _context.prev = 12;
241296
```

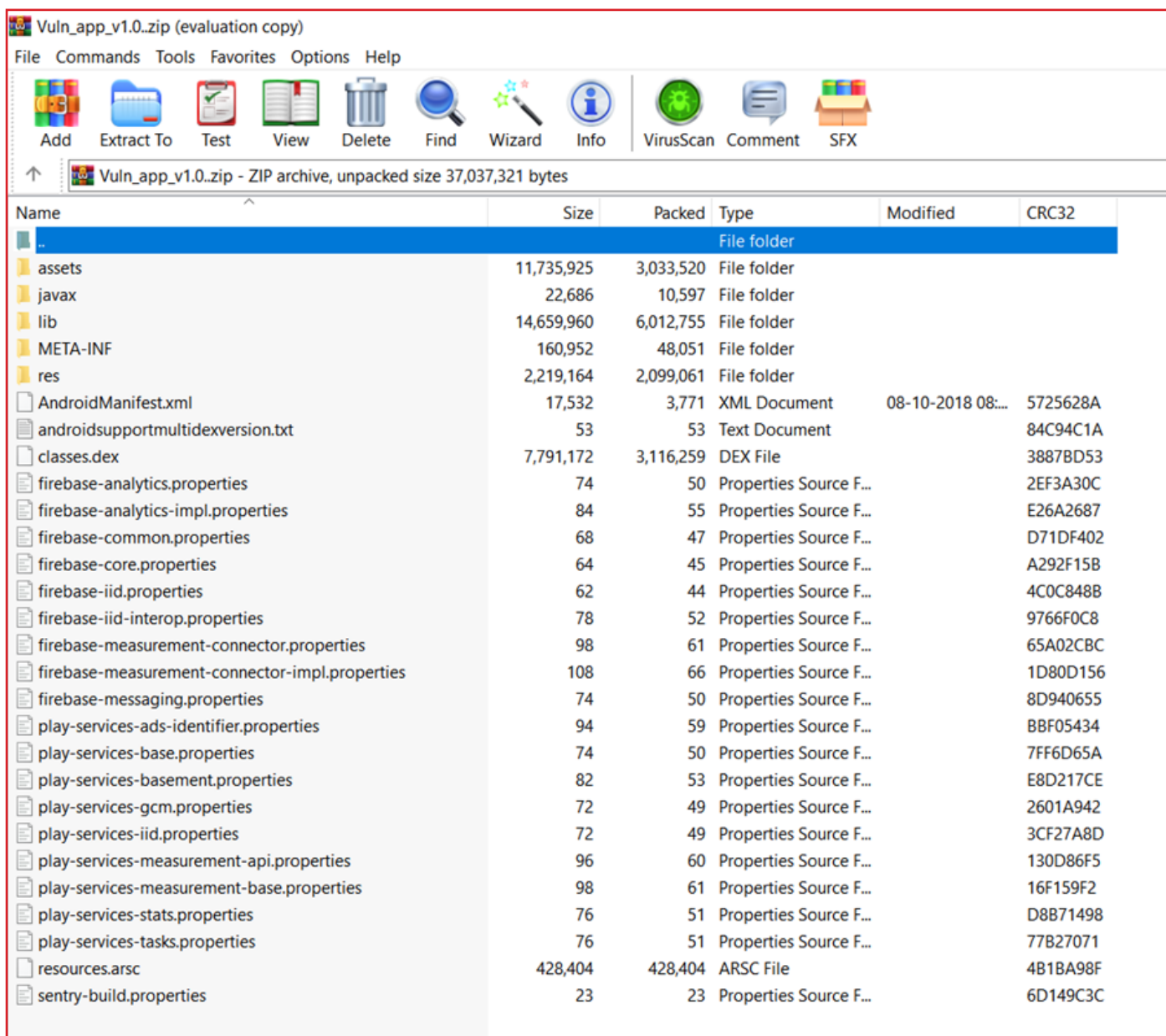




Method 2: Using Any Compression Tool

If you directly want to decompile applications without needing any tool, Method 2 is all you are looking for.

1. Rename the extension of APK file to .zip
2. Now open this file with any compression management tool such as winzip, 7zip

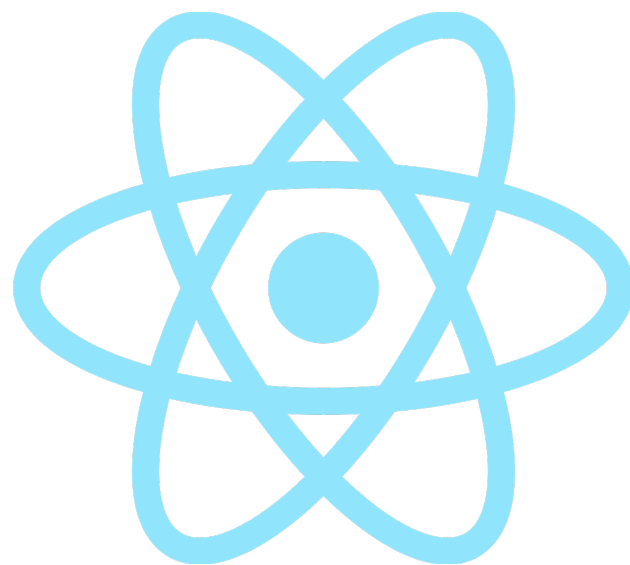




3. Extract all the files in that zip and you will be able to access the files. Some of the files will not be in a human-readable format.

| Name | Type | Size |
|--|------------------------|----------|
| assets | File folder | |
| javax | File folder | |
| lib | File folder | |
| META-INF | File folder | |
| res | File folder | |
| AndroidManifest.xml | XML Document | 18 KB |
| androidsupportmultidexversion.txt | Text Document | 1 KB |
| classes.dex | DEX File | 7,609 KB |
| firebase-analytics.properties | Properties Source File | 1 KB |
| firebase-analytics-impl.properties | Properties Source File | 1 KB |
| firebase-common.properties | Properties Source File | 1 KB |
| firebase-core.properties | Properties Source File | 1 KB |
| firebase-iid.properties | Properties Source File | 1 KB |
| firebase-iid-interop.properties | Properties Source File | 1 KB |
| firebase-measurement-connector.properties | Properties Source File | 1 KB |
| firebase-measurement-connector-impl.properties | Properties Source File | 1 KB |
| firebase-messaging.properties | Properties Source File | 1 KB |
| play-services-ads-identifier.properties | Properties Source File | 1 KB |
| play-services-base.properties | Properties Source File | 1 KB |
| play-services-basement.properties | Properties Source File | 1 KB |
| play-services-gcm.properties | Properties Source File | 1 KB |
| play-services-iid.properties | Properties Source File | 1 KB |
| play-services-measurement-api.properties | Properties Source File | 1 KB |
| play-services-measurement-base.properties | Properties Source File | 1 KB |
| play-services-stats.properties | Properties Source File | 1 KB |
| play-services-tasks.properties | Properties Source File | 1 KB |
| resources.arsc | ARSC File | 419 KB |
| sentry-build.properties | Properties Source File | 1 KB |





Chapter 4

How to Find out if the Application is Built on React Native?





1. Check the presence of the “index.android.bundle” file

- a. Follow the steps mentioned above to decompile the application.
- b. Among the extracted folders, check if the “/assets/index.android.bundle” file is present. This confirms that the application is built on React Native.

| Name | Type | Size |
|-----------------------------------|-----------------------|-----------|
| fonts | File folder | |
| include | File folder | |
| shared | File folder | |
| DigiCertHighAssuranceEVRootCA.crt | Security Certificate | 1 KB |
| entrust_g2_ca.cer | Security Certificate | 2 KB |
| help_center_article_style.css | Cascading Style Sh... | 9 KB |
| index.android.bundle | BUNDLE File | 17,416 KB |
| supplierconfig.json | JSON Source File | 1 KB |

2. Check the “com.facebook.react” string in “AndroidManifest.xml” file

- a. Decompile the application using the APKtool as mentioned above.
- b. Open “AndroidManifest.xml” file and search for “com.facebook.react” string.

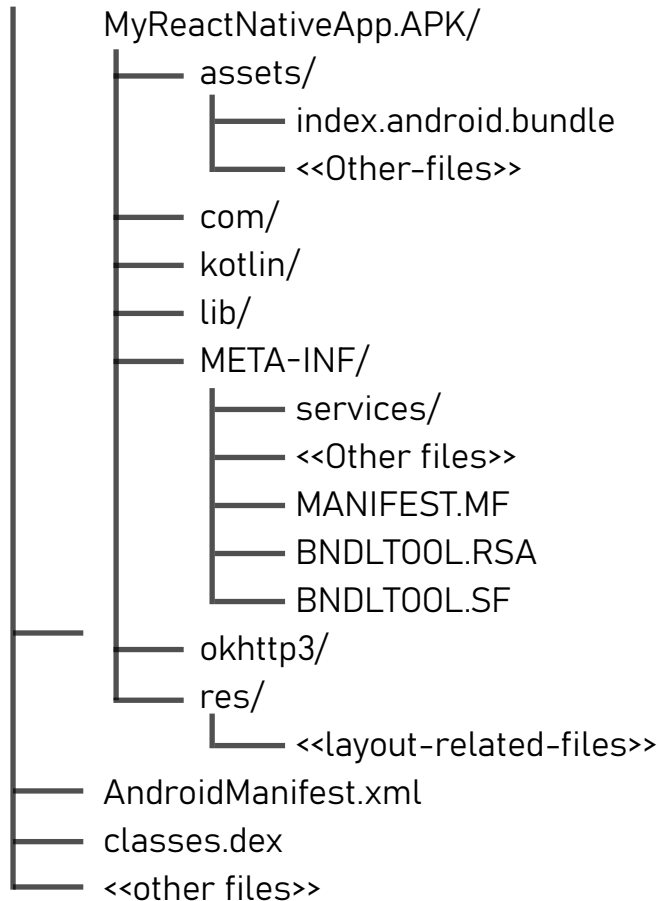
```
183 <action android:name="android.intent.action.VIEW"/>
184 <category android:name="android.intent.category.DEFAULT"/>
185 <category android:name="android.intent.category.BROWSABLE"/>
186 <data android:host="cct.com.wix.android" android:scheme="fbconnect"/>
187 </intent-filter>
188 </activity>
189 <activity android:exported="false" android:name="com.facebook.react.devsupport.DevSettingsActivity"/>
190 <activity android:excludeFromRecents="true" android:exported="false" android:label="@string/app_name"
  android:name="com.stripe.stripeterminal.UsbEventReceiverActivity" android:noHistory="true"
  android:process=":UsbEventReceiverActivityProcess" android:taskAffinity=
  "com.stripe.stripeterminal.taskAffinityUsbEventReceiver" android:theme="@style/Theme.Transparent">
191 <intent-filter>
192 <action android:name="android.hardware.usb.action.USB_DEVICE_ATTACHED"/>
193 </intent-filter>
194 <meta-data android:name="android.hardware.usb.action.USB_DEVICE_ATTACHED" android:resource=
  "@xml/usb_device_filter"/>
195 </activity>
196 <activity android:name="wix.com.mediamanager.newupload.camera.CameraActivity" android:screenOrientation=
  "portrait"/>
197 <activity android:configChanges="keyboardHidden|orientation|screenSize" android:exported="true"
```





React Native APK file structure:

Let's sneak into a release build of a sample React Native Android application. Once decompiled, the basic React Native Android application consists of the following contents:



Note: There are lots of files and directories in this hierarchy. However, only important files and directories are shown in the above diagram.





Let's understand some important files and directories:

assets/

- The “assets/” folder is the important directory to look for while reversing the React Native applications. It contains assets such as fonts, .json files, .properties files, extended JavaScript files along with “index.android.bundle” file.
- index.android.bundle: This file is the heart and soul of React Native applications as it contains the entire core logic of the application. It's a JavaScript bundle file and all of the application's JavaScript+JSX code is compiled into this file in minified format. We will learn more about this file in upcoming sections.

| Name | Type | Size |
|-------------------------------|-----------------------|----------|
| containers | File folder | |
| fonts | File folder | |
| aps_mobile_client_config.json | JSON Source File | 1 KB |
| aps-mraid.js | JavaScript File | 11 KB |
| baseman.txt | Text Document | 1 KB |
| comScore.properties | Properties Source ... | 1 KB |
| dtb-m.js | JavaScript File | 15 KB |
| index.android.bundle | BUNDLE File | 8,664 KB |
| supplierconfig.json | JSON Source File | 1 KB |
| template.html | Microsoft Edge HT... | 2 KB |

kotlin/

- Contains Kotlin code files. These files contain data for declarations of standard ("built-in") Kotlin classes which are not compiled to .class files, but rather are mapped to the existing types on the platform (in this case, JVM). For example, kotlin/kotlin.kotlin_builtins contains the information for non-physical classes in package Kotlin: Int, String, Enum, Annotation, Collection, etc.





META-INF/

- This directory is also one of the important directories while reversing React Native applications. The META-INF folder contains the manifest information and other metadata about the java package carried by the jar file.
- This means it also contains application signing-related files such as mentioned below. We will come back to this folder again in the pentesting part of the article.
 - MANIFEST.MF
 - It contains various information used by the java run-time environment when loading the jar file, such as which is the main class to be run from the jar file, the version of the package, build number, creator of the package, security policies/permissions of java applets and java webstart packages, the list of file names in the jar along with their SHA1 digests, etc.
 - BNDLTOOL.RSA
 - This contains the list of all files along with their SHA-1 digest.
 - BNDLTOOL.SF
 - This contains the signed contents of the CERT.SF file along with the certificate chain of the public key used for signing the contents.

| Name | Type | Size |
|--|--------------|--------|
| services | File folder | |
| MANIFEST.MF | MF File | 403 KB |
| BNDLTOOL.RSA | RSA File | 2 KB |
| BNDLTOOL.SF | SF File | 403 KB |
| androidx.activity_activity.version | VERSION File | 1 KB |
| androidx.activity_activity-ktx.version | VERSION File | 1 KB |
| androidx.annotation_annotation-experim... | VERSION File | 1 KB |
| androidx.appcompat_appcompat.version | VERSION File | 1 KB |
| androidx.appcompat_appcompat-resour... | VERSION File | 1 KB |
| androidx.arch.core_core-runtime.version | VERSION File | 1 KB |
| androidx.asynclayoutinflater_asynclayouti... | VERSION File | 1 KB |
| androidx.autofill_autofill.version | VERSION File | 1 KB |
| androidx.biometric_biometric.version | VERSION File | 1 KB |
| androidx.browser_browser.version | VERSION File | 1 KB |
| androidx.cardview_cardview.version | VERSION File | 1 KB |
| androidx.coordinatorlayout_coordinatorl... | VERSION File | 1 KB |





res/

- Contains all non-code resources, such as XML layouts, UI strings, and bitmap images, divided into corresponding sub-directories.

| Name | Type | Size |
|---------------------|-------------|------|
| anim | File folder | |
| animator | File folder | |
| animator-v21 | File folder | |
| anim-v21 | File folder | |
| color | File folder | |
| color-night-v8 | File folder | |
| color-v23 | File folder | |
| color-v26 | File folder | |
| color-v31 | File folder | |
| drawable | File folder | |
| drawable-anydpi-v21 | File folder | |
| drawable-anydpi-v23 | File folder | |
| drawable-anydpi-v24 | File folder | |
| drawable-hdpi-v4 | File folder | |
| drawable-hdpi-v23 | File folder | |

AndroidManifest.xml

- AndroidManifest.xml is one of the most important file in any android application. If you know even a little bit about android app pentesting, you may know the importance of this file. The AndroidManifest.xml file contains information about your package, including components of the application such as activities, services, broadcast receivers, content providers, etc.
- While developing and compiling React Native applications, this file is automatically generated by Gradle. Therefore, some of the configurations in this file are vulnerable by default. We will check this file out later in the pentesting part.

| Name | Date modified | Type | Size |
|-----------------------------------|---------------------|---------------|----------|
| assets | 25-08-2022 11:30 AM | File folder | |
| com | 25-08-2022 11:30 AM | File folder | |
| google | 25-08-2022 11:30 AM | File folder | |
| kotlin | 25-08-2022 11:30 AM | File folder | |
| lib | 25-08-2022 11:30 AM | File folder | |
| META-INF | 25-08-2022 11:30 AM | File folder | |
| okhttp3 | 25-08-2022 11:30 AM | File folder | |
| res | 25-08-2022 11:30 AM | File folder | |
| AndroidManifest.xml | 01-01-1981 01:01 AM | XML Document | 39 KB |
| androidsupportmultidexversion.txt | 01-01-1981 01:01 AM | Text Document | 1 KB |
| classes.dex | 01-01-1981 01:01 AM | DEX File | 9,563 KB |



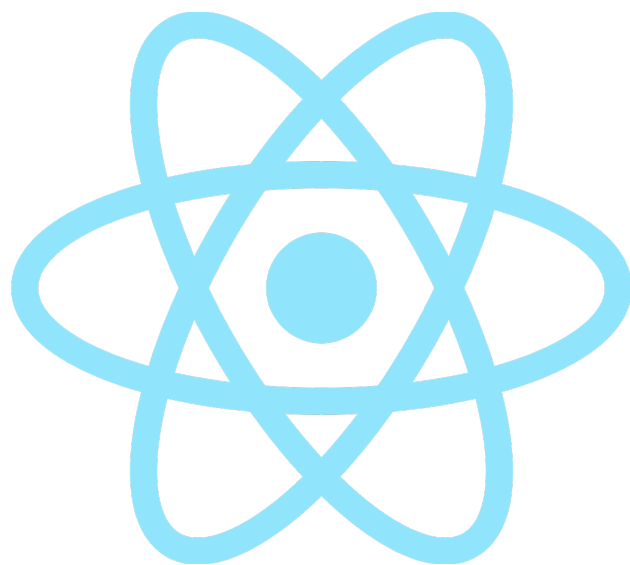


classes.dex

- Even if the React Native applications are written in JavaScript, when they get translated into the android application, Java bytecode code is generated automatically to run the application using ART. This file contains the Dalvik bytecode of this Java bytecode.
- You may find multiple classes.dex files in the apk because of the limitation of dex size (65K) for a single dex file. Multidexing is used in this situation and that's why you will find multiple dex files in the apk.

| Name | Type | Size |
|-----------------------------------|-----------------------|----------|
| assets | File folder | |
| com | File folder | |
| google | File folder | |
| kotlin | File folder | |
| lib | File folder | |
| META-INF | File folder | |
| okhttp3 | File folder | |
| res | File folder | |
| AndroidManifest.xml | XML Document | 39 KB |
| androidsupportmultidexversion.txt | Text Document | 1 KB |
| classes.dex | DEX File | 9,563 KB |
| classes2.dex | DEX File | 7,544 KB |
| classes3.dex | DEX File | 6,263 KB |
| classes4.dex | DEX File | 1,241 KB |
| core.properties | Properties Source ... | 1 KB |
| firebase-abt.properties | Properties Source ... | 1 KB |





Chapter 5

The Fun Part - Attack Surfaces & Static Analysis

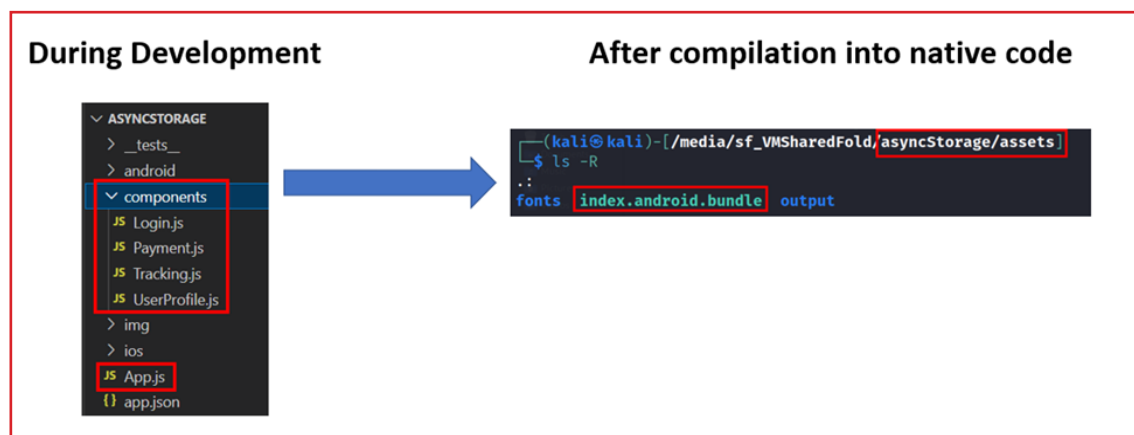




Sensitive information in the “index.android.bundle” file

What is the “index.android.bundle” file?

- In React Native, App.js acts like Main.java. When React Native apps get compiled into an APK file, the React Native index files and components get converted into JS code via JS bridge.
- In React Native applications, all of the JavaScript code written in the project gets compiled into the “index.android.bundle” file when the application is built. Thus, this file contains all of the JavaScript code of the application in minified format.
- When you decompile the React Native apk, the contents of the main ‘App.js’ file and all other components will be bundled together in JS format in the “index.android.bundle” file as mentioned above. This means the “index.android.bundle” file contains all of the source code of React Native application. We can search for hardcoded stuff in this file.



Steps:

1. Decompile application using APKtool.
2. Locate “index.android.bundle” file in /assets folder.

```
/<appfolder>/assets/
```





There is a lot of sensitive information that might be hardcoded in the application component files which later gets compiled into the "index.android.bundle" file. We will look at some types of information that we can find in this file.

1. Hardcoded credentials and tokens:

The poor management of the credentials and tokens is a naïve mistake that we find happening in lots of Android applications. The React Native application is no exception. In fact, hardcoding of the stuff is much higher in React Native as compared to regular native Java apps, and "index.android.bundle" is a goldmine for the hardcoded stuff.

You can search for keywords such as "secrets, tokens, password, apikey, username, login" etc. to find such goofy hidden secrets in the "index.android.bundle" file.

```
    this.resetError = function (callback) {
      this.setState({ error: null }, function () {
        if ('function' == typeof callback) callback();
      });
    };

    this.api = _Api2.default.create(props.access_token);
    var fsxToken =
    "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IjY4RG91IiwiaWF0IjoxNTE2MjM5MD0",
    this.state = {
      current: {
        text: '',
        placeholder: _this.getPlaceholder(),
        valid: false
      },
      password: {
        text: ''
```

2. Third-party database credentials:

Most React Native applications use third-party databases such as firebase to store information. There have been numerous instances of hardcoding credentials of these third-party databases. Lots of credentials are too permissive within their React Native application.

The following keywords can be used to grab these credentials within the target React Native application:

- apiKey
- FIREBASE_API_KEY
- endpoint
- storageBucket
- FIREBASE_API_KEY
- aws
- firebase
- databaseURL





```
index.android.bundle x
136548
136549 var firebaseConfig = {
136550   apiKey: "AIzaSyA8IRdH9FRATKCFBIQmGafE [REDACTED] ",
136551   authDomain: [REDACTED].firebaseapp.com",
136552   databaseURL: "https://[REDACTED].firebaseio.com",
136553   storageBucket: "[REDACTED].appspot.com"
136554 };
136555
136556 firebase.initializeApp(firebaseConfig);
136557
136558 exports.default = firebase;
136559 }, 2621, null, "[REDACTED]/App/Config/Firebase.js");
136560 __d(/* firebase/index.react-native.js */function(global, require, module, exports) {
136561
136562 var firebase = require(2623  ); // 2623 = ./app
136563 require(2629  ); // 2629 = ./auth
```

3. Hidden backdoors and URLs

Developers tend to add backdoors or URLs in the code for various purposes such as debugging, shortcuts to the functionality for convenience, etc. Sometimes they forgot to remove those URLs, and shortcuts while deploying built to the productions. We can scratch through the file to find these hidden URLs and shortcuts.

```
}, [(0, i.default)()]);
e.default = p
, 6289, [2, 61, 142, 16, 2014, 2030, 2836, 4880, 2029, 1806, 1808]];
_d(function(g, r, i, a, m, e, d) {
  m.exports = r(d[0]).registerAsset({
    __packager_asset: 10,
    httpServerLocation: "https://92.135.43.12/dev/prod-server/",
    width: 73.33333333333333,
    height: 72,
    scales: [1.5, 2, 3, 4],
    hash: "d4f9d8d58b3067de8eaf80d5d8e3b8cc",
    name: "cards_activate",
    type: "png"
```

The "index.android.bundle" file contains the core code of the application. Thus, sometimes this file can be huge to analyse. We can be more creative with custom keywords depending upon the type of application, technology, frameworks used in the application, etc.





Splitting “index.android.bundle” code into multiple JS components

As we saw above, all of the JS code is crunched into one “index.android.bundle” file. Navigating through this bulk code is a headache. Fortunately, there is a way to break down this bundle code into multiple JS files with the help of the following npm module.

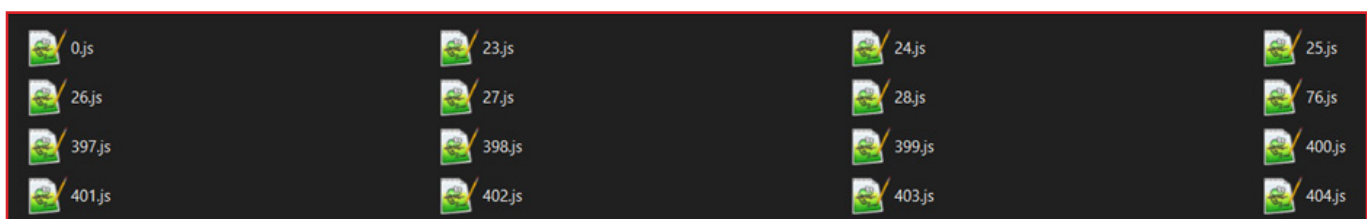
[react-native-decompiler](#)

Steps:

1. Install above mentioned “react-native-decompiler” module. Check the installation instruction in the URL of the module above.
2. Unzip the contents of the vulnerable application into a folder and go to the “assets” folder.
3. Now open command prompt in “assets” folder and type following command:

```
npx react-native-decompiler -i ./index.android.bundle -o ./output
```

4. Wait for the process to complete and the “index.android.bundle” file will be decompiled into multiple JS modules in the “output” folder.



5. Unfortunately, most of the React Native android application does not generate source-map file unlike React js web applications. Thus, we have to manually navigate through various components of the application.



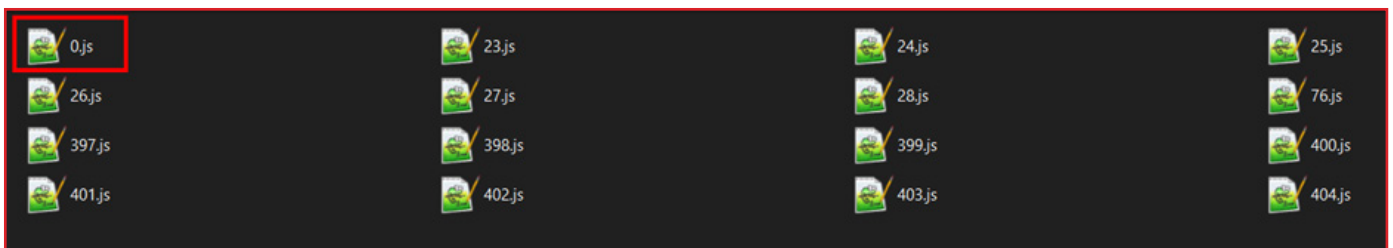


Navigating through multiple decompiled JS modules:

You got decompiled JS files but things are still messy. Let's simplify things. We will navigate through these files to reach to right code.

Steps:

1. Once you decompile the "index.android.bundle" file, you will see multiple .js files in the "output" folder. We can start with "0.js" file. Consider this file as main component file of the application. (App.js)



2. Open this file and check which files are imported in the beginning of the file. It should look like the below:

```
var ReactNative = require('react-native'),
    module397 = (require('./397'));

ReactNative.AppRegistry.registerComponent(require('./404').name, function () {
  return module397.default;
});
```

or in case of multiple imported components:

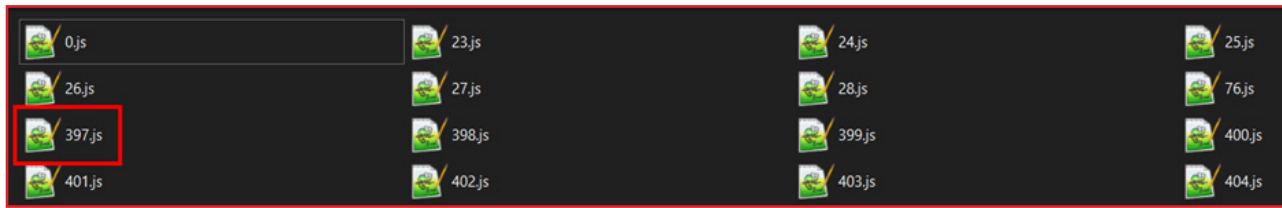
```
var ReactNative = require('react-native'),
    module409 = require('./409'),
    module413 = require('./413'),
    module425 = require('./425'),
    module426 = require('./426');

ReactNative.StatusBar.setBarStyle('light-content', false);
if (module413.device.isAndroid) ReactNative.StatusBar.setBackgroundColor('#000000');
module409.default.enforce();
ReactNative.AppRegistry.registerComponent(module425.name, function () {
  return module426.default;
});
```





3. We can spot these files from the list of multiple .js files.



4. Open this file and you will see the bundled JS code (bundled via webpack).

```
exports.default = function () {
  var c = React.useState(''),
      f = module23.default(c, 2),
      p = f[0],
      v = f[1],
      y = React.useState(''),
      b = module23.default(y, 2),
      x = b[0],
      h = b[1],
      w = function () {
        return module76.default.async(
          function (n) {
            for (;;) {
              switch ((n.prev = n.next)) {
                case 0:
                  n.prev = 0;
                  n.next = 3;
                  return module76.default.awrap(module398.default.setItem('itemList', p));

                case 3:
                  v('');
                  n.next = 6;
                  return module76.default.awrap(0());

                case 6:
                  alert('Data is saved');
                  n.next = 12;
                  break;

                case 9:
                  n.prev = 9;
                  n.t0 = n.catch(0);
                  console.log(n.t0);
              }
            }
          }
        );
      }
}
```

Decompiling Hermes bytecode

As we saw above, the “index.android.bundle” file contains the core logic of the entire application. Thus, the React Native team created their own JavaScript engine called Hermes. This engine is used to run React Native applications. The JS source code is often compiled into the Hermes bytecode, obstructing JS code to some extent.

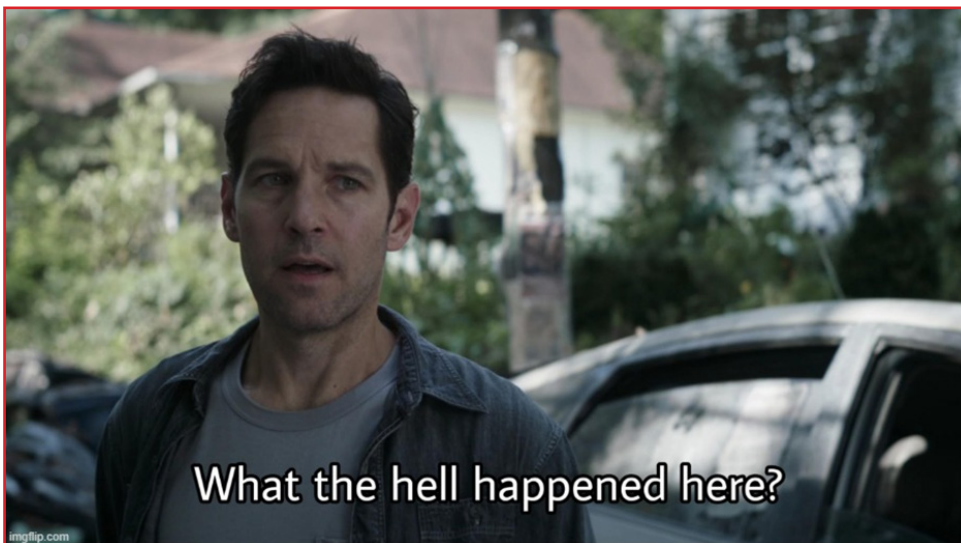




What is Hermes?

[Hermes](#) is an open-source JavaScript engine optimized for React Native. For many apps, enabling Hermes will result in improved start-up time, decreased memory usage, and smaller app size. Refer: <https://reactnative.dev/docs/hermes>

Thus, when you decompile the React Native application that uses Hermes during compilation, the code in the file “index.android.bundle” will be converted into Hermes byte. The contents of the file will look like this:



Fortunately, there is a way to convert this mess into a human-readable format. Shoutout to: [*https://github.com/bongtrop*](https://github.com/bongtrop) for creating *hbctool*. This tool lets us disassemble encrypted bundle file back to Hermes instruction set which is in human-readable bytecode.





Pre-requisite:

hbctool: <https://github.com/bongtrop/hbctool>

Challenge APK: <https://github.com/ErbaZZ/hermes-reversing-lab/blob/main/HermesReversingLab.APK>

Steps:

1. Decompile APK and go to the “/assets” folder.
2. There you will find the “index.android.bundle” file.
3. Install the hbctool with the following command

```
pip install hbctool
```

4. Open the command terminal in the “/assets” folder and type the following command to disassemble Hermes bytecode into human-readable format:

```
hbctool disasm <index.android.bundle> <output_folder_name>
```

5. A folder will be created containing disassembled Hermes bytecode. Now go to the output folder (dis_code)

```
Directory of C:\Users\payatu\Desktop\HermesReversingLab\assets\dis_code
09-08-2022 05:12 PM <DIR> .
09-08-2022 05:12 PM <DIR> ..
09-08-2022 05:12 PM          6,717,740 instruction.hasm
09-08-2022 05:12 PM          3,049,996 metadata.json
09-08-2022 05:12 PM          478,952 string.json
                3 File(s)      10,246,688 bytes
                2 Dir(s)    56,401,403,904 bytes free
```

- **metadata.json:** stores the important information of Hermes bytecode file
- **instruction.hasm:** stores the application instructions or logic in HASM format (edit application logic in this file)
- **string.json:** store the application strings or texts (edit strings in this file)





6. Open the “instructions.hasm” file and analyze instructions sets.

```
instruction.hasm x
182884   GetByIdShort      Reg8:1, Reg8:1, UInt8:1, UInt8:151
182885   ; Oper[3]: String(151) 'state'
182886
182887   GetById           Reg8:2, Reg8:1, UInt8:2, UInt16:2795
182888   ; Oper[3]: String(2795) 'counter'
182889
182890   LoadConstInt     Reg8:1, Imm32:1336
182891   JNotGreaterEqual Addr8:43, Reg8:2, Reg8:1
182892   GetGlobalObject  Reg8:1
182893   TryGetById       Reg8:2, Reg8:1, UInt8:3, UInt16:3716
182894   ; Oper[3]: String(3716) 'alert'
182895
182896   LoadFromEnvironment Reg8:4, Reg8:0, UInt8:0
182897   GetById           Reg8:3, Reg8:4, UInt8:5, UInt16:4072
182898   ; Oper[3]: String(4072) 'decrypt'
182899
182900   LoadConstString  Reg8:1, UInt16:1724
182901   ; Oper[1]: String(1724) 'ZXxZt3UWNXVYadJ2XJZzm25vJFX93ZXnX2fhzZP3ZI5lomX0k20=hJpt'
182902
182903   LoadConstString  Reg8:0, UInt16:219
182904   ; Oper[1]: String(219) 'onPress'
182905
182906   Call3            Reg8:1, Reg8:3, Reg8:4, Reg8:1, Reg8:0
182907   LoadConstUndefined Reg8:0
182908   Call2            Reg8:0, Reg8:2, Reg8:0, Reg8:1
182909   LoadConstUndefined Reg8:0
182910   Ret              Reg8:0
182911 EndFunction
182912
182913 Function<>3846(1 params, 20 registers, 0 symbols):
182914   LoadThisNS      Reg8:3
182915   LoadConstUndefined Reg8:2
182916   LoadConstUndefined Reg8:4
182917   GetEnvironment  Reg8:0, UInt8:1
182918   LoadFromEnvironment Reg8:1, Reg8:0, UInt8:4
182919   GetByIdShort     Reg8:6, Reg8:1, UInt8:1, UInt8:147
182920   ; Oper[3]: String(147) 'default'
```

7. You can find secrets that are stored in String constants by searching for specific keywords such as password, tokens, secret, apikey etc.





8. You can use keyword: `Oper[1]: String(` to grep all of the strings in the bytecode.

```

instruction:hasm
182855  GetGlobalObject      Reg8:1
182856  TryGetById           Reg8:3, Reg8:1, UInt8:3, UInt16:3716
182857  ; Oper[3]: String(3716) 'alert'
182858
182859  LoadConstUndefined  Reg8:2
182860  LoadConstString     Reg8:1, UInt16:787
182861  ; Oper[1]: String(787) 'Increase button has already been broken.'
182862
182863  Call2               Reg8:1, Reg8:3, Reg8:2, Reg8:1
182864  Jmp                 Addr8:109
182865  LoadFromEnvironment Reg8:3, Reg8:0, UInt8:0
182866  GetByIdShort        Reg8:2, Reg8:3, UInt8:4, UInt8:242
182867  ; Oper[3]: String(242) 'setState'
182868

Search results - (3394 hits)
Line 182776: ; Oper[1]: String(1997) ''
Line 182861: ; Oper[1]: String(787) 'Increase button has already been broken.'
Line 182901: ; Oper[1]: String(1724) 'ZxxZt3UwXVradJ2xJzm25vJFX93Zxnx2fHzSP3Zi51omX0k20=hJpt'
Line 182904: ; Oper[1]: String(219) 'onPress'
Line 183006: ; Oper[1]: String(1224) 'Counter: '
Line 183030: ; Oper[1]: String(1361) 'Increase to 1337 for getting a flag'
Line 183069: ; Oper[1]: String(998) '*'
Line 183103: ; Oper[1]: String(2188) 'object'
Line 183158: ; Oper[1]: String(1385) 'InvalidCharacterError'
Line 183166: ; Oper[1]: String(994) 'ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/'
Line 183170: ; Oper[1]: String(1741) '[\\t\\n\\f\\r ]'
Line 183184: ; Oper[1]: String(929) '1.0.0'
Line 183193: ; Oper[1]: String(546) 'function'
Line 183281: ; Oper[1]: String(1997) ''
Line 183298: ; Oper[1]: String(1743) '[^\\0-\\xFF]'
Line 183309: ; Oper[1]: String(1611) 'The string to be encoded contains characters outside of the Latin1 range.'
Line 183404: ; Oper[1]: String(257) '='
Line 183445: ; Oper[1]: String(2524) 'm'
Line 183456: ; Oper[1]: String(1997) ''
Line 183486: ; Oper[1]: String(258) '==2$'
Line 183498: ; Oper[1]: String(1742) '[^+a-zA-Z0-9/]'
Line 183508: ; Oper[1]: String(1377) 'Invalid character: the string to be decoded is not correctly encoded.'

```

Hermes is a custom JavaScript engine created by Facebook. Therefore, the only way to understand this bytecode is to analyse the code patterns. We will see how to read and understand Hermes code in dynamic exploitation in the upcoming chapters.

Grabbing files stored using AsyncStorage

What is AsyncStorage in React Native?

According to the [official react native document](#)-

AsyncStorage is an unencrypted, asynchronous, persistent, key-value storage system that is global to the app. It should be used instead of **LocalStorage**.

AsyncStorage is also asynchronous, i.e., its methods run concurrently with your code. It is also persistent, meaning that the stored data will always be available globally even if you log out or restart the application.





When it becomes a concern?

The data which is stored via AsyncStorage is unencrypted, thus data stored is accessible to anyone with access to the device who can get this data in cleartext. If the application is storing any such credentials of services, user's session token, passwords, or any other sensitive information via AsyncStorage, then it is easy to access this data for an attacker with access to the device.

Where do these files get stored on the device?

On Android, AsyncStorage will use either SQLite or RocksDB based on availability. You can find the databases in the following location:

```
/data/data/<Your-Application-Package-Name>/databases/<your-data-base-name>
```

How to Test:

Steps:

1. Install the vulnerable app on the emulator/physical device and make sure data is getting stored in AsyncStorage.





2. Run the following command to run ADB daemon as root:

```
adb root
```

3. Access the shell of the device via `adb shell`

4. Navigate to the databases folder of the application

```
cd /data/data/<com.your.package>/databases
```

5. You will find 3 files in this folder. You can check the contents of each file with

```
strings RKStorage
strings RKStorage-wal
strings RKStorage-shm
```

```
vbox86p:/data/data/com.xdx/databases # ls -l
total 76
-rw-rw---- 1 u0_a8 u0_a8 20480 2022-08-13 12:37 RKStorage
-rw-rw---- 1 u0_a8 u0_a8 32768 2022-08-13 12:37 RKStorage-shm
-rw-rw---- 1 u0_a8 u0_a8  8272 2022-08-13 12:37 RKStorage-wal
```

6. You will be able to see data stored in AsyncStorage in cleartext.

```
vbox86p:/data/data/com.xdx/databases # strings RKStorage-wal
SQLite format 3
ctableandroid_metadataandroid_metadata
CREATE TABLE android_metadata (locale TEXT)
en_US
ZSQLite format 3
'tablecatalystLocalStoragecatalystLocalStorage
CREATE TABLE catalystLocalStorage (key TEXT PRIMARY KEY, value TEXT NOT NULL)G
indexsqlite_autoindex_catalystLocalStorage_1catalystLocalStorage
ctableandroid_metadataandroid_metadata
CREATE TABLE android_metadata (locale TEXT)
-itemlist s3cr3t_data1234
      itemList
Aitemlist enc_key_AQIPWQR4543B34DDFE
itemlist
```




Sensitive information in XML files

In Android applications, XML files play important roles in defining layouts of components, storing recurring strings, providing ids to the assets, etc. Developers sometimes store sensitive information in these XML files in plaintext. We can go through these XML files to find hardcoded secrets of the application.

Strings.xml

While looking for sensitive information in XML files, "Strings.xml" should be the first place to look for. For convenience, developers might include frequently needed sensitive information such as credentials, static tokens, passwords, secrets, and hidden URLs in the Strings.xml file which later can be referenced within the application. This file can be located in the "/res/values/" folder of the decompiled application.

```
58 <string name="common_google_play_services_enable_button">Enable</string>
59 <string name="common_google_play_services_enable_text">❗️This won't work unless you enable Google Play services.</string>
60 <string name="common_google_play_services_enable_title">Enable Google Play services</string>
61 <string name="common_google_play_services_install_button">Install</string>
62 <string name="common_google_play_services_install_text">❗️This won't run without Google Play services, which are missing from your device.</string>
63 <string name="common_google_play_services_install_title">Get Google Play services</string>
64 <string name="common_google_play_services_notification_channel_name">Google Play services availability</string>
65 <string name="common_google_play_services_notification_ticker">Google Play services error</string>
66 <string name="common_google_play_services_unknown_issue">❗️This is having trouble with Google Play services. Please try again.</string>
67 <string name="common_google_play_services_unsupported_text">❗️This won't run without Google Play services, which are not supported by your device.</string>
68 <string name="common_google_play_services_update_button">Update</string>
69 <string name="common_google_play_services_update_text">❗️This won't run unless you update Google Play services.</string>
70 <string name="common_google_play_services_update_title">Update Google Play services</string>
71 <string name="common_google_play_services_updating_text">❗️This won't run without Google Play services, which are currently updating.</string>
72 <string name="common_google_play_services_wear_update_text">❗️New version of Google Play services needed. It will update itself shortly.</string>
73 <string name="common_open_on_phone">Open on phone</string>
74 <string name="common_signin_button_text">Sign in</string>
75 <string name="common_signin_button_text_long">Sign in with Google</string>
76 <string name="default_web_client_id">852584096874-go2t7tuleihkj158ep7m214kvqepnd8l.apps.googleusercontent.com</string>
77 <string name="facebook_app_id">1540909672882379</string>
78 <string name="fcm_fallback_notification_channel_label">Miscellaneous</string>
79 <string name="firebase_database_url">https://[REDACTED].firebaseio.com</string>
80 <string name="gcm_defaultSenderId">852584096874</string>
81 <string name="gcm_fallback_notification_channel_label">Miscellaneous</string>
82 <string name="google_api_key">AIzaSyDHZ6Z0hCax9bL8BppQ[REDACTED]</string>
83 <string name="google_app_id">1:852584096874:android:89038771e68a73fc</string>
84 <string name="google_crash_reporting_api_key">AIzaSyDHZ6Z0hCax9bL8B[REDACTED]</string>
85 <string name="google_storage_bucket">[REDACTED].appspot.com</string>
86 <string name="search_menu_title">Search</string>
87 <string name="status_bar_notification_info_overflow">999+</string>
88 <string name="ucrop_error_input_data_is_absent">Both input and output Uri must be specified</string>
89 <string name="ucrop_label_edit_photo">Edit Photo</string>
90 <string name="ucrop_label_original">Original</string>
91 <string name="ucrop_menu_crop">Crop</string>
92 <string name="ucrop_mutate_exception_hint">Therefore, override color resource (ucrop_color_toolbar_widget) in your app to make it work on pre-L devices</string>
93 </resources>
```

AndroidManifest.xml

The "AndroidManifest.xml" file contains information on the application package, including components of the application such as activities, broadcast receivers, services, content providers, etc. This file may contain some sensitive hardcoded strings such as keys, secrets, tokens, etc.





```
        <action android:name="com.google.android.c2dm.intent.RECEIVE"/>
    </intent-filter>
</receiver>
<service android:exported="true" android:name="com.google.firebase.iid.FirebaseInstanceIdService">
    <intent-filter android:priority="-500">
        <action android:name="com.google.firebase.INSTANCE_ID_EVENT"/>
    </intent-filter>
</service>
<provider android:authorities="com.donaldaters.firebaseinitprovider" android:exported="false" android:initOrder="100"
android:name="com.google.firebase.provider.FirebaseInitProvider"/>
<activity android:exported="false" android:name="com.google.android.gms.common.api.GoogleApiActivity" android:theme=
"@android:style/Theme.Translucent.NoTitleBar"/>
<meta-data android:name="com.google.android.gms.version" android:value="@integer/google_play_services_version"/>
<meta-data android:name="android.support.VERSION" android:value="26.1.0"/>
<meta-data android:name="com.android.vending.derived_apk_id" android:value="1"/>
<meta-data android:name="com.test.API.KEY" android:value="AIzaEZYip3218rrtqetbmgoetn8ytrg5457n6ui"/>
</application>
</manifest>
```

Uncovering unencrypted HTTP data in the cache

Why is there HTTP data in the cache folder?

Android applications can keep all kinds of stuff in the cache folder of the package. This helps to boost the performance of the app. However, sometimes due to misconfiguration, the application may save sensitive information in the cache folder, more specifically in the “http-cache” folder.

In the React Native applications, “http-cache” contains the GET-based HTTP request+response data. This may expose sensitive data if it is being transferred over an unencrypted or insecure channel.

Limitations of data in the http-cache folder:

1. Only “GET” based HTTP request+response data is stored in the cache folder of the application. Post request data is not cached in the cache folder.
2. Only unencrypted (non-https) requests are cached in plaintext. If the URL is having SSL certificate implemented, then the data will be cached in an encrypted format.





Exploit scenario of http-cache folder:

1. Any GET HTTP request is saved in this folder. If the application is sending sensitive data such as OAuth tokens, credentials, etc. over the GET request type, then we can grab that data in plaintext.
2. Both request and response headers and their values are cached in plaintext. Thus, if the application is sending any sensitive information in request/response headers of the GET request, we can grab that data.

How to test?

1. Open the vulnerable application which has the feature to transfer data in HTTP requests and issue some HTTP requests.

2. Open a command prompt and start the ADB server as root with the following command

```
adb root
```

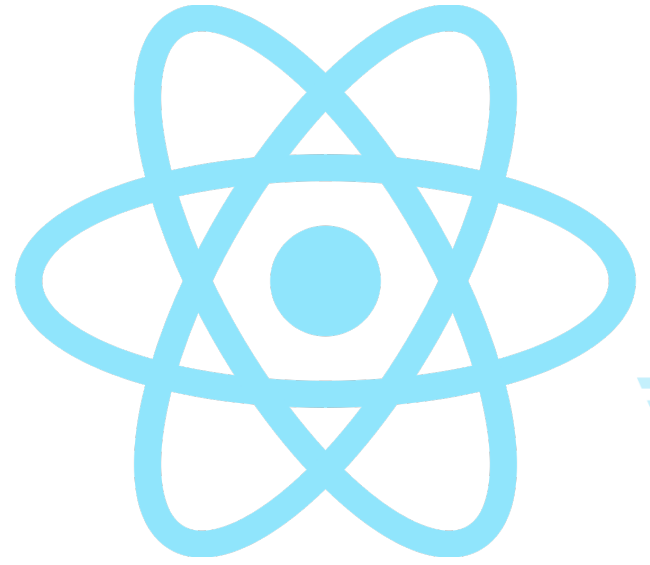
3. Now access the shell of the device with `adb shell` and navigate to the following directory

```
/data/data/<com.package.name>/cache/http-cache
```

4. Open all files in this folder with `cat *` and you can now scrap through cached data

```
C:\Users\payatu\Desktop>adb shell
generic_x86_arm: / # cd /data/data/com.rnapi/cache/http-cache
generic_x86_arm: /data/data/com.rnapi/cache/http-cache # cat *
http://172.25.0.1/101/secret.json
GET
0
HTTP/1.1 200 OK
11
Date: Mon, 29 Aug 2022 18:11:44 GMT
Server: Apache/2.4.53 (Win64) OpenSSL/1.1.1n PHP/7.4.29
Last-Modified: Mon, 29 Aug 2022 18:10:26 GMT
ETag: "3e4-5e7652da02dfb"
Accept-Ranges: bytes
Content-Length: 996
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: application/json
OkHttp-Sent-Millis: 1661796706559
OkHttp-Received-Millis: 1661796706571
{"page":1,"per_page":6,"total":12,"total_pages":2,"data":[{"id":1,"email":"george.bluth@reqres.in","first_name":"George","last_name":"Bluth","avatar":"https://reqres.in/img/faces/1-image.jpg"}, {"id":2,"email":"janet.weaver@reqres.in","first_name":"Janet","last_name":"Weaver","avatar":"https://reqres.in/img/faces/2-image.jpg"}, {"id":3,"email":"emma.wong@reqres.in","first_name":"Emma","last_name":"Wong","avatar":"https://reqres.in/img/faces/3-image.jpg"}, {"id":4,"email":"eve.holt@reqres.in","first_name":"Eve","last_name":"Holt","avatar":"https://reqres.in/img/faces/4-image.jpg"}, {"id":5,"email":"charles.morris@reqres.in","first_name":"Charles","last_name":"Morris","avatar":"https://reqres.in/img/faces/5-image.jpg"}, {"id":6,"email":"tracey.ramos@reqres.in","first_name":"Tracey","last_name":"Ramos","avatar":"https://reqres.in/img/faces/6-image.jpg"}],"support":{"url":"https://reqres.in/#support-heading","text":"To keep ReqRes free, contributions towards server costs are appreciated!"}}libcore.io.DiskLruCache
1
201105
2
DIRTY 1745dc2bba903dbf5aea011fb06c8961
CLEAN 1745dc2bba903dbf5aea011fb06c8961 420 996
generic_x86_arm:/data/data/com.rnapi/cache/http-cache #
```





Chapter 6

Editing and Patching React Native Application





Modifying and patching React Native applications is relatively easier than Java native Android applications. As we already learned, when an APK of React Native project is built, all of the React Native JavaScript code gets compiled into one single file i.e. "index.android.bundle".

We have to find the correct piece of code in the "index.android.bundle" file and then we can modify the code right away. We can find the code block by searching for specific keywords which we can find in the application. For example, we can search for text (like the text on the button, touchableopacity, etc.) shown in the UI of the application to find specific functions associated with that text.

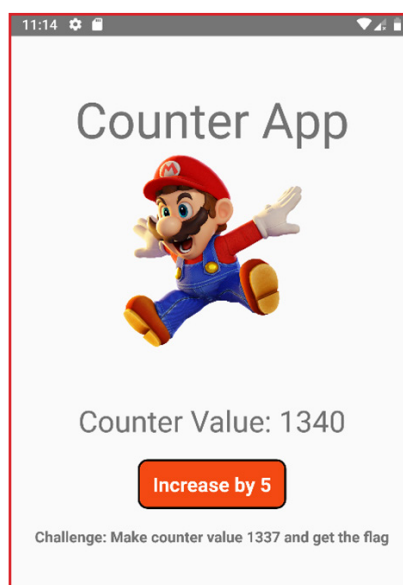
We can utilize the [react-native-decompiler](#) module to analyze the code more efficiently and then later modify it by referencing it into "index.android.bundle".

Steps:

There are several ways to edit and patch the React Native Android application. Below are the two most effective methods demonstrated. You can go with any method as per convenience.

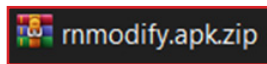
Method 1: Modification using any simple compression tool

1. Open the installed application and you will notice that the counter only increases by 5 digits.

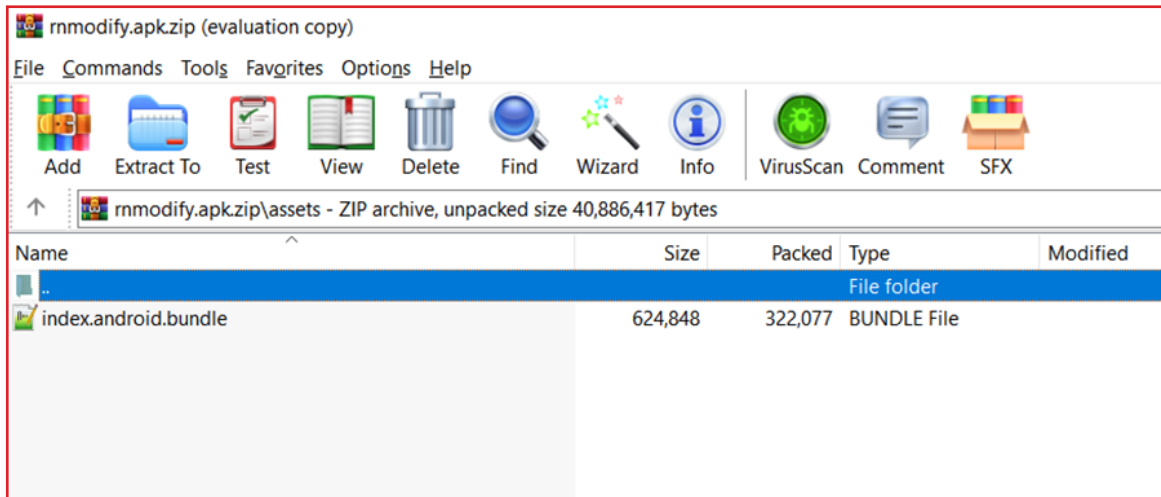




2. Change the extension of the vulnerable APK to “.zip”



3. Open zip in winzip and open /assets/index.android.bundle. *Note that, you have to open the zip file in WinZip. Extracting and again compressing zip might throw an error.*



5. As per our challenge, we have to change the counter value to 1337. Thus, we will change the increment value from 5 to 1 so the counter will increase only by 1 digit per button click.

Tip: You can search in code with custom keywords that you see in the application. Usually, the “hand-written” code can be found at bottom of this file.





```

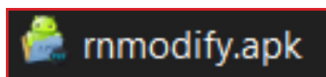
style: {
  textAlign: 'center',
  fontSize: 30,
  marginTop: 50
},
children: ["counter Value: ", u]
}), (0, r(d[4]).jsx)(o.Text, {}), (0, r(d[4]).jsx)(o.TouchableOpacity, {
  style: f.floatingButton,
  onPress: function() {
    s(u + 5), 1336 == u && alert(
  ),
  children: (0, r(d[4]).jsx)(o.Text, {
    style: {
      textAlign: 'center',
      color: '#fff',
      fontSize: 20,
      fontWeight: 'bold'
    },
    children: "increase by 5"
  })
}), (0, r(d[4]).jsx)(o.Text, {
  style: {
    textAlign: 'center',
    fontSize: 15,
    fontWeight: 'bold'
  },
  children: ["\n", "Challenge: Make counter value 1337 and get the flag"]
}))
))
);
var t = r(d[0])(r(d[1])),
n = function(t, n) {
  if (ln && t && t.__esModule) return t;
  if (null === t || "object" != typeof t && "function" != typeof t) return {
    default: t
  };
  var o = l(n);

```

6. Now, we have to delete previous signing certificates. Go to the "META-INF" file and delete the following files:

1. CERT.RSA
2. CERT.SF
3. MANIFEST.MF

7. Exit the "winzip" app and rename the file extension back to ".APK"



8. Now we need to sign the modified APK with a new certificate. To generate custom certificate, run following command and fill out the details:

```
keytool -genkey -v -keystore <keystore_name>.keystore -alias <keystore_alias_name> -keyalg RSA -keysize 2048 -validity 10000
```

```

C:\Users\payatu\Desktop\rnmodify>keytool -genkey -v -keystore rnmodify.keystore -alias rnmodify -keyalg RSA -keysize 2048 -validity 10000
Enter keystore password:
Re-enter new password:
What is your first and last name?
[Unknown]: 1
What is the name of your organizational unit?
[Unknown]: 1
What is the name of your organization?
[Unknown]: 1
What is the name of your City or Locality?
[Unknown]: 1
What is the name of your State or Province?
[Unknown]: 1
What is the two-letter country code for this unit?
[Unknown]: 1
Is CN=1, OU=1, O=1, L=1, ST=1, C=1 correct?
[no]: y
Generating 2,048 bit RSA key pair and self-signed certificate (SHA256withRSA) with a validity of 10,000 days
for: CN=1, OU=1, O=1, L=1, ST=1, C=1
[Storing rnmodify.keystore]

```





9. We will sign our APK with the generated keystore. Run the following command and enter keystore password that is set while creating keystore in step 6.

```
jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore  
<my-keyname>.keystore  
<modify.APK> <alias_name>
```

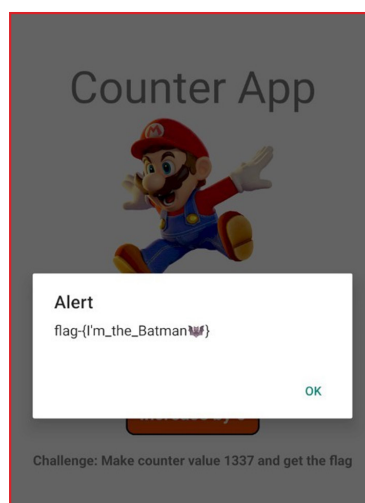
```
C:\Users\payatu\Desktop\rnmodify>jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore rnmodify.keystore rnmodify.apk rnmodify  
Enter Passphrase for keystore:  
adding: META-INF/MANIFEST.MF  
adding: META-INF/RNMODIFY.SF  
adding: META-INF/RNMODIFY.RSA  
signing: META-INF/com/android/build/gradle/app-metadata.properties  
signing: META-INF/androidx.appcompat_appcompat.version  
signing: META-INF/androidx.arch.core_core-runtime.version  
signing: META-INF/androidx.asynclayoutinflater_asynclayoutinflater.version  
signing: META-INF/androidx.autofill_autofill.version  
signing: META-INF/androidx.coordinatorlayout_coordinatorlayout.version  
signing: META-INF/androidx.core_core.version  
signing: META-INF/androidx.cursoradapter_cursoradapter.version  
signing: META-INF/androidx.customview_customview.version  
signing: META-INF/androidx.documentfile_documentfile.version  
signing: META-INF/androidx.drawerlayout_drawerlayout.version  
signing: META-INF/androidx.fragment_fragment.version  
signing: META-INF/androidx.interpolator_interpolator.version  
signing: META-INF/androidx.legacy_legacy-support-core-ui.version  
signing: META-INF/androidx.legacy_legacy-support-core-utils.version  
signing: META-INF/androidx.lifecycle_lifecycle-livedata-core.version  
signing: META-INF/androidx.lifecycle_lifecycle-livedata.version  
signing: META-INF/androidx.lifecycle_lifecycle-runtime.version  
signing: META-INF/androidx.lifecycle_lifecycle-viewmodel.version  
signing: META-INF/androidx.loader_loader.version  
signing: META-INF/androidx.localbroadcastmanager_localbroadcastmanager.version  
signing: META-INF/androidx.print_print.version  
signing: META-INF/androidx.slidingpanelayout_slidingpanelayout.version  
signing: META-INF/androidx.swiperefreshlayout_swiperefreshlayout.version  
signing: META-INF/androidx.vectordrawable_vectordrawable-animated.version  
signing: META-INF/androidx.vectordrawable_vectordrawable.version
```

10. Install the modified APK with adb.

```
adb install modified.APK
```

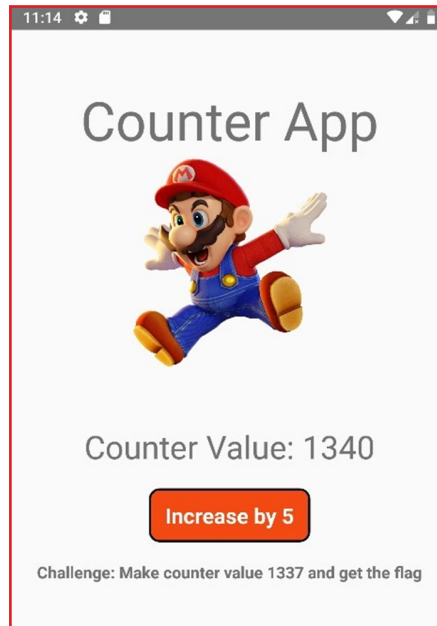
```
C:\Users\payatu\Desktop\rnmodify>adb install rnmodify.apk  
Performing Streamed Install  
Success
```

11. The modified application will be successfully installed.





Method 2: Modification using APKtool:



1. Open the installed application and you will notice that the counter only increases by 5 digits.
2. Run the following command to decompile the application with APKTool:

```
APKtool d VulnerableApp.APK
```

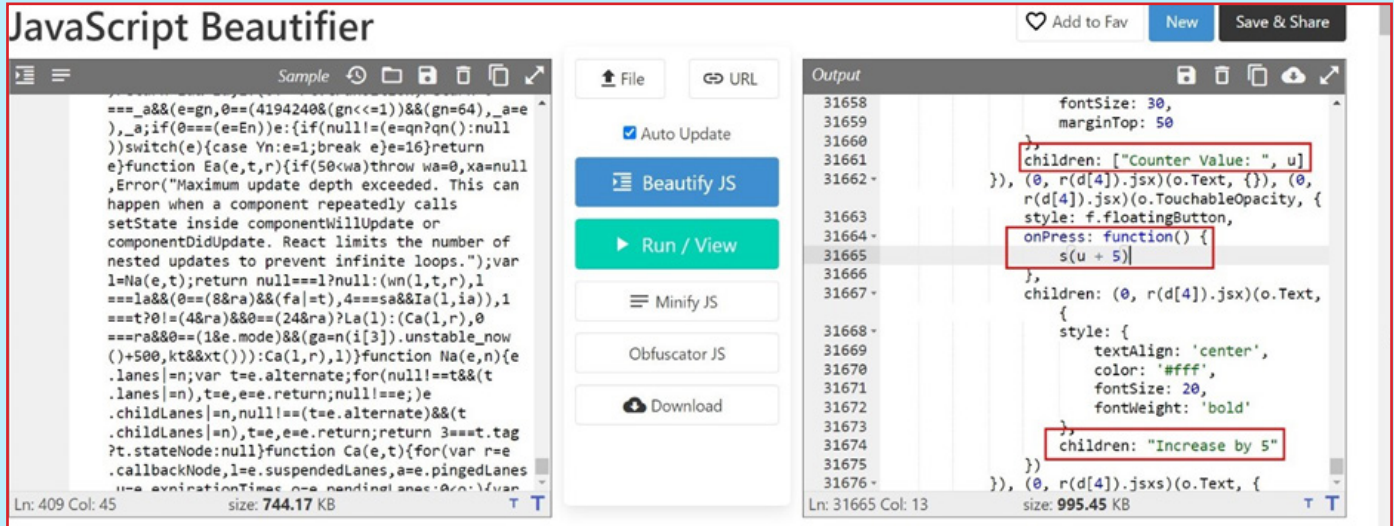
```
C:\Users\payatu\Desktop\Raw>apktool d VulnerableApp.apk
I: Using Apktool 2.6.1 on VulnerableApp.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: C:\Users\payatu\AppData\Local\apktool\framework\1.apk
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Baksmaling classes.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...
```

3. Go to “/VulnerableApp/assets” folder and open the “index.android.bundle” file
4. Search for the keywords such as “Increase by 5” and then search for the “onPress” function. You can copy the entire code and beautify it for convenience.

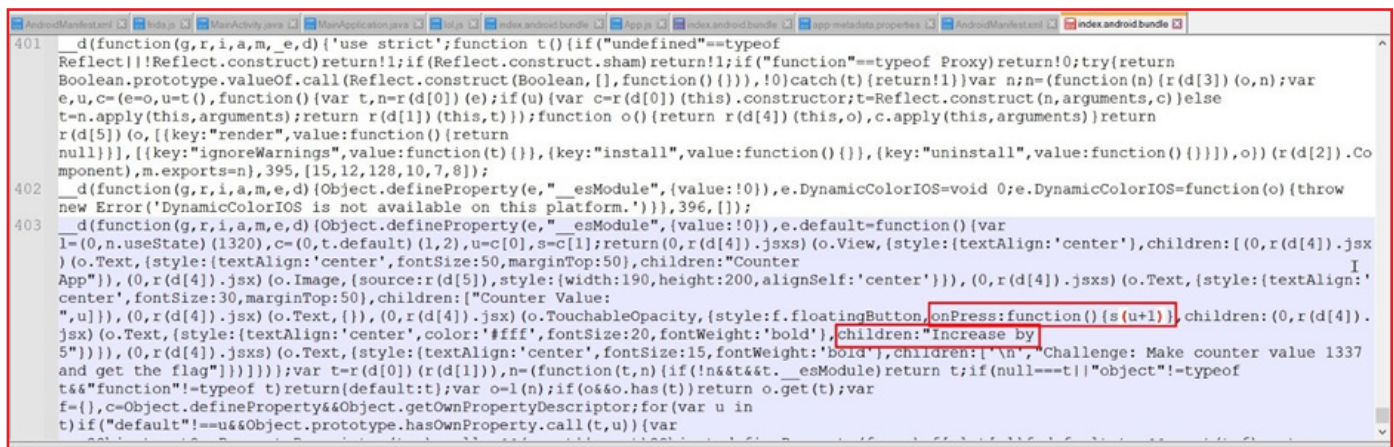




Tip: You can search in code with custom keywords that you see in the application. Usually, the “hand-written” code can be found at bottom of this file.



5. Change the counter value from “5” to “1” in the “index.android.bundle” file.



6. Save this file and run the following APKTool command:

```
APKtool b VulnerableApp
```

7. Modified APK will be generated in the “/VulnerableApp/dist” folder.

8. Go to this folder and create a keystore with the following command:

```
keytool -genkey -v -keystore <keystoreName>.keystore -alias <key-
storeAlias> -keyalg RSA -keysize 2048 -validity 10000
```





9. Sign the APK with “jarsigner”.

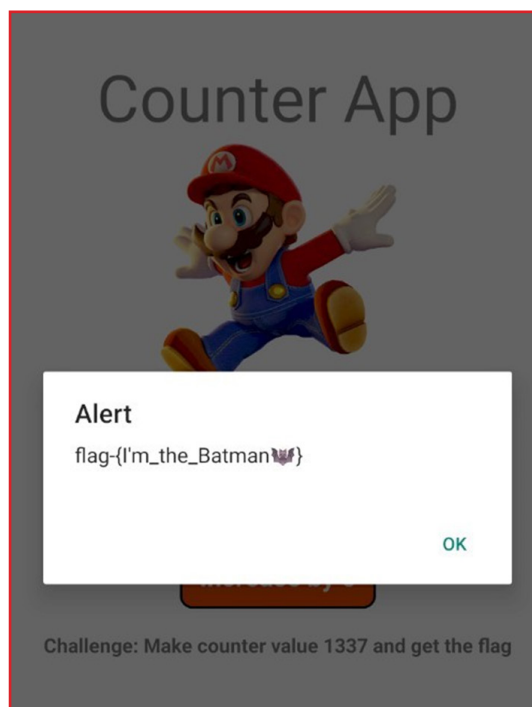
```
jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore <keystoreName>.keystore VulnerableApp.APK <keystoreAlias>
```

```
C:\Users\payatu\Desktop\Raw\VulnerableApp\dist>jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore loki.keystore VulnerableApp.apk loki
Enter Passphrase for keystore:
  adding: META-INF/MANIFEST.MF
  adding: META-INF/LOKI.SF
  adding: META-INF/LOKI.RSA
  signing: AndroidManifest.xml
  signing: classes.dex
  signing: kotlin/annotation/annotation.kotlin_builtins
  signing: kotlin/collections/collections.kotlin_builtins
  signing: kotlin/coroutines/coroutines.kotlin_builtins
  signing: kotlin/internal/internal.kotlin_builtins
  signing: kotlin/kotlin.kotlin_builtins
  signing: kotlin/ranges/ranges.kotlin_builtins
  signing: kotlin/reflect/reflect.kotlin_builtins
  signing: lib/arm64-v8a/libbutter.so
  signing: lib/arm64-v8a/libc++_shared.so
  signing: lib/arm64-v8a/libfabricjni.so
  signing: lib/arm64-v8a/libfb.so
```

10. Install the signed application with:

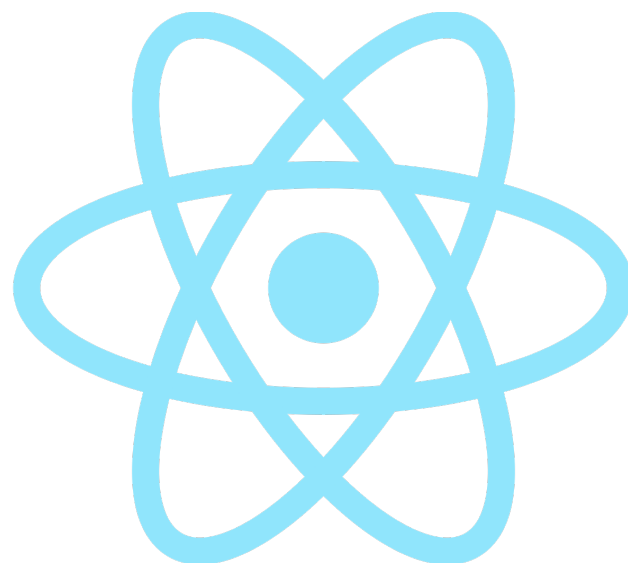
```
adb install VulnerableApp.APK
```

11. Open the application and you will be able to increase the counter by 1 digit now.



Note: You can use either methods demonstrated above to modify and patch the React Native application.





Chapter 7

Modifying Hermes Bytecode



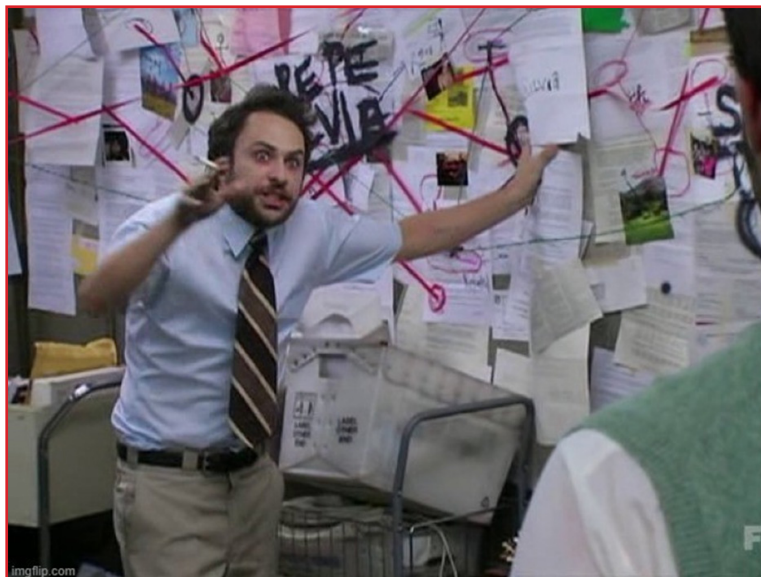


When you decompile the React Native application that uses Hermes during compilation, the code in the “index.android.bundle” file will be converted into Hermes code. The contents of the file will look like this:

```
1 1
2 "h811R, 881r (88rghb"
3
4
5
6
7
8
9
10
11
12
13
```

Understanding Hermes bytecode

As of now, there is no way to convert disassembled Hermes bytecode to readable JavaScript code. We have to understand the bytecode in bits and pieces in order to modify the behavior of a specific function and eventually of the application. The bytecode consists of a bunch of constants and functions which make up the logic of the application.





- **LoadConstInt**: This element stores all integer values created within the application.

```

LoadConstInt      Reg8:1, Imm32:1336
JNotGreaterEqual  Addr8:43, Reg8:2, Reg8:1
GetGlobalObject   Reg8:1
TryGetById        Reg8:2, Reg8:1, UInt8:3, UInt16:3716
; Oper[3]: String(3716) 'alert'

LoadFromEnvironment  Reg8:4, Reg8:0, UInt8:0
GetById            Reg8:3, Reg8:4, UInt8:5, UInt16:4072
; Oper[3]: String(4072) 'decrypt'

LoadConstString    Reg8:1, UInt16:1724
; Oper[1]: String(1724) 'ZXxZt3UWNXVYadJ2XJZzm25vJFX93ZXnX2fhzZP3ZI5lomX0k20=hJpt'

LoadConstString    Reg8:0, UInt16:219
; Oper[1]: String(219) 'onPress'

Call3              Reg8:1, Reg8:3, Reg8:4, Reg8:1, Reg8:0
LoadConstUndefined Reg8:0
Call2              Reg8:0, Reg8:2, Reg8:0, Reg8:1
LoadConstUndefined Reg8:0
Ret               Reg8:0
adFunction

Function<>3846(1 params, 20 registers, 0 symbols):
  LoadThisNS       Reg8:3
  LoadConstUndefined Reg8:2
  LoadConstUndefined Reg8:4
  .
  .
  .
60 hits)
178422: LoadConstInt      Reg8:2, Imm32:4294967295
180646: LoadConstInt      Reg8:0, Imm32:4294967295
181900: LoadConstInt      Reg8:6, Imm32:400
181902: LoadConstInt      Reg8:4, Imm32:4294967295
182890: LoadConstInt      Reg8:1, Imm32:1336

```

● Relational Operators identification:

The instruction code has different keywords for relational operators. Below are some of the important keywords of relational operators and their meanings.

| Keyword | Operator | Meaning |
|---------------|----------|-----------------------|
| JEqual | == | Equal to |
| JNotEqual | != | Not equal to |
| JLess | < | Less than |
| JGreater | > | Greater than |
| JLessEqual | <= | Lesser or equal than |
| JGreaterEqual | >= | Greater or equal than |

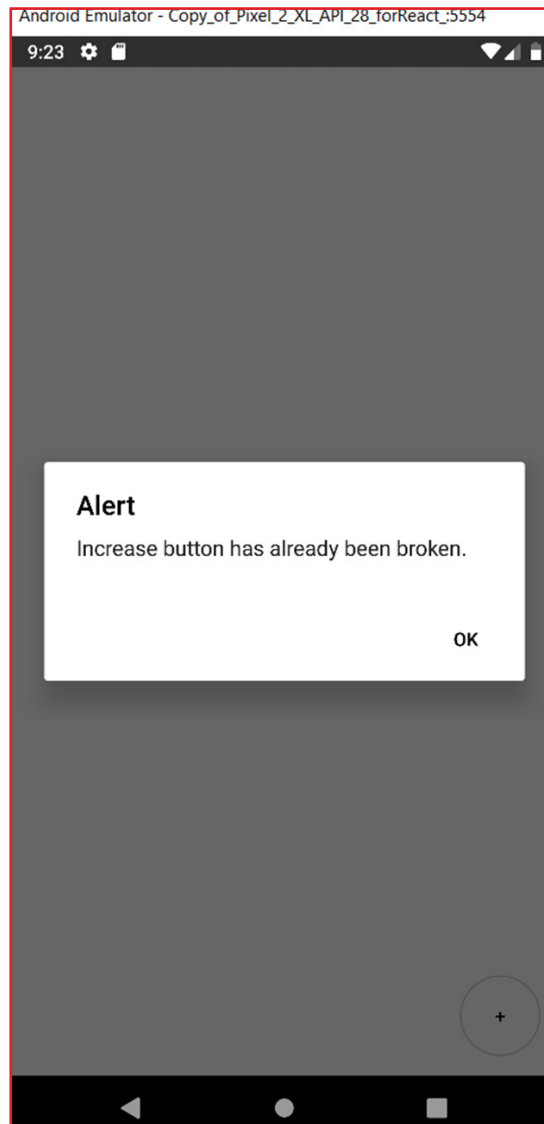




| Keyword | Operator | Meaning |
|------------------|----------|-----------------------------|
| JNotLessEqual | !<= | Not lesser or equal than |
| JNotGreaterEqual | !>= | Not greater or equal than |
| JEqualLong | == | Equal to long data type |
| JNotEqualLong | != | Not equal to long data type |
| JStrictEqual | === | Strict Equal to |
| JStrictNotEqual | !== | Strict not equal to |

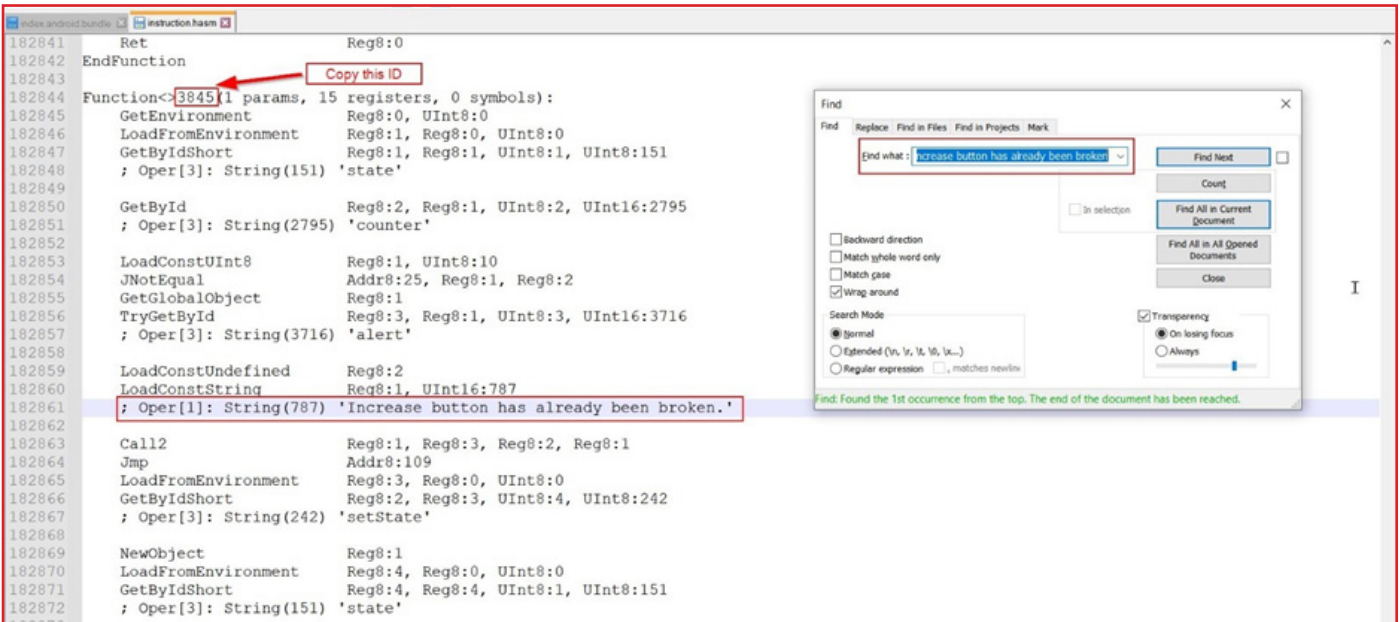
● Find a function name with a string: We can search for any specific function with the help of a string.

o For example, look at the application screenshot below:

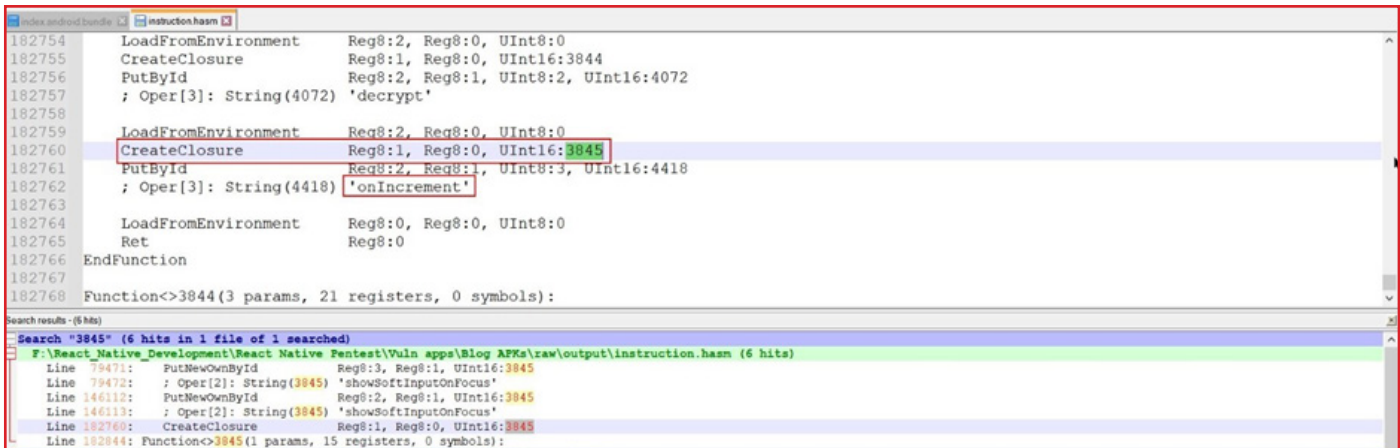




o We can search with any keyword in the string shown in the screenshot below:



o Copy the ID of the function as shown above and search for this ID in the file.



o You will get the name of the function. For reference, here is side by side comparison of React Native JSX code and Hermes bytecode





The Comparison:

React Native JSX code of “onIncrement” function:

Hermes Bytecode of the “onIncrement” function:

```

onIncrement = () => {
  if (this.state.counter == 10) {
    alert("Increase button has already been broken.");
    return;
  }

  this.setState({
    counter: this.state.counter + 1,
  });

  if (this.state.counter >= 1336) {
    var flag = "ZXxZt3UWNXVYadJ2XJZzm25vJFX93ZXnX2fhzP3ZI5lomX0k20=hJpt";
    alert(this.decrypt(flag, "onPress"));
  }
};

```

```

1 Function<>3845(1 params, 15 registers, 0 symbols):_
2   GetEnvironment      Reg8:0, UInt8:0
3   LoadFromEnvironment Reg8:1, Reg8:0, UInt8:0
4   GetByIdShort       Reg8:1, Reg8:1, UInt8:1, UInt8:151
   ; Oper[3]: String(151) 'state'
5
6   GetById            Reg8:2, Reg8:1, UInt8:2, UInt16:2795
   ; Oper[3]: String(2795) 'counter'
7
8   LoadConstUInt8    Reg8:1, UInt8:10
9   JNotEqual         Addr8:25, Reg8:1, Reg8:2
10  GetGlobalObject    Reg8:1
11  TryGetById        Reg8:3, Reg8:1, UInt8:3, UInt16:3716
   ; Oper[3]: String(3716) 'alert'
12
13  LoadConstUndefined Reg8:2
14  LoadConstString    Reg8:1, UInt16:787
   ; Oper[1]: String(787) 'Increase button has already been broken.'
15
16  Call2              Reg8:1, Reg8:3, Reg8:2, Reg8:1
17  Jump              Addr8:109
18  LoadFromEnvironment Reg8:3, Reg8:0, UInt8:0
19  GetByIdShort       Reg8:2, Reg8:3, UInt8:4, UInt8:242
   ; Oper[3]: String(242) 'setState'
20
21  NewObject          Reg8:1
22  LoadFromEnvironment Reg8:4, Reg8:0, UInt8:0
23  GetByIdShort       Reg8:4, Reg8:4, UInt8:1, UInt8:151
   ; Oper[3]: String(151) 'state'
24
25  GetById            Reg8:5, Reg8:4, UInt8:2, UInt16:2795
   ; Oper[3]: String(2795) 'counter'
26
27  LoadConstUInt8    Reg8:4, UInt8:1
28  Add                Reg8:4, Reg8:5, Reg8:4
29  PutNewOwnById     Reg8:1, Reg8:4, UInt16:2795
   ; Oper[2]: String(2795) 'counter'
30
31  Call2              Reg8:1, Reg8:2, Reg8:3, Reg8:1
32  LoadFromEnvironment Reg8:1, Reg8:0, UInt8:0
33  GetByIdShort       Reg8:1, Reg8:1, UInt8:1, UInt8:151
   ; Oper[3]: String(151) 'state'
34
35  GetById            Reg8:2, Reg8:1, UInt8:2, UInt16:2795
   ; Oper[3]: String(2795) 'counter'
36
37  LoadConstInt      Reg8:1, Imm32:1336
38  JNotGreaterEqual   Addr8:43, Reg8:2, Reg8:1
39  GetGlobalObject    Reg8:1
40  TryGetById        Reg8:2, Reg8:1, UInt8:3, UInt16:3716
   ; Oper[3]: String(3716) 'alert'
41
42  LoadFromEnvironment Reg8:4, Reg8:0, UInt8:0
43  GetById            Reg8:3, Reg8:4, UInt8:5, UInt16:4072
   ; Oper[3]: String(4072) 'decrypt'
44
45  LoadConstString   Reg8:1, UInt16:1724
   ; Oper[1]: String(1724)
   'ZXxZt3UWNXVYadJ2XJZzm25vJFX93ZXnX2fhzP3ZI5lomX0k20=hJpt'
46
47  LoadConstString   Reg8:0, UInt16:219
   ; Oper[1]: String(219) 'onPress'
48
49  Call3              Reg8:1, Reg8:3, Reg8:4, Reg8:1, Reg8:0
50  LoadConstUndefined Reg8:0
51  Call2              Reg8:0, Reg8:2, Reg8:0, Reg8:1
52  LoadConstUndefined Reg8:0
53  Ret                Reg8:0
54  EndFunction

```

- o This way we can link any function with its properties.





If you want to learn more about Hermes bytecode, there is a great playground for it: hermesengine.dev

```

H Hermes Docs Playground
-0 -dump-bytecode 27 ms

1  const requestOptions = {
2  method:'get',
3  headers:{
4    'Content-Type':'application/json',
5  },
6  },
7  body: JSON.stringify({ email: "eve.holt@reqres.in", password:
8  })
9  }
10 useEffect(() => {
11   fetch('https://reqres.in/api/users', {
12     sslPinning: {
13       certs:["reqres"]
14     })
15   .then((response) => response.text())
16   .then((json) => setData(JSON.stringify(json)))
17   .catch((error) => console.error(error))
18   .finally(() => setLoading(false));
19 }, []);

63  NCFUNCTION<(1 params, 14 registers, 0 symbols):
64  Offset in debug table: source 0x0013, lexical 0x0000
65  CreateEnvironment r1
66  GetGlobalObject r0
67  TryGetById r4, r0, 1, "fetch"
68  NewObject r0
69  NewArrayWithBuffer r2, 1, 1, 0
70  PutNewOwnByIdShort r0, r2, "certs"
71  NewObject r3
72  PutNewOwnByIdShort r3, r0, "sslPinning"
73  LoadConstUndefined r0
74  LoadConstString r2, "https://reqres.in"...
75  Call3 r4, r4, r0, r2, r3
76  GetByIdShort r3, r4, 2, "then"
77  CreateClosure r2, r1, NCFUNCTION<
78  Call2 r4, r3, r4, r2
79  GetByIdShort r3, r4, 2, "then"
80  CreateClosure r2, r1, NCFUNCTION<
81  Call2 r4, r3, r4, r2
82  GetByIdShort r3, r4, 3, "catch"
83  CreateClosure r2, r1, NCFUNCTION<
84  Call2 r3, r3, r4, r2
85  GetByIdShort r2, r3, 4, "finally"
86  CreateClosure r1, r1, NCFUNCTION<
87  Call2 r1, r1, r3, r1
88  Ret r0
89
90
91  NCFUNCTION<(2 params, 9 registers, 0 symbols):
92  Offset in debug table: source 0x003b, lexical 0x0000
93  LoadParam r1, 1
94  GetByIdShort r0, r1, 1, "text"
95  Call1 r0, r0, r1
96  Ret r0

```

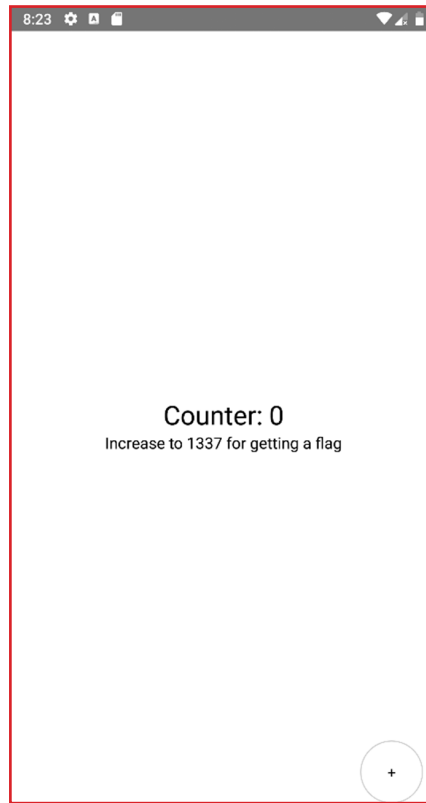
Now, let's disassemble/assemble the obfuscated code into bytecode.

Steps:

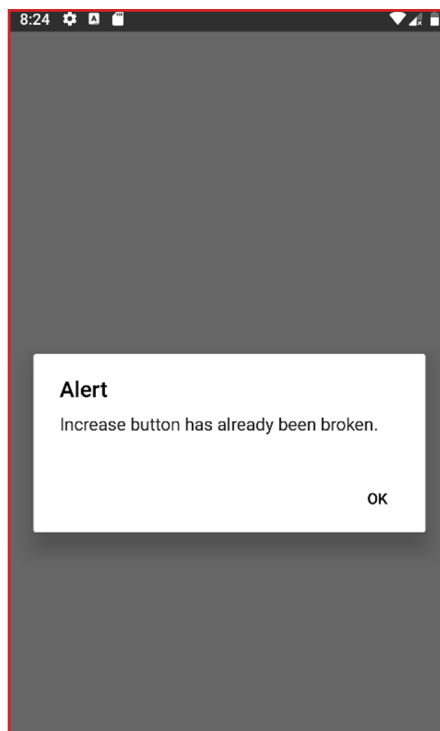
Note: We will solve a challenge created by ["bongtrop"](#). More info here: ["suam.wtf"](#)

1. Install the vulnerable application and you will get the following screen:



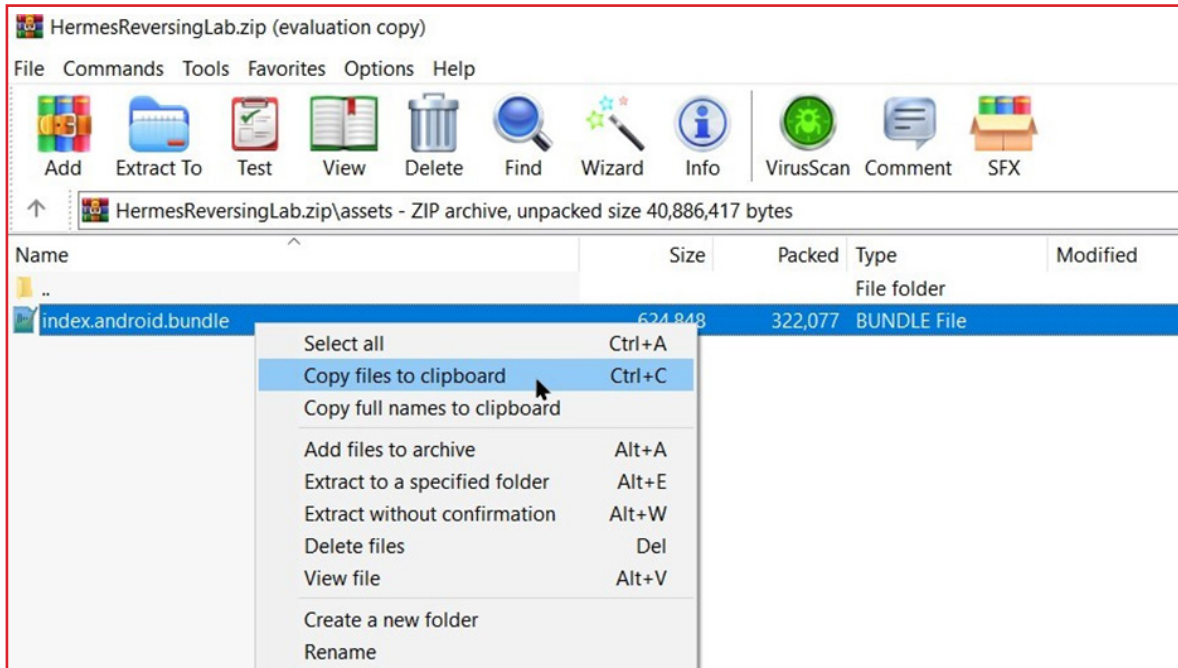


2. We have to increase counter value to 1337 in order to get the flag. But if we do try to increase the counter value with the “+” button at the bottom, we get the following error:

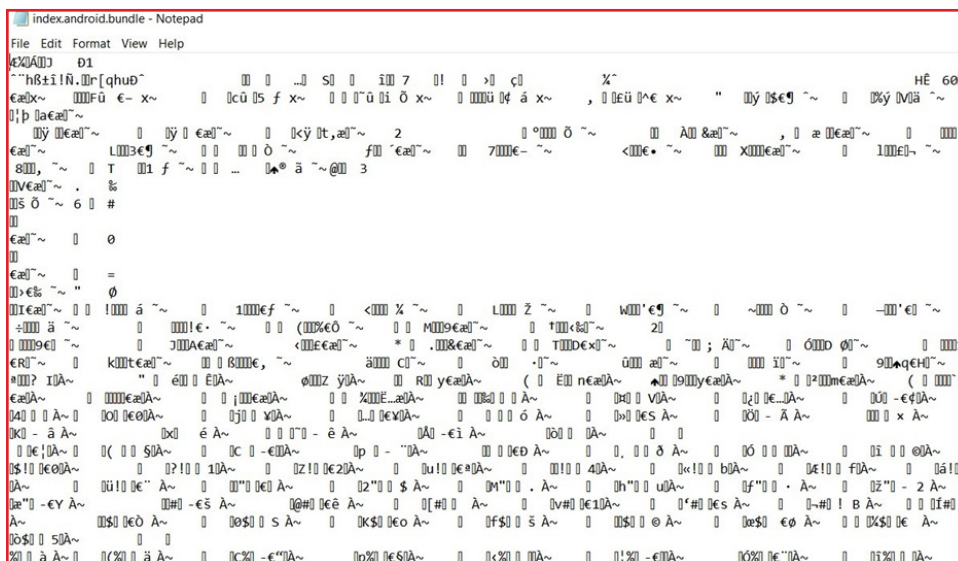




- 3. This means we have to directly set the counter value as 1337.
- 4. Change the extension of .APK file to .zip and open this file with winzip.
- 5. Go to the “/assets/” folder and copy the “index.android.bundle” file in any folder on the system.



6. If you open this file, you will find a gibberish code.





7. Let's convert this mess into a bytecode. Install hbctool with the following command:

```
pip install hbctool
```

8. Open command prompt in folder where "index.android.bundle" file is pasted and run the following command to disassemble the file:

```
//hbctool disasm <path-to-index.android.bundle> <output-folder>  
hbctool disasm index.android.bundle output
```



```
C:\Users\payatu\Desktop\rnmodify>hbctool disasm index.android.bundle output  
[*] Disassemble 'index.android.bundle' to 'output' path  
[*] Hermes Bytecode [ Source Hash: d0310a88a868dfb1ee21d12e9011725b1f716875, HBC Version: 74 ]  
[*] Done
```





9. Go to the “output” folder created and there you will find “instructions.iasm” file. You will find all the React Native application’s JS code in bytecode format.

```
instruction.iasm x
182837 ; Oper[3]: String(190) 'join'
182838
182839 Call2 Reg8:0, Reg8:0, Reg8:3, Reg8:4
182840 Call2 Reg8:0, Reg8:1, Reg8:2, Reg8:0
182841 Ret Reg8:0
182842 EndFunction
182843
182844 Function<>3845(1 params, 15 registers, 0 symbols):
182845 GetEnvironment Reg8:0, UInt8:0
182846 LoadFromEnvironment Reg8:1, Reg8:0, UInt8:0
182847 GetByIdShort Reg8:1, Reg8:1, UInt8:1, UInt8:151
182848 ; Oper[3]: String(151) 'state'
182849
182850 GetById Reg8:2, Reg8:1, UInt8:2, UInt16:2795
182851 ; Oper[3]: String(2795) 'counter'
182852
182853 LoadConstUInt8 Reg8:1, UInt8:10
182854 JNotEqual Addr8:25, Reg8:1, Reg8:2
182855 GetGlobalObject Reg8:1
182856 TryGetById Reg8:3, Reg8:1, UInt8:3, UInt16:3716
182857 ; Oper[3]: String(3716) 'alert'
182858
182859 LoadConstUndefined Reg8:2
182860 LoadConstString Reg8:1, UInt16:787
182861 ; Oper[1]: String(787) 'Increase button has already been broken.'
182862
182863 Call2 Reg8:1, Reg8:3, Reg8:2, Reg8:1
182864 Jmp Addr8:109
182865 LoadFromEnvironment Reg8:3, Reg8:0, UInt8:0
182866 GetByIdShort Reg8:2, Reg8:3, UInt8:4, UInt8:242
182867 ; Oper[3]: String(242) 'setState'
182868
182869 NewObject Reg8:1
182870 LoadFromEnvironment Reg8:4, Reg8:0, UInt8:0
182871 GetByIdShort Reg8:4, Reg8:4, UInt8:1, UInt8:151
182872 ; Oper[3]: String(151) 'state'
182873
182874 GetById Reg8:5, Reg8:4, UInt8:2, UInt16:2795
182875 ; Oper[3]: String(2795) 'counter'
182876
```





10. As we have to increase the counter value to 1337, first find the function that deals with the counter value. We can search with keywords of the error “Increase button has already been broken.”

```
Function<>3845 1 params, 15 registers, 0 symbols):
  GetEnvironment      Reg8:0, UInt8:0
  LoadFromEnvironment Reg8:1, Reg8:0, UInt8:0
  GetByIdShort        Reg8:1, Reg8:1, UInt8:1, UInt8:151
  ; Oper[3]: String(151) 'state'

  GetById             Reg8:2, Reg8:1, UInt8:2, UInt16:2795
  ; Oper[3]: String(2795) 'counter'

  LoadConstUInt8     Reg8:1, UInt8:10
  JNotEqual           Addr8:25, Reg8:1, Reg8:2
  GetGlobalObject     Reg8:1
  TryGetById          Reg8:3, Reg8:1, UInt8:3, UInt16:3716
  ; Oper[3]: String(3716) 'alert'

  LoadConstUndefined Reg8:2
  LoadConstString     Reg8:1, UInt16:787
  ; Oper[1]: String(787) 'Increase button has already been broken.'

  Call2              Reg8:1, Reg8:3, Reg8:2, Reg8:1
  Jump               Addr8:109
  LoadFromEnvironment Reg8:3, Reg8:0, UInt8:0
  GetByIdShort        Reg8:2, Reg8:3, UInt8:4, UInt8:242
  ; Oper[3]: String(242) 'setState'

  NewObject           Reg8:1
  LoadFromEnvironment Reg8:4, Reg8:0, UInt8:0
  GetByIdShort        Reg8:4, Reg8:4, UInt8:1, UInt8:151
  ; Oper[3]: String(151) 'state'

  GetById             Reg8:5, Reg8:4, UInt8:2, UInt16:2795
  ; Oper[3]: String(2795) 'counter'

  LoadConstUInt8     Reg8:4, UInt8:1
  Add                 Reg8:4, Reg8:5, Reg8:4
  PutNewOwnById       Reg8:1, Reg8:4, UInt16:2795
  ; Oper[2]: String(2795) 'counter'

  Call2              Reg8:1, Reg8:2, Reg8:3, Reg8:1
  LoadFromEnvironment Reg8:1, Reg8:0, UInt8:0
  GetByIdShort        Reg8:1, Reg8:1, UInt8:1, UInt8:151
  ; Oper[3]: String(151) 'state'
```




11. As observed above, the counter breaks when we try to increase the counter value beyond 10. Thus, the application is performing a “Relational operation” to verify if the counter value is greater than 10 or not.

```
Function<>3845 1 params, 15 registers, 0 symbols):
  GetEnvironment      Reg8:0, UInt8:0
  LoadFromEnvironment  Reg8:1, Reg8:0, UInt8:0
  GetByIdShort        Reg8:1, Reg8:1, UInt8:1, UInt8:151
  ; Oper[3]: String(151) 'state'

  GetById            Reg8:2, Reg8:1, UInt8:2, UInt16:2795
  ; Oper[3]: String(2795) 'counter'

  LoadConstUInt8     Reg8:1, UInt8:10
  JNotEqual          Addr8:25, Reg8:1, Reg8:2
  GetGlobalObject    Reg8:1
  TryGetById         Reg8:3, Reg8:1, UInt8:3, UInt16:3716
  ; Oper[3]: String(3716) 'alert'

  LoadConstUndefined Reg8:2
  LoadConstString    Reg8:1, UInt16:787
  ; Oper[1]: String(787) 'Increase button has already been broken.'

  Call2              Reg8:1, Reg8:3, Reg8:2, Reg8:1
  Jump               Addr8:109
  LoadFromEnvironment  Reg8:3, Reg8:0, UInt8:0
  GetByIdShort        Reg8:2, Reg8:3, UInt8:4, UInt8:242
  ; Oper[3]: String(242) 'setState'

  NewObject          Reg8:1
  LoadFromEnvironment  Reg8:4, Reg8:0, UInt8:0
  GetByIdShort        Reg8:4, Reg8:4, UInt8:1, UInt8:151
  ; Oper[3]: String(151) 'state'

  GetById            Reg8:5, Reg8:4, UInt8:2, UInt16:2795
  ; Oper[3]: String(2795) 'counter'

  LoadConstUInt8     Reg8:4, UInt8:1
  Add                 Reg8:4, Reg8:5, Reg8:4
  PutNewOwnById      Reg8:1, Reg8:4, UInt16:2795
  ; Oper[2]: String(2795) 'counter'

  Call2              Reg8:1, Reg8:2, Reg8:3, Reg8:1
  LoadFromEnvironment  Reg8:1, Reg8:0, UInt8:0
  GetByIdShort        Reg8:1, Reg8:1, UInt8:1, UInt8:151
  ; Oper[3]: String(151) 'state'
```

12. Instead of increasing the value of the counter, we can change target value i.e. 1337 to 4. For this, we have to find the relational operator in the same function which is checking if the counter value is greater than 1336 or equal to 1337 or not.





```

GetById      Reg8:5, Reg8:4, UInt8:2, UInt16:2795
; Oper[3]: String(2795) 'counter'

LoadConstUInt8  Reg8:4, UInt8:133
Add            Reg8:4, Reg8:5, Reg8:4
PutNewOwnById  Reg8:1, Reg8:4, UInt16:2795
; Oper[2]: String(2795) 'counter'

Call2        Reg8:1, Reg8:2, Reg8:3, Reg8:1
LoadFromEnvironment  Reg8:1, Reg8:0, UInt8:0
GetByIdShort  Reg8:1, Reg8:1, UInt8:1, UInt8:151
; Oper[3]: String(151) 'state'

GetById      Reg8:2, Reg8:1, UInt8:2, UInt16:2795
; Oper[3]: String(2795) 'counter'

LoadConstInt  Reg8:1, Imm32:1336
JNotGreaterEqual  Addr8:43, Reg8:2, Reg8:1
GetGlobalObject  Reg8:1
TryGetById    Reg8:2, Reg8:1, UInt8:3, UInt16:3716
; Oper[3]: String(3716) 'alert'

LoadFromEnvironment  Reg8:4, Reg8:0, UInt8:0
GetById          Reg8:3, Reg8:4, UInt8:5, UInt16:4072
; Oper[3]: String(4072) 'decrypt'

LoadConstString  Reg8:1, UInt16:1724
; Oper[1]: String(1724) 'ZXzT3UWNXVYadJ2XJZzm25vJFX93ZXnX2fhzZP3ZI5lomX0k20=hJpt'
    
```

13. We have found a relational operation that is saying if the counter value reaches 1336 or above, then decrypt and alert the flag. (Note that the flag is encrypted in this case). Thus, we can change this value from 1336 to 10 or less. We are changing it to 4 here.

```

NewObject      Reg8:1
LoadFromEnvironment  Reg8:4, Reg8:0, UInt8:0
GetByIdShort   Reg8:4, Reg8:4, UInt8:1, UInt8:151
; Oper[3]: String(151) 'state'

GetById        Reg8:5, Reg8:4, UInt8:2, UInt16:2795
; Oper[3]: String(2795) 'counter'

LoadConstUInt8  Reg8:4, UInt8:133
Add            Reg8:4, Reg8:5, Reg8:4
PutNewOwnById  Reg8:1, Reg8:4, UInt16:2795
; Oper[2]: String(2795) 'counter'

Call2        Reg8:1, Reg8:2, Reg8:3, Reg8:1
LoadFromEnvironment  Reg8:1, Reg8:0, UInt8:0
GetByIdShort  Reg8:1, Reg8:1, UInt8:1, UInt8:151
; Oper[3]: String(151) 'state'

GetById        Reg8:2, Reg8:1, UInt8:2, UInt16:2795
; Oper[3]: String(2795) 'counter'

LoadConstInt  Reg8:1, Imm32:4
JNotGreaterEqual  Addr8:43, Reg8:2, Reg8:1
GetGlobalObject  Reg8:1
TryGetById    Reg8:2, Reg8:1, UInt8:3, UInt16:3716
; Oper[3]: String(3716) 'alert'

LoadFromEnvironment  Reg8:4, Reg8:0, UInt8:0
GetById          Reg8:3, Reg8:4, UInt8:5, UInt16:4072
; Oper[3]: String(4072) 'decrypt'

LoadConstString  Reg8:1, UInt16:1724
; Oper[1]: String(1724) 'ZXzT3UWNXVYadJ2XJZzm25vJFX93ZXnX2fhzZP3ZI5lomX0k20=hJpt'
    
```

o This means, if the counter value reaches greater than 4 then we will get the flag.





14. Save this file after making any changes and open the “.zip” file of the APK.

15. Now, we need to assemble the index file back to Hermes bytecode format.

16. Open a command prompt and run the following command:

```
//hbctool asm <folder-with-instructions.hasm-file> index.Android.bundle
```

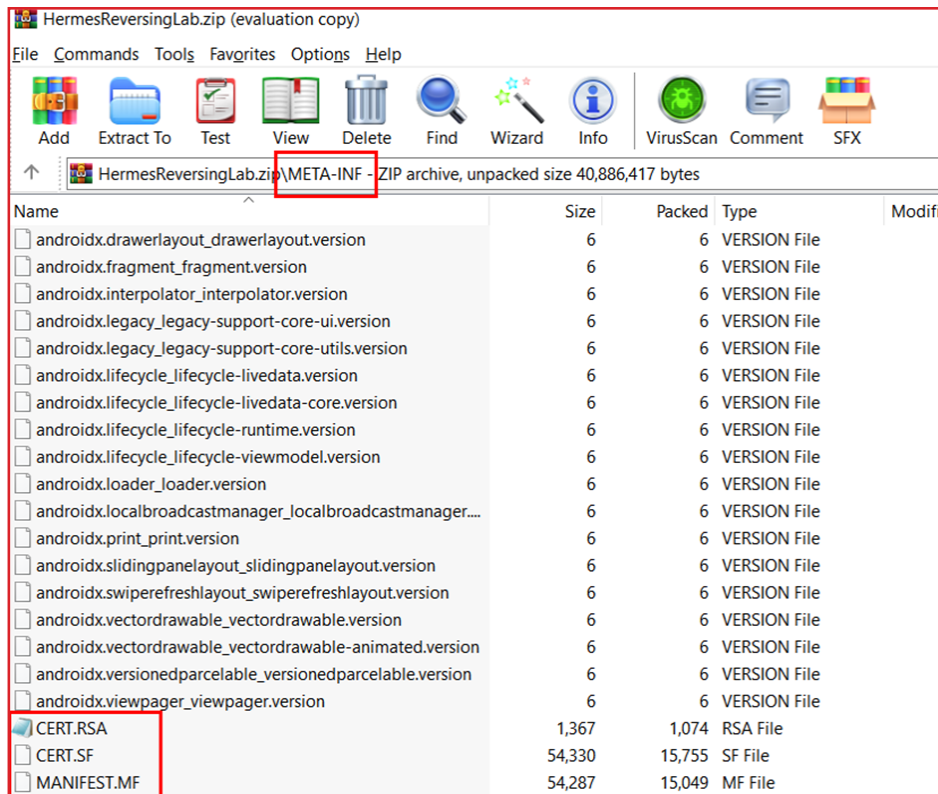
hbctool asm output index.android.bundle

```
C:\Users\payatu\Desktop\rnmodify>hbctool asm output index.android.bundle
[*] Assemble 'output' to 'index.android.bundle' path
[*] Hermes Bytecode [ Source Hash: d0310a88a868dfb1ee21d12e9011725b1f716875, HBC Version: 74 ]
[*] Done
```

17. Go to “/assets” folder. Delete the original “index.android.bundle” file and paste this newly created file there.

18. As usual, we also need to remove the signature files. Go to the “/META-INF” folder and remove the following files:

- o CERT.RSA
- o CERT.SF
- o MANIFEST.MF





19. Exit the “winzip” app and rename the file extension back to “.apk”

20. Now we need to sign the modified APK with the certificate. To generate a custom certificate, run the following command and fill out the details:

```
keytool -genkey -v -keystore <keystore_name>.keystore -alias <key-store_alias_name> -keyalg RSA -keysize 2048 -validity 10000
```

```
C:\Users\payatu\Desktop\rnmodify>keytool -genkey -v -keystore rnmodify.keystore -alias rnmodify -keyalg RSA -keysize 2048 -validity 10000
Enter keystore password:
Re-enter new password:
What is your first and last name?
[Unknown]: 1
What is the name of your organizational unit?
[Unknown]: 1
What is the name of your organization?
[Unknown]: 1
What is the name of your City or Locality?
[Unknown]: 1
What is the name of your State or Province?
[Unknown]: 1
What is the two-letter country code for this unit?
[Unknown]: 1
Is CN=1, OU=1, O=1, L=1, ST=1, C=1 correct?
[no]: y
Generating 2,048 bit RSA key pair and self-signed certificate (SHA256withRSA) with a validity of 10,000 days
for: CN=1, OU=1, O=1, L=1, ST=1, C=1
[Storing rnmodify.keystore]
```

21. We will sign our APK with the generated keystore. Run the following command and enter the keystore password that is set while creating the keystore in step 6.

```
jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore <my-keyname>.keystore <modify.APK> <alias_name>
```

```
C:\Users\payatu\Desktop\rnmodify>jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore rnmodify.keystore rnmodify.apk rnmodify
Enter Passphrase for keystore:
adding: META-INF/MANIFEST.MF
adding: META-INF/RNMODIFY.SF
adding: META-INF/RNMODIFY.RSA
signing: META-INF/com/android/build/gradle/app-metadata.properties
signing: META-INF/androidx.appcompat_appcompat.version
signing: META-INF/androidx.arch.core_core-runtime.version
signing: META-INF/androidx.asynclayoutinflater_asynclayoutinflater.version
signing: META-INF/androidx.autofill_autofill.version
signing: META-INF/androidx.coordinatorlayout_coordinatorlayout.version
signing: META-INF/androidx.core_core.version
signing: META-INF/androidx.cursoradapter_cursoradapter.version
signing: META-INF/androidx.customview_customview.version
signing: META-INF/androidx.documentfile_documentfile.version
signing: META-INF/androidx.drawerlayout_drawerlayout.version
signing: META-INF/androidx.fragment_fragment.version
signing: META-INF/androidx.interpolator_interpolator.version
signing: META-INF/androidx.legacy_legacy-support-core-ui.version
signing: META-INF/androidx.legacy_legacy-support-core-utils.version
signing: META-INF/androidx.lifecycle_lifecycle-livedata-core.version
signing: META-INF/androidx.lifecycle_lifecycle-livedata.version
signing: META-INF/androidx.lifecycle_lifecycle-runtime.version
signing: META-INF/androidx.lifecycle_lifecycle-viewmodel.version
signing: META-INF/androidx.loader_loader.version
signing: META-INF/androidx.localbroadcastmanager_localbroadcastmanager.version
signing: META-INF/androidx.print_print.version
signing: META-INF/androidx.slidingpanelayout_slidingpanelayout.version
signing: META-INF/androidx.swiperefreshlayout_swiperefreshlayout.version
signing: META-INF/androidx.vectordrawable_vectordrawable-animated.version
signing: META-INF/androidx.vectordrawable_vectordrawable.version
```

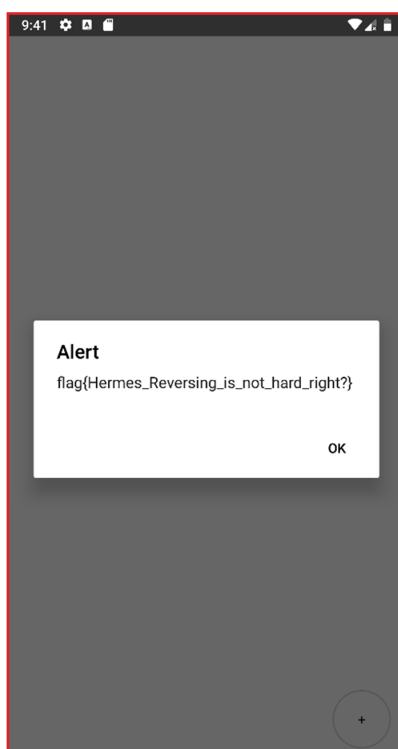




22. Install the modified APK with adb and the modified APK file will successfully get installed.

```
adb install <modified.APK>
```

23. Increase the counter value by tapping “+” button 5 times and you will get the flag.



Understanding and analysing the Hermes bytecode can be a hassle. However, there are certain patterns in the bytecode that help us understand the flow of the functions, methods, and constants.

Root detection bypass

In React Native applications, the JailMonkey npm package is widely used for detecting rooted Android devices. It is also used to detect mocked locations, hooking statuses, and some basic integrity checks of the device.

What is JailMonkey?

JailMonkey is a third-party npm package that provides functionality to check or detect whether the device is rooted or not. It utilizes API “isJailBroken” to check the root status





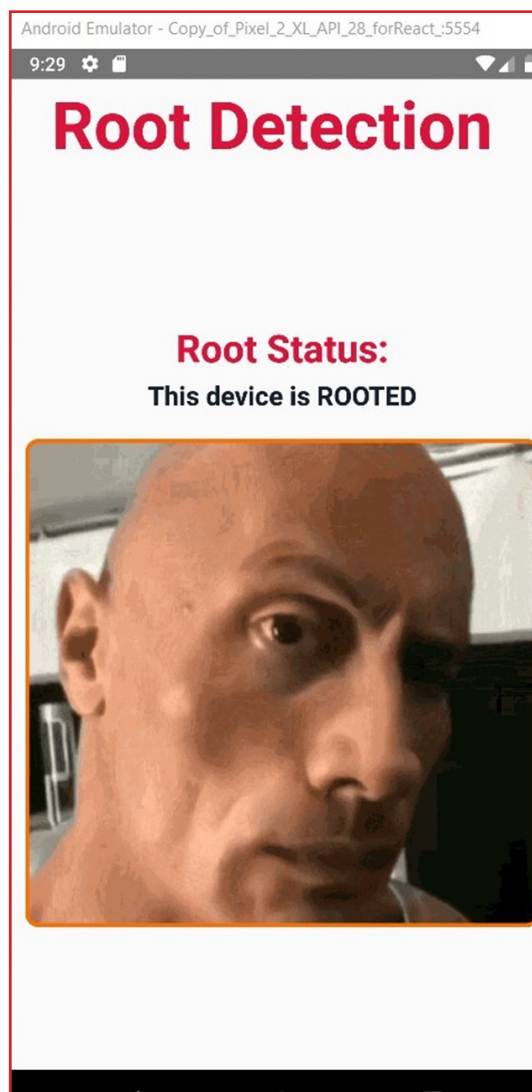
of the device by checking various pieces of information throughout the device such as whether “su” binary exists in the device, whether “busybox” is installed, alternate paths for “su” binaries. etc.

We can bypass this check by modifying the “isJailBroken” function in the “index.android.bundle” file. The steps to do it are shown below.

Note: Always try to modify the function instead of removing it altogether as there might have been some references in rest of the code.

Steps:

1. Open the vulnerable application and you will see it detecting the root status of the device.





2. Now change the extension of the APK file from “.apk” to “.zip” and open this file with any file compression tool such as 7z, winzip.

3. Open “/assets/index.android.bundle” file and search for isJailBroken keyword. You can search the below keyword for reaching the correct code line:

```
isJailBroken: function()
```

```
index.android.bundle
', color: '#DC143C', alignSelf: 'center', children: [{"Root
Detection", "\n\n"}], (0, r(d[6])).jsx (t.Text, {style: {fontSize: 29, fontWeight: 'bold', textAlign: 'center', color: '#DC143C', textAlig
er'}, children: "Root
Status: "}), (0, r(d[6])).jsx (t.Text, {style: {fontSize: 20, fontWeight: 'bold', textAlign: 'center', color: '#17202A', textAlig
ren: c}), (0, r(d[6])).jsx (t.Image, {source: o, style: l.rootImgStyle})); ! (function(t, n) { if (!n && t.__esModule) return
t; if (null === t || "object" !== typeof t && "function" !== typeof t) return (default: t); var l = n(n); if (l && l.has(t)) return l.get(t); var
c = {}, f = Object.defineProperty && Object.getOwnPropertyDescriptor; for (var s in
t) if ("default" !== s && Object.prototype.hasOwnProperty.call(t, s)) { var
u = f ? Object.getOwnPropertyDescriptor(t, s) : null; u && (u.get || u.set) ? Object.defineProperty(c, s, u) : c[s] = t[s]; c.default = t; l && l.set(t, c)} (r(d[0]));
var t = r(d[1]), n = (d[2]) (r(d[3])); function o(t) { if ("function" !== typeof WeakMap) return null; var n = new WeakMap, l = new
WeakMap; return (o = function(t) { return t ? l : n })(t) } var
l = t.StyleSheet.create({ container: { flex: 1, fontSize: 20, fontWeight: 'bold', alignItems: 'center'}, rootImgStyle: { width: 390, height: 370, borderColor: "#FF6F00", borderWid
th: 3, borderRadius: 10, overlayColor: 'white', alignSelf: 'center'}}), 397, [128, 1, 3, 398, 399, 401, 176]);
404 _d(function(g, r, i, a, m, e, d) { Object.defineProperty(e, "__esModule", {value: !0}), e.default = void 0; var
n = r(d[0]), o = n.NativeModules.JailMonkey; null !== o && console.warn("JailMonkey is not available, check your native dependencies have linked
correctly and ensure your app has been rebuilt"); var t = { jailBrokenMessage: function() { return
o.jailBrokenMessage || "" }, isJailBroken: function() { return o.isJailBroken || !1 }, hookDetected: function() { return
o.hookDetected || !1 }, canMockLocation: function() { return o.canMockLocation || !1 }, trustFall: function() { return
o.isJailBroken || o.canMockLocation || !1 }, isOnExternalStorage: function() { return o.isOnExternalStorage || !1 }, isDebuggedMode: function() { return
o.isDebuggedMode() }, isDevelopmentSettingsMode: function() { return "android" !== n.Platform.OS ? Promise.resolve(!1) : o.isDevelopmentSettingsMode() },
AdbEnabled: function() { return o.AdbEnabled || !1 }; e.default = t }, 398, [1]);
405 _d(function(e, t, s, a, r, i, o) { r.exports = t(o[0]).registerAsset({ __packager_asset: !0, httpServerLocation: "/assets/img", width: 640, height: 640, scale
s: [1], hash: "937006f319a0dd8b3f21lac5e5283a90", name: "rockroot", type: "gif"}), 399, [400]);
406 _d(function(g, r, i, a, m, e, d) { 'use strict'; m.exports = r(d[0]), 400, [154]);
407 _d(function(e, s, t, a, d, i, r) { d.exports = s(r[0]).registerAsset({ __packager_asset: !0, httpServerLocation: "/assets/img", width: 640, height: 364, scale
s: [1], hash: "df697c15dd1328f9e9f1815d0bed1d817", name: "rocksafe", type: "gif"}), 401, [400]);
408 _d(function(o, t, e, d, a, n, r) { a.exports = { name: "rootdet", displayName: "rootdet"}, 402, [1];
409 r(52);
410 r(0);
411 // # sourceMappingURL=index.android.bundle.map
```

4. Modify the function as shown below:

```
index.android.bundle
c = {}, f = Object.defineProperty && Object.getOwnPropertyDescriptor; for (var s in
t) if ("default" !== s && Object.prototype.hasOwnProperty.call(t, s)) { var
u = f ? Object.getOwnPropertyDescriptor(t, s) : null; u && (u.get || u.set) ? Object.defineProperty(c, s, u) : c[s] = t[s]; c.default = t; l && l.set(t, c)} (r(d[0]));
var t = r(d[1]), n = (d[2]) (r(d[3])); function o(t) { if ("function" !== typeof WeakMap) return null; var n = new WeakMap, l = new
WeakMap; return (o = function(t) { return t ? l : n })(t) } var
l = t.StyleSheet.create({ container: { flex: 1, fontSize: 20, fontWeight: 'bold', alignItems: 'center'}, rootImgStyle: { width: 390, height: 370, borderColor: "#FF6F00", borderWid
th: 3, borderRadius: 10, overlayColor: 'white', alignSelf: 'center'}}), 397, [128, 1, 3, 398, 399, 401, 176]);
404 _d(function(g, r, i, a, m, e, d) { Object.defineProperty(e, "__esModule", {value: !0}), e.default = void 0; var
n = r(d[0]), o = n.NativeModules.JailMonkey; null !== o && console.warn("JailMonkey is not available, check your native dependencies have linked
correctly and ensure your app has been rebuilt"); var t = { jailBrokenMessage: function() { return
o.jailBrokenMessage || "" }, isJailBroken: function() { return false }, hookDetected: function() { return
o.hookDetected || !1 }, canMockLocation: function() { return o.canMockLocation || !1 }, trustFall: function() { return
o.isJailBroken || o.canMockLocation || !1 }, isOnExternalStorage: function() { return o.isOnExternalStorage || !1 }, isDebuggedMode: function() { return
o.isDebuggedMode() }, isDevelopmentSettingsMode: function() { return "android" !== n.Platform.OS ? Promise.resolve(!1) : o.isDevelopmentSettingsMode() },
AdbEnabled: function() { return o.AdbEnabled || !1 }; e.default = t }, 398, [1]);
405 _d(function(e, t, s, a, r, i, o) { r.exports = t(o[0]).registerAsset({ __packager_asset: !0, httpServerLocation: "/assets/img", width: 640, height: 640, scale
s: [1], hash: "937006f319a0dd8b3f21lac5e5283a90", name: "rockroot", type: "gif"}), 399, [400]);
406 _d(function(g, r, i, a, m, e, d) { 'use strict'; m.exports = r(d[0]), 400, [154]);
407 _d(function(e, s, t, a, d, i, r) { d.exports = s(r[0]).registerAsset({ __packager_asset: !0, httpServerLocation: "/assets/img", width: 640, height: 364, scale
s: [1], hash: "df697c15dd1328f9e9f1815d0bed1d817", name: "rocksafe", type: "gif"}), 401, [400]);
408 _d(function(o, t, e, d, a, n, r) { a.exports = { name: "rootdet", displayName: "rootdet"}, 402, [1];
409 r(52);
410 r(0);
411 // # sourceMappingURL=index.android.bundle.map
```

We are modifying the function such that it returns a “false” boolean value to the “isJail-Broken” function.





5. Go to the “META-INF” folder and delete the following files

1. CERT.RSA
2. CERT.SF
3. MANIFEST.SF

6. Change file extension back to “.APK” and run the following command to generate the keystore file,

```
keytool -genkey -v -keystore <keyStoreName>.keystore -alias <keyStoreAlias> -keyalg RSA -keysize 2048 -validity 10000
```

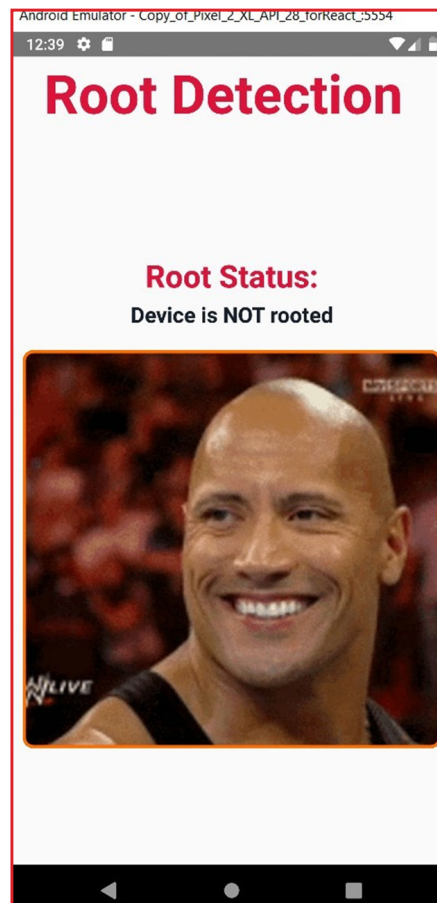
7. Now sign the APK with the newly generated keystore. Run the following command:

```
jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore <my-keyname>.keystore <VulnerableApp.APK> <alias_name>
```

8. Install the application into the device with

```
adb install VulnerableApp.APK
```

9. Open the application and you will see root detection has been bypassed.





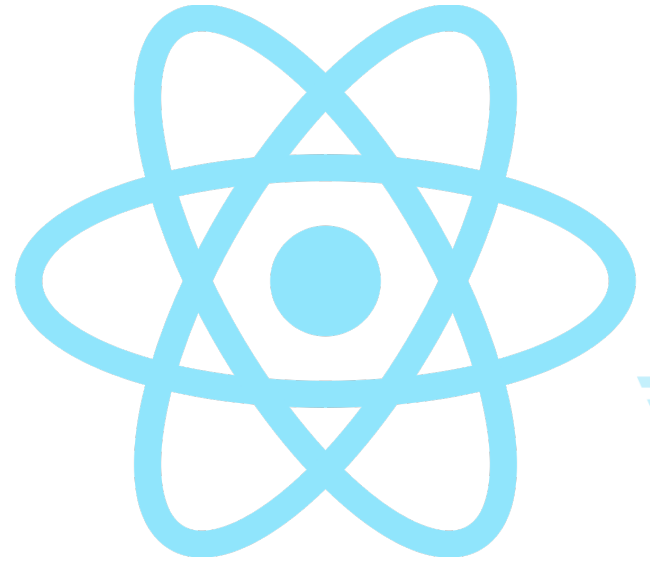
The example shown above is not limited to the shown test case. The implementation of the “isJailBroken” function may vary. It is important to understand the function implementation to modify as per our requirement.

Bonus:

For reference, below is the actual project code snippet vs webpack compiled code.

| Actual Project Code Snippet | Webpack Compiled Code Snippet |
|--|--|
| <pre>if (JailMonkey.isJailBroken){ //code to execute if device is rooted } else { //code to execute if device is not rooted }</pre> | <pre>n.default.isJailBroken ? (//code to execute if device is rooted) : (//code to execute is device is not rooted)); isJailBroken:function(){ return o.isJailBroken !1 }</pre> |





Chapter 8

SSL Certificate Pinning Bypass





What is SSL certificate pinning?

You might already be aware of SSL certificate pinning in the Android application. SSL certificate pinning in short is a process of associating a host with its expected *X509 certificate or public key*.

In certificate pinning, the application is configured to accept only the certificate of a specific domain instead of any trusted CA root certificate in the device (such as PortSwigger CA certificate).



SSL pinning flow diagram

source: <https://www.indusface.com/learning/what-is-ssl-pinning-a-quick-walk-through/>

Bypassing certificate pinning with Frida

Frida by codeshare is the go-to tool to bypass the certificate pinning in runtime. The famous "[Universal Android SSL Pinning bypass script](#)" also works great with React Native applications. You can refer to the article below to perform a pinning bypass like a normal Android application:

["Hail Frida!! The Universal SSL pinning bypass for Android applications"](#)

But.

What if due to any circumstances, we are not able to dynamically hook the application and bypass certificate pinning or we want to permanently bypass the certificate pinning?



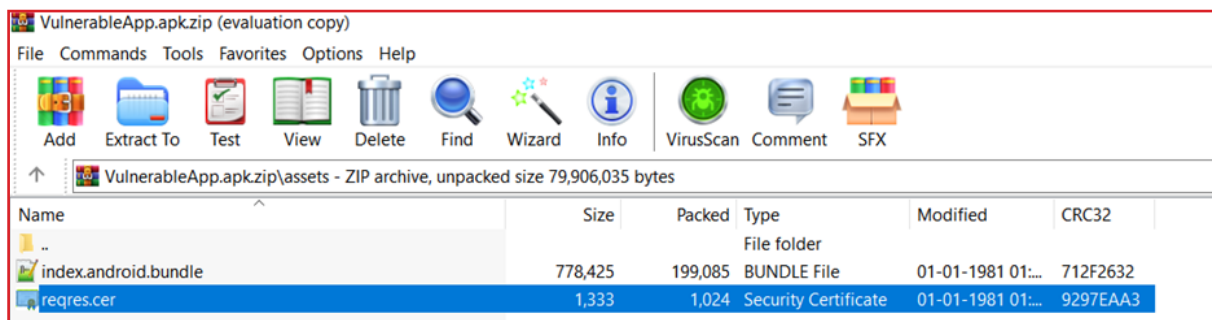


Manually Patching React Native application to bypass certificate pinning

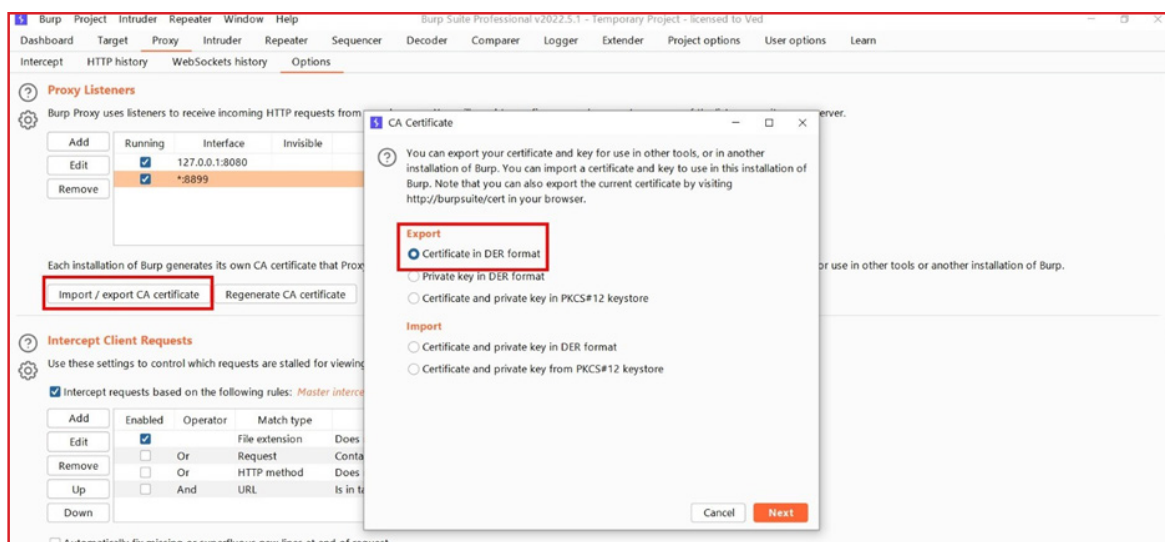
The most used technique to implement certificate pinning in React Native applications is by utilizing the “[react-native-ssl-pinning](#)” node module. The major disadvantage (perhaps an advantage for us 😊) of certificate pinning in React Native applications is that the pinned certificate can be found in the “/assets” folder of the application. Hence an attacker having control over this certificate completely demolishes the certificate pinning implementation.

Steps:

1. Change the extension of the .apk file to .zip and open the zip file in any compression tool such as WinRAR or 7zip.
2. Go to the “/assets” folder and note the name of .cer certificates.

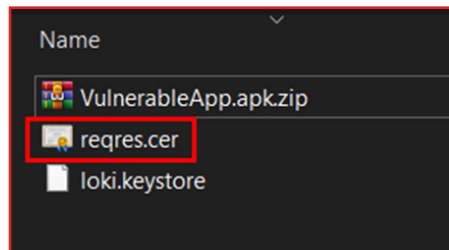


3. Delete all “.cer” certificates from the “/assets” folder.
4. Now configure BurpSuite with an Android device and generate a .der certificate from BurpSuite.

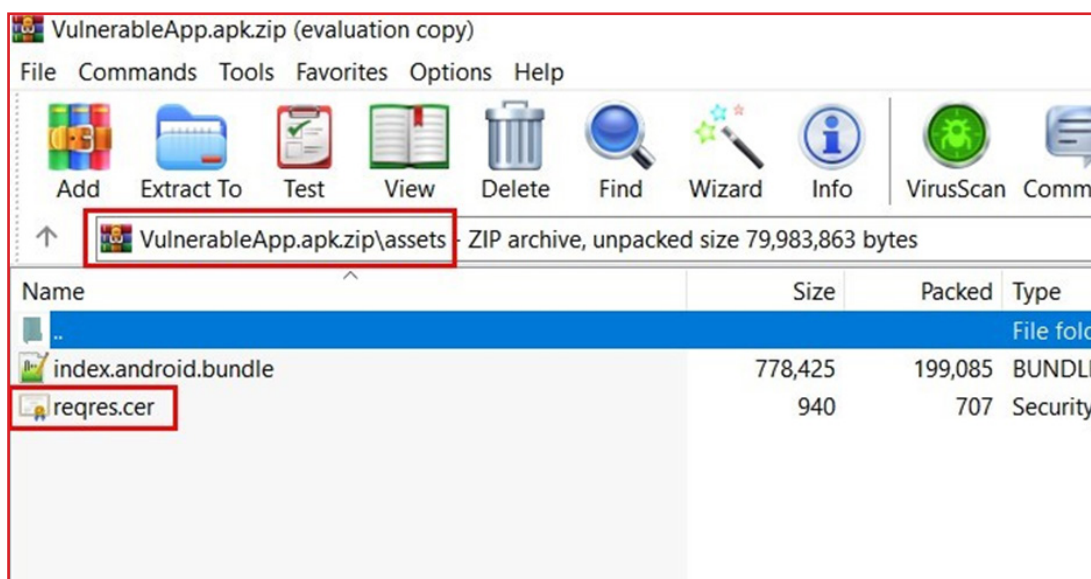




5. Change the certificate extension from “.cer” to “.der” and rename the newly generated “.cer” certificate from BurpSuite with the name copied in step 2.



6. Paste these new certificates in the “/assets” folder.



7. Delete files in META-INF and sign APK as instructed earlier.

```
F:\React_Native_Development\RNApi\android\app\build\outputs\apk\release\rw>jarsigner -verbose -sigalg SHA1withRSA -digestalg SHA1 -keystore loki.keystore VulnerableApp.apk loki
Enter Passphrase for keystore:
updating: META-INF/MANIFEST.MF
adding: META-INF/LOKI.SF
adding: META-INF/LOKI.RSA
signing: META-INF/com/android/build/gradle/app-metadata.properties
signing: META-INF/androidx.appcompat_appcompat.version
signing: META-INF/androidx.arch.core_core-runtime.version
signing: META-INF/androidx.asynclayoutinflater_asynclayoutinflater.version
signing: META-INF/androidx.autofill_autofill.version
signing: META-INF/androidx.coordinatorlayout_coordinatorlayout.version
signing: META-INF/androidx.core_core.version
signing: META-INF/androidx.cursoradapter_cursoradapter.version
signing: META-INF/androidx.customview_customview.version
signing: META-INF/androidx.documentfile_documentfile.version
signing: META-INF/androidx.drawerlayout_drawerlayout.version
signing: META-INF/androidx.fragment_fragment.version
signing: META-INF/androidx.interpolator_interpolator.version
signing: META-INF/androidx.legacy_legacy-support-core-ui.version
signing: META-INF/androidx.legacy_legacy-support-core-ui.version
```





8. Install the application and intercept the encrypted HTTP traffic.

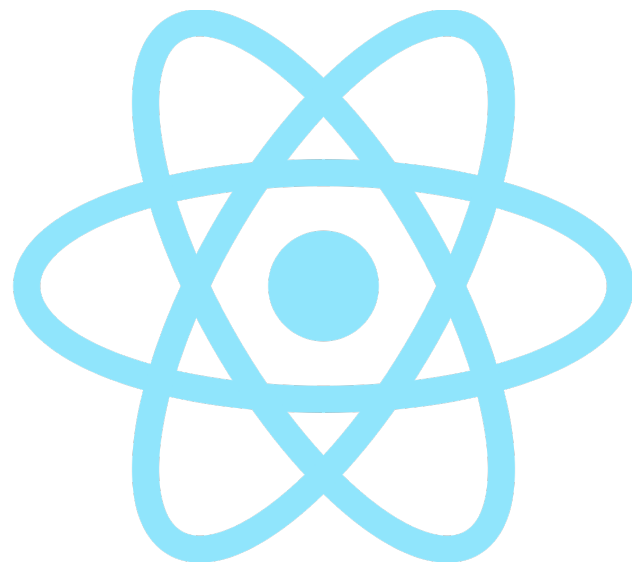
The screenshot shows Burp Suite Professional v2022.5.1 in the background, with a browser window in the foreground displaying the results of an intercepted request. The browser window is titled "Demo" and shows "Fetched Data:" followed by a JSON response. The JSON response contains a list of user profiles, each with an ID, email, first name, last name, and avatar URL. The first profile is highlighted in red in the browser view.

| # | Host | Method | URL | Params | Edited | Status | Length | MIME type | Extension | Title |
|----|---------------------------------|--------|---------------|--------|--------|--------|--------|-----------|-----------|-------|
| 86 | https://reqres.in | GET | /api/users | | | 200 | 1660 | JSON | | |
| 83 | https://connectivitycheck.gs... | GET | /generate_204 | | | 204 | 298 | | | |
| 81 | https://connectivitycheck.gs... | GET | /generate_204 | | | 204 | 298 | | | |

```
Request
1 GET /api/users HTTP/2
2 Host: reqres.in
3 Accept-encoding: gzip, deflate
4 User-Agent: okhttp/4.9.2
5
6

Response
1 HTTP/2 200 OK
2 Date: Tue, 27 Sep 2022 17:08:56 GMT
3 Content-type: application/json; charset=utf-8
4 X-Powered-By: Express
5 Access-Control-Allow-Origin: *
6 Etag: W/"3e4-2RLXvr5WtG9Y6aH95CKYoFNUo8"
7 Via: 1.1 vegur
8 Cache-Control: max-age=14400
9 Cf-Cache-Status: HIT
10 Age: 1455
11 Report-To:
12 {"endpoints":[{"url":"https://a.nel.cloudflare.com/report/v3?v6kQrk6Ea07175zRv1XQj7BqMtc4uQ3mQI2qA9kuKEXNhfVwJ771dI3FhwWCQ9aJnAI1Iiggh0E7X9wJvpX2m3YyeejyMmqHVHFk7KcyFeOucFg3D%3D"}],"group":"cf-nel","max_age":604800}
13 Nel: {"success_fraction":0,"report_to":"cf-nel","max_age":604800}
14 Vary: Accept-Encoding
15 Server: cloudflare
16 Cf-Ray: 7515d81aea9f9a6f-NAG
17 [{"page":1,"per_page":6,"total":12,"total_pages":2,"data":[{"id":1,"email":"george.bluth@reqres.in","first_name":"George","last_name":"Bluth","avatar":"https://reqres.in/img/faces/1-image.jpg"},"id":2,"email":"janet.weaver@reqres.in","first_name":"Janet","last_name":"Weaver","avatar":"https://reqres.in/img/faces/2-image.jpg"},"id":3,"email":"emma.wong@reqres.in","first_name":"Emma","last_name":"Wong","avatar":"https://reqres.in/img/faces/3-image.jpg"},"id":4,"email":"eve.holt@reqres.in","first_name":"Eve","last_name":"Holt","avatar":"https://reqres.in/img/faces/4-image.jpg"},"id":5,"email":"charles.morris@reqres.in","first_name":"Charles","last_name":"Morris","avatar":"https://reqres.in/img/faces/5-image.jpg"},"id":6,"email":"tracey.ramos@reqres.in","first_name":"Tracey","last_name":"Ramos","avatar":"https://reqres.in/img/faces/6-image.jpg"}],"support":{"url":"https://reqres.in/#support"}
```





Chapter 9

Identify Manually Installed npm Packages





React Native provides a set of built-in [Core Components and APIs](#) ready to use in the app. We are not limited to these built-in packages, as React Native has a community of thousands of developers. If the core packages don't have what we are looking for, we may be able to find and install a library from the [community](#) to add the functionality to our app.

React Native packages are typically installed from the [npm registry](#) using a Node.js package manager such as [npm CLI](#) or [Yarn Classic](#).

Application may use some of the packages which are either outdated or contains critical vulnerabilities. Either way, we can identify these packages to find any known vulnerabilities/loopholes in them which can help in our exploitation journey of the React Native applications. Below are the two types of npm packages we can find in the React Native application.

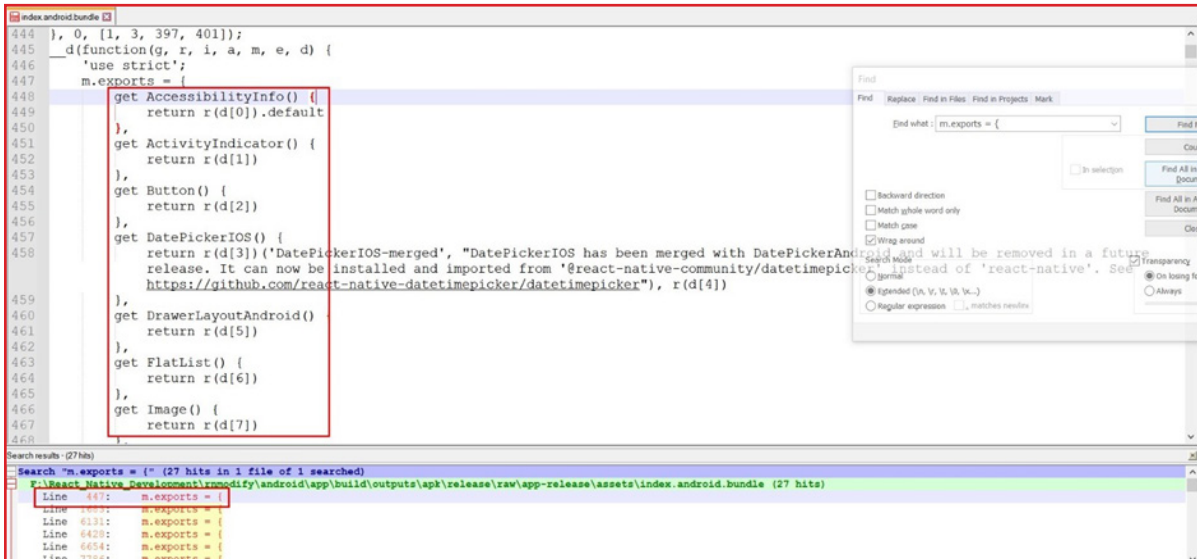
Types of npm packages in React Native application:

1. Pre-installed:

- Pre-installed npm packages are those packages that get installed during the project creation of a new React Native application. These packages are core packages that provide basic features for any React Native application.
- Examples of pre-installed packages are “StyleSheet, AsyncStorage, FlatList, TextInput” etc.
- Usually, these pre-installed packages can be found at the beginning of the “index.android.bundle” file. We can search for the following exact keyword to find the list of pre-installed npm packages: `m.exports = {`



- The very first instance of the above-mentioned keyword contains the list of pre-installed npm packages.



- We can scroll down to see the entire list of pre-installed npm modules in the “m.export” array.



```
m.exports = {
  get AccessibilityInfo() {
    return r(d[0]).default
  },
  get ActivityIndicator() {
    return r(d[1])
  },
  get Button() {
    return r(d[2])
  },
  get DatePickerIOS() {
    return r(d[3])('DatePickerIOS-merged', "DatePickerIOS has been merged with
    DatePickerAndroid and will be removed in a future release. It can now be installed and
    imported from '@react-native-community/datetimepicker' instead of 'react-native'. See
    https://github.com/react-native-datetimepicker/datetimepicker"), r(d[4])
  },
  get DrawerLayoutAndroid() {
    return r(d[5])
  },
  get FlatList() {
    return r(d[6])
  },
  get Image() {
    return r(d[7])
  },
  get ImageBackground() {
    return r(d[8])
  },
  get InputAccessoryView() {
    return r(d[9])
  },
  get KeyboardAvoidingView() {
    return r(d[10]).default
  },
  get MaskedViewIOS() {
    return r(d[3])('maskedviewios-moved', "MaskedViewIOS has been extracted from
    react-native core and will be removed in a future release. It can now be installed and
    imported from '@react-native-masked-view/masked-view' instead of 'react-native'. See
    https://github.com/react-native-masked-view/masked-view"), r(d[11])
  },
  get Modal() {
    return r(d[12])
  },
  get Pressable() {
    return r(d[13]).default
  },
  get ProgressBarAndroid() {
    return r(d[3])('progress-bar-android-moved', "ProgressBarAndroid has been extracted
    from react-native core and will be removed in a future release. It can now be
    installed and imported from '@react-native-community/progress-bar-android' instead of
    'react-native'. See
    https://github.com/react-native-progress-view/progress-bar-android"), r(d[14])
  },
  get ProgressViewIOS() {
    return r(d[3])('progress-view-ios-moved', "ProgressViewIOS has been extracted from
    react-native core and will be removed in a future release. It can now be installed and
    imported from '@react-native-community/progress-view' instead of 'react-native'. See
    https://github.com/react-native-progress-view/progress-view"), r(d[15])
  },
  get RefreshControl() {
    return r(d[16])
  },
  get SafeAreaView() {
    return r(d[17]).default
  },
  get ScrollView() {
    return r(d[18])
  },
  get SectionList() {
    return r(d[19]).default
  },
  get SegmentedControlIOS() {
    return r(d[3])('segmented-control-ios-moved', "SegmentedControlIOS has been extracted
    from react-native core and will be removed in a future release. It can now be
    installed and imported from '@react-native-segmented-control/segmented-control'
    instead of 'react-native'. See
    https://github.com/react-native-segmented-control/segmented-control"), r(d[20])
  },
  get Slider() {
    return r(d[3])('slider-moved', "Slider has been extracted from react-native core and
    will be removed in a future release. It can now be installed and imported from
    '@react-native-community/slider' instead of 'react-native'. See
    https://github.com/callstack/react-native-slider"), r(d[21])
  },
  get StatusBar() {
    return r(d[22])
  }
}
```





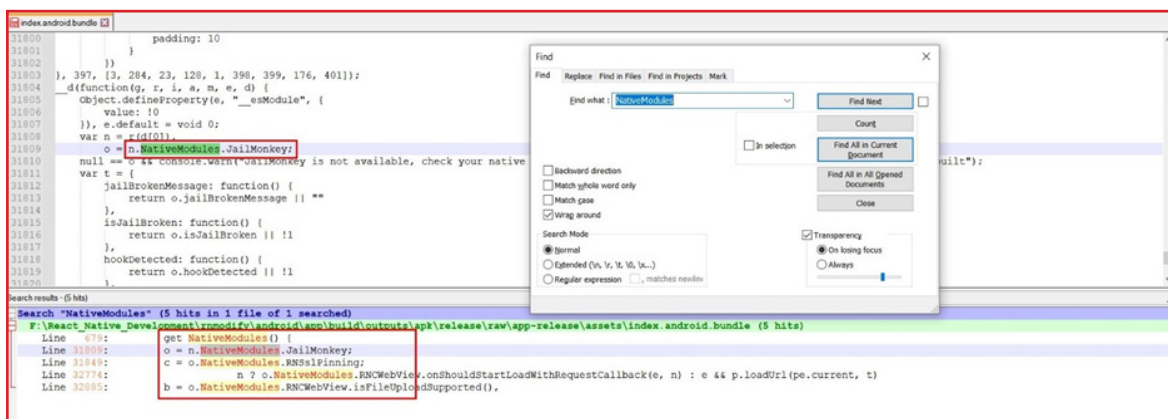
2. Manually installed:

- Contrary to the pre-installed npm packages, manually installed npm packages are installed manually during the development phase of the React Native application. This means these packages do not come pre-installed when React Native application creation is initialized.
- These packages are created and released by awesome community members of React Native and can be found on "[npmjs.com](https://www.npmjs.com)"
- As mentioned above, manually installed packages get installed during the development phase of a product, or the application team may install a single or number of packages as per their convenience.

For example,

- "Stark Technologies" want to implement root detection in its React Native application. Thus, it may use the "[jail-monkey](#)" package.
- "Pym Technologies" want to implement SSL pinning in its React Native application. Thus, it may use the "[react-native-ssl-pinning](#)" package.
- On another side, "S.H.I.E.L.D. Technologies" want to implement both, root detection & SSL pinning. Thus, it may use both the "jail-monkey" and "react-native-ssl-pinning" npm packages.
- We can find these packages in "index.android.bundle" file with the following keyword:

NativeModules



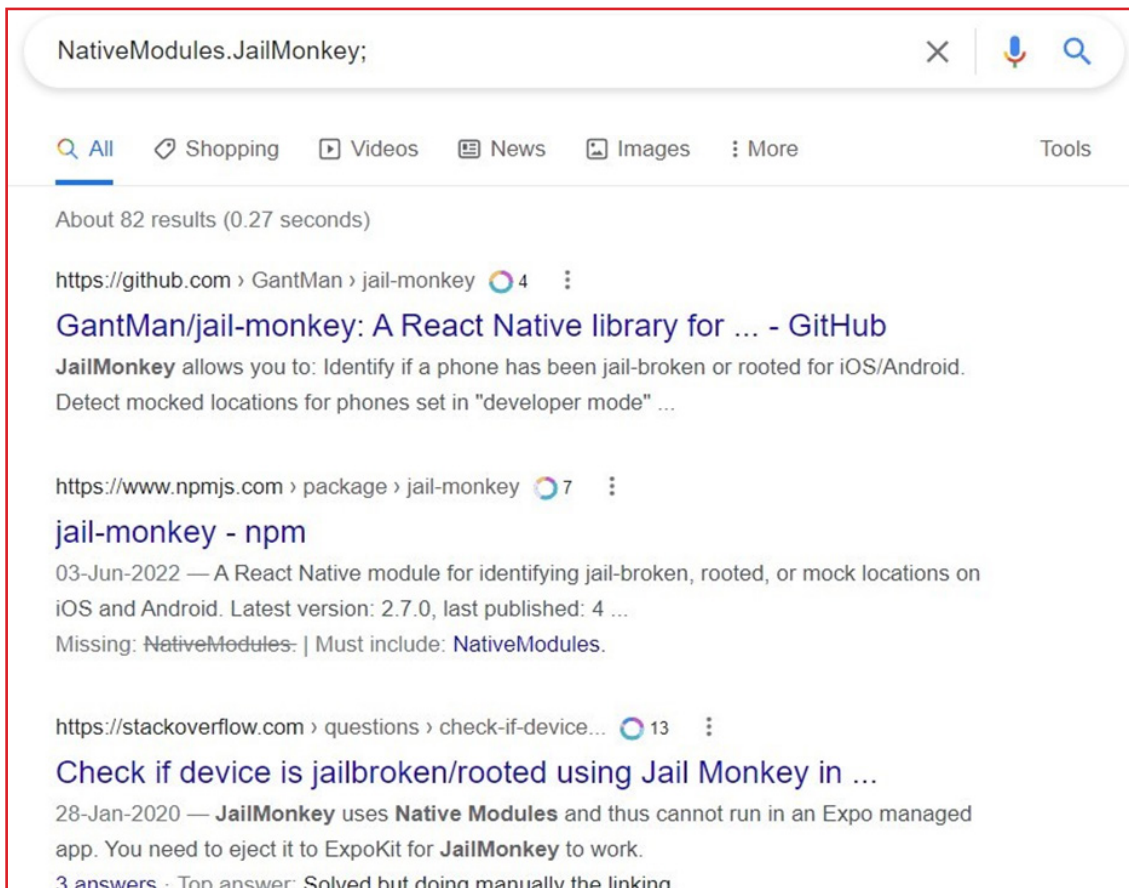


- Here we can see that we have manually installed and used 3 packages:
 - JailMonkey which is "jail-monkey"
 - RNSSLPinning which is "react-native-ssl-pinning"
 - RNCWebView which is "react-native-webview"

Searching for any known CVEs or vulnerabilities on found packages:

1. Once we identify a list of packages, we can search for more information on these packages with following example keyword:

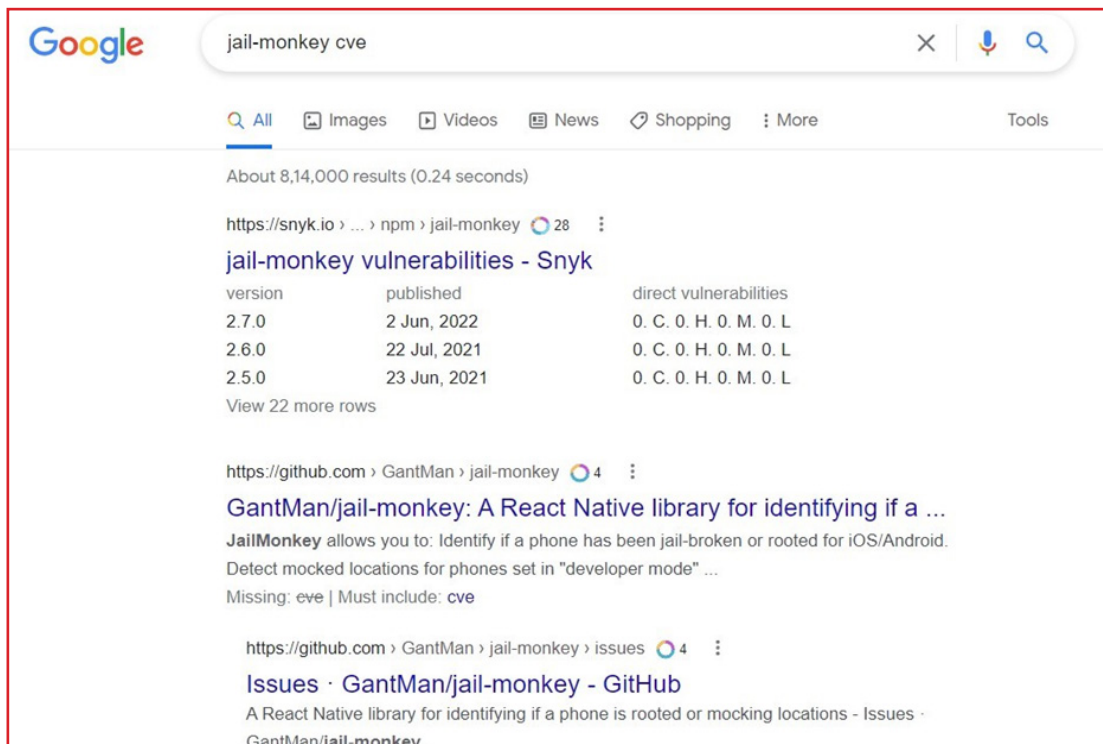
```
//NativeModules.<moduleName>  
NativeModules.JailMonkey
```

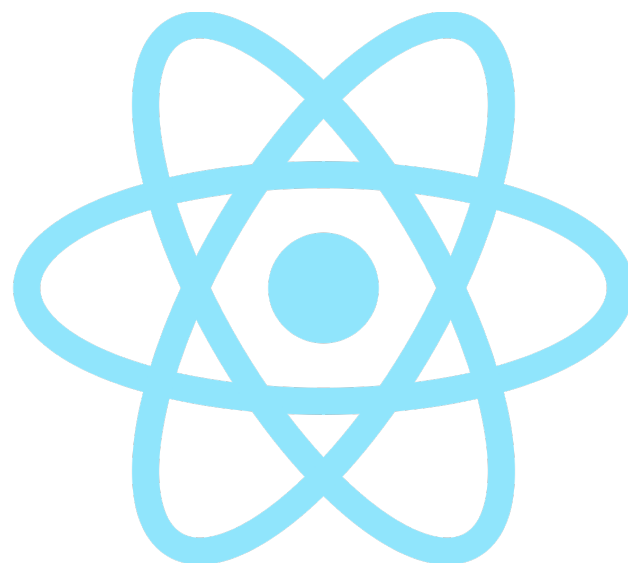




2. Unfortunately, there is no way to identify the version details of the packages used in vulnerable React Native applications. However, we can search for any known CVEs or vulnerabilities, or open issues for npm package we found in the application:

```
//<package name> cve  
jail-monkey cve
```





Chapter 10

React Native npm Package CVEs Walkthrough





React Native applications are built using multiple npm modules. Some of them are officially released and maintained by Facebook, however some of them are created by community members. We will review some of the zero-day vulnerabilities identified in npm packages specifically used to build some components of React Native applications.

1. [CVE-2020-6506](#) Android WebView Universal Cross-site Scripting

- A universal XSS (cross-site scripting) vulnerability has been identified in the Android WebView system component. “[react-native-webview](#)” npm package which is used for webview component in React Native applications is also affected as it utilizes the same component for WebView implementation. This component allows cross-origin iframes to execute arbitrary JavaScript.
- This UXSS vulnerability affects React Native applications which use a “react-native-webview” npm package that allows navigation to arbitrary URLs and when that app runs on systems with an Android WebView version prior to 83.0.4103.106.

Affected npm package: [react-native-webview](#)

Affected version: 10.0.0 or below

Description:

- In the WebView component in React Native applications, `setSupportMultipleWindows` is used to handle new windows with javascript: URLs in the same way as new windows with https:// URLs, which is to navigate the top document to the provided URL. This leads to JavaScript being executed in the top document context.
- To exploit this issue, an iframe can call `windows.open()` with `javascript:<url>`. Successful exploitation of this attack requires a user interaction such as tap or click or keypress because WebView requires interaction to open a new window.

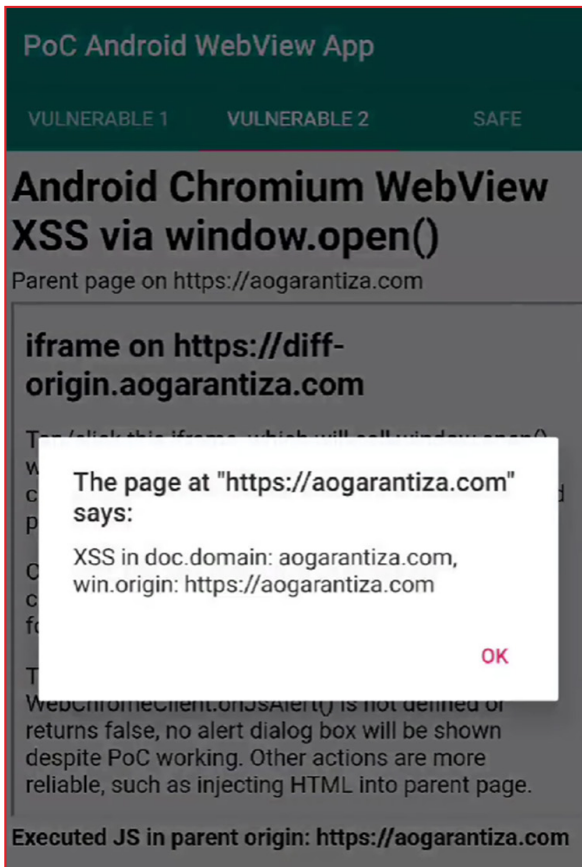




Demo:

- Vulnerable:

- Safe:



Images Reference: <https://alesandroortiz.com/articles/uxss-Android-webview-cve-2020-6506/#sidenote-1>

Mitigation:

- Ensure users update their Android WebView system component via the Google Play Store to 83.0.4103.106 or higher to avoid this UXSS. 'react-native-webview' is working on a mitigation but it could take some time.

Read more:

- <https://alesandroortiz.com/articles/uxss-Android-webview-cve-2020-6506/>
- <https://github.com/advisories/GHSA-36j3-xxf7-4pqq>





2. CVE-2020-7696 Information Exposure Affecting react-native-fast-image

- “react-native-fast-image” npm package is an image processing component which improves the image processing ability of an React Native application. It reduces flickering, cache misses, improves performance loading from cache and performance in general.
- The affected version of this package has been vulnerable to information exposure while rendering the image from uri. When an image with `source={{uri: "...", headers: { host: "[somehost.com](<http://somehost.com/>)", authorization: "..."}}}` is loaded, all other subsequent images will use same headers. Thus, authorization token, cookies or any sort of headers will be leaked to the servers of subsequent images.

Affected npm package: [react-native-fast-image](#)

Affected version: 8.2.2 or below

Demo:

- React Native Code:

```
export default function App(){
  return(
    <View>
      <Text style={{fontWeight:'bold', fontSize:30, textAlign:'center', color:'#145252', textAlignVertical: 'center', margin:10}}>CVE-2020-7696</Text>
      <Text style={{fontWeight:'bold', fontSize:23, textAlign:'center', color:'#145252', textAlignVertical: 'center', margin:10}}> Information Exposure Affecting
      "react-native-fast-image" npm package</Text>
      <FastImage
        style={{ width: 200, height: 200 }}
        source={{
          uri: "https://webhook.site/a394f591-73a0-40b8-8f9b-4136fe1a3db7",
          headers: { Authorization: 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9',
          priority: FastImage.priority.normal,
        }}
        resizeMode={FastImage.resizeMode.contain}
      />
      <FastImage
        style={{ width: 200, height: 200 }}
        source={{
          uri: "https://webhook.site/6a36bf49-bec4-487e-83d1-c6370477a49f",
          priority: FastImage.priority.normal,
        }}
        resizeMode={FastImage.resizeMode.contain}
      />
    </View>
  )
}
```

image source uri #1

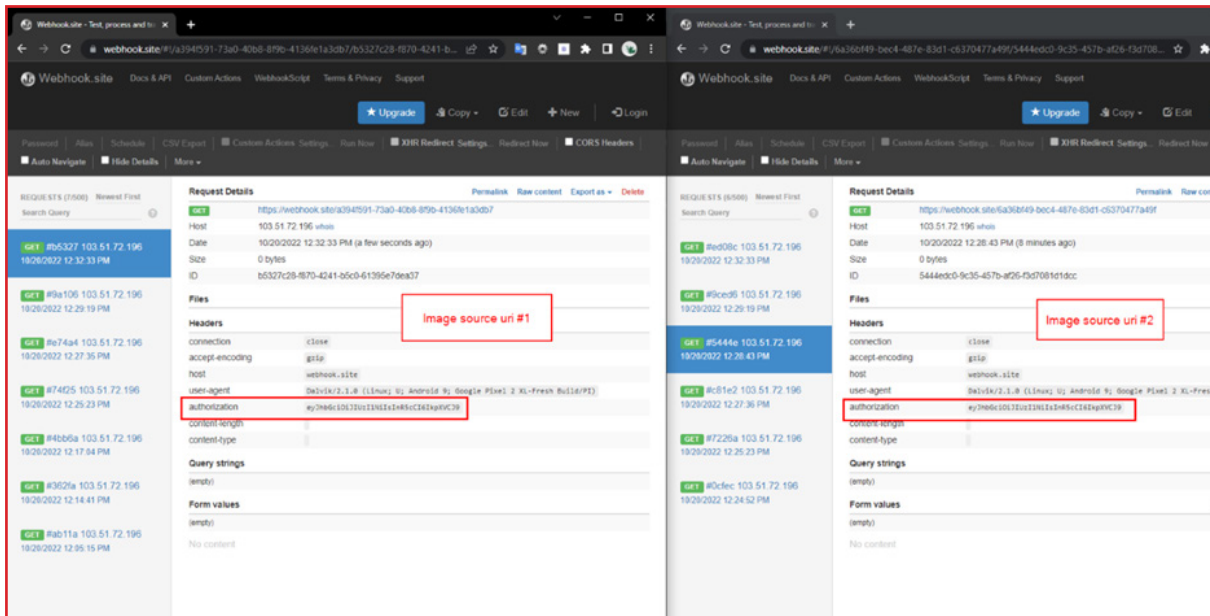
Authorization header for image source uri #1

image source uri #2





- Call-back listener uri:



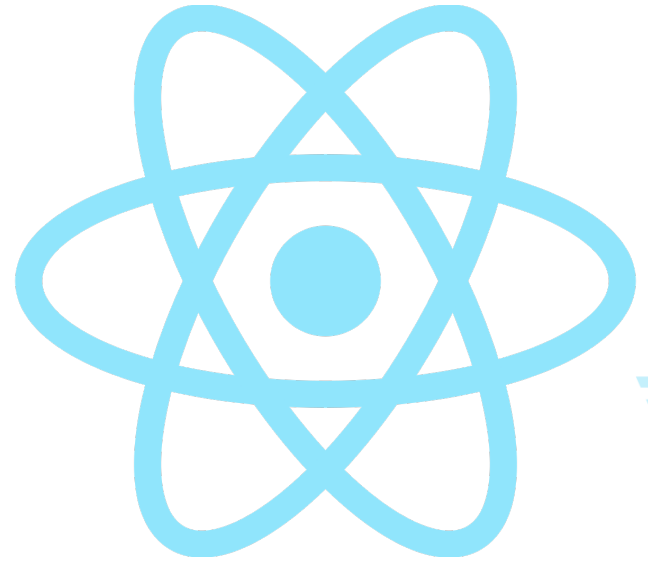
Mitigation:

- Upgrade "react-native-fast-image" to version 8.3.0 or higher.

Reference:

- <https://www.cve.org/CVERecord?id=CVE-2020-7696>
- <https://security.snyk.io/vuln/SNYK-JS-REACTNATIVEFASTIMAGE-572228>





Final Thoughts





In the past few years, we have seen a huge expansion of new technologies in mobile application development. While it is hard to keep up with everything going at a “mach-10” speed, it is important to figure out the differences between the technologies to hit hard at the weakest link within them. React Native framework is evolving with full thrust due to Facebook and the support of a strong community. So, it becomes crucial to identify the pain points of this framework.

We as pentesters are always curious about new technologies and it’s no different in the case of React Native. The technology is still new and needs more research, trial & error, to uncover the nastiest loopholes for the purpose of exploiting them for fun and profit.

Also, we have released two React Native CTF applications which you can find below:

- 1. [VulnerableRN.apk](#) (Without Hermes)
- 2. [RNHermesCTF.apk](#) (With Hermes)

Do check these out!

Finally, thank you for taking the time to read this ebook. I hope you had fun trying out these test cases on our React Native CTF application. Do let us know if you have any feedback or comments. Until next time, Adios!!



About Payatu

Payatu is a Research-powered cybersecurity services and training company specialized in IoT, Embedded Web, Mobile, Cloud, & Infrastructure security assessments with a proven track record of securing software, hardware and infrastructure for customers across 20+ countries.



Mobile Security Testing [↗](#)

Detect complex vulnerabilities & security loopholes. Guard your mobile application and user's data against cyberattacks, by having Payatu test the security of your mobile application.



IoT Security Testing [↗](#)

IoT product security assessment is a complete security audit of embedded systems, network services, applications and firmware. Payatu uses its expertise in this domain to detect complex vulnerabilities & security loopholes to guard your IoT products against cyberattacks.



Cloud Security Assessment [↗](#)

As long as cloud servers live on, the need to protect them will not diminish. Both cloud providers and users have a shared. As long as cloud servers live on, the need to protect them will not diminish.

Both cloud providers and users have a shared responsibility to secure the information stored in their cloud Payatu's expertise in cloud protection helps you with the same. Its layered security review enables you to mitigate this by building scalable and secure applications & identifying potential vulnerabilities in your cloud environment.



Web Security Testing [↗](#)

Internet attackers are everywhere. Sometimes they are evident. Many times, they are undetectable. Their motive is to attack web applications every day, stealing personal information and user data. With Payatu, you can spot complex vulnerabilities that are easy to miss and guard your website and user's data against cyberattacks.



DevSecOps Consulting [↗](#)

DevSecOps is DevOps done the right way. With security compromises and data breaches happening left, right & center, making security an integral part of the development workflow is more important than ever. With Payatu, you get an insight to security measures that can be taken in integration with the CI/CD pipeline to increase the visibility of security threats.



Code Review [↗](#)

Payatu's Secure Code Review includes inspecting, scanning and evaluating source code for defects and weaknesses. It includes the best secure coding practices that apply security consideration and defend the software from attacks.



Red Team Assessment [↗](#)

Red Team Assessment is a goal-directed, multidimensional & malicious threat emulation. Payatu uses offensive tactics, techniques, and procedures to access an organization's crown jewels and test its readiness to detect and withstand a targeted attack.



Product Security [↗](#)

Save time while still delivering a secure end-product with Payatu. Make sure that each component maintains a uniform level of security so that all the components “fit” together in your mega-product.



Critical Infrastructure Assessment [↗](#)

There are various security threats focusing on Critical Infrastructures like Oil and Gas, Chemical Plants, Pharmaceuticals, Electrical Grids, Manufacturing Plants, Transportation systems etc. and can significantly impact your production operations. With Payatu's OT security expertise you can get a thorough ICS Maturity, Risk and Compliance Assessment done to protect your critical infrastructure.



CTI [↗](#)

The area of expertise in the wide arena of cybersecurity that is focused on collecting and analyzing the existing and potential threats is known as Cyber Threat Intelligence or CTI. Clients can benefit from Payatu's CTI by getting – social media monitoring, repository monitoring, darkweb monitoring, mobile app monitoring, domain monitoring, and document sharing platform monitoring done for their brand.

More Services Offered

- [AI/ML Security Audit](#) [↗](#)
- [Trainings](#) [↗](#)

More Products Offered


- [EXPLIoT](#) [↗](#)
- [CloudFuzz](#) [↗](#)



Payatu Security Consulting Pvt. Ltd.

 www.payatu.com

 info@payatu.com

 +91 20 41207726

