

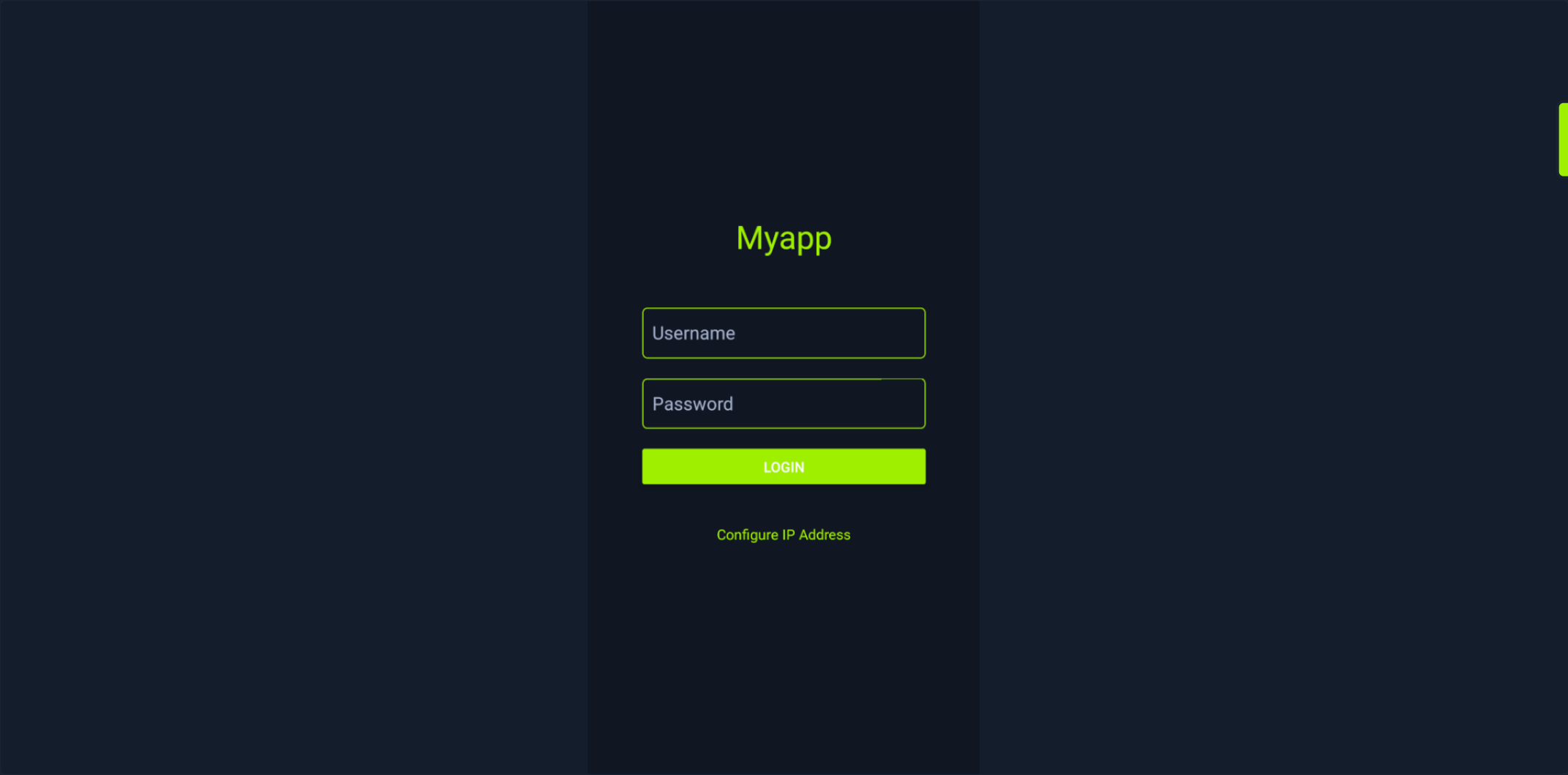
Reversing Hybrid Apps

Android applications can be categorized into three main types: native apps, which are built using platform-specific languages like Java or Kotlin; web apps, which run in a browser and are typically built with HTML, CSS, and JavaScript; and hybrid apps, which combine elements of both and often utilize WebViews to render web content within a native container. Hybrid apps are often developed using HTML, CSS, and JavaScript through frameworks that leverage WebViews. In this example, we will focus on applications built with React Native, a widely-used open-source framework developed by Facebook. React Native enables developers to build cross-platform mobile applications using JavaScript and React, allowing them to maintain a single codebase for both Android and iOS platforms.

Using JADX to analyze the source code of hybrid apps created using React Native is not sufficient on its own. JADX can decompile Java bytecode and produce Java-like pseudocode, but is unable to process the Javascript files present in Hybrid apps. Furthermore, React Native developers can utilize JavaScript obfuscator to further protect and secure the app from reverse engineering and potential tampering. Additionally, when using the [Hermes engine](#), the JavaScript code is transformed into Hermes bytecode—a more compact and optimized format—adding a layer of obfuscation. In React Native applications, the JavaScript code is typically bundled into a file named `index.android.bundle`, which becomes the primary focus of analysis in this section.

Reading Minified JS Code

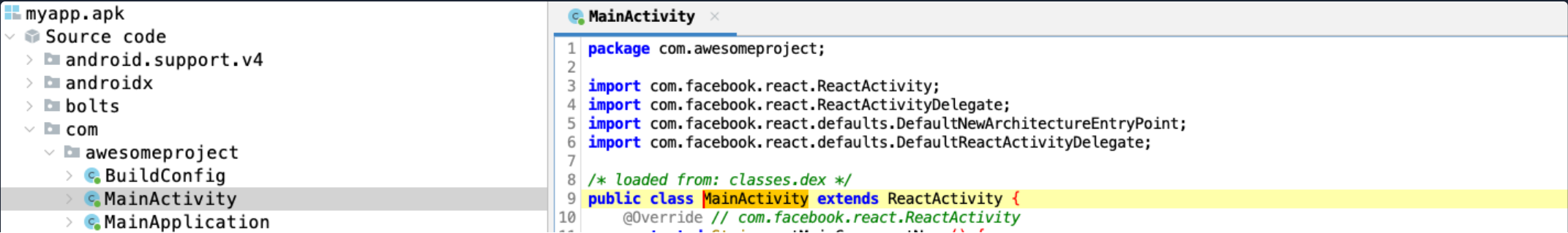
The following app is created using React Native, and once it's started, the user can see a login screen.



Let's decompile APK and read the application's source code.

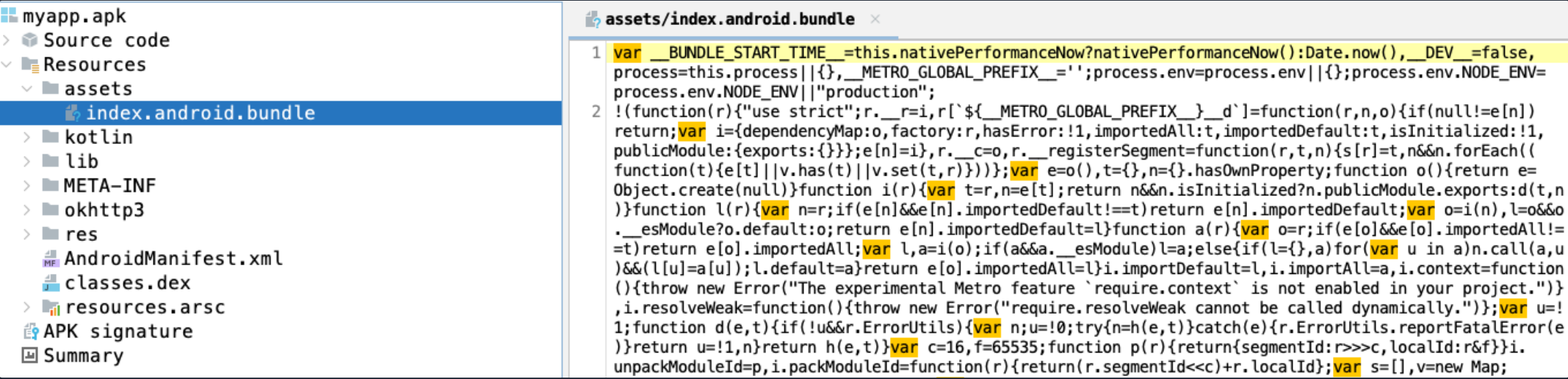
Reversing Hybrid Apps

```
r11k@htb[/htb]$ jadx-gui ~/myapp.apk
```

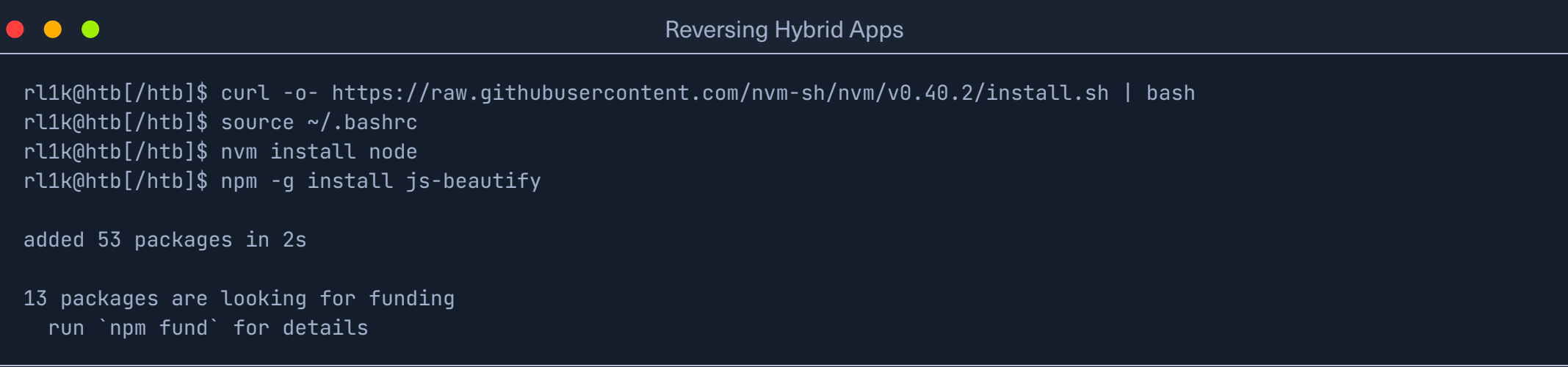




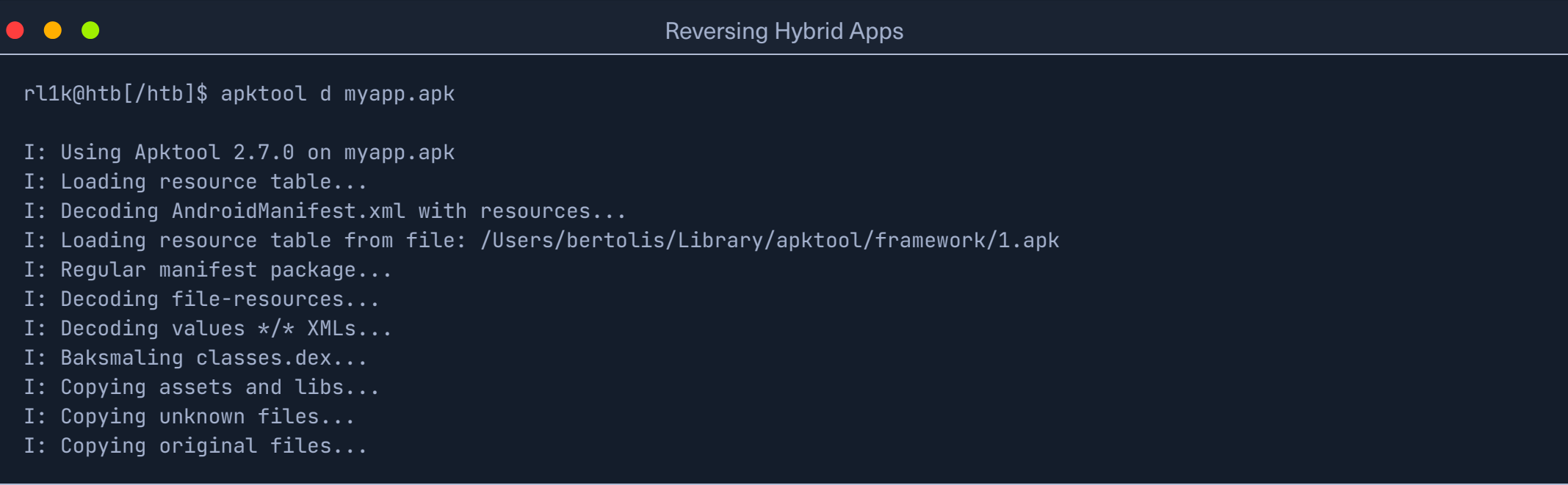
The `MainActivity` and `MainApplication` classes in React Native projects serve as bridges between the native Android environment and the JavaScript runtime. They are responsible for initializing and configuring the essential settings and components required for the React Native framework to function properly within the Android application. By navigating to `Source code` -> `Resources` -> `assets` -> `index.android.bundle`, we can access the bundled JavaScript code, as shown below.



When React Native builds the app, the Javascript code undergoes a minification process to reduce its size. This includes removing whitespace, renaming variables to shorter names, and performing other optimizations to decrease the file size. Although one could search for hardcoded strings, reading the application's source code requires substantial effort. To convert minified code back to a more readable form, we need to employ tools known as `beautifiers`. Such tools can be found online, like the `JSBeautifier` and `Pretty Print`. However, `index.android.bundle` is usually a large file, and it might be difficult for our web browser to display all this content effectively. On the other hand, command line tools like `js-beautify` can act more efficiently. Let's install this tool and try to read the content of the file `index.android.bundle`.



And of course, we will use `APKTool` to extract the files from the APK file.



Once files are extracted, we can run the following command to "beautify" the content of the minified JS code.



```
r11k@htb[/htb]$ js-beautify myapp/assets/index.android.bundle -o beautified_index.android.bundle.js

beautified beautified_index.android.bundle.js
```

Even after beautification, the file contains not only the application code written by the developer, but also the bundled code from all the `node_modules` used in the project. Fortunately, in most cases, the developer-written code appears toward the end of the file. Examining the contents of the extracted bundle reveals the following snippet.



```
r11k@htb[/htb]$ vim beautified_index.android.bundle.js
```

Code: `javascript`

```
x = (function() {
  var o = (0, t.default)((function*() {
    try {
      var t = yield fetch('http://10.10.10.10/login', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/x-www-form-urlencoded'
        },
        body: `username=${encodeURIComponent(f)}&password=${encodeURIComponent(b)}`
      }), o = yield t.text();
      t.ok ? console.log('Login Successful:', o, '. temp_debugging_key: DebuggingPassword') : console.log('Login Failed
    } catch (t) {
      console.error('Error:', t)
    }
  }));
  return function() {
    return o.apply(this, arguments)
  }
})();
```

The above snippet indicates that this is a remote authentication function and reveals the hardcoded debug password `DebuggingPassword`.

Reading Compiled JS Code

As mentioned before, React Native apps built using the Hermes engine are considered to have an extra security layer. The Hermes engine can be enabled or disabled by setting the `hermesEnabled` value to `true` or `false` accordingly under the `gradle.properties` file in an Android project.

Gradle Scripts

build.gradle (Project: AwesomeProject)

build.gradle.kts (Included build: gradle-plugin)

settings.gradle.kts (Included build: gradle-plugin)

build.gradle (Module :app)

proguard-rules.pro (ProGuard Rules for ":app")

gradle.properties (Project Properties)

gradle-wrapper.properties (Gradle Version)

local.properties (SDK Location)

settings.gradle (Project Settings)

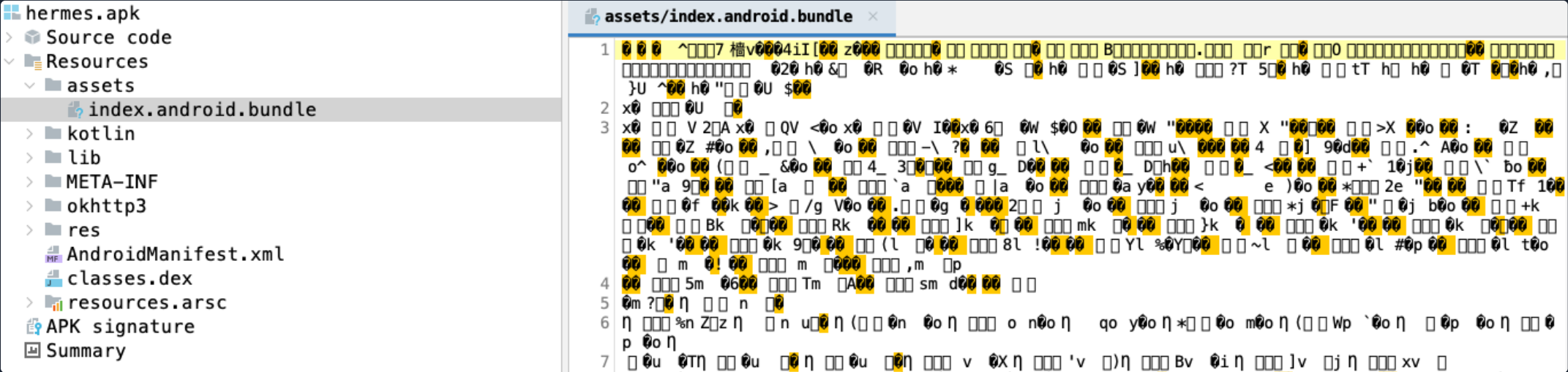
```
20 # AndroidX package structure to make it clearer which packages are bundled with the
21 # Android operating system, and which are packaged with your app's APK
22 # https://developer.android.com/topic/libraries/support-library/androidx-rn
23 android.useAndroidX=true
24 # Automatically convert third-party libraries to use AndroidX
25 android.enableJetifier=true
26
27 # Version of flipper SDK to use with React Native
28 FLIPPER_VERSION=0.182.0
29
30 # Use this property to specify which architecture you want to build.
31 # You can also override it from the CLI using
32 # ./gradlew <task> -PreactNativeArchitectures=x86_64
33 reactNativeArchitectures=armeabi-v7a,arm64-v8a,x86,x86_64
34
35 # Use this property to enable support to the new architecture.
36 # This will allow you to use TurboModules and the Fabric render in
37 # your application. You should enable this flag either if you want
38 # to write custom TurboModules/Fabric components OR use libraries that
39 # are providing them.
40 newArchEnabled=false
41
42 # Use this property to enable or disable the Hermes JS engine
```

```
42 # Use this property to enable or disable the Hermes JS engine.
43 # If set to false, you will be using JSC instead.
44 hermesEnabled=false
45
```

Reading the content of the `index.android.bundle` file using JADX or APKTool isn't possible.

Reversing Hybrid Apps

```
r11k@htb[/htb]$ jadx-gui hermes.apk
```



Decompiling the APK file using APKTool and running the `file` command on the `index.android.bundle` reveals the following.

Reversing Hybrid Apps

```
r11k@htb[/htb]$ apktool d hermes.apk
r11k@htb[/htb]$ cd hermes/assets/
r11k@htb[/htb]$ file index.android.bundle

index.android.bundle: Hermes JavaScript bytecode, version 94
```

This indicates that the file's content is `Hermes JavaScript bytecode`, meaning it's compiled non-human readable code, while the previous file containing only minified code would return `ASCII text`. Fortunately, some tools can disassemble files compiled into Hermes VM bytecode. In this example, we will use `hermes-dec`. Let's install it and try to decompile the `index.android.bundle` file.

Reversing Hybrid Apps

```
r11k@htb[/htb]$ pip3 install --upgrade git+https://github.com/P1sec/hermes-dec
r11k@htb[/htb]$ hbc-decompiler index.android.bundle output.js

[+] Decompiled output wrote to "output.js"
```

The file is successfully decompiled. Let's now read its content.

Reversing Hybrid Apps

```
r11k@htb[/htb]$ vim output.js
```

```

r3 = 'username=';
r2 = '&password=';
r2 = r9.bind(r3)(r11, r2, r10);
r3 = {};
r9 = 'POST';
r3['method'] = r9;
r3['headers'] = r8;
r3['body'] = r2;
r2 = 'http://10.10.10.10/login';
r2 = r7.bind(r1)(r2, r3);
SaveGenerator(address=131);
```

Examination of the code near the end of the file reveals a POST request to the URL `https://10.10.10.10/login`. Reading a few more lines below reveals a possible remote authentication functionality that uses the hardcoded key-value pair `debug_key: tempDebuggingPassword` to authenticate the user.

```

r9 = r5.console;
```

```
r8 = r9.log;
r7 = r4;
r6 = 'Login Failed: ';
r6 = r8.bind(r9)(r6, r7);
_fun4334_ip = 226; continue _fun4334;

r8 = r5.console;
r7 = r8.log;
r6 = r4;
r5 = 'Login Successful: ';
r4 = '. debug_key: tempDebuggingPassword';
r4 = r7.bind(r8)(r5, r6, r4);
```


Let's issue an HTTP request using the query parameter `debug_key` with the value `tempDebuggingPassword`. Note that although it's a POST request, the debug key is not being sent as form data.

Reversing Hybrid Apps

```
r11k@htb[/htb]$ curl -X POST 'http://10.10.10.10/login?debug_key=tempDebuggingPassword'

{
  "message": "Logged in using debug key!",
  "status": "success"
}
```

With the correct debug key identified in the source code, we successfully authenticated and gained access to the application.



Connect to Pwnbox

Your own web-based Parrot Linux instance to play our labs.

Pwnbox Location

UK

28ms

Terminate Pwnbox to switch location


Start Instance

/ 1 spawns left

Waiting to start...

Answer the question(s) below to complete this Section and earn cubes!


Target(s): [Click here to spawn the target system!](#)


+ 3  Analyze the APK found inside the attached ZIP file. What is the value of the "message" key after logging into the remote service using the debugging code?

Submit your answer here...

+10 Streak pts

Submit


 [myapp_hybrid_1.zip](#)

+ 5  Analyze the APK found inside the attached ZIP file. What is the value of the "message" key after logging into the remote service using the debugging code?

Submit your answer here...


+10 Streak pts

Submit

 [myapp_hybrid_2.zip](#)

← Previous

Next →

 Cheat Sheet

? [Go to Questions](#)

Table of Contents

Extracting and Enumerating APK Files


 Introduction

 Disassembling the APK


Understanding Smali

Analyzing Application's Source Code

 Reading Hardcoded Strings


 Bad Cryptography Implementation


 [Reversing Hybrid Apps](#)

 Reading Obfuscated Code

 Deobfuscating Code

Analyzing Native Libraries

 Reversing Shared Objects

 Reversing DLL Files

Application Patching

 Authentication Bypass

 Modifying Game Apps

 License Verification Bypass

 Root Detection Bypass

Skills Assessment

 Skills Assessment



My Workstation

OFFLINE

▶ Start Instance

∞ / 1 spawns left

