



WOW an API?

A data leak vulnerability

ZenTech

Note that Information contained in this document is for educational purposes.

+Contents

1	Introduction	1
1.1	Background	1
1.2	Aim	1
2	Procedure.....	2
2.1	Overview of Procedure	2
2.2	Converting the APK File.....	2
2.3	Traversing the Directory	3
2.4	Understanding the .bundle File and Finding the Flag	3
3	Results.....	6
3.1	Outcome of the Challenge	6
4	Countermeasures.....	7
4.1	General Overview	7
4.2	Obfuscation.....	7
4.3	Serverless functions	7
	References	8

1 INTRODUCTION

1.1 BACKGROUND

Reverse Engineering is a method which allows a person to understand how intellectual (various software) or physical goods operate. This is achieved by entirely or partially extracting data about the design and its implementation.

Depending on the language and the complexity of an application, reverse engineering it could take a long time and a lot of experience. A good example would be the popular programming language C Plus Plus (C++). Analysing software coded in this language would require the use of specialized software called disassemblers (Ghidra, IDA Pro, Cutter, etc.). The disassembler will break down the executable into instructions executed by the CPU (Central Processing Unit) and try to implement them into a specific language (usually C). Partial data can be extracted (i.e., specific strings) but an experience analyst will be required to accurately analyse the binary file.

On the contrary, there is also software coded in other languages which is significantly easier to reverse engineer. One of them is JavaScript and its React Native framework – the language and framework used to develop the DVAA (Damn Vulnerable Android Application).

1.2 AIM

This documentation has two main goals – providing a walkthrough of the vulnerability for the tester and appropriate countermeasures. The first aim will be met by giving a step-by-step guide on the exploitation of the vulnerability with images to support the reading material. This can be found in **Section 2 Procedure**. The countermeasures will cover steps which should be taken to prevent malicious attackers from exploiting the vulnerability and obtaining sensitive data about the application and the organisation which owns it. Further information can be seen in **Section 4.2 Countermeasures**.

2 PROCEDURE

2.1 OVERVIEW OF PROCEDURE

The procedure section will provide the reader with a step-by-step walkthrough of the vulnerability and how it will be exploited. Specialised software and knowledge (other than basic JavaScript and data analysis) will not be required due to the simple process. This, however, applies only if countermeasures were not taken to secure the data (such as obfuscation and encryption).

2.2 CONVERTING THE APK FILE

To start with the exploitation, you must first convert the APK file into an archive file (.zip, .rar or .7z). As mentioned previously, specialised software will not be required. Successfully converting the file can be easily achieved – right click on it, press on “Rename” and change the extension to whichever you would like to use. In this case the tester has changed the file extension to **.zip**.

This will convert the file to an archive. From this point you can either directly work with the .zip file or extract it and convert the file back to an **.apk**. Choosing the former option will leave you at risk of corrupting the file. This is not an issue as you can simply obtain a new copy of the **.apk**. The latter option will be more comfortable to work with as you will still have a running **.apk** file and all the extracted files will remain inside a separate folder. Altering any of them will not affect the application in any way. An image of the extracted folder can be seen on **Figure 2.2.1**.

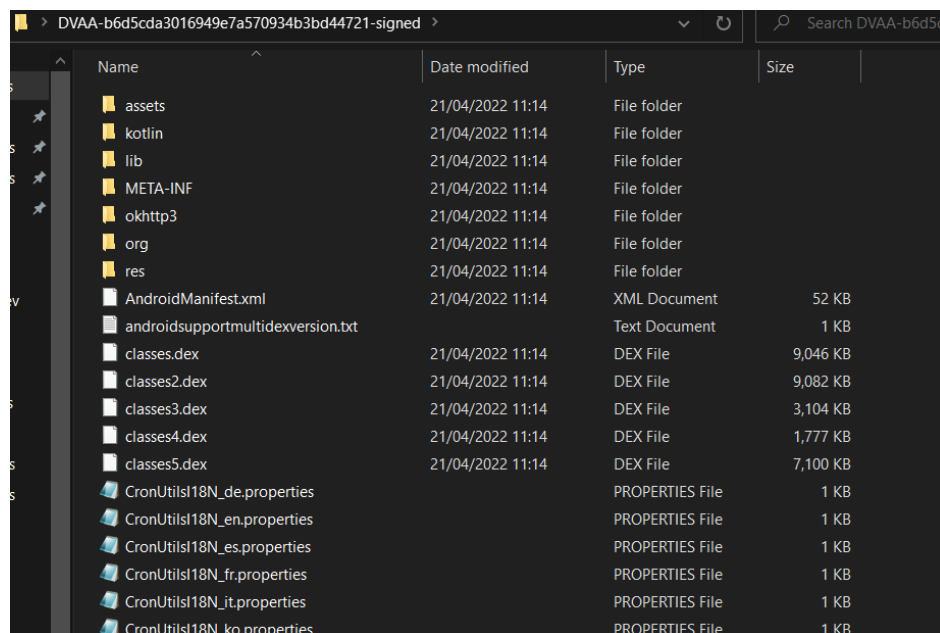
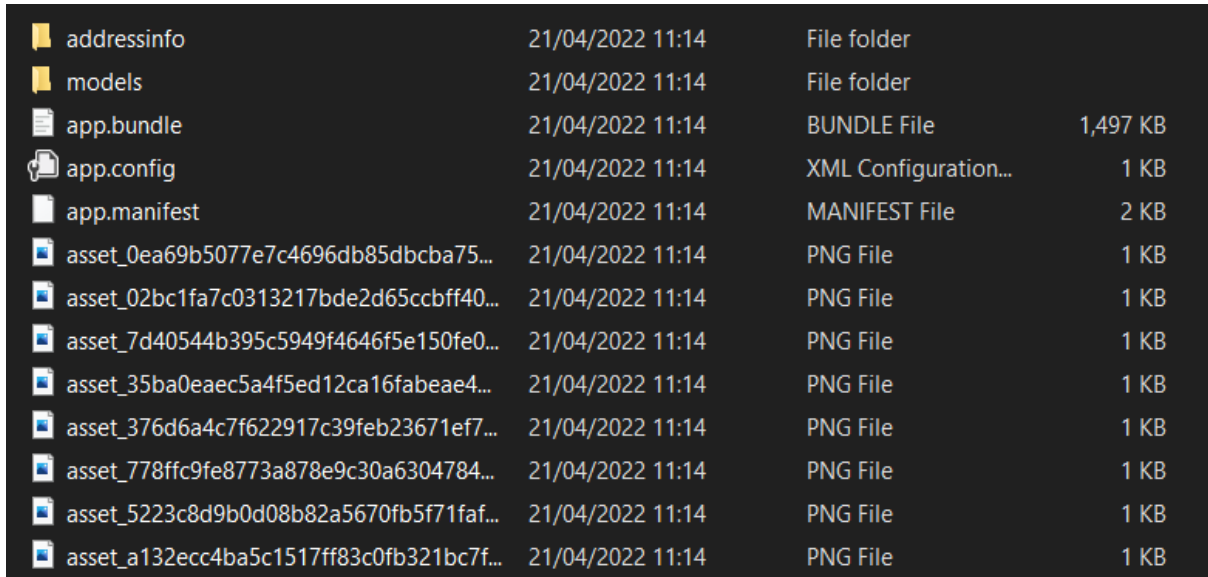


Figure 2.2.1 – Folder extracted from APK.

2.3 TRAVERSING THE DIRECTORY

Seeing the directory for the first time may cause confusion. Fear not, analysing software takes time and dedication so work through all files and subdirectories at your own pace. In this specific situation we need to check one specific directory – the **assets** directory. (**Figure 2.3.1**)



addressinfo	21/04/2022 11:14	File folder	
models	21/04/2022 11:14	File folder	
app.bundle	21/04/2022 11:14	BUNDLE File	1,497 KB
app.config	21/04/2022 11:14	XML Configuration...	1 KB
app.manifest	21/04/2022 11:14	MANIFEST File	2 KB
asset_0ea69b5077e7c4696db85dbcba75...	21/04/2022 11:14	PNG File	1 KB
asset_02bc1fa7c0313217bde2d65ccbff40...	21/04/2022 11:14	PNG File	1 KB
asset_7d40544b395c5949f4646f5e150fe0...	21/04/2022 11:14	PNG File	1 KB
asset_35ba0eaec5a4f5ed12ca16fabebae4...	21/04/2022 11:14	PNG File	1 KB
asset_376d6a4c7f622917c39feb23671ef7...	21/04/2022 11:14	PNG File	1 KB
asset_778ffc9fe8773a878e9c30a6304784...	21/04/2022 11:14	PNG File	1 KB
asset_5223c8d9b0d08b82a5670fb5f71faf...	21/04/2022 11:14	PNG File	1 KB
asset_a132ecc4ba5c1517ff83c0fb321bc7f...	21/04/2022 11:14	PNG File	1 KB

Figure 2.3.1 – Image of assets folder.

The **assets** folder contains all the resources for an application's proper workflow – fonts, models, etc. You can look through the files and directories if you wish to learn more about what they contain. You will also notice a **.bundle** file. It will have different names in specific situations but the most common ones are **index.android.bundle** and **app.bundle**.

2.4 UNDERSTANDING THE .BUNDLE FILE AND FINDING THE FLAG

The **.bundle** file contains a minified version of all the code used to create the application. (**Figure 2.4.1**) It may look confusing and hard to read but do not worry – it still contains the plain JavaScript code without any obfuscation (unless special tools are used to obfuscate it) or encryption. Additionally, you may also find a **.bundle.map** file. This file contains mapping of the functions which will make the **.bundle** file easier to read.

```

1 var _BUNDLE_START_TIME =this.nativePerformanceNow?nativePerformanceNow().Date.now(), _DEV =false,process=this.process||{},_METRO_GLOBAL_PREFIX ='';process.env.process.
2 !function(r){if("strict"===r._r_o,r[_METRO_GLOBAL_PREFIX +_d]=function(r,i,n){if(!n||e[i])return;var o={dependencyMap:n,factory:r,hasError:1,importedAll:t,imported
3 !function(n){var e=function(){function n(n,e){return n}function e(n){var e={};return n.forEach(function(n,r){e[n]=10}),e}function r(n,r,u){if(n.formatValueCalls++,n.format
4 !function(n){var r=0,t=function(n,r){throw n},l=(setGlobalHandler:function(n){t=n},getGlobalHandler:function(t){return t},reportError:function(n){t&&t(n,1)},reportFatalError
5 'undefined'!=typeof globalThis?globalThis:'undefined'!=typeof global?global:'undefined'!=typeof window&&window,function(){'use strict';var e=Object.prototype.hasOwnProperty
6 _d=function(g,r,i,a,m,e,d){var o=r(d[0]),t=r(d[1]),n=o(r(d[2]));(0,t.registerRootComponent)(n.default),0,[1,2,463]};
7 _d=function(g,r,i,a,m,e,d){m.exports=function(t){return t&&t._esModule?t:(default:t),m.exports._esModule=!0,m.exports.default=m.exports,1,[1]};
8 _d=function(g,r,i,a,m,e,d){var t=r(d[0]);Object.defineProperty(e,"_esModule",{value:10}),e.Logs=void 0,Object.defineProperty(e,"registerRootComponent",{enumerable:10,get:
9 _d=function(g,r,i,a,m,e,d){var t=r(d[0]),t(r(d[1]),r(d[2]),r(d[3]),r(d[4]),r(d[5]),var n,o,p,r(d[6]),u,p(r(d[7])),l=r(d[8]),s=t(r(d[9])),r(d[10]),t(r(d[11]));function i
10 _d=function(g,r,i,a,m,e,d){m.exports=function(t,n,o){return n in t?Object.defineProperty(t,n,{value:o,enumerable:10,configurable:10,writable:10}):t[n]=o,t},m.exports._esM
11 _d=function(g,r,i,a,m,e,d){var t=function(t,n){if(!n&&t&&t._esModule)return t;if(!t||t===t||"object"!=typeof t&&"function"!=typeof t)return t;if(!t||t===t||"object"!=
12 _d=function(g,r,i,a,m,e,d){var n=r(d[0]);Object.defineProperty(e,"_esModule",{value:10}),Object.defineProperty(e,"AppOwnership",{enumerable:10,get:function(){return f.App
13 _d=function(g,r,i,a,m,e,d){var t=r(d[0]);m.exports=function(o,n){if(!n||!o)return{};var l,p,s,t(o,n);if(Object.getOwnPropertySymbols){var u=Object.getOwnPropertySymbols(o)
14 _d=function(g,r,i,a,m,e,d){m.exports=function(t,n){if(!n||!t)return{};var o,u,f={};var o,u,f={};for(u=0;u<s.length;u++)o=s[u],n.indexOf(o)>=0||(f[o]=t[o]);return f},m.
15 _d=function(g,r,i,a,m,e,d){var t=r(d[0]);Object.defineProperty(e,"_esModule",{value:10});var n={RCTDeviceEventEmitter:10,DeviceEventEmitter:10,EventEmitter:10,NativeModule
16 _d=function(g,r,i,a,m,e,d){'use strict';r(d[0]),var t=r(d[1]);m.exports={get AccessibilityInfo(){return r(d[2])},get ActivityIndicator(){return r(d[3])},get Button(){return
17 _d=function(g,r,i,a,m,e,d){'use strict';m.exports=function(n,o,t,f,s,u,c,l){if(!n){var v;if(void 0===v)new Error("Minified exception occurred; use the non-minified ver
18 _d=function(g,r,i,a,m,e,d){'use strict';var n={};m.exports=function(o,t){n[o]}(console.warn(t),n[o]=10)},12,[1]};
19 _d=function(g,r,i,a,m,e,d){'use strict';var t=r(d[0]);Object.defineProperty(e,"_esModule",{value:10}),e.default=void 0;var u=t(r(d[1])),n=t(r(d[2])),l=t(r(d[3])),o=t(r(d[
20 _d=function(g,r,i,a,m,e,d){m.exports=function(o,n){if(!o instanceof n)throw new TypeError("Cannot call a class as a function");m.exports._esModule=!0,m.exports.default=
21 _d=function(g,r,i,a,m,e,d){function t(t,o){for(var n=0;n<o.length;n++){var p=o[n];p.enumerable=p.enumerable||1,p.configurable=10,"value" in p&&(p.writable=10),Object.define
22 _d=function(g,r,i,a,m,e,d){var t=r(d[0]);function o(){return"undefined"!=typeof Reflect&&Reflect.get?m.exports=Reflect.get,m.exports._esModule=!0,m.exports.default=m.
23 _d=function(g,r,i,a,m,e,d){var t=r(d[0]);m.exports=function(o,n){for(;!Object.prototype.hasOwnProperty.call(o,n)&&null!==(o=t(o));)return o;return o},m.exports._esModule=!0,m.exp
24 _d=function(g,r,i,a,m,e,d){function t(o){return m.exports=Object.setPrototypeOf?Object.getPrototypeOf:function(t){return t.__proto__||Object.getPrototypeOf(t)},m.exports
25 _d=function(g,r,i,a,m,e,d){var t=r(d[0]);m.exports=function(o,n){if("function"!=typeof n&&null!==(n)throw new TypeError("Super expression must either be null or a function")
26 _d=function(g,r,i,a,m,e,d){function t(o,s){return m.exports=Object.setPrototypeOf?function(t,o){return t.__proto__=o,t},m.exports._esModule=!0,m.exports.default=m.exp
27 _d=function(g,r,i,a,m,e,d){var o=r(d[0]),default,t=r(d[1]);m.exports=function(n,u){if(!u&&"object"===o(u)||"function"!=typeof u)return u;if(void 0!==(u)throw new TypeError(
28 _d=function(g,r,i,a,m,e,d){function o(t){if(!t)return t;if("function"!=typeof t)return t;if("function"!=typeof t)return t;if("function"!=typeof t)return t;if("function"!=typeof t)return t;if("function"!=
29 _d=function(g,r,i,a,m,e,d){m.exports=function(t){if(void 0===t)throw new ReferenceError("this hasn't been initialised - super() hasn't been called");return t},m.exports._e
30 _d=function(g,r,i,a,m,e,d){'use strict';Object.defineProperty(e,"_esModule",{value:10}),e.default=void 0;var t=r(d[0]),e.default=t,25,[26]};
31 _d=function(g,r,i,a,m,e,d){'use strict';var t=r(d[0]),s=t(r(d[1])),n=t(r(d[2])),u=t(r(d[3])),o=t(r(d[4])),c=r(d[5]),l=function(){return 10},b=function(){function t(n){(0,s
32 _d=function(g,r,i,a,m,e,d){'use strict';var t=r(d[0]),n=t(r(d[1])),u=t(r(d[2])),c=t(r(d[3])),f=t(r(d[4])),g=t(r(d[5])),function i(A){if("function"===typeof A&&A instanceof Reflect)10;Reflect

```

Figure 2.4.1 – Image of app.bundle

Now that you know what the file is and what it contains, you have two options. The first option is analysing the entirety of the file. This will take more time and will be a more tedious process, but with this you may obtain more in-depth information about the workflow of the application. You can also identify vulnerabilities in the code which you could find during a testing phase of the application.

The second option would require you to use for specific keywords like **API** or strings if you know that an API key starts with a specific set of characters. In our case, this method will be beneficial as we are looking for the flag. Opening the file in **Visual Studio Code** and pressing **Ctrl+F** will open a search dialogue window. There you can try to look for specific keywords. We are looking for a flag so the string **"FLAG{"** will be used in the search. (Figure 2.4.2)

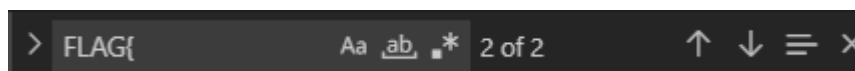


Figure 2.4.2 – String search.

The editor will then move the cursor to the exact location of the located string – there you can see the flag and the API key used for the Google Dialog Flow package. (Figure 2.4.3) Going back to the screen of this vulnerability and inputting the flag shows an alert indicating that this is the correct flag. Congratulations, you solved the CTF! (Figure 2.4.4)

```

IN PRIVATE KEY-----nMIIEvglBADA...1oq0awdddddiojdq9wg09wd81u1ijzcjklasd-----END PRIVATE KEY-----\n',y.DialogFlow_V2.LANG_ENGLISH,'testv2-3b4dss','FLAG(WON_API)')

```

Figure 2.4.3

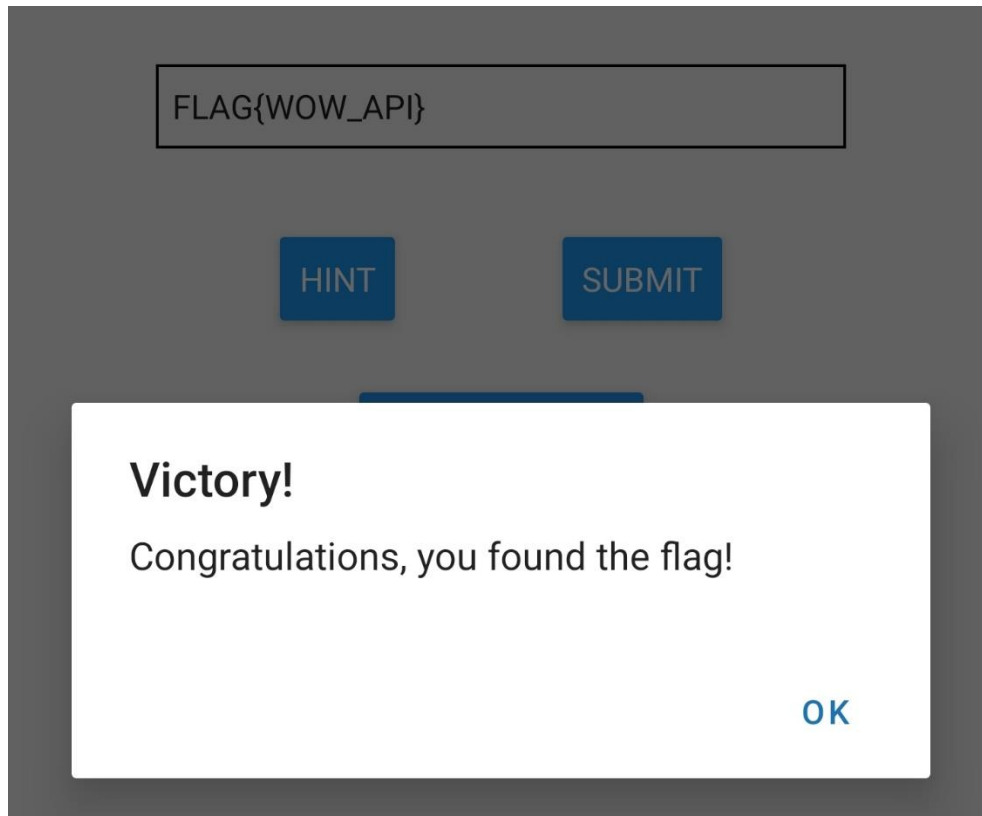


Figure 2.4.4 – Solving the CTF challenge

3 RESULTS

3.1 OUTCOME OF THE CHALLENGE

Solving the CTF challenge resulted in the following outcomes:

- Knowledge about the structure of React Native apk file
- Knowledge about the **.bundle** and **.bundle.map** files
- The structure of the files and where to obtain sensitive data

Reverse engineering react native applications is not a difficult process if no countermeasures have been taken. Considering this and the knowledge you obtained from above, you now know how to exploit this and obtain sensitive information. In the following section you will learn about the countermeasures which can be taken to prevent this issue.

4 COUNTERMEASURES

4.1 GENERAL OVERVIEW

Overcoming this vulnerability can be achieved in a multitude of ways. One of the easiest and most efficient ways, however, are with the use of obfuscation and serverless functions. Additionally, sensitive data should never be directly inserted into the code but should be kept as separate resources and the indirectly accessed through variables.

Let us cover each of the beforementioned methods separately.

4.2 OBFUSCATION

Obfuscation is the process of making something unclear, unreadable. This will make reverse engineering software a significantly harder task as the code will not be available to the attacker in plain sight. You can easily do this by using ProGuard.

ProGuard is a tool integrated into the Android SDK. It is easy to use, and not only does it obfuscate the code, but it also performs additional code optimisation which will reduce the final size of the application.

Successfully implementing the tool can be achieved by including the following line inside your android/app/build.gradle file: **def enableProguardInReleaseBuilds = true**. Further and more detailed information can be obtained in the React Native documentation. (React Native, 2022)

4.3 SERVERLESS FUNCTIONS

Serverless functions use the FaaS model (Function as a Service) which makes it easy to run code on cloud without providing any compute infrastructure. Those functions are stateless and are not aware of any other functions within the software.

The most secure way to use them and handle API keys is by building an orchestration layer between the app and the used resource. How is this secure? Secrets in server-side code do not have the same accessibility rules for API consumers in comparison to the app code. Further information can be obtained in the React Native documentation. (React Native, 2022)

REFERENCES

React Native. (2022). *Enabling Proguard to reduce the size of the APK*. Available: <https://reactnative.dev/docs/signed-apk-android#enabling-proguard-to-reduce-the-size-of-the-apk-optional>. Last accessed 4th Mar 2022.

React Native. (2022). *Security*. Available: <https://reactnative.dev/docs/security>. Last accessed 4th Mar 2022.