License Verification Bypass

As we examine more scenarios of application patching, a significant area of focus is the license verification bypass process. This mechanism is designed to ensure that the users have legitimately purchased or are authorized to use an application, and it refers to paid apps or apps with in-app purchases. License Verification Bypass is a method that can be used by malicious actors who want to use paid features without actually purchasing the license. Various techniques can be used to bypass mechanisms like these, including modifying the application's code (application patching), using third-party tools to emulate a valid license, or intercepting and altering the communication between the application and the licensing server. Below are the steps that show how the app is getting verified.

Step	Description
`License Verification Library (LVL)`	Android provides the License Verification Library (LVL) to facilitate communication between an app and Google's licensing servers in order to verify the app's licensing status.
`Verification Process`	When an app with license verification is launched, it attempts to contact Google's servers to confirm whether the user has legitimately purchased the app or a specific feature.
`Server Response`	The server responds with the licensing status, and the app uses this response to determine whether to grant access to its full functionality or restrict certain features.

The main purpose of using license verification in applications is to prevent piracy and unauthorized use. Piracy in Android applications can occur as follows: Imagine a premium application with its license verification altered and redistributed via third-party app stores. In this modified version, the app's internal license verification checks are patched to always return a valid status, regardless of whether the user has legitimately purchased the license. This enables users who download the app from these third-party stores to access paid features without payment.

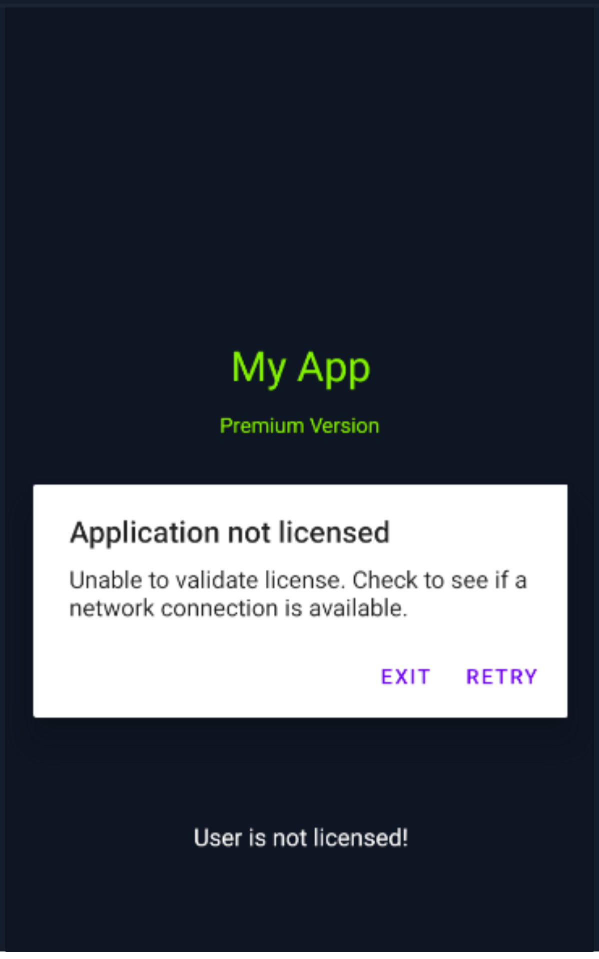
In this exercise, we will simulate the abovementioned piracy scenario and attempt to bypass the an app's license verification. Once your AVD (or emulator of your choice) is running, use the following commands to connect to the device via `ADB` and install the application:

License Verification Bypass

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

Performing Streamed Install
Success
```

After the app launches, tapping the `ACCESS PREMIUM FEATURE` button returns the following message.



My App

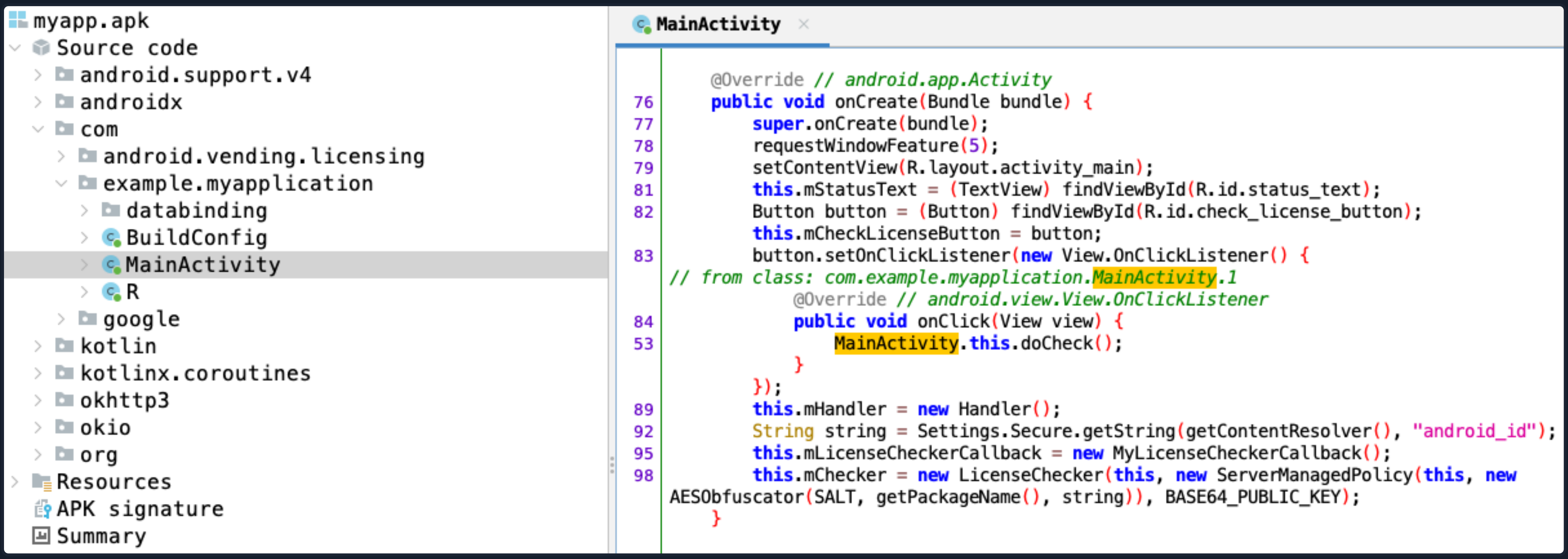
Premium Version

Application not licensed

Unable to validate license. Check to see if a network connection is available.

User is not licensed!

The message states that in order to have access to the premium features the app provides, we need to purchase a license. Let's start by reading the application's source code using JADX.

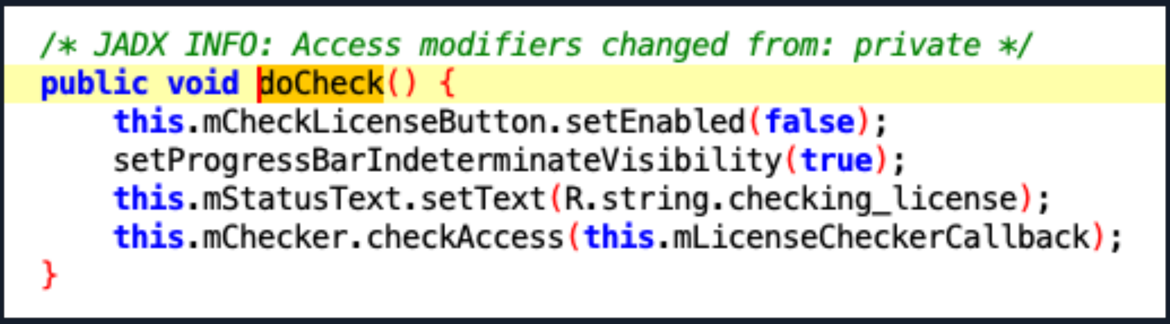


```
myapp.apk
  Source code
    android.support.v4
    androidx
    com
      android.vending.licensing
      example.myapplication
        databinding
        BuildConfig
        MainActivity
        R
      google
    kotlin
    kotlinx.coroutines
    okhttp3
    okio
    org
  Resources
  APK signature
  Summary
```

MainActivity

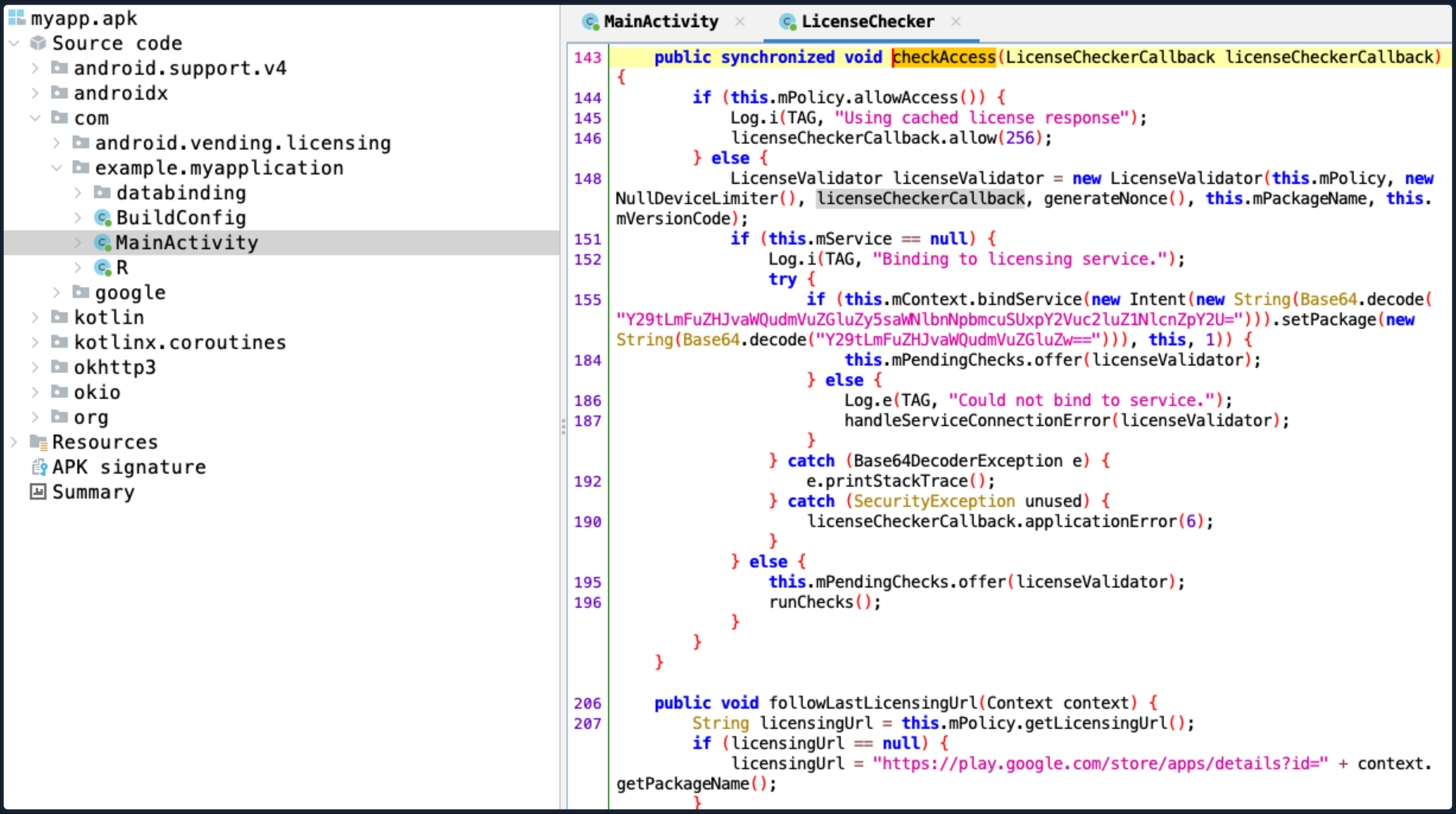
```java
    @Override // android.app.Activity
    public void onCreate(Bundle bundle) {
        super.onCreate(bundle);
        requestWindowFeature(5);
        setContentView(R.layout.activity_main);
        this.mStatusText = (TextView) findViewById(R.id.status_text);
        Button button = (Button) findViewById(R.id.check_license_button);
        this.mCheckLicenseButton = button;
        button.setOnClickListener(new View.OnClickListener() {
// from class: com.example.myapplication.MainActivity.1
            @Override // android.view.View.OnClickListener
            public void onClick(View view) {
                MainActivity.this.doCheck();
            }
        });
        this.mHandler = new Handler();
        String string = Settings.Secure.getString(getContentResolver(), "android_id");
        this.mLicenseCheckerCallback = new MyLicenseCheckerCallback();
        this.mChecker = new LicenseChecker(this, new ServerManagedPolicy(this, new
AESObfuscator(SALT, getPackageName(), string)), BASE64_PUBLIC_KEY);
    }
```

Reading the `onCreate()` method of the `MainActivity` class, we see that the `doCheck()` method is called.



```java
/* JADX INFO: Access modifiers changed from: private */
public void doCheck() {
    this.mCheckLicenseButton.setEnabled(false);
    setProgressBarIndeterminateVisibility(true);
    this.mStatusText.setText(R.string.checking_license);
    this.mChecker.checkAccess(this.mLicenseCheckerCallback);
}
```

Double-clicking on it reveals the `checkAccess()` method of the class `LicenseChecker`.



```
myapp.apk
  Source code
    android.support.v4
    androidx
    com
      android.vending.licensing
      example.myapplication
        databinding
        BuildConfig
        MainActivity
        R
      google
    kotlin
    kotlinx.coroutines
    okhttp3
    okio
    org
  Resources
  APK signature
  Summary
```

MainActivity   LicenseChecker

```java
    public synchronized void checkAccess(LicenseCheckerCallback licenseCheckerCallback)
{
        if (this.mPolicy.allowAccess()) {
            Log.i(TAG, "Using cached license response");
            licenseCheckerCallback.allow(256);
        } else {
            LicenseValidator licenseValidator = new LicenseValidator(this.mPolicy, new
NullDeviceLimiter(), licenseCheckerCallback, generateNonce(), this.mPackageName, this.
mVersionCode);
            if (this.mService == null) {
                Log.i(TAG, "Binding to licensing service.");
                try {
                    if (this.mContext.bindService(new Intent(new String(Base64.decode(
"Y29tLmFuZHJvaWQudmVuZGluZy5saWNlbnNpbmcuSUxpY2Vuc2luZ1NlcnZpY2U="))).setPackage(new
String(Base64.decode("Y29tLmFuZHJvaWQudmVuZGluZw=="))), this, 1)) {
                        this.mPendingChecks.offer(licenseValidator);
                    } else {
                        Log.e(TAG, "Could not bind to service.");
                        handleServiceConnectionError(licenseValidator);
                    }
                } catch (Base64DecoderException e) {
                    e.printStackTrace();
                } catch (SecurityException unused) {
                    licenseCheckerCallback.applicationError(6);
                }
            } else {
                this.mPendingChecks.offer(licenseValidator);
                runChecks();
            }
        }
    }

    public void followLastLicensingUrl(Context context) {
        String licensingUrl = this.mPolicy.getLicensingUrl();
        if (licensingUrl == null) {
            licensingUrl = "https://play.google.com/store/apps/details?id=" + context.
getPackageName();
        }
```

The method seems to check if the user is verified, calling the method `licenseCheckerCallback.allow(256)` if the verification succeeds. Otherwise, the app will make the verification request to the remote server. On the left side of the window, we can also see the package `android.vending.license`.

Searching online reveals this GitHub repository as the third result, which, according to the project's description, is the client library for the Google Play licensing server. The same snippet of code (but in the original Java format) is also available, and can help us further understand the functionality of the validation process.

```java
public synchronized void checkAccess(LicenseCheckerCallback callback) {
    // If we have a valid recent LICENSED response, we can skip asking
    // Market.
    if (mPolicy.allowAccess()) {
        Log.i(TAG, "Using cached license response");
        callback.allow(Policy.LICENSED);
    } else {
        LicenseValidator validator = new LicenseValidator(mPolicy, new NullDeviceLimiter(),
                callback, generateNonce(), mPackageName, mVersionCode);
```

Let's try to edit the Smali code of the application and make it always call the `licenseCheckerCallback.allow(256)` method. First, let's decompile the APK file using APKTool.

License Verification Bypass

```
rl1k@htb[/htb]$ apktool d myapp.apk

I: Using Apktool 2.7.0 on myapp.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: /Users/bertolis/Library/apktool/framework/1.apk
I: Regular manifest package...
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Baksmaling classes.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...
I: Copying META-INF/services directory
```

Listing the directory `./myapp/smali/com/example/myapplication` won't reveal the `LicenseChecker` class, as it's part of the `android.vending.license` package. However, listing the content of the directory `./myapp/smali/com/google/android/vending/licensing/` returns the following Smali files.

License Verification Bypass

```
rl1k@htb[/htb]$ ls -l ./myapp/smali/com/google/android/vending/licensing/

total 368
-rw-r--r--  1 bertolis  bertolis  12441 Nov 15 01:22 AESObfuscator.smali
-rw-r--r--  1 bertolis  bertolis  32069 Nov 15 01:22 APKExpansionPolicy.smali
-rw-r--r--  1 bertolis  bertolis    530 Nov 15 01:22 BuildConfig.smali
-rw-r--r--  1 bertolis  bertolis    234 Nov 15 01:22 DeviceLimiter.smali
-rw-r--r--  1 bertolis  bertolis   3746 Nov 15 01:22 LicenseChecker$ResultListener$1.smali
-rw-r--r--  1 bertolis  bertolis   5939 Nov 15 01:22 LicenseChecker$ResultListener$2.smali
-rw-r--r--  1 bertolis  bertolis   5420 Nov 15 01:22 LicenseChecker$ResultListener.smali
-rw-r--r--  1 bertolis  bertolis  25754 Nov 15 01:22 LicenseChecker.smali
<SNIP>
```

Our target function is contained in the `LicenseChecker.smali` file. Just as we did in the previous section, we will try to find the `if` condition and subsequently patch it. Searching for the message `Using cached license response` reveals the following snippet.

Code: smali

```smali
<SNIP>
    if-eqz v0, :cond_0

    const-string v0, "LicenseChecker"
    const-string v1, "Using cached license response"

    .line 145
    invoke-static {v0, v1}, Landroid/util/Log;->i(Ljava/lang/String;Ljava/lang/String;)I

    const/16 v0, 0x100

    .line 146
    invoke-interface {p1, v0}, Lcom/google/android/vending/licensing/LicenseCheckerCallback;->allow(I)V

    goto :goto_0

    .line 148
    :cond_0
    new-instance v7, Lcom/google/android/vending/licensing/LicenseValidator;
<SNIP>
```

Let's copy the snippet of code between the lines `if-eqz v0, :cond_0` and `goto :goto_0` and paste it right before the `if-eqz v0, :cond_0`. This way, the app will always return a valid status. The updated code snippet appears like this.

Code: smali

```
<SNIP>
    const-string v0, "LicenseChecker"

    const-string v1, "Using cached license response"

    .line 145
    invoke-static {v0, v1}, Landroid/util/Log;->i(Ljava/lang/String;Ljava/lang/String;)I

    const/16 v0, 0x100

    .line 146
    invoke-interface {p1, v0}, Lcom/google/android/vending/licensing/LicenseCheckerCallback;->allow(I)V

    if-eqz v0, :cond_0
<SNIP>
```

Changing the `if-eqz v0, :cond_0` condition to `if-nez v0, :cond_0` can also work, but the validation would only apply when the user is not verified. The updated snippet is shown below.

Code: smali

```
<SNIP>
    if-nez v0, :cond_0

    const-string v0, "LicenseChecker"
<SNIP>
```

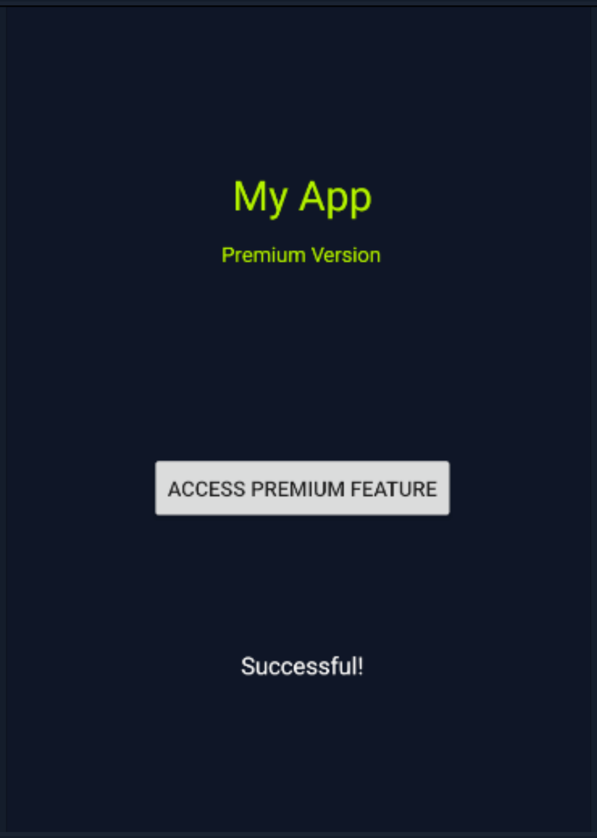
After patching the smali code, we can proceed to recompile, sign, and install the modified application.

License Verification Bypass

```
rl1k@htb[/htb]$ apktool b myapp
rl1k@htb[/htb]$ echo -e "password\npassword\njohn doe\ntest\ntest\ntest\ntest\ntest\nyes" > params.txt
rl1k@htb[/htb]$ cat params.txt | keytool -genkey -keystore key.keystore -validity 1000 -keyalg RSA -alias john
rl1k@htb[/htb]$ zipalign -p -f -v 4 myapp/dist/myapp.apk myapp_aligned.apk
rl1k@htb[/htb]$ echo password | apksigner sign --ks key.keystore myapp_aligned.apk
rl1k@htb[/htb]$ adb uninstall com.hackthebox.myapp
rl1k@htb[/htb]$ adb install myapp_aligned.apk
```

```
Performing Incremental Install
Serving...
All files should be loaded. Notifying the device.
Success
Install command complete in 619 ms
```

Once the app is installed, let's run it and tap the `ACCESS PREMIUM FEATURE` button once again.



The license check verification is successfully bypassed.

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

Pwnbox Location

UK 34ms

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!

+ 3 What is the message displayed on the screen after bypassing the license check verification mechanism?

Submit your answer here...

+10 Streak pts Submit myapp_license.zip

Previous Next

Cheat Sheet

? Go to Questions

Table of Contents

My Workstation

OFFLINE

Start Instance

∞ / 1 spawns left