Abusing Google Play Billing for fun and unlimited credits!

Guillaume Lopes - @Guillaume_Lopes

29/11/2019

Who am I?

- Senior Penetration Tester at RANDORISEC
  - 10 years of experience in different fields (Active Directory, Windows, Linux, Web applications, Wifi, Android)
- Member of the Checkmarx Application Security Research Team
  - https://www.checkmarx.com/category/blog/technical-blog/
- Security Researcher at Cobalt.io
- Play CTF (Hackthebox, Insomni’hack, Nuit du Hack, …)
  - Gives a hand to the Tipi’hack team

Agenda

1. Google Play Billing Presentation
2. Known Vulnerabilities
3. Vulnerable Applications
4. Conclusion

DEEPSEC

Google Play Billing Presentation

How does it work?

DEEPSEC

Google Play Billing Presentation

- Google Play Billing (or Google InApp Billing)
  - “service that lets you sell digital content inside your Android app”

- Android framework that allows to easily monetize applications
  - One-time products
  - Subscriptions

Google Play Billing Presentation

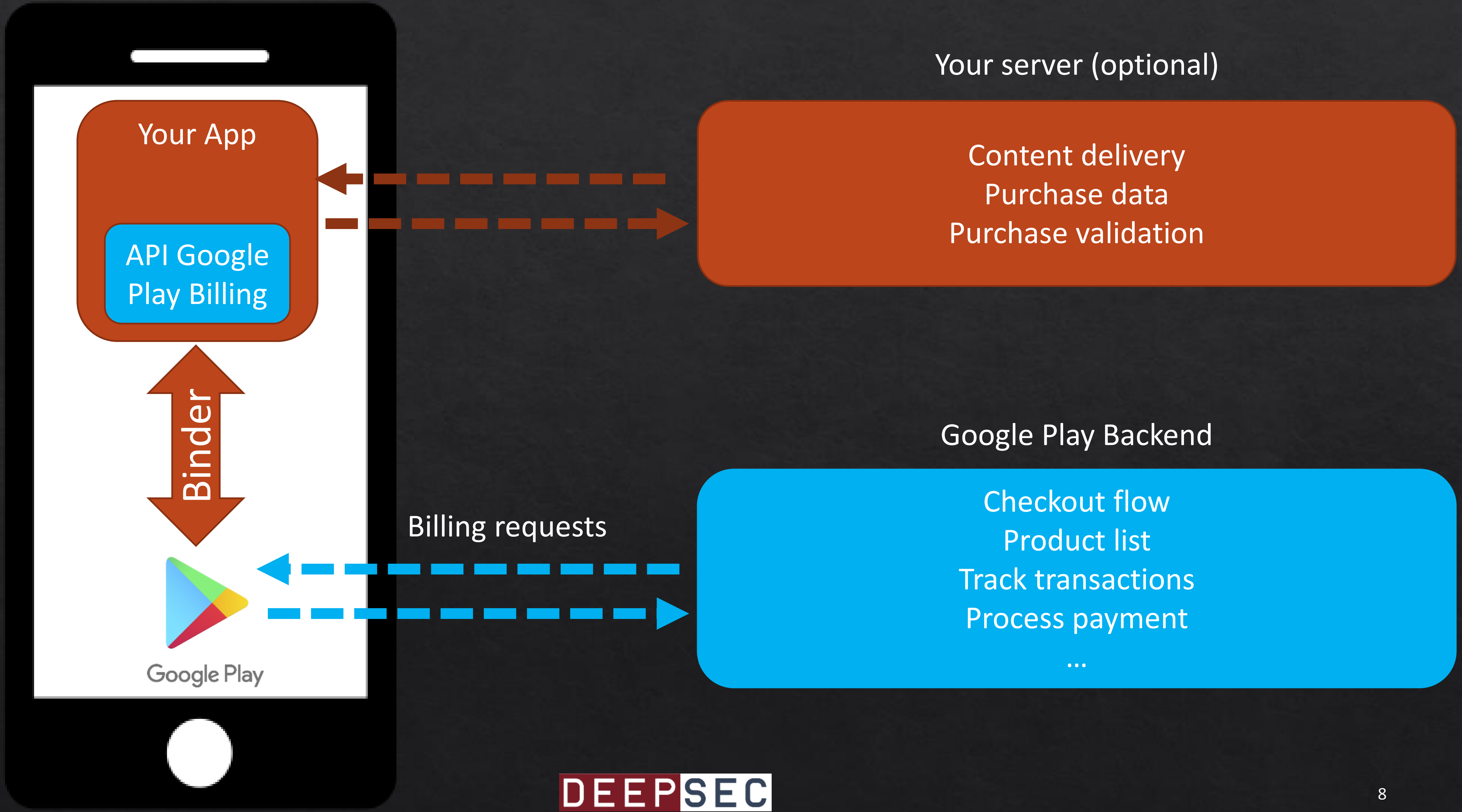
- One-time products
  - You can buy it only once
    - Premium version of an app
    - Extra products (coins, credits, music, etc.)

- Subscriptions
  - Need a regular payment (daily, weekly or monthly)
    - Subscription to a magazine or a newspaper

Google Play Billing Presentation

- Payment is handled by Google
  - Need to have Google Play services in your device
  - Credit card not exposed to the developers
  - Products need to be defined in the Google Play Console
  - Tracking made by Google

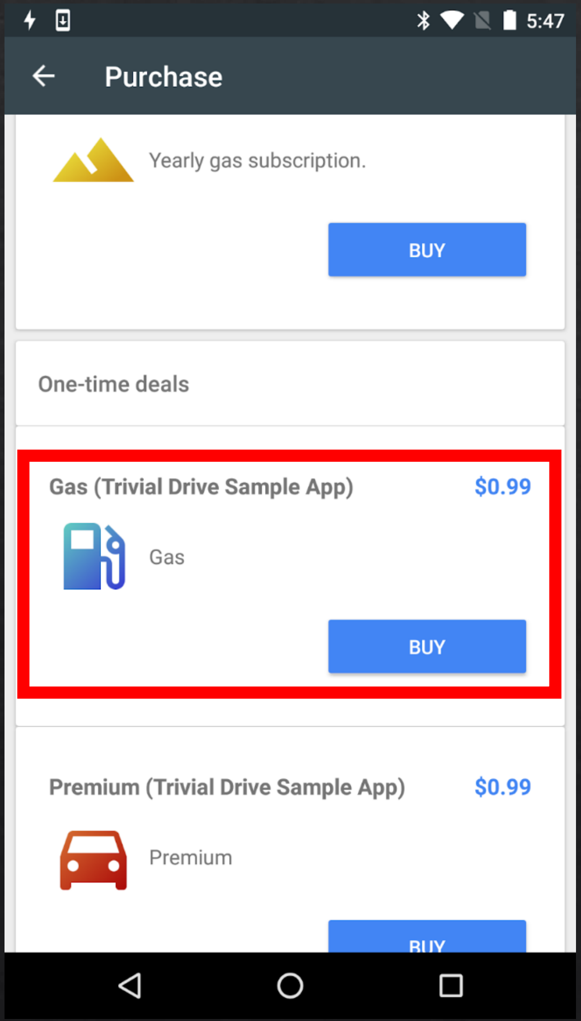
Google Play Billing Presentation



DEEPSEC

Google Play Billing Presentation

- Payment process handled by Google



Purchase

Yearly gas subscription.

BUY

One-time deals

Gas (Trivial Drive Sample App) $0.99

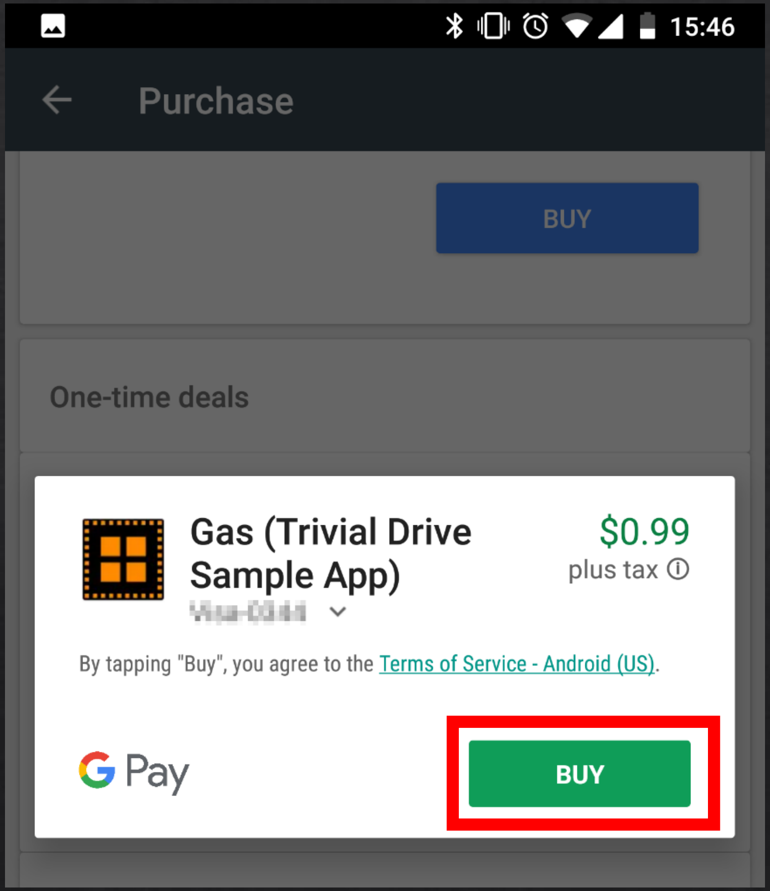
Gas

BUY

Premium (Trivial Drive Sample App) $0.99

Premium

BUY



Purchase

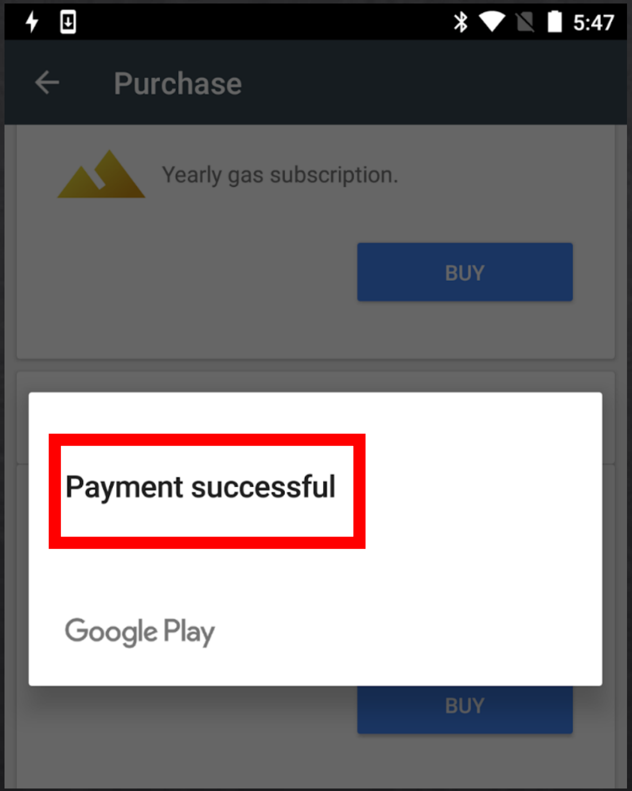
BUY

One-time deals

Gas (Trivial Drive Sample App) $0.99 plus tax

By tapping "Buy", you agree to the Terms of Service - Android (US).

G Pay BUY



Purchase

Yearly gas subscription.

BUY

Payment successful

Google Play

BUY

DEEPSEC

Google Play Billing Presentation

- Purchase validation can be performed in 2 ways
  - Verify purchase on a server
  - Verify purchase on a device

- In any case, Google returns a JSON object containing the response data of the purchase
  - INAPP_PURCHASE_DATA

Google Play Billing Presentation

- The JSON object contains
  - **orderId: I**dentifier of the transaction
  - **packageName:** App name
  - **productId:** Product identifier
  - **purchaseState**: Integer with 2 possible values 0 (Purchased) or 1 (Canceled)
  - **purchaseToken**: String generated by Google Play to uniquely identify the transaction

Google Play Billing Presentation

- Example of INAPP_PURCHASE_DATA

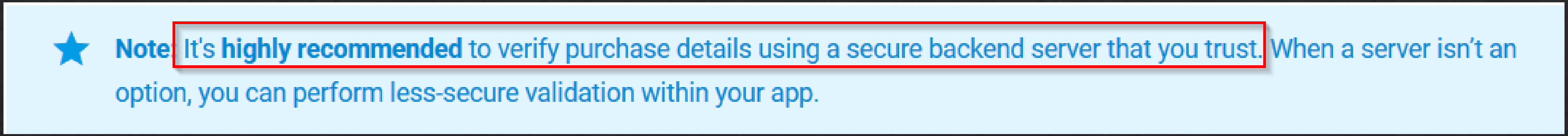
```
{
   "orderId":"63955112435364424 09.37657008969468 63",
   "packageName":"com.lima.doodlejump",
   "productId":"doodlejump.candys1000",
   "purchaseTime":1574248391379,
   "purchaseState":0,
   "developerPayload":"",
   "purchaseToken":"ijkgeytwuimhgxidsloanbnc"
}
```

Google Play Billing Presentation

- Google Play signs the JSON object
  - The Google Play Console generates an RSA key pair for each application
  - The private key is associated to the application used
  - The application uses the public key to validate the signature

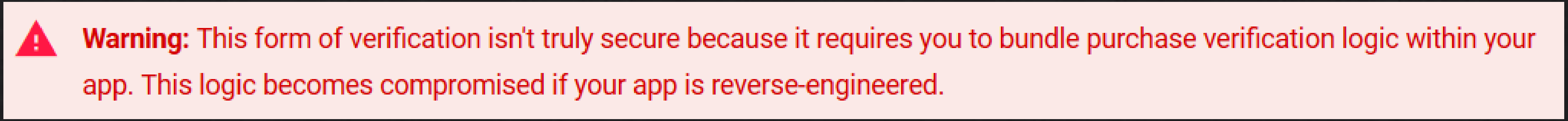
Google Play Billing Presentation

- Google recommends to validate purchase details on a server controlled by the developer



Note: It's **highly recommended** to verify purchase details using a secure backend server that you trust. When a server isn't an option, you can perform less-secure validation within your app.

- However, it is still possible to verify the purchase on the device by validating the signature

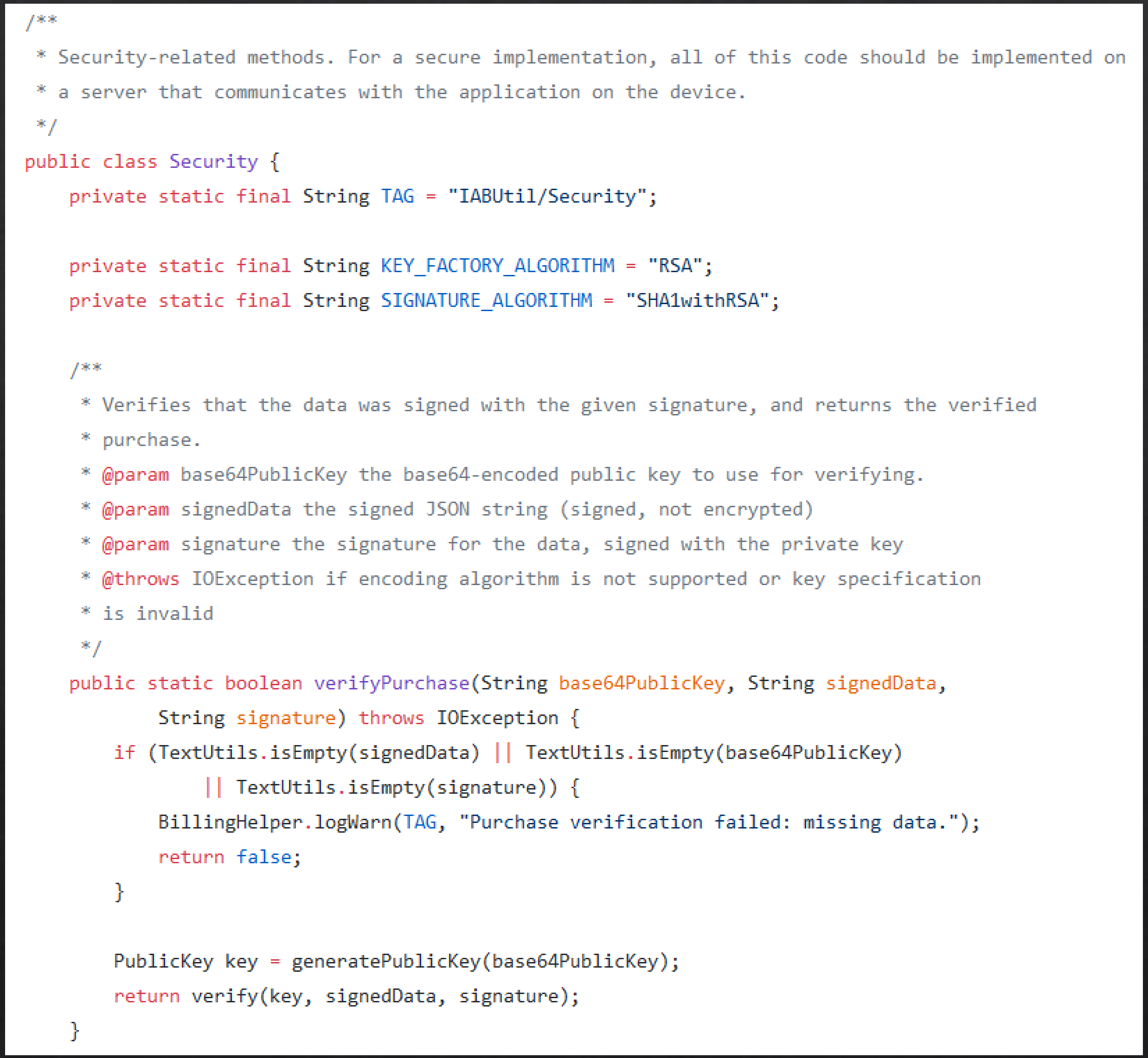


Warning: This form of verification isn't truly secure because it requires you to bundle purchase verification logic within your app. This logic becomes compromised if your app is reverse-engineered.

DEEPSEC

Google Play Billing Presentation

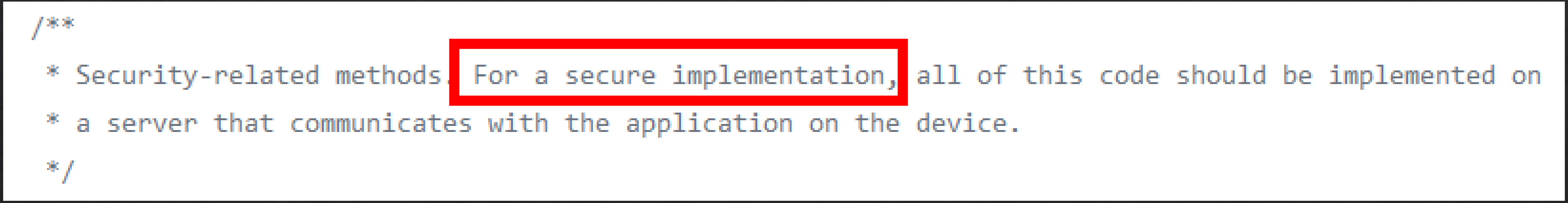
- Trivial Drive v2
  - Sample app
  - Example on how to use the Google Play Billing API



```
/**
 * Security-related methods. For a secure implementation, all of this code should be implemented on
 * a server that communicates with the application on the device.
 */
public class Security {
    private static final String TAG = "IABUtil/Security";

    private static final String KEY_FACTORY_ALGORITHM = "RSA";
    private static final String SIGNATURE_ALGORITHM = "SHA1withRSA";

    /**
     * Verifies that the data was signed with the given signature, and returns the verified
     * purchase.
     * @param base64PublicKey the base64-encoded public key to use for verifying.
     * @param signedData the signed JSON string (signed, not encrypted)
     * @param signature the signature for the data, signed with the private key
     * @throws IOException if encoding algorithm is not supported or key specification
     * is invalid
     */
    public static boolean verifyPurchase(String base64PublicKey, String signedData,
            String signature) throws IOException {
        if (TextUtils.isEmpty(signedData) || TextUtils.isEmpty(base64PublicKey)
                || TextUtils.isEmpty(signature)) {
            BillingHelper.logWarn(TAG, "Purchase verification failed: missing data.");
            return false;
        }

        PublicKey key = generatePublicKey(base64PublicKey);
        return verify(key, signedData, signature);
    }
```

DEEPSEC

Google Play Billing Presentation

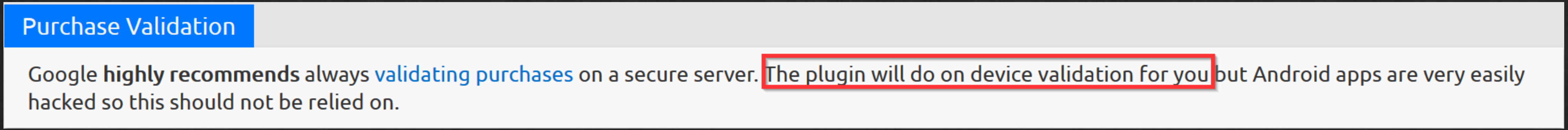


```
/**
 * Security-related methods. For a secure implementation, all of this code should be implemented on
 * a server that communicates with the application on the device.
 */
```

DEEPSEC

Google Play Billing Presentation

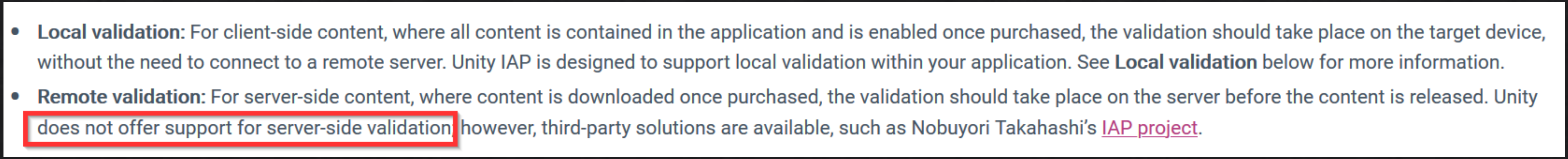
- And what about 3rd party libraries?
  - Prime31



Purchase Validation

Google **highly recommends** always validating purchases on a secure server. The plugin will do on device validation for you but Android apps are very easily hacked so this should not be relied on.

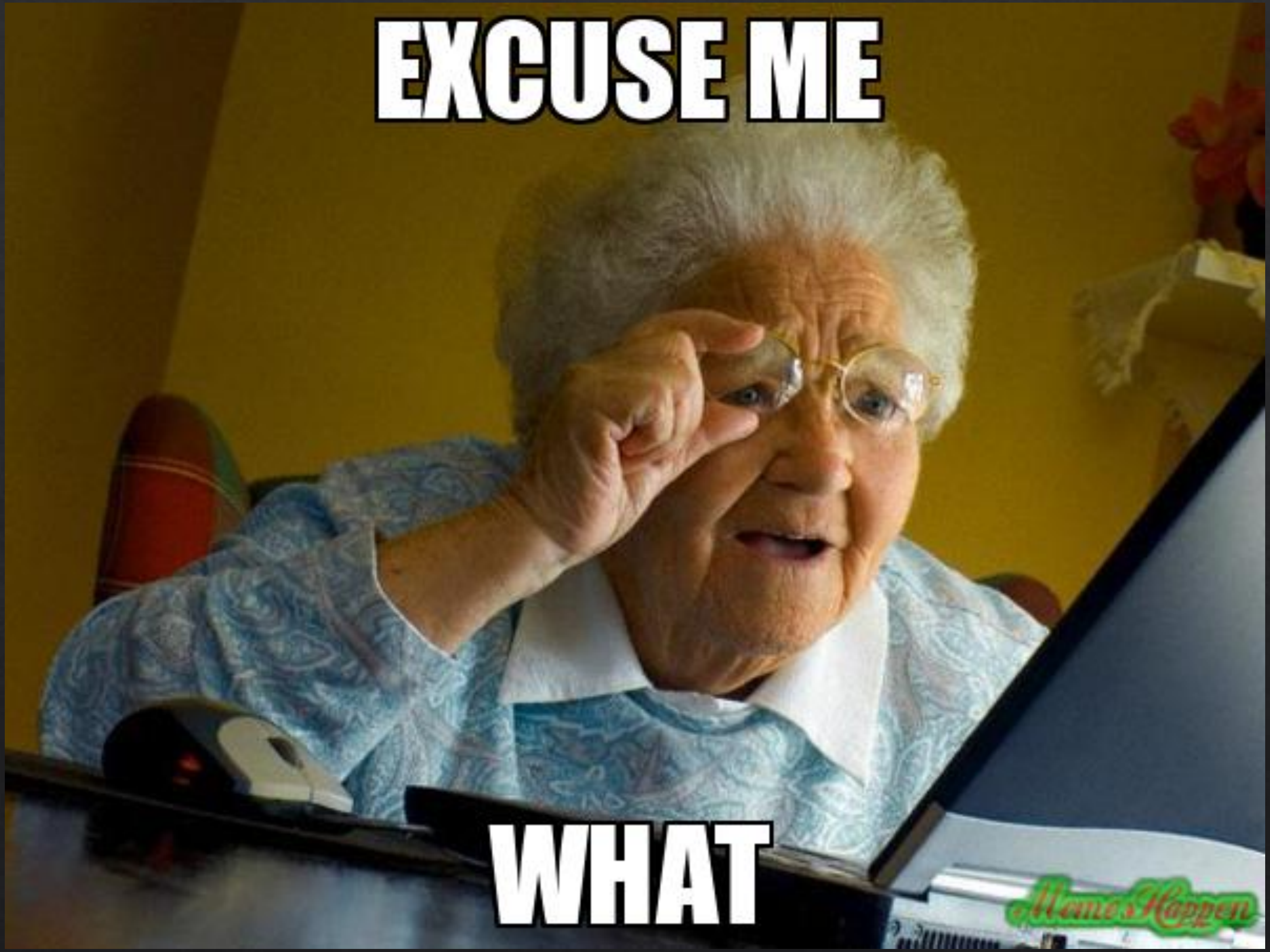
  - UnityIAP



- **Local validation:** For client-side content, where all content is contained in the application and is enabled once purchased, the validation should take place on the target device, without the need to connect to a remote server. Unity IAP is designed to support local validation within your application. See **Local validation** below for more information.
- **Remote validation:** For server-side content, where content is downloaded once purchased, the validation should take place on the server before the content is released. Unity does not offer support for server-side validation; however, third-party solutions are available, such as Nobuyori Takahashi's IAP project.

DEEPSEC

Google Play Billing Presentation



DEEPSEC

Google Play Billing Presentation

- Google gives the possibility to validate the purchase locally
  - In the documentation, Google performs the validation on the device
  - Well know 3rd party libraries only support device validation
- So what do you think developers are going to do?

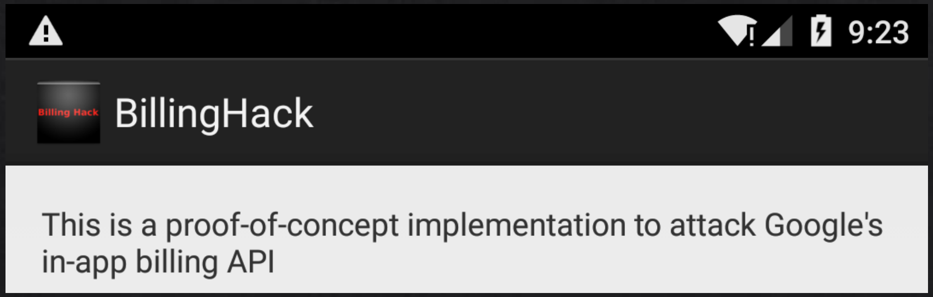
DEEPSEC

Known Vulnerabilities

A little bit of history

Known Vulnerabilities

- 2013: Dominik Schürmann found 2 vulnerabilities allowing to bypass the payment process
  - Bug disclosed to Google (Hall of Fame)
- Dominik developed an app as PoC called BillingHack
  - Just need to launch the app in background



This is a proof-of-concept implementation to attack Google's in-app billing API

DEEPSEC

Known Vulnerabilities

1. A malicious app can impersonate the Google Play Billing service (com.android.vending)

   - Define an Intent filter with a high priority

```
<intent-filter android:priority="2147483647" >
   <action android:name="com.android.vending.billing.InAppBillingService.BIND" />
</intent-filter>
```

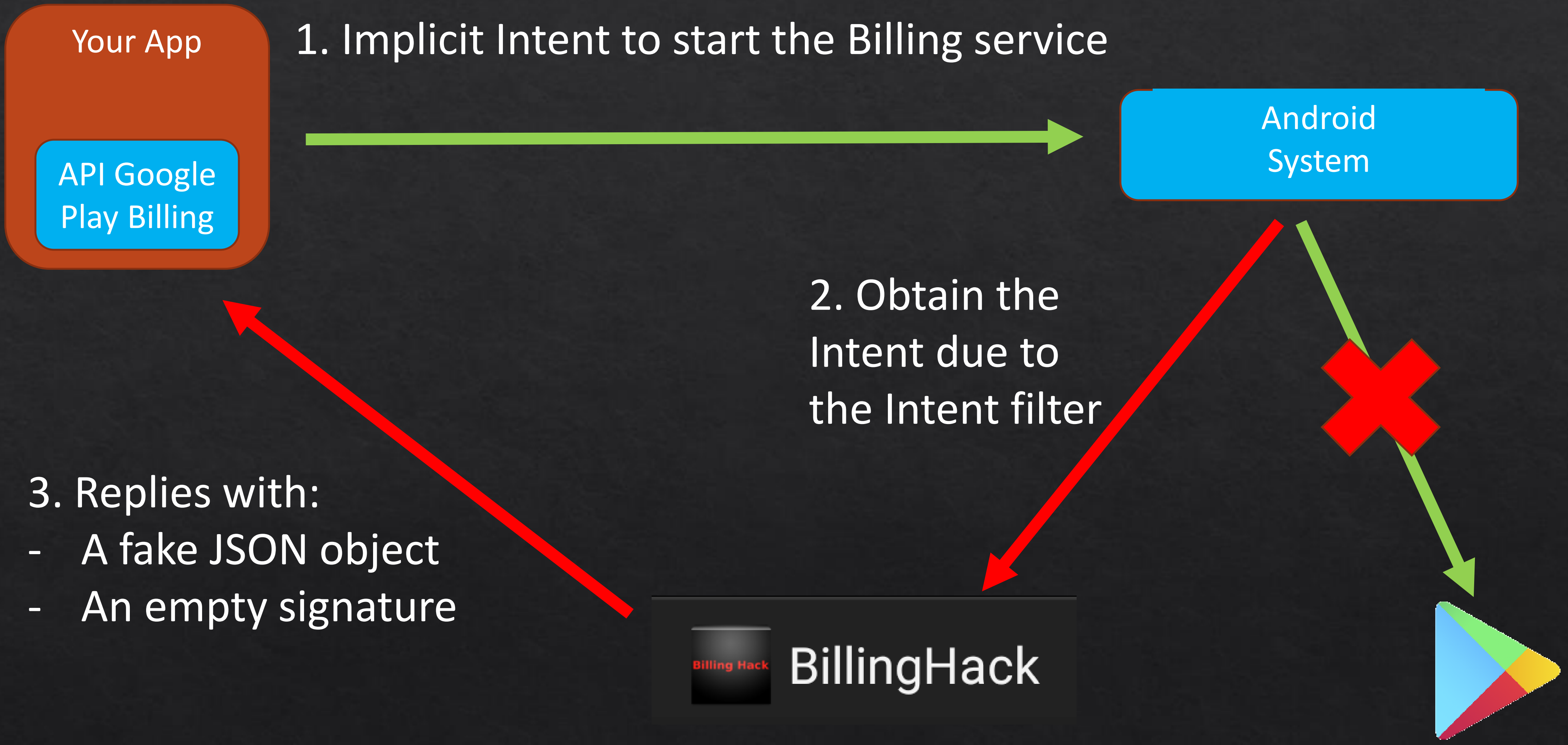
- Note
  - An implicit Intent is sent to initiate the Billing service
  - Android decides which component should receive the intent based on the intent information

Known Vulnerabilities

2. If the signature is an empty string, the validation is true

```
public static boolean verifyPurchase(String base64PublicKey, String signedData, String signature) {
        if (signedData == null) {
            Log.e(TAG, "data is null");
            return false;
        }
        boolean verified = false;
        if (!TextUtils.isEmpty(signature)) {
            PublicKey key = Security.generatePublicKey(base64PublicKey);
            verified = Security.verify(key, signedData, signature);
            if (!verified) {
                Log.w(TAG, "signature does not match data.");
                return false;
            }
        }
        return true;
}
```

Known Vulnerabilities



1. Implicit Intent to start the Billing service

2. Obtain the Intent due to the Intent filter

3. Replies with:
- A fake JSON object
- An empty signature

Known Vulnerabilities

- Google fixed these 2 vulnerabilities by applying the following modifications

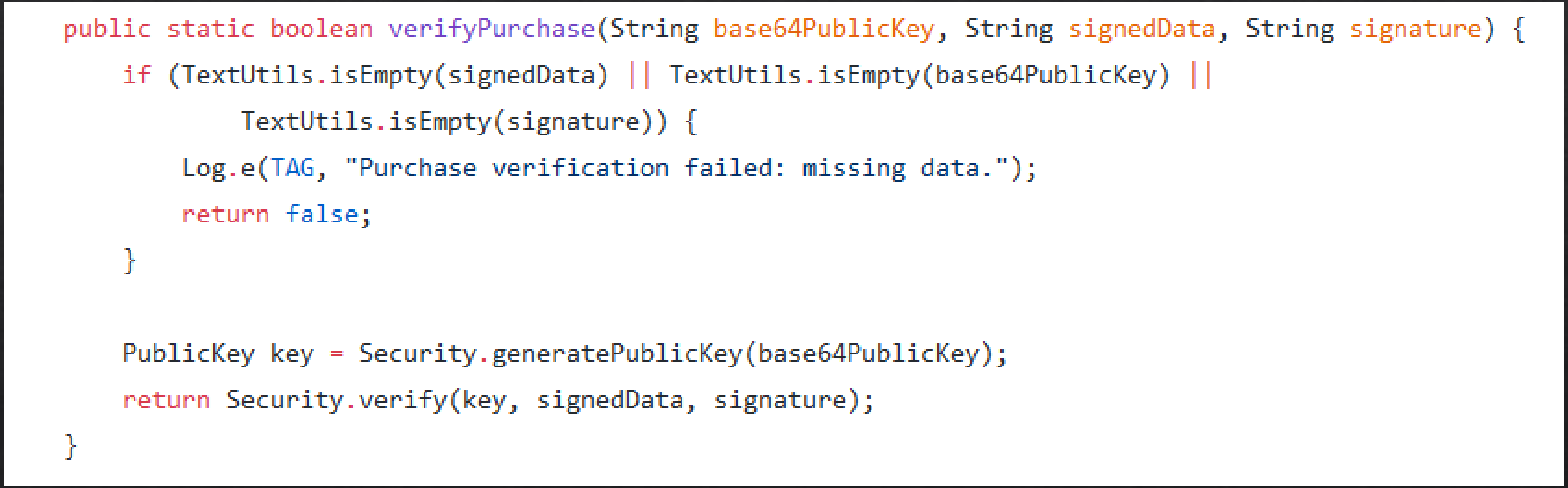
  1. Every app using the Google Play Billing API should define which is the targeted package for the intent

```
Intent intent = new Intent("com.android.vending.billing.InAppBillingService.BIND");
intent.setPackage(« com.android.vending");
```

  - If an explicit Intent is not defined, Google won't allow the app to be published on the Play Store

Known Vulnerabilities

2. The function checking the signature was modified in order to return true only if the signature is valid



```
public static boolean verifyPurchase(String base64PublicKey, String signedData, String signature) {
    if (TextUtils.isEmpty(signedData) || TextUtils.isEmpty(base64PublicKey) ||
            TextUtils.isEmpty(signature)) {
        Log.e(TAG, "Purchase verification failed: missing data.");
        return false;
    }

    PublicKey key = Security.generatePublicKey(base64PublicKey);
    return Security.verify(key, signedData, signature);
}
```

Source: Trivial Drive v2

DEEPSEC

Known Vulnerabilities

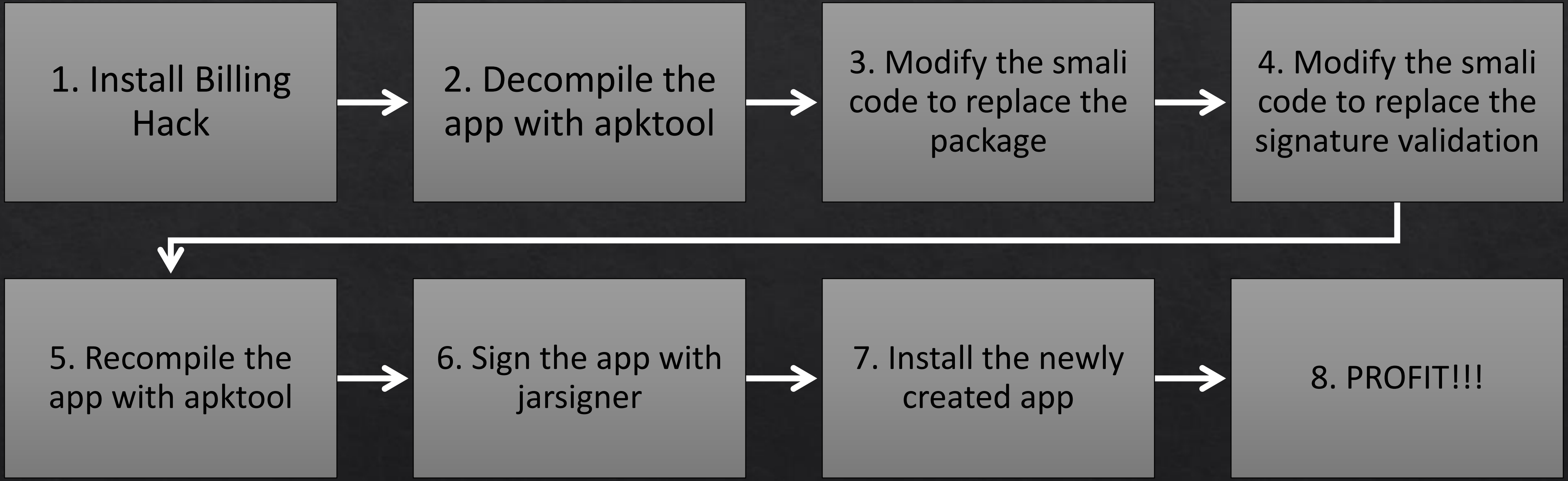
Known Vulnerabilities

- If your app is performing the verification process locally, you can always circumvent the payment by
  1. Binding the Billing service to an app you control
  2. Modify the signature verification in order to return always true

- The main “problem” is to find how the app is performing the signature verification!

Known Vulnerabilities

- Hacking Steps


1. Install Billing Hack
2. Decompile the app with apktool
3. Modify the smali code to replace the package
4. Modify the smali code to replace the signature validation
5. Recompile the app with apktool
6. Sign the app with jarsigner
7. Install the newly created app
8. PROFIT!!!

Known Vulnerabilities

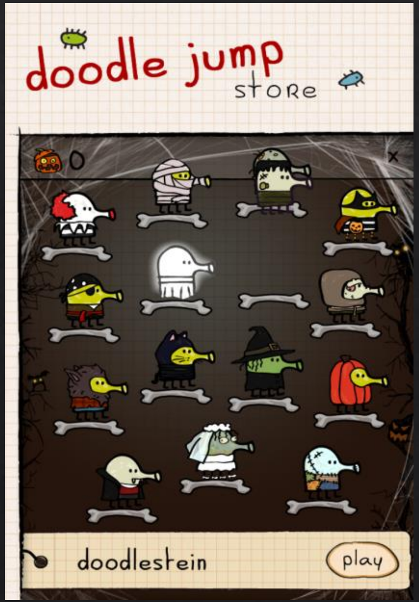
- Some tools allows you to perform this process automatically
  - LuckyPatcher is a tool allowing to modify an Android app in order to bypass the payment
  - This app uses the PoC provided by Dominik

Vulnerable Applications

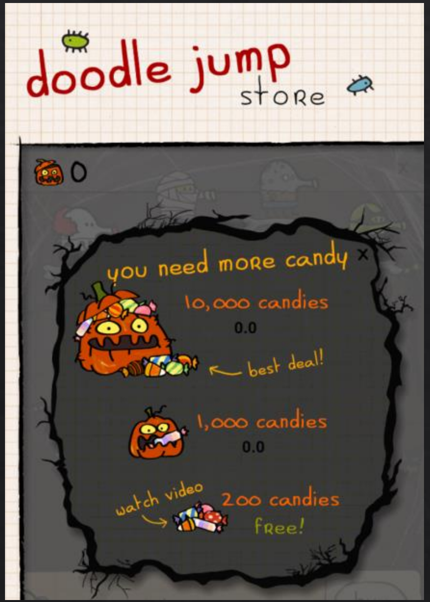
How to obtain unlimited credits?

Vulnerable Applications

- Doodle Jump (com.lima.doodlejump)
  - Platform game ("How high can you get?")
  - "Named Best of 2015 by Google Play editors"
  - Buy different items, but you need candies!

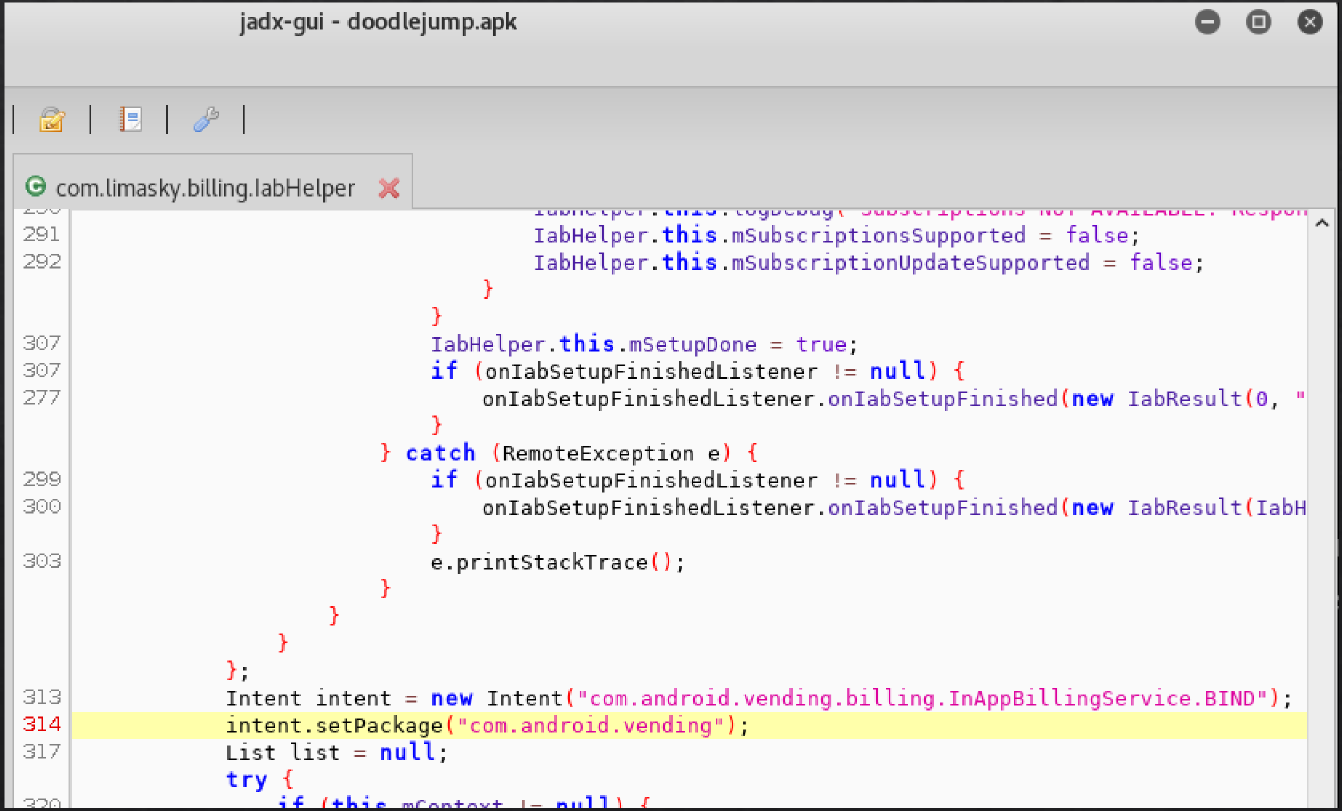


DEEPSEC



Vulnerable Applications

- Very easy to modify in order to buy items for free!
  - Replace “com.android.vending” by “org.billinghack”



jadx-gui - doodlejump.apk

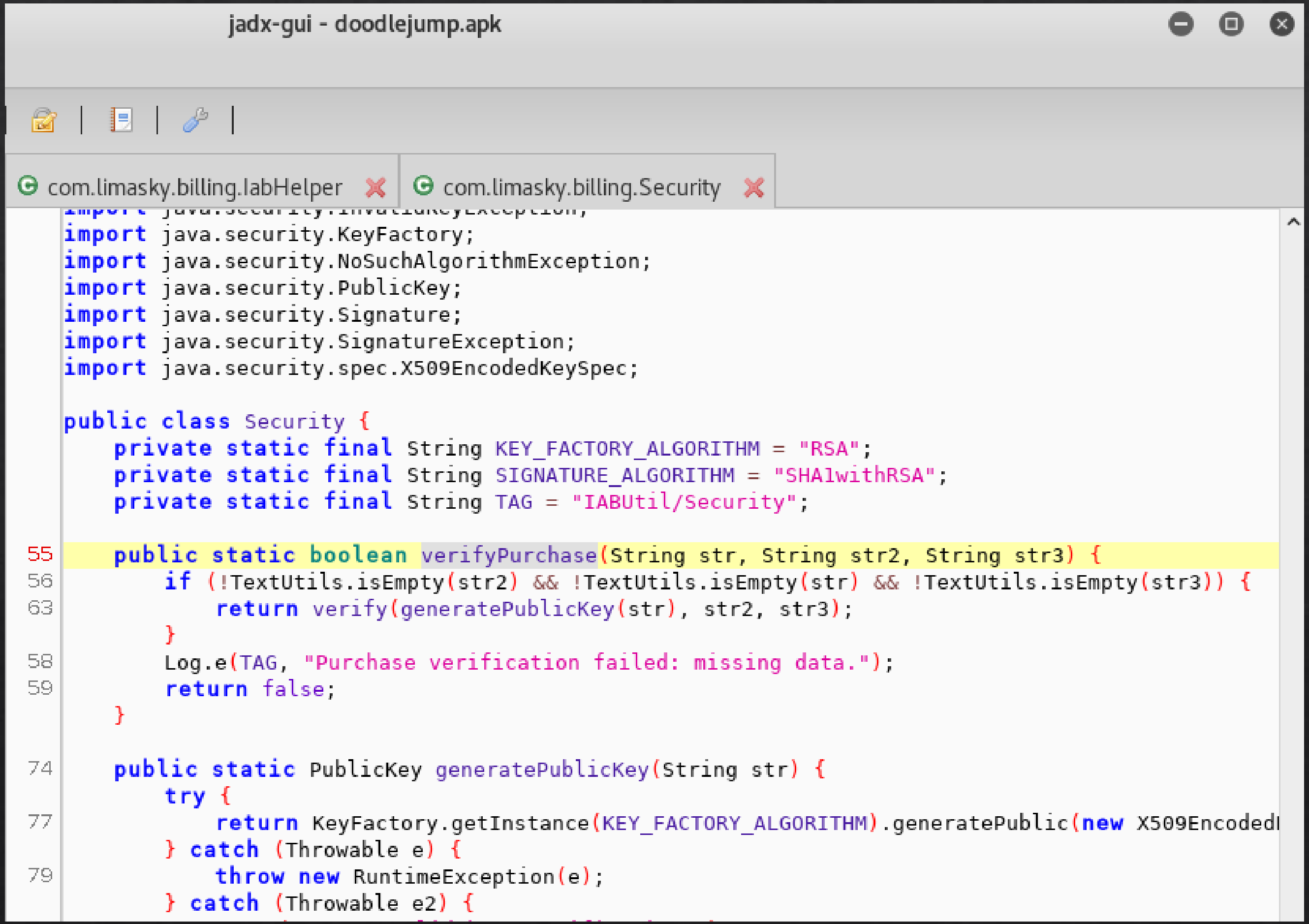
com.limasky.billing.IabHelper

```
                        IabHelper.this.logDebug("Subscriptions NOT AVAILABLE. Respon
                        IabHelper.this.mSubscriptionsSupported = false;
                        IabHelper.this.mSubscriptionUpdateSupported = false;
                    }
                }
                IabHelper.this.mSetupDone = true;
                if (onIabSetupFinishedListener != null) {
                    onIabSetupFinishedListener.onIabSetupFinished(new IabResult(0, "
                }
            } catch (RemoteException e) {
                if (onIabSetupFinishedListener != null) {
                    onIabSetupFinishedListener.onIabSetupFinished(new IabResult(IabH
                }
                e.printStackTrace();
            }
        }
    }
};
Intent intent = new Intent("com.android.vending.billing.InAppBillingService.BIND");
intent.setPackage("com.android.vending");
List list = null;
try {
    if (this.mContext != null) {
```

DEEPSEC

Vulnerable Applications

- Then, modify the “verifyPurchase” function to return true



jadx-gui - doodlejump.apk

com.limasky.billing.labHelper | com.limasky.billing.Security

```
import java.security.InvalidKeyException;
import java.security.KeyFactory;
import java.security.NoSuchAlgorithmException;
import java.security.PublicKey;
import java.security.Signature;
import java.security.SignatureException;
import java.security.spec.X509EncodedKeySpec;

public class Security {
    private static final String KEY_FACTORY_ALGORITHM = "RSA";
    private static final String SIGNATURE_ALGORITHM = "SHA1withRSA";
    private static final String TAG = "IABUtil/Security";

    public static boolean verifyPurchase(String str, String str2, String str3) {
        if (!TextUtils.isEmpty(str2) && !TextUtils.isEmpty(str) && !TextUtils.isEmpty(str3)) {
            return verify(generatePublicKey(str), str2, str3);
        }
        Log.e(TAG, "Purchase verification failed: missing data.");
        return false;
    }

    public static PublicKey generatePublicKey(String str) {
        try {
            return KeyFactory.getInstance(KEY_FACTORY_ALGORITHM).generatePublic(new X509Encoded
        } catch (Throwable e) {
            throw new RuntimeException(e);
        } catch (Throwable e2) {
```

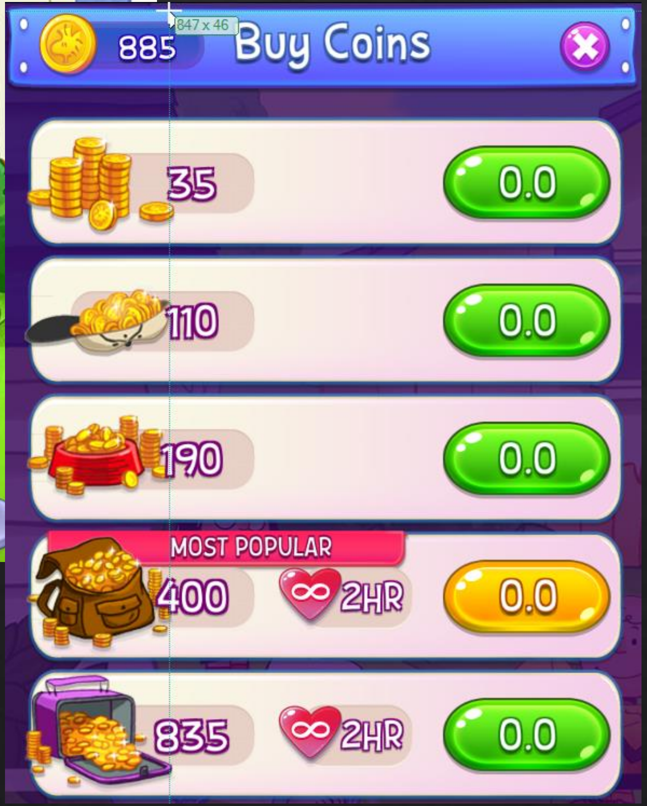
DEEPSEC

Vulnerable Applications

DEMO

Vulnerable Applications

- Snoopy Pop (com.jamcity.snoopypop)
  - Game similar to Bubble Witch but with Snoopy
  - You can buy coins and lives



Vulnerable Applications

- Unity library is used for the graphics
    - But Unity also offers a Google Play Billing interface
    - However Unity does not offer server-side validation

DEEPSEC

Vulnerable Applications

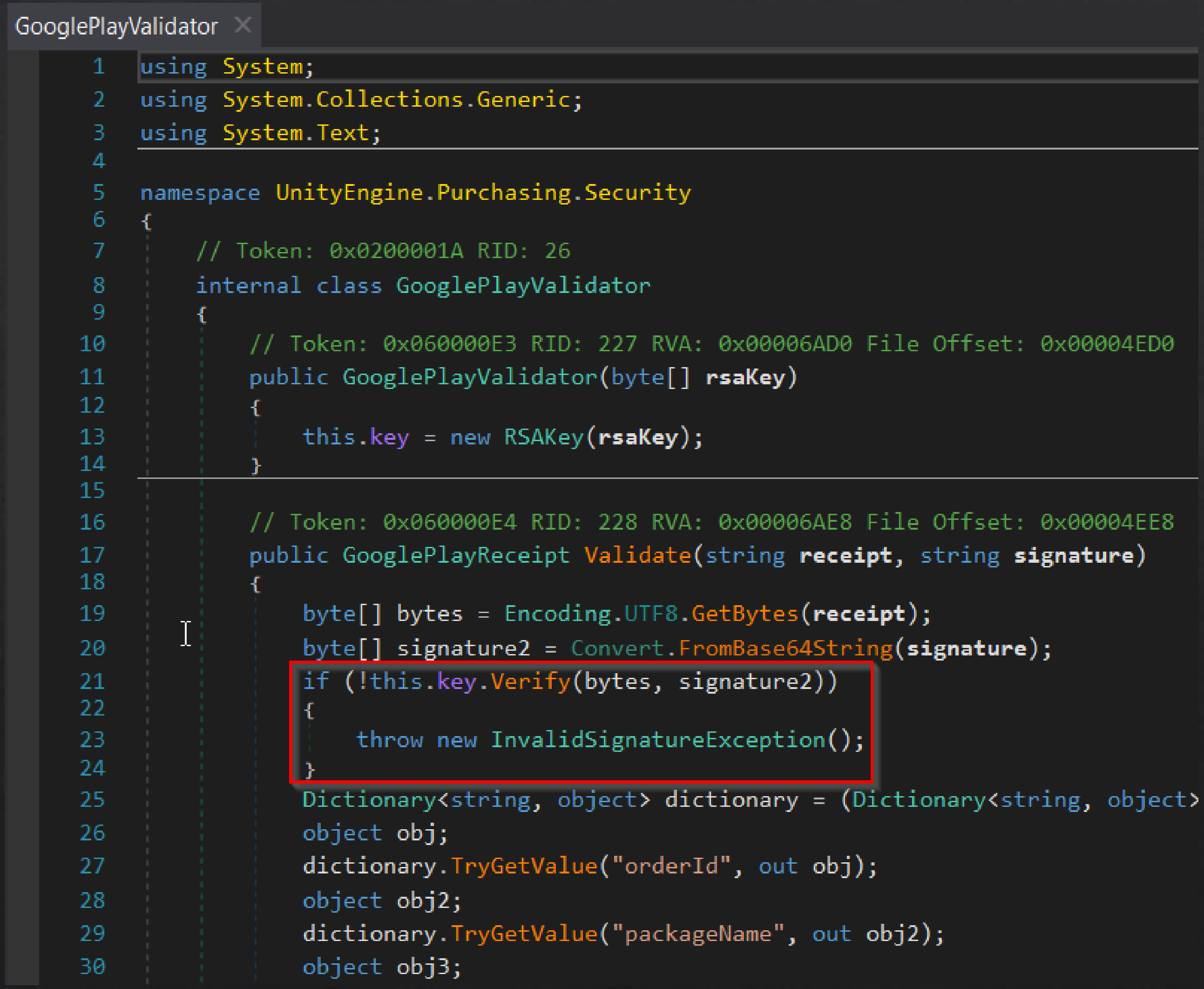
- Most of the Unity’s code is written in Mono .NET
  - These DLLs are stored on /assets/bin/Data/Managed

```
# ls assets/bin/Data/Managed/
Analytics.dll   Assembly-CSharp-firstpass.dll  Facebook.Unity.dll   mscorlib.dll  Stores.dll  System.Xml.dll
UnityEngine.Analytics.dll   UnityEngine.Purchasing.dll  winrt.dll    Apple.dll    Common.dll   Facebook.Unity.IOS.dll
P31RestKit.dll  System.Core.dll  System.Xml.Linq.dll  UnityEngine.dll    UnityEngine.UI.dll  Assembly-CSharp.dll
Facebook.Unity.Android.dll  Mono.Security.dll   Security.dll  System.dll   Tizen.dll UnityEngine.Networking.dll
Validator.dll
```

  - The most interesting one is Security.dll
    - This DDL contains a function called “Validate” which verify the signature of the purchase

Vulnerable Applications

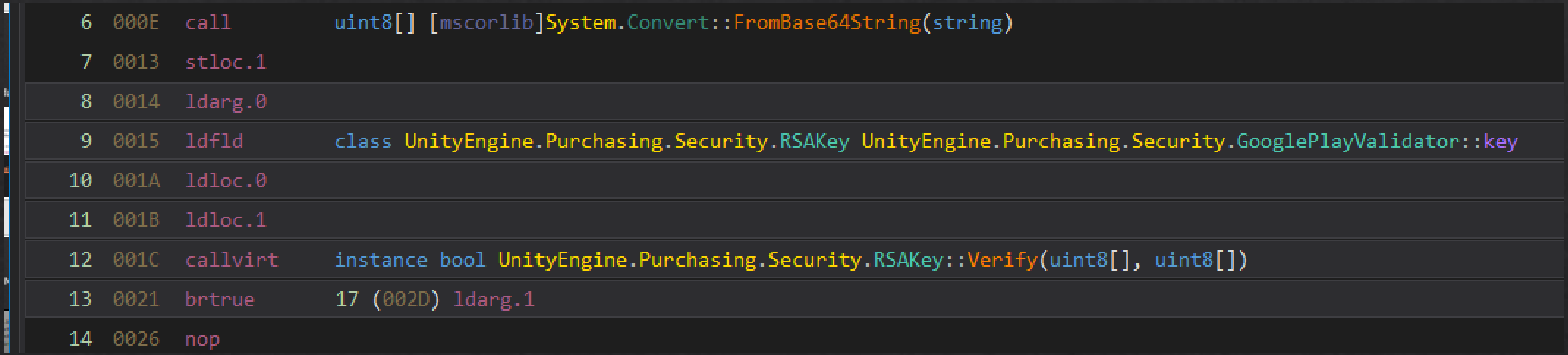
- With DnSpy, a .NET decompiler
  - It’s trivial to obtain and modify the .NET code
- The “Validate” function throws an exception when the signature is invalid



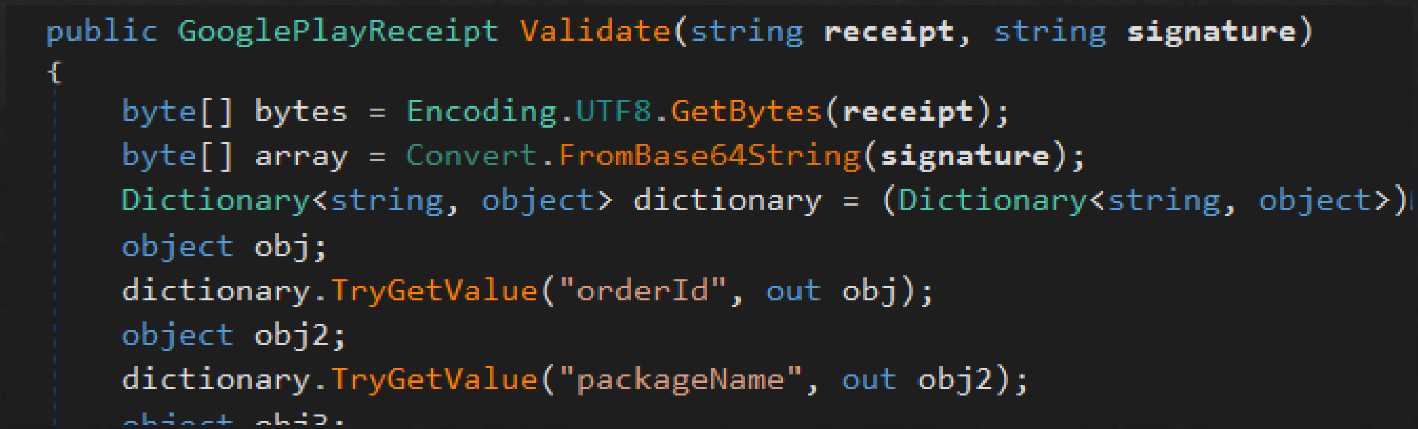
```
using System;
using System.Collections.Generic;
using System.Text;

namespace UnityEngine.Purchasing.Security
{
    // Token: 0x0200001A RID: 26
    internal class GooglePlayValidator
    {
        // Token: 0x060000E3 RID: 227 RVA: 0x00006AD0 File Offset: 0x00004ED0
        public GooglePlayValidator(byte[] rsaKey)
        {
            this.key = new RSAKey(rsaKey);
        }

        // Token: 0x060000E4 RID: 228 RVA: 0x00006AE8 File Offset: 0x00004EE8
        public GooglePlayReceipt Validate(string receipt, string signature)
        {
            byte[] bytes = Encoding.UTF8.GetBytes(receipt);
            byte[] signature2 = Convert.FromBase64String(signature);
            if (!this.key.Verify(bytes, signature2))
            {
                throw new InvalidSignatureException();
            }
            Dictionary<string, object> dictionary = (Dictionary<string, object>
            object obj;
            dictionary.TryGetValue("orderId", out obj);
            object obj2;
            dictionary.TryGetValue("packageName", out obj2);
            object obj3;
```

Vulnerable Applications

- So, let’s just remove the code performing the check



```
000E  call      uint8[] [mscorlib]System.Convert::FromBase64String(string)
0013  stloc.1
0014  ldarg.0
0015  ldfld     class UnityEngine.Purchasing.Security.RSAKey UnityEngine.Purchasing.Security.GooglePlayValidator::key
001A  ldloc.0
001B  ldloc.1
001C  callvirt  instance bool UnityEngine.Purchasing.Security.RSAKey::Verify(uint8[], uint8[])
0021  brtrue    17 (002D) ldarg.1
0026  nop
```



```
public GooglePlayReceipt Validate(string receipt, string signature)
{
    byte[] bytes = Encoding.UTF8.GetBytes(receipt);
    byte[] array = Convert.FromBase64String(signature);
    Dictionary<string, object> dictionary = (Dictionary<string, object>)
    object obj;
    dictionary.TryGetValue("orderId", out obj);
    object obj2;
    dictionary.TryGetValue("packageName", out obj2);
    object obj3;
```

DEEPSEC

Vulnerable Applications

- At the end, we replace your modified DLL in the app
  - Don’t forget to replace with your package
  - Rebuild with apktool
  - And PROFIT!

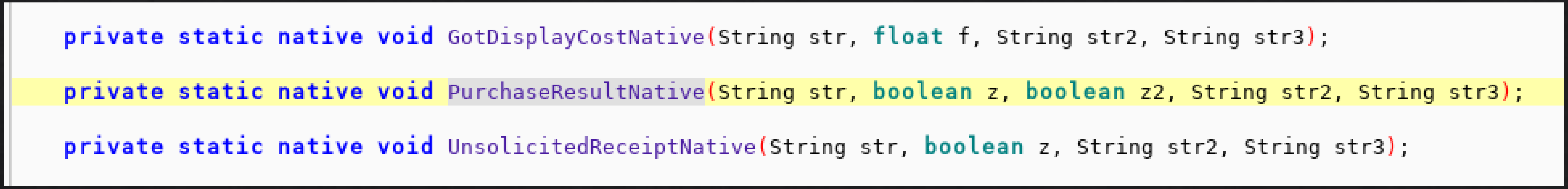
- DEMO

Vulnerable Applications

- Fruit Ninja (com.halfbrick.fruitninjafree)
  - Famous game where you need to cut fruits (like a ninja!)
  - More than 100 millions of downloads

Vulnerable Applications

- Java Native Interface (JNI)
  - JNI allows to interact with native code (C/C++) from Java/Kotlin
  - In short, you can embed a shared library and your app can call functions from this library
- FruitNinja implements sensitive functions using JNI
  - And mostly for Google Play Billing functions

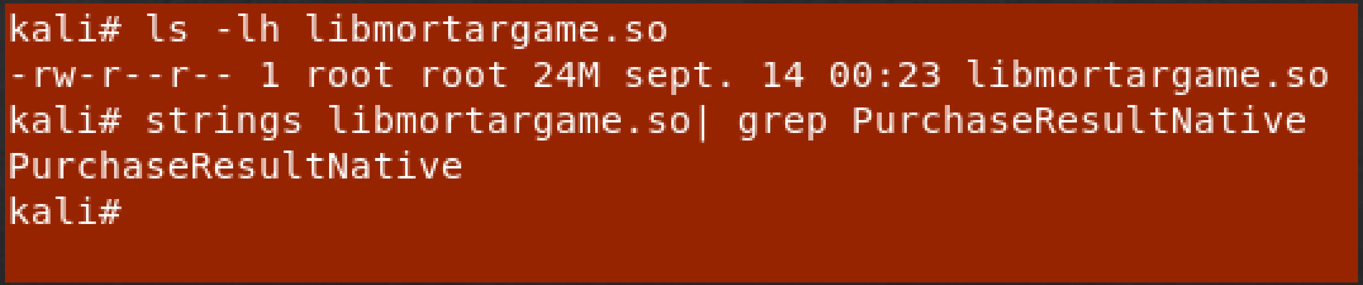


```
private static native void GotDisplayCostNative(String str, float f, String str2, String str3);
private static native void PurchaseResultNative(String str, boolean z, boolean z2, String str2, String str3);
private static native void UnsolicitedReceiptNative(String str, boolean z, String str2, String str3);
```

DEEPSEC

Vulnerable Applications

- Need to reverse engineer the shared library



```
kali# ls -lh libmortargame.so
-rw-r--r-- 1 root root 24M sept. 14 00:23 libmortargame.so
kali# strings libmortargame.so| grep PurchaseResultNative
PurchaseResultNative
kali#
```

- Shared library coded in C++
  - Time consuming!
  - Difficult to rebuild a new shared library

Vulnerable Applications

- However, it seems that the signature validation is poorly made
  - So it’s possible to bypass the payment

- DEMO

Vulnerable Applications

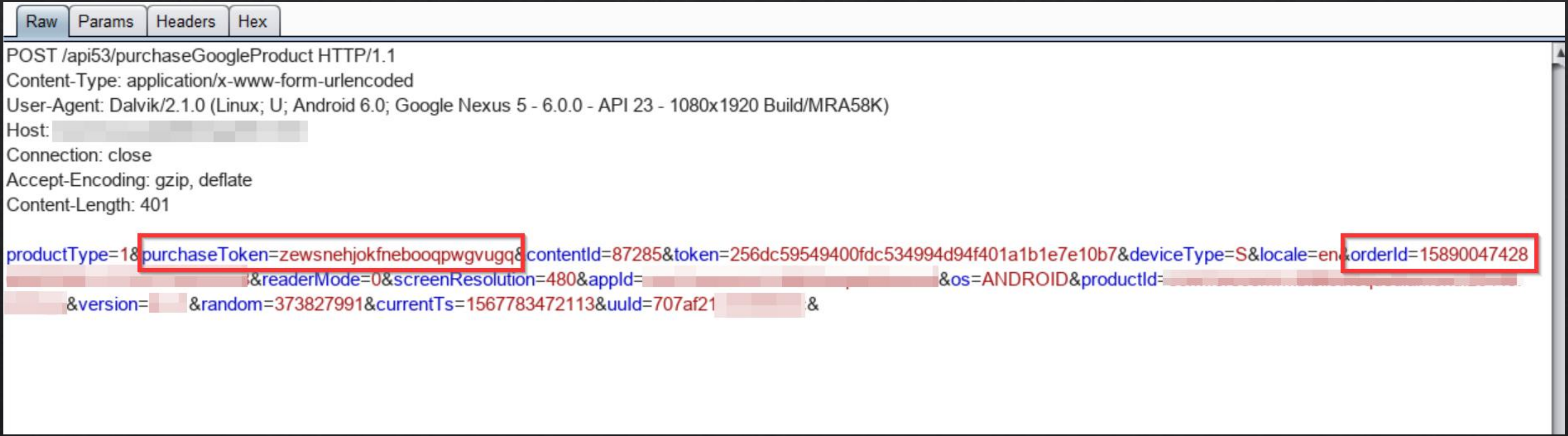
- Anonymous app
  - Magazine reader
  - You can buy magazines or get a subscription
- Purchase process
  1. The user selects a magazine
  2. The Google Play Billing allows to buy it
  3. After payment validation, the app downloads the PDF of your magazine

Vulnerable Applications

- In the same manner as Doodle Jump or Snoopy Pop
  - The signature is validated locally
- It is possible to bypass the purchase validation as shown before
- However, an addition verification step was added

Vulnerable Applications

- The *purchaseToken* and *orderId* are validated on a server



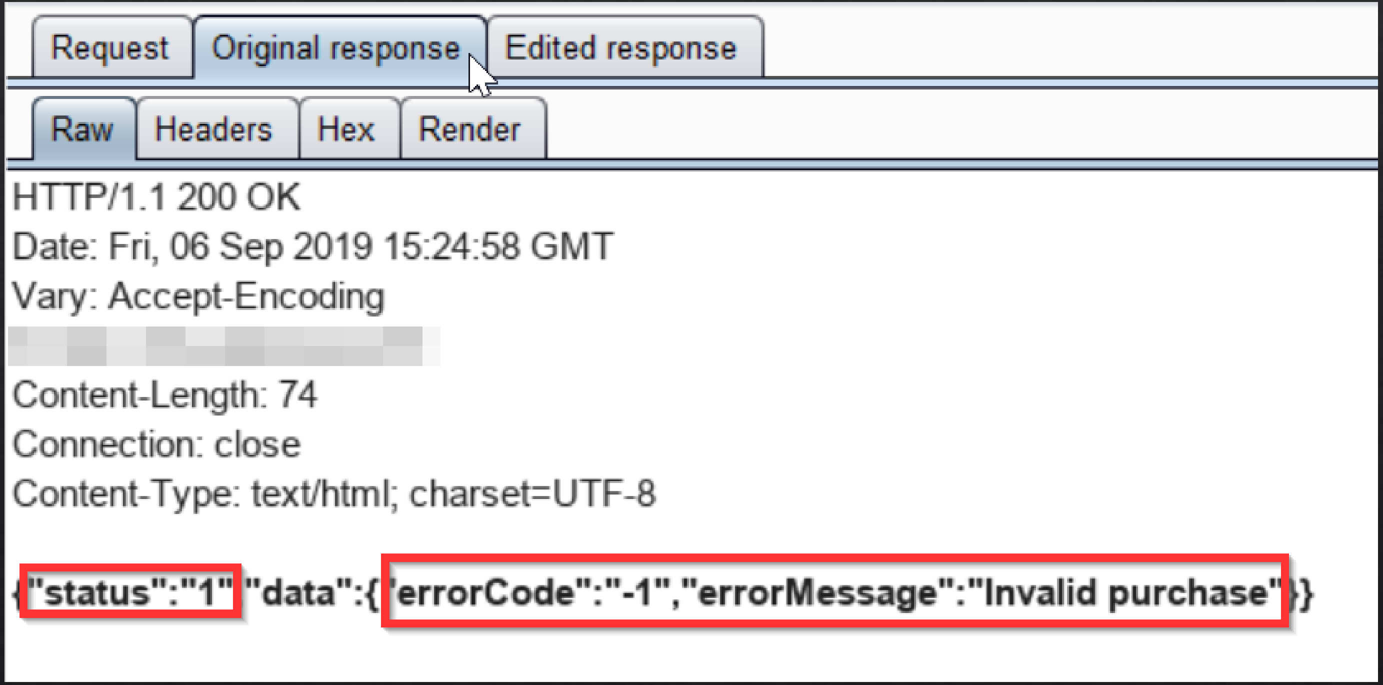
Raw | Params | Headers | Hex

```
POST /api53/purchaseGoogleProduct HTTP/1.1
Content-Type: application/x-www-form-urlencoded
User-Agent: Dalvik/2.1.0 (Linux; U; Android 6.0; Google Nexus 5 - 6.0.0 - API 23 - 1080x1920 Build/MRA58K)
Host: [redacted]
Connection: close
Accept-Encoding: gzip, deflate
Content-Length: 401

productType=1&purchaseToken=zewsnehjokfnebooqpwgvugq&contentId=87285&token=256dc59549400fdc534994d94f401a1b1e7e10b7&deviceType=S&locale=en&orderId=15890047428
[redacted]&readerMode=0&screenResolution=480&appId=[redacted]&os=ANDROID&productId=[redacted]
[redacted]&version=[redacted]&random=373827991&currentTs=1567783472113&uuId=707af21[redacted]&
```

DEEPSEC

Vulnerable Applications

- When *purchaseToken* and *orderId are not valid, the* server replies with an error code

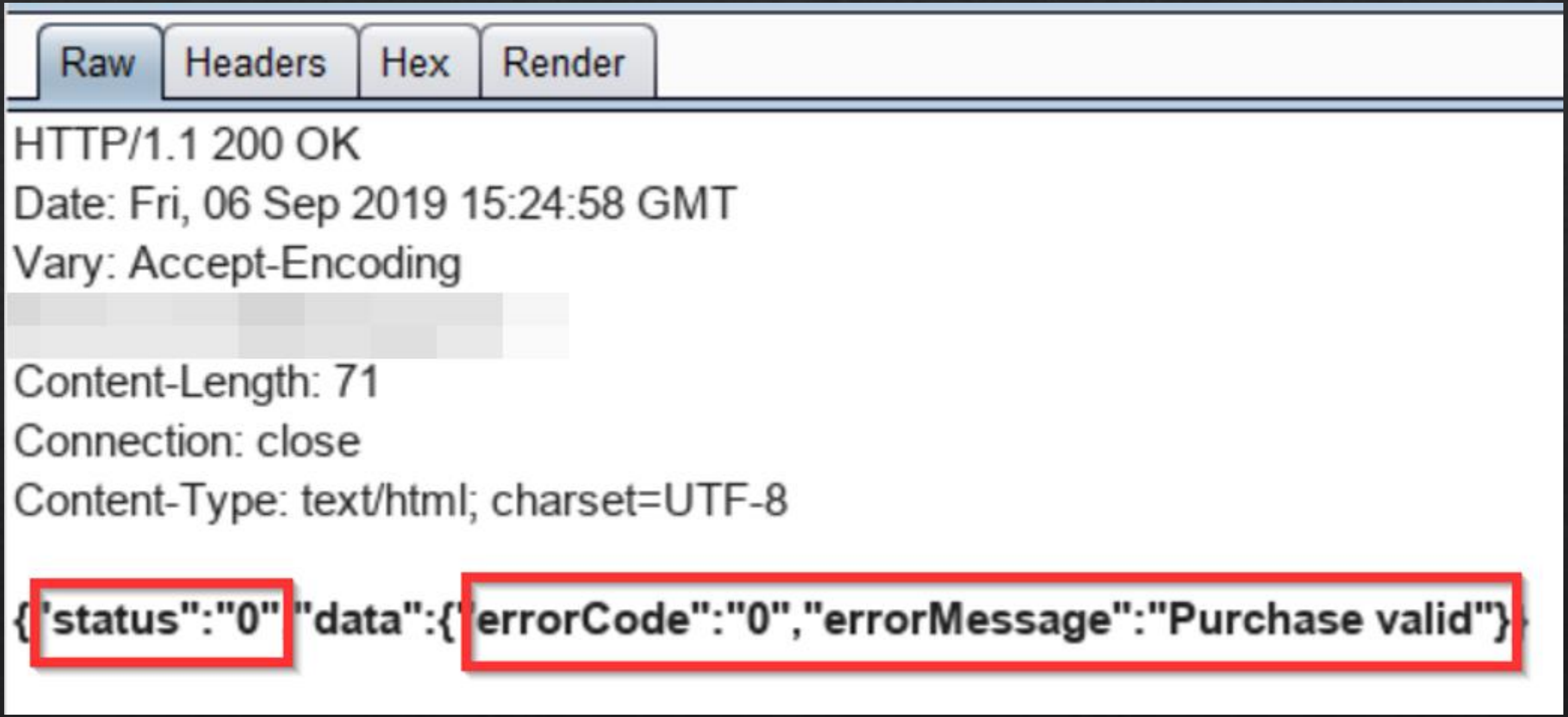


```
HTTP/1.1 200 OK
Date: Fri, 06 Sep 2019 15:24:58 GMT
Vary: Accept-Encoding
[redacted]
Content-Length: 74
Connection: close
Content-Type: text/html; charset=UTF-8

{"status":"1","data":{"errorCode":"-1","errorMessage":"Invalid purchase"}}
```

DEEPSEC

Vulnerable Applications

- So what happens, when we change the status code?



```
HTTP/1.1 200 OK
Date: Fri, 06 Sep 2019 15:24:58 GMT
Vary: Accept-Encoding
[redacted]
Content-Length: 71
Connection: close
Content-Type: text/html; charset=UTF-8

{"status":"0","data":{"errorCode":"0","errorMessage":"Purchase valid"}}
```

Vulnerable Applications

- The app downloads the PDF without any issue!

DEEPSEC

Conclusion

That’s it?

DEEPSEC

Conclusion

- Developers use different techniques to protect the Google Play Billing payment
  - Obfuscation
  - Shared library
  - Nothing!
- However, the signature validation is mainly performed locally inside the app

Conclusion

- On 50 apps tested
  - 29 apps were vulnerable (bypass payment)
  - Only 5 apps used an external endpoint to perform additional checks
- I contacted the developers, but I only get an answer from the magazine app
  - The issues are still present

Conclusion

- Regarding other Billing libraries, Google is the only one allowing local validation
  - Amazon IAP (In-App Purchase) and Samsung In-App Purchase enforces to use a server in order to retrieve the content

Thanks for the support!

Questions?

DEEPSEC

References 1/4

- Google Play Billing documentation
  - https://developer.android.com/google/play/billing/billing_overview
- Google Play Billing Best Practices
  - https://developer.android.com/google/play/billing/billing_best_practices.html
- Google Play In-App Billing Library Hacked
  - https://www.schuermann.eu/2013/10/29/google-play-billing-hacked.html

References 2/4

- Google Play Billing – Verify purchase
  - https://developer.android.com/google/play/billing/billing_library_overview#Verify
- Prime 31 documentation – Validate purchase
  - https://prime31.com/docs#androidIAB
- Unity documentation – Validate Purchase
  - https://docs.unity3d.com/Manual/UnityIAPValidatingReceipts.html

References 3/4

- Billing Hack Source Code
  - https://github.com/dschuermann/billing-hack
- Google prevents vulnerable apps on the Play Store
  - https://support.google.com/faqs/answer/7054270?hl=en
- Amazon documentation
  - https://developer.amazon.com/fr/docs/in-app-purchasing/iap-rvs-for-android-apps.html
- Samsung documentation
  - https://developer.samsung.com/iap#overview

References 4/4

- Get Freebies by Abusing the Android InApp Billing API
  - https://www.checkmarx.com/blog/abusing-android-inapp-billing-api/
- Abusing Android In-app Billing feature thanks to a misunderstood integration
  - https://www.securingapps.com/blog/BsidesLisbon17_AbusingAndroidInappBilling.pdf