

乱取り

# Contournement de l'API Google Play Billing

Guillaume Lopes - @Guillaume\_Lopes



# Guillaume Lopes

- ▶ Consultant Sécurité chez **RandoriSec**
- ▶ +10 ans d'expérience dans le domaine du test d'intrusion et de l'audit
  - ▶ Spécialités : Windows/Active Directory, Applications Web et Android
- ▶ Membre de l'équipe de recherche de Checkmarx
  - ▶ <https://www.checkmarx.com/category/blog/technical-blog/>
- ▶ Joueur de CTF (Hackthebox, Insomni'hack, Nuit du Hack, etc.)
  - ▶ Donne un coup main à l'équipe Tipi'hack

# Agenda

- 1) Présentation de l'API Google Play Billing
- 2) Historique sur les vulnérabilités connues
- 3) Exemples d'applications vulnérables
- 4) Conclusion

1

# Google Play Billing

Présentation

# ► Présentation

- ▶ **Google Play Billing** (ou anciennement Google InApp Billing)
  - ▶ Service permettant aux développeurs de vendre des produits au sein de leur application mobile
- ▶ Cette API permet de facilement vendre des produits au sein de son application
- ▶ En résumé, 2 grandes catégories de produits peuvent être vendus
  - ▶ Les « one-time products »
  - ▶ Les « subscriptions »

# ► Présentation

- ▶ ***One-time products***

- ▶ Produits nécessitant un paiement en une seule fois
- ▶ Exemple : achat d'une version premium ou de vies supplémentaires dans un jeu

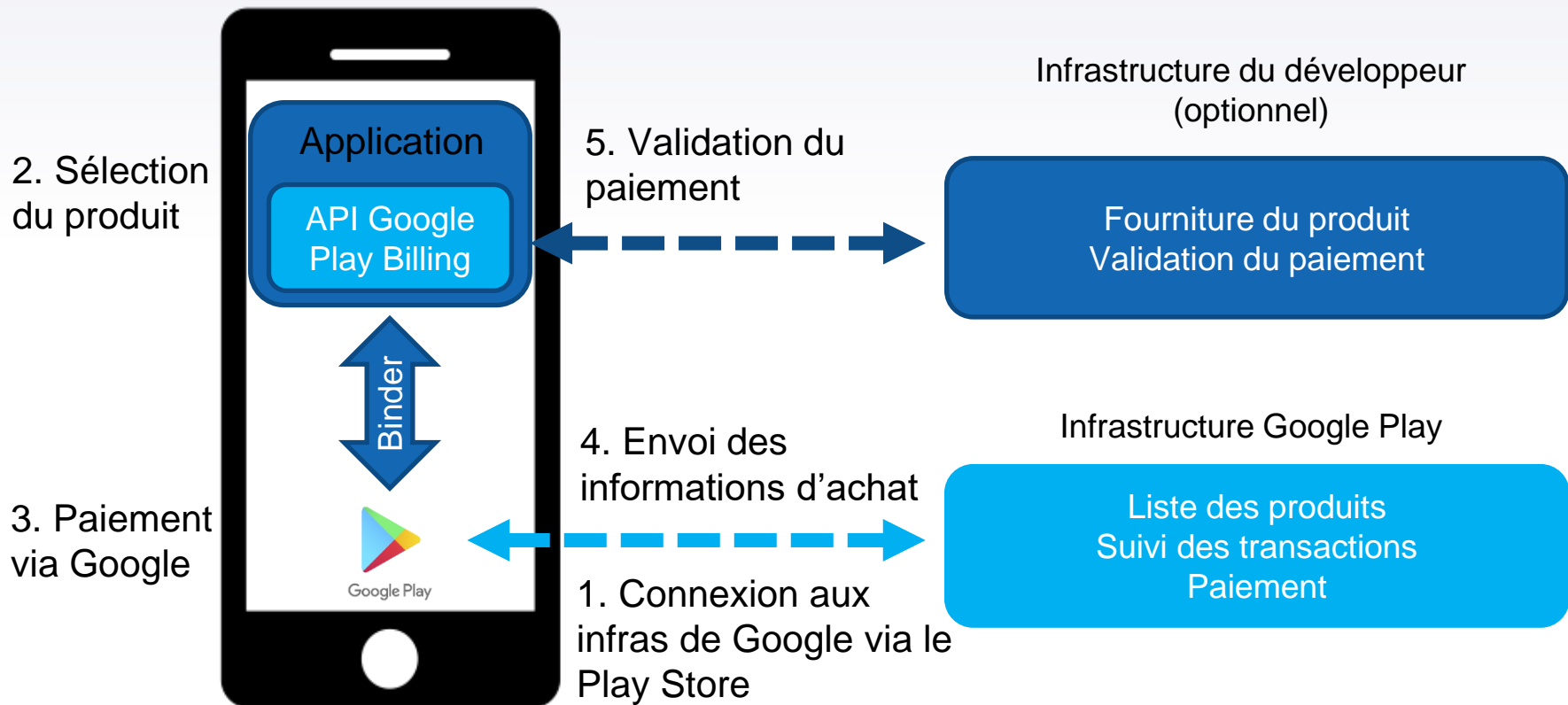
- ▶ ***Subscriptions***

- ▶ Il s'agit d'un abonnement qui va nécessiter un paiement régulier (hebdomadaire, mensuel, etc.)
- ▶ Exemple : abonnement à un journal

# ► Présentation

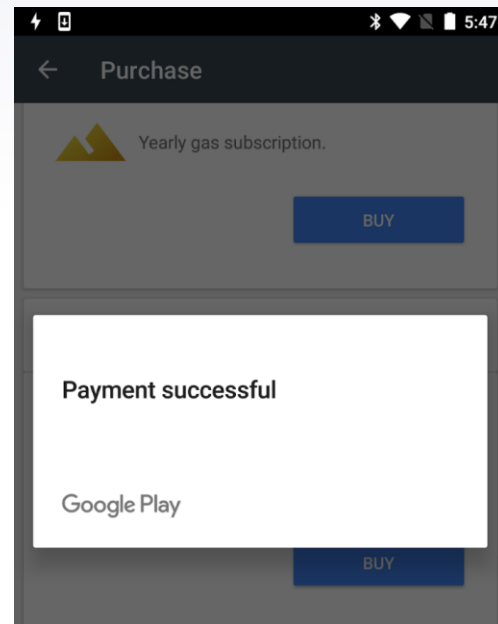
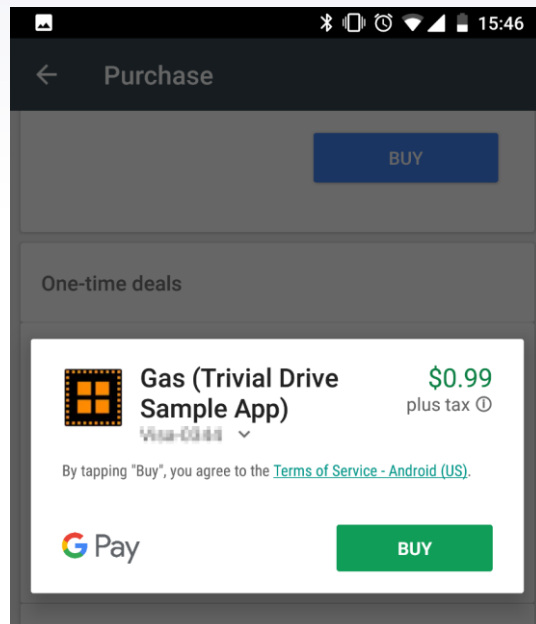
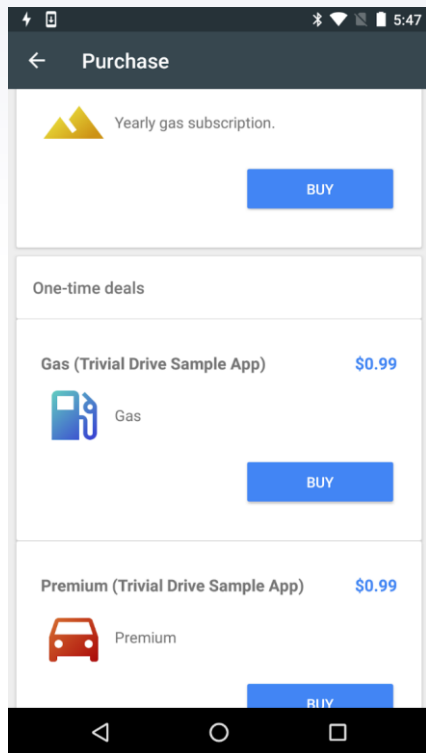
- ▶ Principal avantage du Google Play Billing
  - ▶ Le paiement est complètement géré par Google
  - ▶ Le développeur n'a pas besoin de gérer des cartes de paiements ou des données bancaires
- ▶ L'ensemble des produits vendus par l'application doit être renseigné dans la console Google Play

# Présentation





# Présentation



# ► Présentation

- ▶ Validation du paiement peut être réalisée de 2 façons
  - ▶ Soit validation via un serveur externe
  - ▶ Soit validation localement au sein de l'application
- ▶ Dans tous les cas, Google renvoie un objet JSON contenant les informations de l'achat effectué par l'utilisateur
  - ▶ INAPP\_PURCHASE\_DATA
- ▶ **Google recommande de valider le paiement via un serveur externe**

# Présentation

- ▶ Suite au paiement, Google renvoie un objet JSON contenant :
  - ▶ **orderId** : identifiant unique de la transaction
  - ▶ **packageName** : nom de l'application
  - ▶ **productId** : identifiant du produit
  - ▶ **purchaseState** : entier indiquant l'état du paiement
    - ▶ 0 (paiement réalisé) ou 1 (paiement annulé)
  - ▶ **purchaseToken** : un jeton à usage unique permettant d'identifier un achat

# Présentation

- ▶ Exemple

```
{  
  "orderId":"6395511243536442409.3765700896946863",  
  "packageName":"com.lima.doodlejump",  
  "productId":"doodlejump.candys1000",  
  "purchaseTime":1574248391379,  
  "purchaseState":0,  
  "developerPayload":"","  
  "purchaseToken":"ijkgeytwuimhgxedsloanbnc"  
}
```

# ► Présentation

- ▶ Lorsque l'application est publiée sur le Play Store
  - ▶ Une paire de clés RSA est générée par Google
  - ▶ L'objet JSON est signé par Google afin de garantir son authenticité en utilisant la clé privée
  - ▶ Le développeur peut valider la signature en utilisant la clé publique (qui doit être incluse dans l'application)

# Présentation

- ▶ Comme évoqué, Google recommande de valider cette signature via un serveur externe

★ **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.

- ▶ Néanmoins, il est toujours possible de valider cette signature localement au sein de l'application

⚠ **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.

# Présentation

- ▶ Google fournit une application de démonstration afin de tester l'API Google Play Billing
  - ▶ Trivial Drive v2

```
/**
 * 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";

    /**
     * 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.
     */

    /**
     * @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);
    }
}
```

# Présentation

- ▶ Et concernant les autres API ?
  - ▶ 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](#).



# ► Présentation

- ▶ Le design de l'API de Google permet de valider le paiement soit via une ressource externe ou localement
  - ▶ Dans la documentation de Google, la validation est effectuée localement
  - ▶ Des API externes ne prennent pas en charge la validation du paiement via un serveur externe
- ▶ Que pensez-vous que les développeurs vont utiliser ?

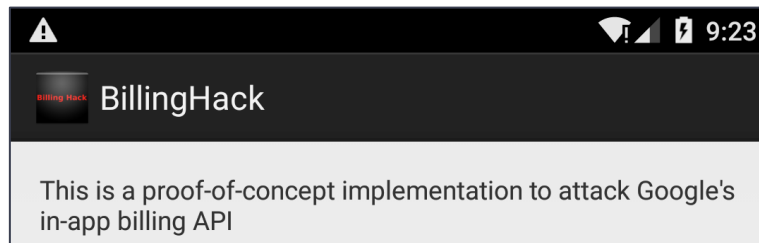
2

# Vulnérabilités connues

Un peu d'histoire

# ▶ Vulnérabilités connues

- ▶ En 2013, Dominik Schürmann a identifié 2 vulnérabilités au sein de l'API Google Play Billing
  - ▶ La combinaison de ces 2 vulnérabilités permettait de contourner le paiement
  - ▶ Les vulnérabilités ont été remontées à Google
- ▶ Une preuve de concept a été développée : **BillingHack**



# ► Vulnérabilités connues

1. Une application malveillante installée sur l'équipement peut usurper l'identité du service Google Play Billing (com.android.vending)
  - Il suffisait que l'application définisse un *Intent filter* avec une priorité élevée

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

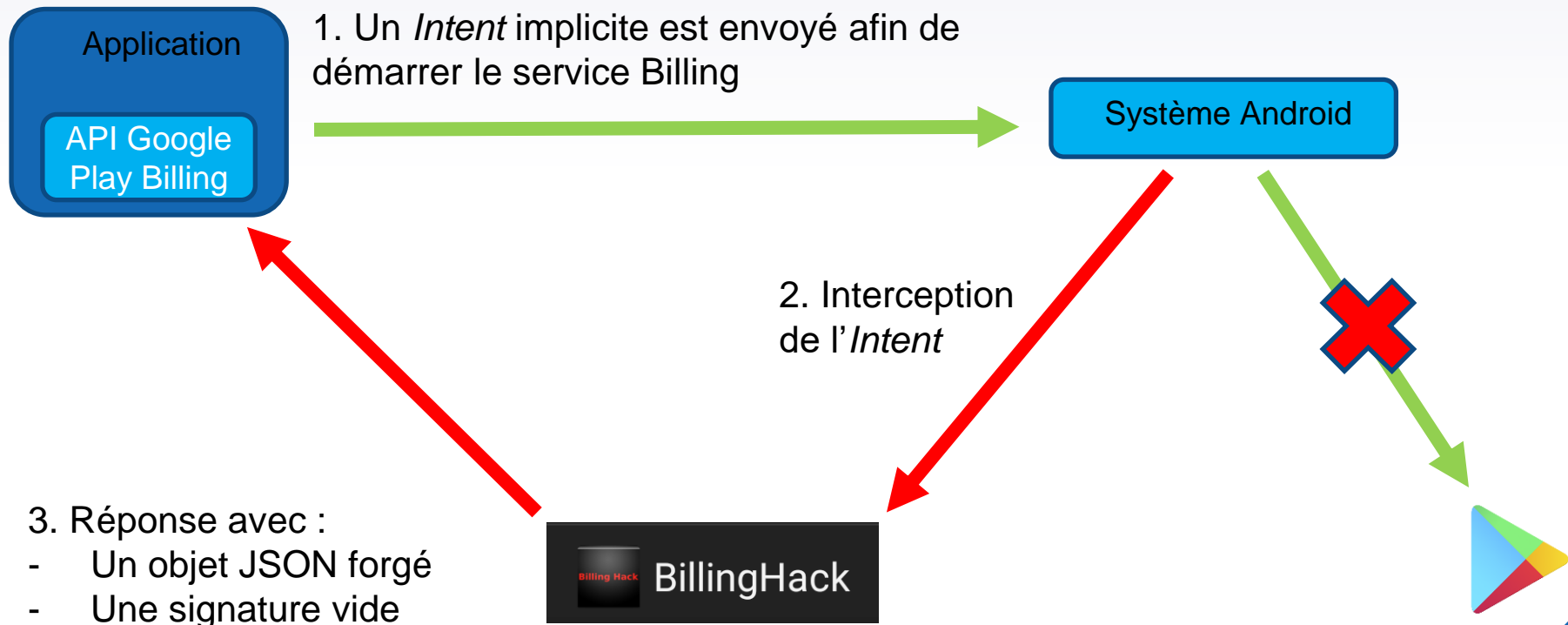
Sous Android, les *Intents* permettent aux applications de communiquer entre elles

# Vulnérabilités connues

2. Un bug dans la fonction de la vérification de la signature renvoyait vrai si la signature était une chaîne de caractères vide

```
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;  
}
```

# Vulnérabilités connues



# Vulnérabilités connues

- ▶ Google a corrigé les vulnérabilités de la façon suivante
  1. Chaque application doit définir un *Intent* explicite

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

Si un *Intent* implicite est utilisé, Google refusera de publier l'application sur le Play Store

# Vulnérabilités connues

2. La fonction validant la signature a été modifiée afin de retourner vrai uniquement si la signature est valide

```
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);  
}
```



# ► Vulnérabilités connues

- ▶ Néanmoins, si l'application effectue la validation du paiement localement, il est toujours possible de contourner le paiement
  1. Modifier l'application afin de communiquer avec notre application malveillante
  2. Modifier la fonction de validation de la signature afin de retourner vrai quoi qu'il arrive
- ▶ La seule « difficulté » est d'identifier la partie du code validant la signature

# ► Vulnérabilités connues

- ▶ Certaines applications permettent d'effectuer cette manipulation
  - ▶ **LuckyPatcher** est une application permettant de contourner le paiement, mais aussi de supprimer les publicités
  - ▶ LuckyPatcher utilise la preuve de concept fournie par Dominik Schurmann

3

# Applications vulnérables

Comment obtenir des crédits de manière illimitée

# Applications vulnérables

## ► Doodle Jump

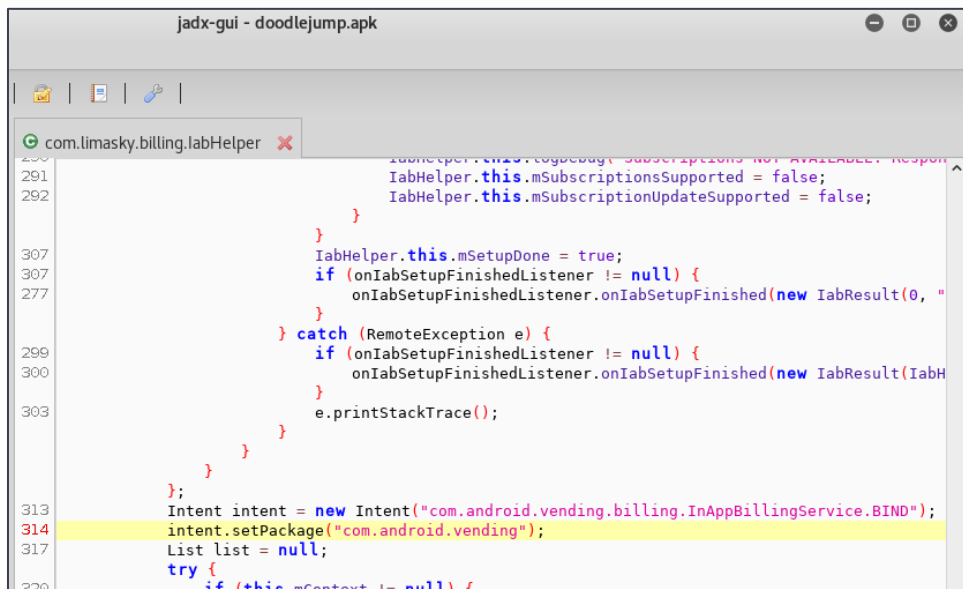
(com.lima.doodlejump)

- Jeu de plateforme
- Élu meilleur jeu en 2015 (Google Play editors)
- Achat de costumes dans le jeu, mais il faut des friandises



# Applications vulnérables

- ▶ La validation du paiement est réalisée localement
  - ▶ Il suffit de modifier l'*Intent* explicite afin d'utiliser BillingHack



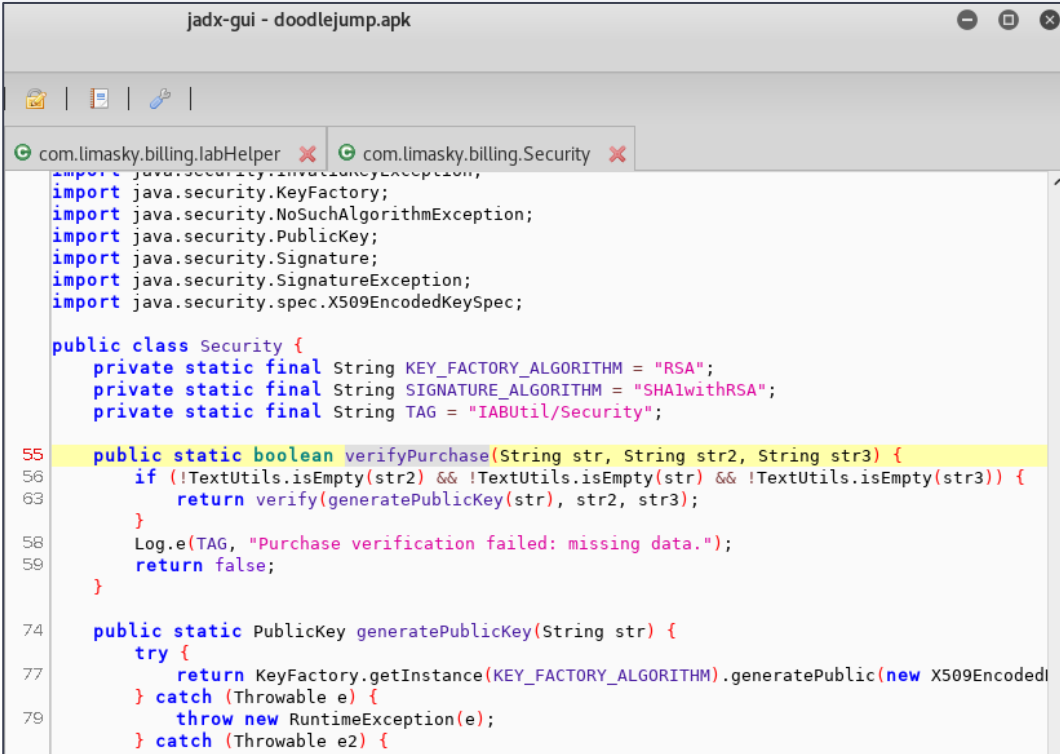
```
jadx-gui - doodlejump.apk

com.limasky.billing.IabHelper

291         IabHelper.this.mSubscriptionsSupported = false;
292         IabHelper.this.mSubscriptionUpdateSupported = false;
    }
    }
307     IabHelper.this.mSetupDone = true;
307     if (onIabSetupFinishedListener != null) {
277         onIabSetupFinishedListener.onIabSetupFinished(new IabResult(0, "
    }
} catch (RemoteException e) {
299     if (onIabSetupFinishedListener != null) {
300         onIabSetupFinishedListener.onIabSetupFinished(new IabResult(IabH
    }
303     e.printStackTrace();
    }
    }
    }
    };
313     Intent intent = new Intent("com.android.vending.billing.InAppBillingService.BIND");
314     intent.setPackage("com.android.vending");
317     List list = null;
    try {
        if (this.mContext != null) {
```

# Applications vulnérables

- ▶ Il suffit de modifier la fonction « verifyPurchase »



```
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";

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

74     public static PublicKey generatePublicKey(String str) {
    try {
77         return KeyFactory.getInstance(KEY_FACTORY_ALGORITHM).generatePublic(new X509EncodedKeySpec(str.getBytes()));
    } catch (Throwable e) {
79         throw new RuntimeException(e);
    } catch (Throwable e2) {
    }
```

# DEMO

# Applications vulnérables

- ▶ **Snoopy Pop**  
([com.jamcity.snooypop](http://com.jamcity.snooypop))
  - ▶ Jeu similaire à BubbleWitch
  - ▶ Achats de pièces ou de vies





# ► Applications vulnérables

- ▶ Ce jeu utilise la bibliothèque Unity pour les graphismes
  - ▶ Elle fournit également une API pour utiliser le Google Play Billing
- ▶ Dans cette version, la majorité du code d'Unity est développée en .NET

```
# 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
```

# Applications vulnérables

## Security.dll

- ▶ Contient une fonction « Validate »
- ▶ Génère une exception si la signature est invalide

Il est possible de décompiler et modifier cette DLL avec DnSpy

```
GooglePlayValidator x
1 using System;
2 using System.Collections.Generic;
3 using System.Text;
4
5 namespace UnityEngine.Purchasing.Security
6 {
7     // Token: 0x0200001A RID: 26
8     internal class GooglePlayValidator
9     {
10         // Token: 0x060000E3 RID: 227 RVA: 0x00006AD0 File Offset: 0x00004ED0
11         public GooglePlayValidator(byte[] rsaKey)
12         {
13             this.key = new RSAKey(rsaKey);
14         }
15
16         // Token: 0x060000E4 RID: 228 RVA: 0x00006AE8 File Offset: 0x00004EE8
17         public GooglePlayReceipt Validate(string receipt, string signature)
18         {
19             byte[] bytes = Encoding.UTF8.GetBytes(receipt);
20             byte[] signature2 = Convert.FromBase64String(signature);
21             if (!this.key.Verify(bytes, signature2))
22             {
23                 throw new InvalidSignatureException();
24             }
25             Dictionary<string, object> dictionary = (Dictionary<string, object>)
26             object obj;
27             dictionary.TryGetValue("orderId", out obj);
28             object obj2;
29             dictionary.TryGetValue("packageName", out obj2);
30             object obj3;
```

# DEMO

# Applications vulnérables

- ▶ **Fruit Ninja**  
**(com.halfbrick.fruitninjafree)**

- ▶ Plus de 100 millions de téléchargement
- ▶ Jeu d'adresse où il faut découper des fruits



# ► Applications vulnérables

- ▶ Java Native Interface (JNI)
  - ▶ JNI permet d'interagir avec du code natif (C/C++)
  - ▶ Permet d'exécuter du code natif depuis du code Java/Kotlin
  - ▶ Embarquer sous forme d'une bibliothèque partagée
- ▶ Fruit Ninja implémente les fonctions sensibles en utilisant JNI
  - ▶ Et plus particulièrement les fonctions liées à Google Play Billing

```
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);
```

# Applications vulnérables

- ▶ Afin de comprendre le fonctionnement du paiement, il est nécessaire de faire de la rétro-ingénierie sur la bibliothèque partagée

```
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#
```

- ▶ Cette bibliothèque est codée en C++ ...
- ▶ Néanmoins, il apparaît que la signature n'est pas validée !

# DEMO

# ► Applications vulnérables

- ▶ **Application anonyme**

- ▶ Lecteur de magazines
- ▶ Achat de magazines à l'unité ou alors via un abonnement

- ▶ Processus d'achat

1. Un utilisateur sélectionne un magazine
2. Achat effectué utilisant le Google Play Billing
3. Après validation du paiement, le PDF est téléchargé par l'application



# ► Applications vulnérables

- ▶ Comme DoodleJump ou SnoopyPop, la validation du paiement est effectuée localement
- ▶ Il est donc trivial de contourner le paiement
- ▶ Néanmoins, une vérification supplémentaire a été implémentée par le développeur
  - ▶ Une validation côté serveur a été mis en place

# Applications vulnérables

- ▶ L'identifiant et le jeton de la transaction sont validés sur un serveur externe
  - ▶ *orderId* et *purchaseToken*

```
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=zewsnehjokfneboopwgvugq8&contentId=87285&token=256dc59549400fdc534994d94f401a1b1e7e10b7&deviceType=S&locale=en&orderId=15890047428
&readerMode=0&screenResolution=480&appId=[REDACTED]&os=ANDROID&productId=[REDACTED]
&version=[REDACTED]&random=373827991&currentTs=1567783472113&uuld=707af21[REDACTED]&
```

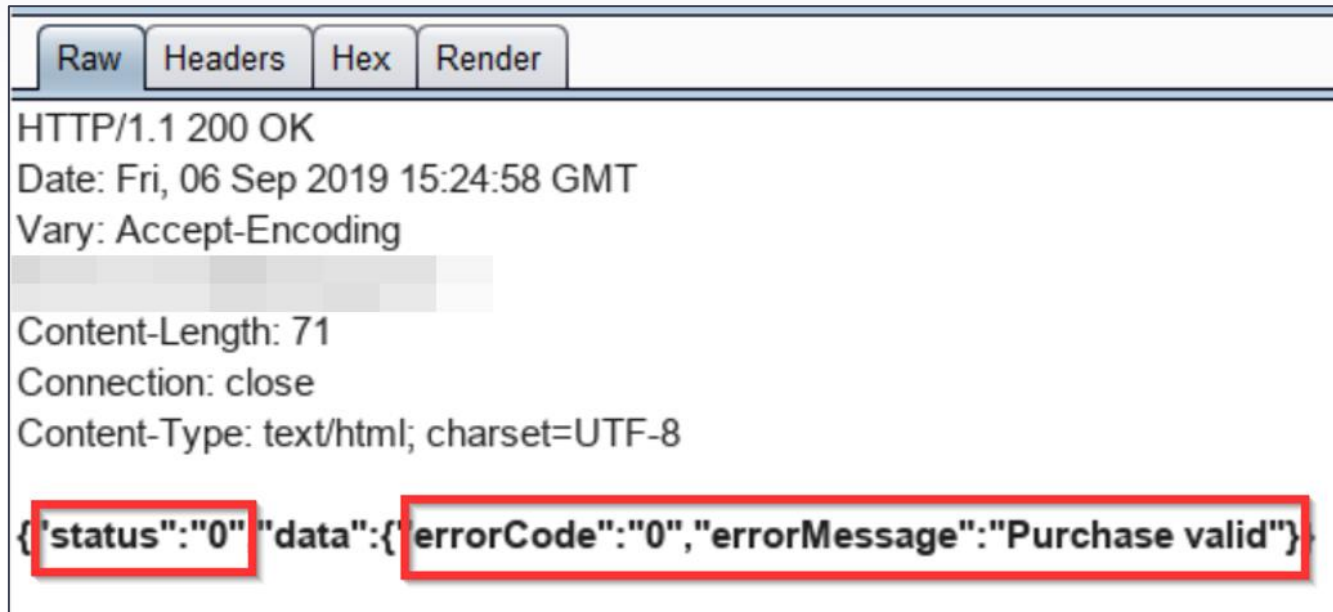
# Applications vulnérables

- ▶ Lorsque l'identifiant et le jeton de la transaction sont invalides
  - ▶ Le paiement n'est pas validé et le PDF n'est pas téléchargé



# Applications vulnérables

- ▶ Que se passe-t-il si l'on change le code de retour du serveur ?



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

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

# ► Applications vulnérables

- L'application télécharge le PDF sans aucun problème !



4

# Conclusion

Et maintenant ?

# Conclusion

- ▶ Les développeurs utilisent différentes techniques afin de protéger la validation du paiement
  - ▶ Obfuscation
  - ▶ Java Native Interface (JNI)
  - ▶ Rien!
- ▶ Le choix laissé par Google de permettre aux développeurs de valider le paiement localement dans l'application n'a pas permis d'assurer la sécurité des paiements dits « in-app » sous Android

# ► Conclusion

- ▶ Lors de notre étude réalisée sur 50 applications
  - ▶ 29 applications étaient vulnérables
  - ▶ Uniquement 5 applications utilisaient un serveur externe
- ▶ L'ensemble des développeurs ont été contactés
  - ▶ Uniquement l'application « anonyme » a été supprimée du Play Store
  - ▶ Les autres développeurs n'ont pas répondu



# MERCI!

## Des questions?



[www.randorisec.fr](http://www.randorisec.fr)



[blog.randorisec.fr](http://blog.randorisec.fr)



[@RandoriSec](https://twitter.com/RandoriSec)



# ▶ Références

Google Play Billing documentation

- ▶ [https://developer.android.com/google/play/billing/billing\\_overview](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](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>

# ▶ Références

Google Play Billing – Verify purchase

- ▶ [https://developer.android.com/google/play/billing/billing\\_library\\_overview#Verify](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>

# ▶ Références

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>

# ▶ Références

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](https://www.securingapps.com/blog/BsidesLisbon17_AbusingAndroidInappBilling.pdf)

# ► Crédits

Merci aux personnes qui ont mis à disposition ces ressources gratuitement :

- ▶ Modèle de présentation PowerPoint par [SlidesCarnival](#)
- ▶ Photographie par [Pixabay](#)