Playing Without Paying: Detecting Vulnerable Payment Verification in Native Binaries of Unity Mobile Games

Chaoshun Zuo, Zhiqiang Lin

Department of Computer Science and Engineering
The Ohio State University

USENIX Security 2022
Game Industry

Video game market revenue worldwide 2021

Revenue: 175.8 billion
- Mobile Game: 52%
- PC game: 20%
- Console game: 28%

Source: www.statista.com
Mobile Game Monetization

Monetization strategies

- Pay-to-play
- In-game advertising
- In-game purchase (most popular one)

Source: blog.instabug.com
In-game Purchase Implementation

Mobile Game

Game Engine IAP SDK

Google Play IAP  Apple Store IAP  etc.
How In-game Purchases Work

1. Payment Request

- Mobile Game
- App Store
- Game Server

Absence of Verification
How In-game Purchases Work

1. Payment Request
2. Payment Processing
How In-game Purchases Work

1. Payment Request
2. Payment Processing
3. Payment Response
How In-game Purchases Work

1. Payment Request
2. Payment Processing
3. Payment Response
4. Transaction Validation

Mobile Game → App Store → Game Server

Absence of Verification
How In-game Purchases Work
How In-game Purchases Work

1. Payment Request
2. Payment Processing
3. Payment Response
4. Transaction Validation
5. Transaction Verification
6. Verification Result
How In-game Purchases Work

1. Payment Request
2. Payment Processing
3. Payment Response
4. Transaction Validation
5. Transaction Verification
6. Verification Result
7. Products Distribution
Vulnerable In-game Purchases

▶ Lack-of-verification
Vulnerable In-game Purchases

1. Payment Request
2. Payment Processing
3. Payment Response
4. Local Verification by Public Key
5. Products Distribution

▶ Local Verification
### Game Engines

**Popularity**
- By analyzing 293,019 mobile games, we found that Unity is the most popular game engine in Android games.

**Advantages**
- Cross platform
- C# programming language
- Powerful and easy-to-use IDE
Unity - IL2CPP Compatible

C# game logic

libil2cpp

C++ compiler

C++ code

Native binary

System Libraries
i.e., UnityEngine.dll

Managed assemblies

Unused Bytecode Stripper

Stripped assemblies

Faster

Securer
Introduction

Background

PaymentScope

Evaluation

Related Work

Summary

References

Running Examples

```csharp
class IAPManager : IStoreListener
{
    public PurchaseProcessingResult ProcessPurchase(PurchaseEventArgs args)
    {
        CrossPlatformValidator validator = new CrossPlatformValidator(GooglePlayTangle.Data(), AppleTangle.Data(), Application.identifier);
        try
        {
            validator.Validate(args.purchasedProduct.get_receipt());
        }
        catch (IAPSecurityException)
        {
            Debug.Log("Invalid receipt, not unlocking content");
        }
        return PurchaseProcessingResult.Complete;
    }
}

Payment Processing

Game Server

App Store

Mobile Game
```
class IAPManager : IStoreListener
{
    public PurchaseProcessingResult ProcessPurchase(PurchaseEventArgs args)
    {
        CrossPlatformValidator validator = new CrossPlatformValidator(
            GooglePlayTangle.Data(), AppleTangle.Data(),
            Application.identifier);
        try
        {
            validator.Validate(args.purchasedProduct.get_receipt());
            ...
        }
        catch (IAPSecurityException)
        {
            Debug.Log("Invalid receipt, not unlocking content");
        }
        return PurchaseProcessingResult.Complete;
    }
}
Running Examples

```
1 Class UnityEngine.Purchasing.IStoreListener
2
3 public PurchaseProcessingResult ProcessPurchase(PurchaseEventArgs args)
4 {
5   ...
6   String receiptstr = args.purchasedProduct.get_receipt();
7   StorePurchaseReceipt receipt = JsonUtility.FromJson<StorePurchaseReceipt>(receiptstr);
8   GooglePayload gpayload = JsonUtility.FromJson<GooglePayload>(receipt.Payload);
9   httpRequest.AddField("signature", gpayload.signature)
10  ...
11 }

12 public class StorePurchaseReceipt
13 {
14   public string Store; // 0x10
15   public string TransactionID; // 0x18
16   public string Payload; // 0x20
17 }
18
19 public class GooglePayload
20 {
21   public string json; // 0x10
22   public string signature; // 0x18
23 }
```
Running Examples

<table>
<thead>
<tr>
<th>Address Bytes</th>
<th>Disassemble</th>
<th>P-code Output</th>
<th>OP Code</th>
<th>P-code Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>0130d49c c0f24794 b1 0x02509f9c</td>
<td>(register, 0x4000, 8) CALL (ram, 0x02509f9c, 8), (register, 0x400b, 8), (const, 0x00, 8)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0130d4ac 52be0c94 b1 0x0163cdf4</td>
<td>(register, 0x4000, 8) CALL (ram, 0x0163cdf4, 8), (register, 0x4000, 8), (register, 0x4008, 8)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0130d4c0 c01240f9 ldr x0, [x22, #0x20]</td>
<td>(unique, 0x100004b4, 8) INT_ADD (register, 0x4000, 8), (const, 0x20, 8)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1 Class UnityEngine.Purchasing.IStoreListener
2
3 public PurchaseProcessingResult ProcessPurchase(PurchaseEventArgs args)
4 {
5    ...
6    String receiptstr = args.purchasedProduct.get_receipt();
7    StoreReceipt receipt = JsonUtility.FromJson<StoreReceipt>(receiptstr);
8    GooglePayload gpayload = JsonUtility.FromJson<GooglePayload>(receipt.Payload);
9    httpRequest.AddField("signature", gpayload.signature)
10    ...
11 }
Running Examples

<table>
<thead>
<tr>
<th>Address</th>
<th>Bytes</th>
<th>Disassemble</th>
<th>P-code Output</th>
<th>OP Code</th>
<th>P-code Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>0130d49c 00f24794</td>
<td>bl 0x02509f9c</td>
<td>(register, 0x4000, 8)</td>
<td>CALL (ram, 0x02509f9c, 8), (register, 0x40b0, 8), (const, 0x00, 8)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0130d4ac 52be0c94</td>
<td>bl 0x0163cfd4</td>
<td>(register, 0x4000, 8)</td>
<td>CALL (ram, 0x0163cfd4, 8), (register, 0x4000, 8), (register, 0x4008, 8)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0130d4c0 c01240f9</td>
<td>ldr x0, [x22, #0x20]</td>
<td>(unique, 0x100004b4, 8)</td>
<td>INT_ADD (register, 0x4000, 8), (const, 0x20, 8)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(unique, 0x0c90, 8)</td>
<td>CAST (unique, 0x100004b4, 8)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0130d4cc 4abe0c94</td>
<td>bl 0x0163cfd4</td>
<td>(register, 0x4000, 8)</td>
<td>CALL (ram, 0x0163cfd4, 8), (register, 0x4000, 8), (register, 0x4008, 8)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0130d510 d70e0f9</td>
<td>ldr x23, [x22, #0x18]</td>
<td>(unique, 0x100004d4, 8)</td>
<td>INT_ADD (register, 0x4000, 8), (const, 0x18, 8)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(unique, 0x0c90, 8)</td>
<td>CAST (unique, 0x100004d4, 8)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0130d538 67eb4794</td>
<td>bl 0x025082d4</td>
<td>--</td>
<td>CALL (ram, 0x025082d4, 8), ..., (register, 0x40b8, 8), (const, 0x00, 8)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

1. Class UnityEngine.Purchasing.IStoreListener
2. public PurchaseProcessingResult ProcessPurchase(PurchaseEventArgs args)
3. {
4.   String receiptstr = args.purchasedProduct.get_receipt();
5.   StoreReceipt receipt = JsonUtility.FromJson<StoreReceipt>(receiptstr);
7.   httpRequest.AddField("signature", gpayload.signature)
8.   ...
### Running Examples

<table>
<thead>
<tr>
<th>Address</th>
<th>Disassemble</th>
<th>P-code Output</th>
<th>OP Code</th>
<th>P-code Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>0130d49c c0f24794</td>
<td>bl 0x02509f9c</td>
<td>(register, 0x4000, 8)</td>
<td>CALL (ram, 0x02509f9c, 8), (register, 0x40b0, 8), (const, 0x00, 8)</td>
<td></td>
</tr>
<tr>
<td>0130d4ac 52be0c94</td>
<td>bl 0x0163cfd4</td>
<td>(register, 0x4000, 8)</td>
<td>CALL (ram, 0x0163cfd4, 8), (register, 0x4000, 8), (register, 0x4008, 8)</td>
<td></td>
</tr>
<tr>
<td>0130d4c0 01240f9f</td>
<td>ldr x0, [x22, #0x20]</td>
<td>(unique, 0x100004b4, 8)</td>
<td>INT_ADD (register, 0x4000, 8), (const, 0x20, 8)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(unique, 0x0c90, 8)</td>
<td>CAST (unique, 0x100004b4, 8)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(register, 0x4000, 8)</td>
<td>LOAD (const, 0x01bl, 4), (unique, 0x0c90, 8)</td>
<td></td>
</tr>
<tr>
<td>0130d4cc 4abe0c94</td>
<td>bl 0x0163cfd4</td>
<td>(register, 0x4000, 8)</td>
<td>CALL (ram, 0x0163cfd4, 8), (register, 0x4000, 8), (register, 0x4008, 8)</td>
<td></td>
</tr>
<tr>
<td>0130d510 070e4bf9</td>
<td>ldr x23, [x22, #0x18]</td>
<td>(unique, 0x100004d4, 8)</td>
<td>INT_ADD (register, 0x4000, 8), (const, 0x18, 8)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(unique, 0x0c90, 8)</td>
<td>CAST (unique, 0x100004d4, 8)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(register, 0x40b8, 8)</td>
<td>LOAD (const, 0x01bl, 4), (unique, 0x0c90, 8)</td>
<td></td>
</tr>
<tr>
<td>0130d538 67eb4794</td>
<td>bl 0x025082d4</td>
<td>--</td>
<td>CALL (ram, 0x025082d4, 8), ..., (register, 0x40b8, 8), (const, 0x00, 8)</td>
<td></td>
</tr>
</tbody>
</table>

1. Class UnityEngine.Purchasing.IStoreListener
2. public PurchaseProcessingResult ProcessPurchase(PurchaseEventArgs args)
3. {
4.   String receiptstr = args.purchasedProduct.get_receipt();
5.   StoreReceipt receipt = JsonUtility.FromJson.StoreReceipt(receiptstr);
7.   httpRequest.AddField("signature", gpayload.signature)
8.   ...
9. }

9 / 17
### Running Examples

<table>
<thead>
<tr>
<th>Address</th>
<th>Bytes</th>
<th>Disassemble</th>
<th>P-code Output</th>
<th>OP Code</th>
<th>P-code Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>0130d49c</td>
<td>c0f24794</td>
<td>bl 0x02509f9c</td>
<td>(register, 0x4000, 8)</td>
<td>CALL (ram, 0x02509f9c, 8), (register, 0x40b0, 8), (const, 0x00, 8)</td>
<td></td>
</tr>
<tr>
<td>0130d4ac</td>
<td>52be0c94</td>
<td>bl 0x0163cdf4</td>
<td>(register, 0x4000, 8)</td>
<td>CALL (ram, 0x0163cdf4, 8), (register, 0x4000, 8), (register, 0x4008, 8)</td>
<td></td>
</tr>
<tr>
<td>0130d4c0</td>
<td>c01240f9f</td>
<td>ldr x0, [x22, #0x20]</td>
<td>(unique, 0x100004b4, 8)</td>
<td>INT_ADD (register, 0x4000, 8), (const, 0x20, 8)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(unique, 0x0c90, 8)</td>
<td>CAST (unique, 0x100004b4, 8)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(register, 0x4000, 8)</td>
<td>LOAD (const, 0x01b1, 4), (unique, 0x0c90, 8)</td>
<td></td>
</tr>
<tr>
<td>0130d4cc</td>
<td>4abe0c94</td>
<td>bl 0x0163cdf4</td>
<td>(register, 0x4000, 8)</td>
<td>CALL (ram, 0x0163cdf4, 8), (register, 0x4000, 8), (register, 0x4008, 8)</td>
<td></td>
</tr>
<tr>
<td>0130d510</td>
<td>d70e0f9f</td>
<td>ldr x23, [x22, #0x18]</td>
<td>(unique, 0x100004d4, 8)</td>
<td>INT_ADD (register, 0x4000, 8), (const, 0x18, 8)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(unique, 0x0c90, 8)</td>
<td>CAST (unique, 0x100004d4, 8)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(register, 0x40b8, 8)</td>
<td>LOAD (const, 0x01b1, 4), (unique, 0x0c90, 8)</td>
<td></td>
</tr>
<tr>
<td>0130d538</td>
<td>67eb4794</td>
<td>bl 0x025082d4</td>
<td>--</td>
<td>CALL (ram, 0x025082d4, 8), ..., (register, 0x40b8, 8), (const, 0x00, 8)</td>
<td></td>
</tr>
</tbody>
</table>

1. `Class UnityEngine.Purchasing.IStoreListener
2. 3 public PurchaseProcessingResult ProcessPurchase(PurchaseEventArgs args)
4. 7 String receiptstr = args.purchasedProduct.get_receipt();
5. 8 StoreReceipt receipt = JsonUtility.FromJson<StoreReceipt>(receiptstr);
6. 9 GooglePayload gpayload = JsonUtility.FromJson<GooglePayload>(receipt.Payload);
7. 10 httpRequest.AddField("signature", gpayload.signature)
### Running Examples

**Class**: `UnityEngine.Purchasing.IStoreListener`

**public** `PurchaseProcessingResult ProcessPurchase(PurchaseEventArgs args)`

```csharp
public PurchaseProcessingResult ProcessPurchase(PurchaseEventArgs args) {
    String receiptstr = args.purchasedProduct.get_receipt();
    StoreReceipt receipt = JsonUtility.FromJson<StoreReceipt>(receiptstr);
    GooglePayload gpayload = JsonUtility.FromJson<GooglePayload>(receipt.Payload);
    HttpClient httpClient = new HttpClient();
    httpClient.AddField("signature", gpayload.signature);
    ...  
}
```
Goal

Remote Verification

Local Verification

Lack-of-verification

Problem Statement

► In-game purchase implemented w/ Unity Engine
► Local verification and lack-of-verification

Assumption

► Android games
► IL2CPP
Challenges and Insights

Remote Verification

- Mobile Game
- App Store
- Game Server

1. Payment Request
2. Payment Response
3. Payment Processing
4. Transaction Validation
5. Transaction Verification
6. Verification Result
7. Products Distribution

Local Verification

- Mobile Game
- App Store
- Game Server

1. Payment Request
2. Payment Response
3. Payment Processing
4. Local Verification by Public Key
5. Products Distribution

Lack-of-verification

- Mobile Game
- App Store
- Game Server

1. Payment Request
2. Payment Response
3. Payment Processing

Challenge

▶ How to identify vulnerable in-game purchases
Challenges and Insights

- **Challenge**
  - How to identify vulnerable in-game purchases

- **Insight**
  - Operating System and Unity Engine APIs

```csharp
1 class IAPManager : IStoreListener
2 {
3     public PurchaseProcessingResult ProcessPurchase(
4         PurchaseEventArgs args)
5     {
6         CrossPlatformValidator validator = new CrossPlatformValidator(
7             GooglePlayTangle.Data(), AppleTangle.Data(),
8             Application.identifier);
9         try
10            validator.Validate(args.purchasedProduct.get_receipt());
11         catch (IAPSecurityException)
12            {
13                Debug.Log("Invalid receipt, not unlocking content");
14                return PurchaseProcessingResult.Complete;
15            }
16     }
17 }
18 }
```

```csharp
1 class PurchaseManager : IStoreListener
2 {
3     public PurchaseProcessingResult ProcessPurchase(
4         PurchaseEventArgs args)
5     {
6         ... 
7         StoreReceipt receipt = JsonUtility.FromJsonStoreReceipt(
8             args.purchasedProduct.get_receipt());
9         GooglePayload gpayload = JsonUtility.FromJsonGooglePayload(
10             receipt, payload);
11         httpRequest.AddField("signature", gpayload.signature)
12         ... 
13     }
14 }
15 }
```

- **CrossPlatformValidator.Validate**
  - *String receipt*

- **WWWForm.AddField**
  - *String key, String value*
### Challenges and Insights

<table>
<thead>
<tr>
<th>Address</th>
<th>Bytes</th>
<th>Disassemble</th>
<th>P-code Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0130d49c</td>
<td>c02479f</td>
<td><code>b1 0x02509f9c</code></td>
<td><code>(register, 0x4000, 8) CALL (ram, 0x02509f9c, 8), (register, 0x40b0, 8), (const, 0x00, 8)</code></td>
</tr>
<tr>
<td>0130d4ac</td>
<td>52be0c9f</td>
<td><code>b1 0x0163df4</code></td>
<td><code>(register, 0x4000, 8) CALL (ram, 0x0163df4, 8), (register, 0x4000, 8), (register, 0x4008, 8)</code></td>
</tr>
<tr>
<td>0130d4c0</td>
<td>c01240f9f</td>
<td><code>ldr x0, [x22, #0x20]</code></td>
<td><code>(unique, 0x10004b4, 8) INT_ADD (register, 0x4000, 8), (const, 0x20, 8)</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><code>(unique, 0x0c90, 8) CAST (unique, 0x10004b4, 8)</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><code>(register, 0x4000, 8) LOAD (const, 0x01bl, 4), (unique, 0x0c90, 8)</code></td>
</tr>
<tr>
<td>0130d4cc</td>
<td>4abe0c9f</td>
<td><code>b1 0x0163df4</code></td>
<td><code>(register, 0x4000, 8) CALL (ram, 0x0163df4, 8), (register, 0x4000, 8), (register, 0x4008, 8)</code></td>
</tr>
<tr>
<td>0130d510</td>
<td>d70e40f9f</td>
<td><code>ldr x23, [x22, #0x18]</code></td>
<td><code>(unique, 0x10004d4, 8) INT_ADD (register, 0x4000, 8), (const, 0x18, 8)</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><code>(unique, 0x0c90, 8) CAST (unique, 0x10004d4, 8)</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><code>(register, 0x40b8, 8) LOAD (const, 0x01bl, 4), (unique, 0x0c90, 8)</code></td>
</tr>
<tr>
<td>0130d538</td>
<td>67eb479f</td>
<td><code>b1 0x025082d4</code></td>
<td>-- CALL (ram, 0x025082d4, 8), ..., (register, 0x40b8, 8), (const, 0x00, 8)</td>
</tr>
</tbody>
</table>

#### Challenge

- **How to pinpoint target APIs from game binaries**
Challenges and Insights

### Address Bytes | Disassemble | P-code Output | OP Code | P-code Input
---|---|---|---|---
0130d49c c024f94 | b1 0x02509f9c | (register, 0x4000, 8) CALL (ram, 0x02509f9c, 8), (register, 0x40b0, 8), (const, 0x00, 8)
0130d4ac 52be0c94 | b1 0x0163cdf4 | (register, 0x4000, 8) CALL (ram, 0x0163cdf4, 8), (register, 0x4000, 8), (register, 0x4008, 8)
0130d4c0 c01240f9 | ldr x0, [x22, #0x20] | (unique, 0x100004b4, 8) INT_ADD (register, 0x4000, 8), (const, 0x20, 8)
0130d4cc 4abe0c94 | b1 0x0163cdf4 | (register, 0x4000, 8) LOAD (const, 0x01b1, 4), (unique, 0x0c90, 8)
0130d510 d70e40f9 | ldr x23, [x22, #0x18] | (unique, 0x100004d4, 8) INT_ADD (register, 0x4000, 8), (const, 0x18, 8)
0130d538 67eb4794 | b1 0x025082d4 | -- | CALL (ram, 0x025082d4, 8), ..., (register, 0x40b8, 8), (const, 0x00, 8)

### Challenge

► How to pinpoint target APIs from game binaries

### Insight

► Extracting meta-data file generated during compilation
Challenges and Insights

<table>
<thead>
<tr>
<th>Address</th>
<th>Bytes</th>
<th>Disassemble</th>
<th>P-code Output</th>
<th>OP Code</th>
<th>P-code Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>0130d49c</td>
<td>c024794</td>
<td>b1 0x02509f9c</td>
<td>(register, 0x4000, 8) CALL</td>
<td>(ram, 0x02509f9c, 8), (register, 0x40b0, 8), (const, 0x00, 8)</td>
<td></td>
</tr>
<tr>
<td>0130d4ac</td>
<td>52b06c94</td>
<td>b1 0x0163df4</td>
<td>(register, 0x4000, 8) CALL</td>
<td>(ram, 0x0163df4, 8), (register, 0x4000, 8), (register, 0x4008, 8)</td>
<td></td>
</tr>
<tr>
<td>0130d4c0</td>
<td>c01240f9</td>
<td>1dr x0, [x22, #0x20]</td>
<td>(unique, 0x100004b4, 8) INT_ADD</td>
<td>(register, 0x4000, 8), (const, 0x20, 8)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(unique, 0x0c90, 8) CAST</td>
<td>(unique, 0x100004b4, 8)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(register, 0x4000, 8) LOAD</td>
<td>(const, 0x01b1, 4), (unique, 0x0c90, 8)</td>
<td></td>
</tr>
<tr>
<td>0130d4cc</td>
<td>4abe0c94</td>
<td>b1 0x0163df4</td>
<td>(register, 0x4000, 8) CALL</td>
<td>(ram, 0x0163df4, 8), (register, 0x4000, 8), (register, 0x4008, 8)</td>
<td></td>
</tr>
<tr>
<td>0130d510</td>
<td>d70e0f9</td>
<td>1dr x23, [x22, #0x18]</td>
<td>(unique, 0x100004d4, 8) INT_ADD</td>
<td>(register, 0x4000, 8), (const, 0x18, 8)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(unique, 0x0c90, 8) CAST</td>
<td>(unique, 0x100004d4, 8)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>(register, 0x40b8, 8) LOAD</td>
<td>(const, 0x01b1, 4), (unique, 0x0c90, 8)</td>
<td></td>
</tr>
<tr>
<td>0130d538</td>
<td>67eb4794</td>
<td>b1 0x025082d4</td>
<td>--</td>
<td>CALL</td>
<td>(ram, 0x025082d4, 8), ..., (register, 0x40b8, 8), (const, 0x00, 8)</td>
</tr>
</tbody>
</table>

Challenge

▶ How to identify the payment-data definition, use, and propagation
Challenges and Insights

### Challenge

- **How to identify the payment-data definition, use, and propagation**

### Insight-direct data flow

- **Using system and Unity APIs summary approaches**
Challenges and Insights

```csharp
1 class unityInAppPurchase_LS : IStoreListener
2 {
3     ... 
4     private string m_LastReceipt; // 0x30
5     public PurchaseProcessingResult ProcessPurchase(
6         PurchaseEventArgs args)
7     {
8         this.m_LastReceipt = args.purchasedProduct.get_receipt();
9     }
10 }
```

### Challenge
- How to identify the payment-data definition, use, and propagation

### Insight-indirect data flow
- Building global class table for propagation
PaymenScope

Three Key Components

1. Metadata Extraction
2. Payment-Aware Data Flow Analysis
3. Vulnerabilities Identification
Experiments

1. 2.1 million apps from Google Play (downloaded from AndroZoo)
2. 39,121 of them are developed based on Unity Engine and compiled by IL2CPP
3. 10,640 of them support in-game purchase.
Experiments

1. DELL server: two E5-2695 v2 CPUs (48 cores in total) and 96GB memory.
2. The experiment took 669 hours (almost 28 days) with 24 threads.
Experiments

1. DELL server: two E5-2695 v2 CPUs (48 cores in total) and 96GB memory.
2. The experiment took 669 hours (almost 28 days) with 24 threads.

Vulnerable Games
- 8,233 games with lack-of-verification
- 721 games with local verification
Experiments

1. DELL server: two E5-2695 v2 CPUs (48 cores in total) and 96GB memory.
2. The experiment took 669 hours (almost 28 days) with 24 threads.

Vulnerable Games

- 8,233 games with lack-of-verification
- 721 games with local verification

10,640 games support in-game purchase

84.15% games are vulnerable
FP and FN Analysis

Dataset - 280 games

- Top-10 local verification games
- Top-10 lack-of-verification games
- Top-10 remote verification games
- Randomly selected 200 games from vulnerable games
- Randomly selected 50 games from non-vulnerable games

- Injecting fake transaction using virtualization
- Disabling local-verification using code patching
## FP and FN Analysis

### FP analysis - 220 vulnerable games
- 30 games cannot be tested
- 190 games are confirmed to be vulnerable

### FN analysis - 60 non-vulnerable games
- 9 games cannot be tested
- 37 games are secure
- 14 games are vulnerable (i.e., FN: 29%)
Vulnerable Game Analysis

![Chart showing the number of apps and their verification types across different payment scopes.]

- Remote-verification
- No-verification
- Local-verification

---

### Vulnerable Game Analysis

- **100M - 500M**: 55
- **50M - 100M**: 61
- **10M - 50M**: 64
- **5M - 10M**: 70
- **1M - 5M**: 79
- **500K - 1M**: 77
- **100K - 500K**: 75
- **50K - 100K**: 81
- **10K - 50K**: 81
- **5K - 10K**: 87
- **1K - 5K**: 90
- **500 - 1K**: 94

---

### Payment Scope

- **100M - 500M**: 9
- **50M - 100M**: 31
- **10M - 50M**: 299
- **5M - 10M**: 313
- **1M - 5M**: 1225
- **500K - 1M**: 635
- **100K - 500K**: 1423
- **50K - 100K**: 671
- **10K - 50K**: 1425
- **5K - 10K**: 611
- **1K - 5K**: 1355
- **500 - 1K**: 2643
Vulnerable Game Analysis

Off-line games
Responsible Disclosure

Unity
- Remove the local verification API
- Offer remote verification API
- Show risk in the documentation

Game developers
- Contacted 5,494 developers of the vulnerable games
Limitation

- PaymentScope cannot handle some indirect propagations
- PaymentScope has FNs (i.e., 29%)
Related Work

1. **Game security.** Numerous efforts have been made to fight game bots [BCR08, LGZ+17, LWK+16, GWXW09]. BlackMirror [PAL20] use Intel SGX to defeat wallhacks. Tian et al. [TCM+16] studied attacks on mobile games.

2. **Binary analysis and payment security.** Over the past decades, a large body of research focusing on binary analysis [NS05, YSE+07, CLZ21, WLZ20, SWS+16, Wei84, KSS17] and payment security [WCWQ11, SXS14, WHS16, RSM+12, LH, MRK14].
Summary

PAYMENTSCOPE

▶ A fully automated system to identify vulnerable in-game purchases
▶ It performs payment-aware data flow analysis on Unity games binaries

Experimental Result w/ 10,640 games
▶ 8,954 (84.15%) games are vulnerable

Open source
▶ github.com/OSUSecLab PAYMENTSCOPE
Playing Without Paying: Detecting Vulnerable Payment Verification in Native Binaries of Unity Mobile Games

Chaoshun Zuo, Zhiqiang Lin

Department of Computer Science and Engineering
The Ohio State University

USENIX Security 2022


Yeh-chi Lai and Mohammad Husain, *A holistic approach for securing in-app purchase (iap) vulnerability in mobile applications*.

Eunjo Lee, Jiyoung Woo, Hyoungshick Kim, Aziz Mohaisen, and Huy Kang Kim, You are a game bot!: Uncovering game bots in mmorpgs via self-similarity in the wild., Ndss, 2016.


Fangqi Sun, Liang Xu, and Zhendong Su, *Detecting logic vulnerabilities in e-commerce applications*, NDSS, 2014.


