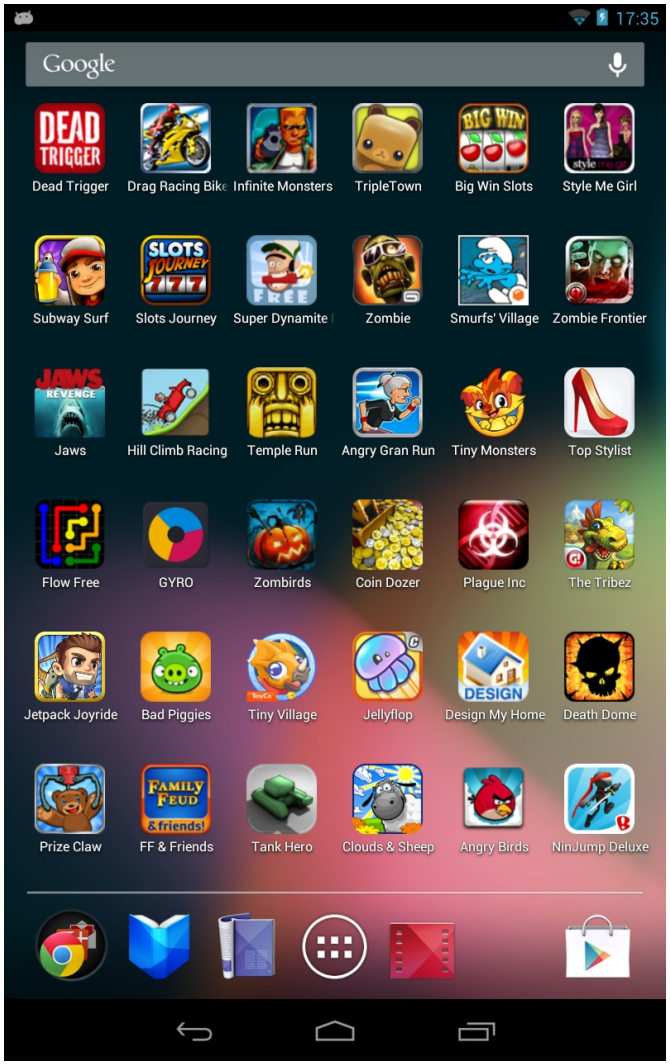
Northeastern University
Systems Security Lab

VirtualSwindle: An Automated Attack Against In-App Billing on Android

ASIACCS 2014

Collin Mulliner, William Robertson, Engin Kirda
{crm,wkr,ek}[at]ccs.neu.edu

Mobile Apps


17:35
Google
Dead Trigger
Drag Racing Bike
Infinite Monsters
TripleTown
Big Win Slots
Style Me Girl
Subway Surf
Slots Journey
Super Dynamite
Zombie
Smurfs' Village
Zombie Frontier
Jaws
Hill Climb Racing
Temple Run
Angry Gran Run
Tiny Monsters
Top Stylist
Flow Free
GYRO
Zombirds
Coin Dozer
Plague Inc
The Tribez
Jetpack Joyride
Bad Piggies
Tiny Village
Jellyflop
Design My Home
Death Dome
Prize Claw
FF & Friends
Tank Hero
Clouds & Sheep
Angry Birds
NinJump Deluxe

Mobile Apps

Apps make the platform, without Apps nobody buys devices

Over 1 Million Apps in Google Play

Mobile Apps → Money

- Buy App
  - Fixed price
  - One-time payment (when you buy the App)

Mobile Apps → Money

- Buy App
  - Fixed price
  - One-time payment (when you buy the App)

- **Buy items + services within the application**
  - **Multiple purchases → multiple payments**
  - **Google calls this “In-App Billing”**

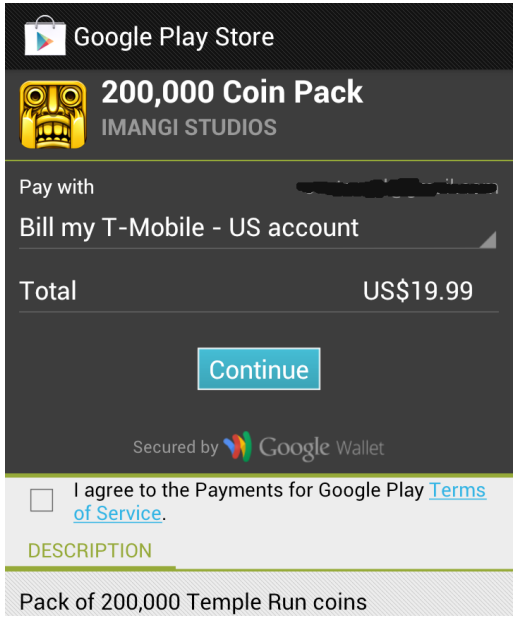
In-App Billing

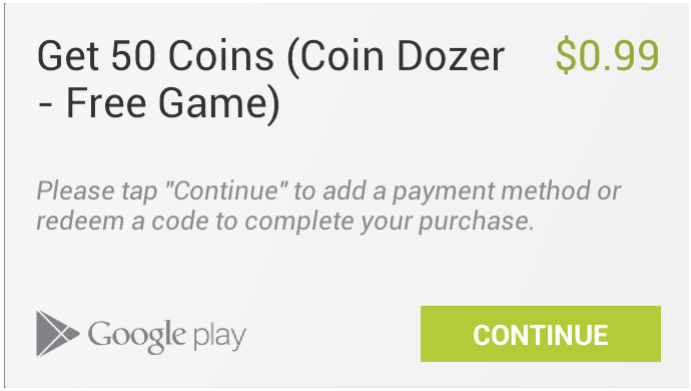
- Android platform service
  - Built into the Play Store
  - API on device
- Google
  - Takes care of payment processing, handles refunds, ...
  - Keeps 30% of all sales
- Developer
  - Configures items in Google Play Store
  - Adds payment functionality to his App
- Both make significant revenue through In-App Billing

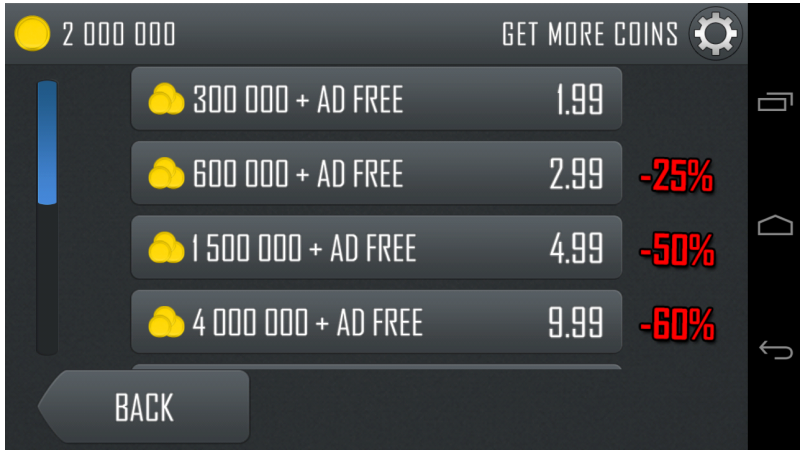
In-App Billing Usecases

- Remove ads
- Full version → more features
- Games
  - Levels
  - Coins
- Application specific
  - Features
  - Content

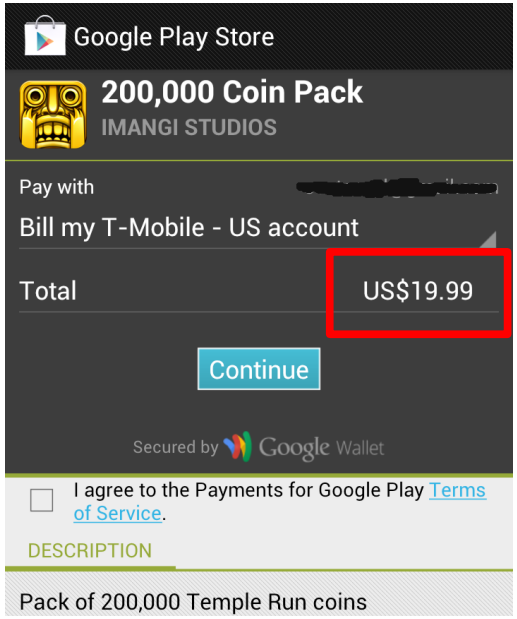
In-App Billing Pricing

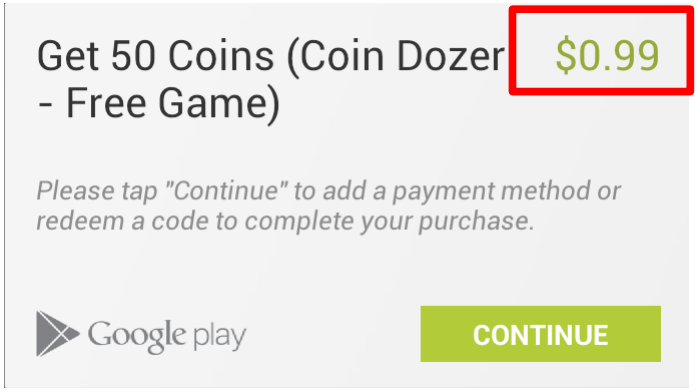


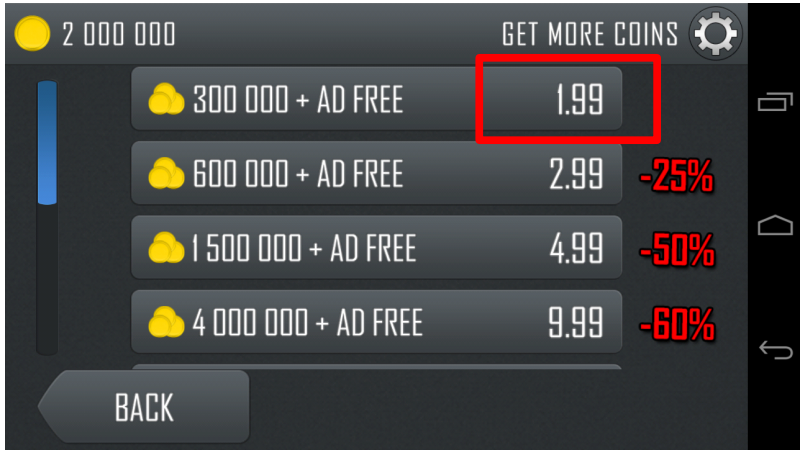




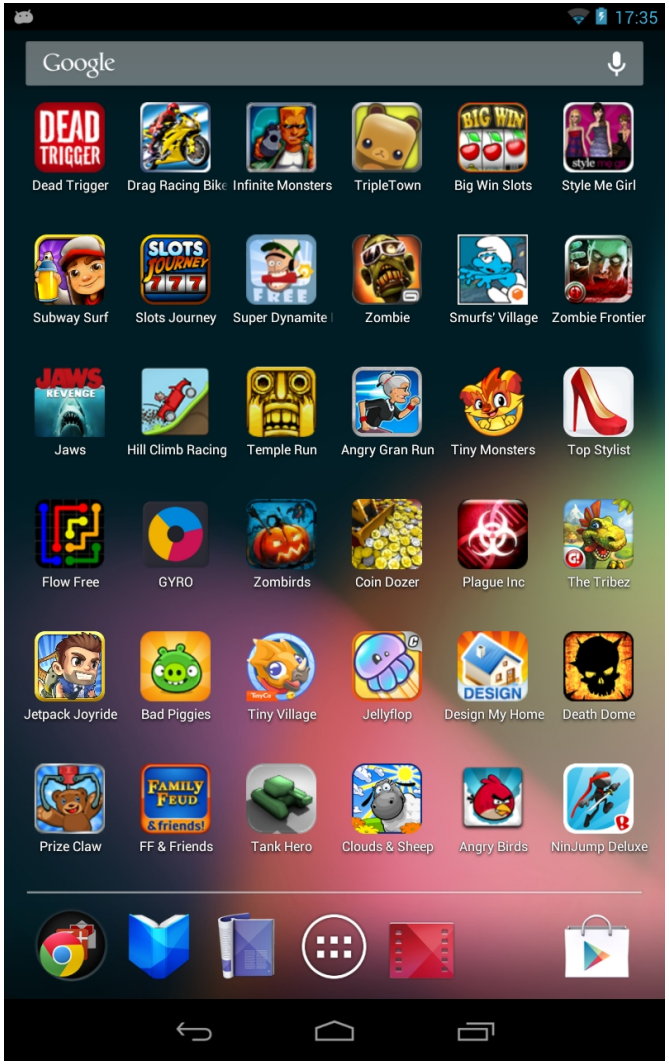
In-App Billing Pricing







Who uses In-App Billing?

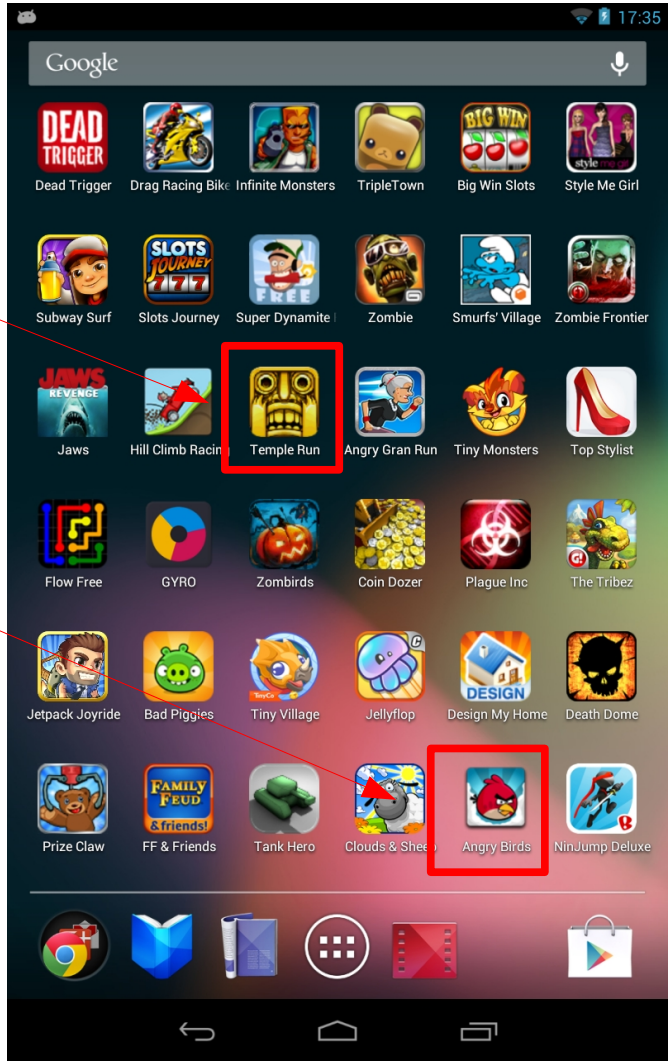


Who uses In-App Billing?

- Temple Run
  +100M installs

- Angry Birds
  +100M installs

- Everybody!
  - developers find possibilities



Attacking In-App Billing

- Significant loss if In-App Billing can be bypassed
- Developers should have high incentive to protect their Apps

- Questions:
  - Can In-App Billing be bypassed?
    - Without reverse engineering and patching App?
  - Are Apps hardened against attacks?
    - What kind of hardening?

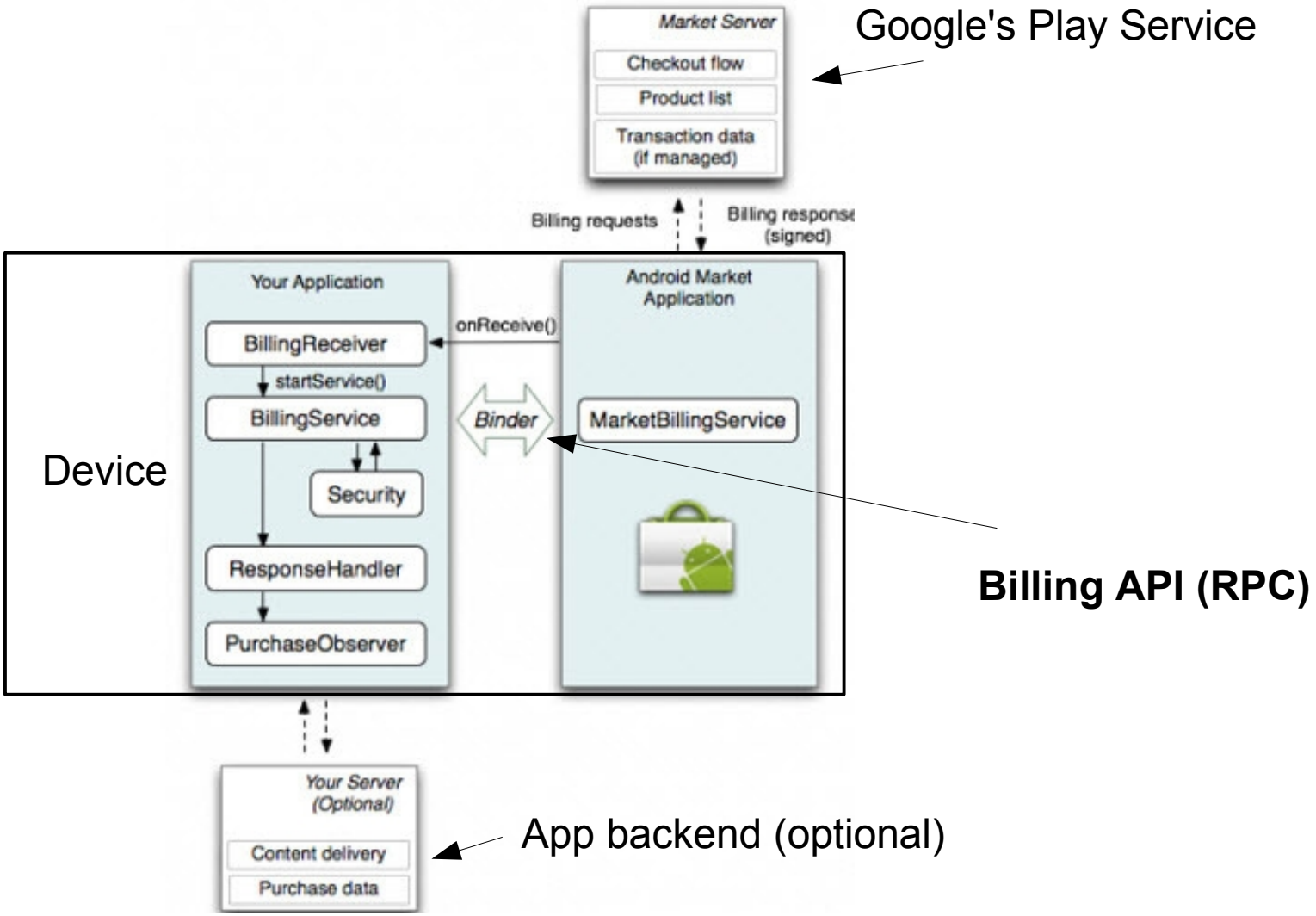
Attacks are possible: VirtualSwindle

- Demo

Contributions

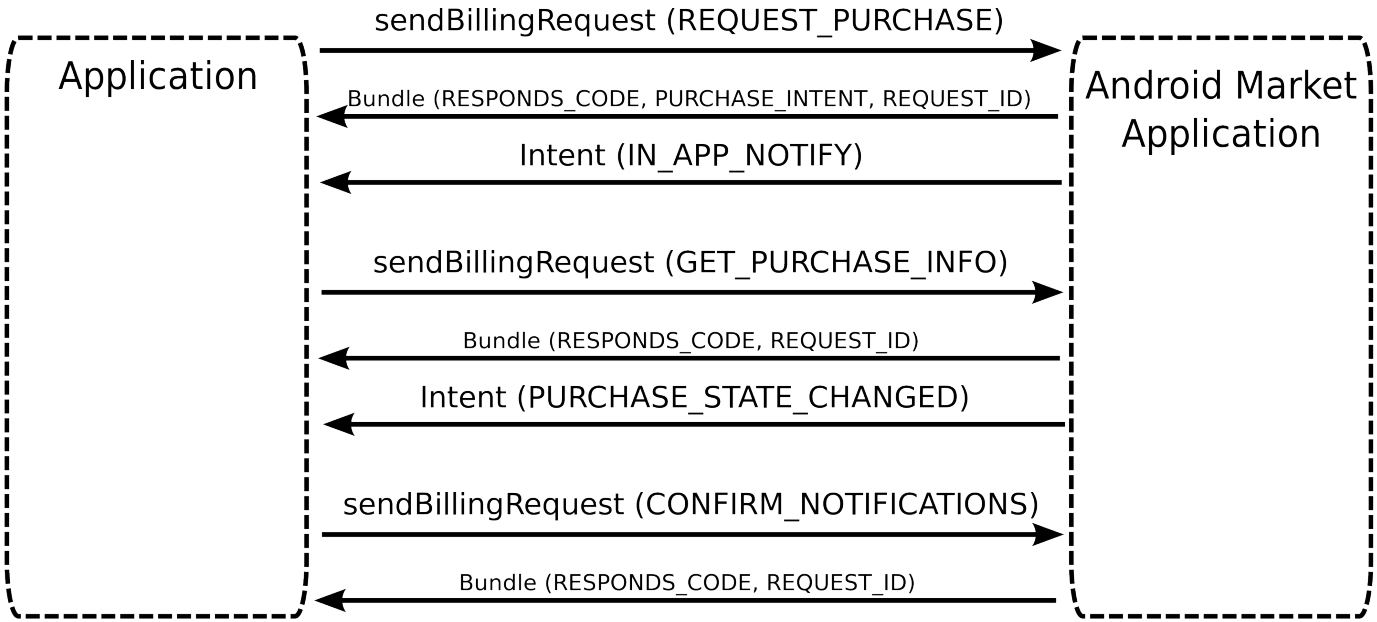
- First to investigate In-App Billing security
  - Systematic study
- Created *VirtualSwindle* an automatic attack against In-App Billing
  - Attack is generic, independent from App
- Developed Dynamic Dalvik Instrumentation (DDI)
  - Allows modifying Dalvik Apps at runtime
- Application robustness study on 85 Apps
  - 60% of Apps can be automatically cracked
- Propose hardening scheme for In-App Billing
  - Practical solutions that target the App code itself

In-App Billing: Overview



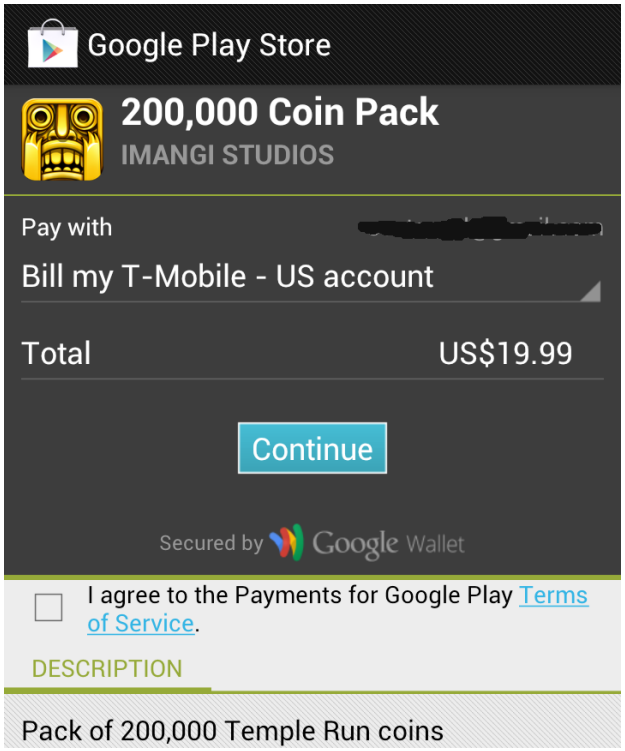
In-App Billing: Purchase Process

- App and Play Store (Market App) exchange messages
  - App tells Play Store which item is to be purchased



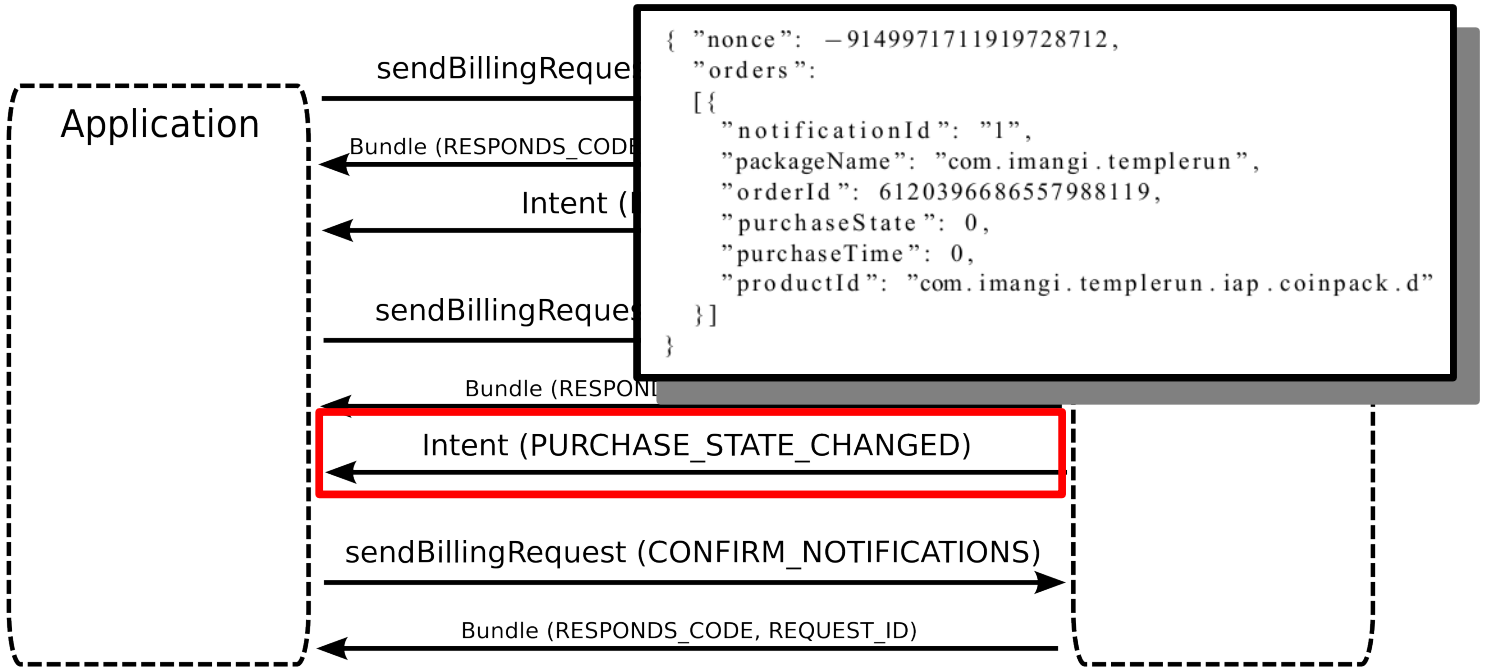
In-App Billing: Purchase Process

- Play Store asks user to complete purchase



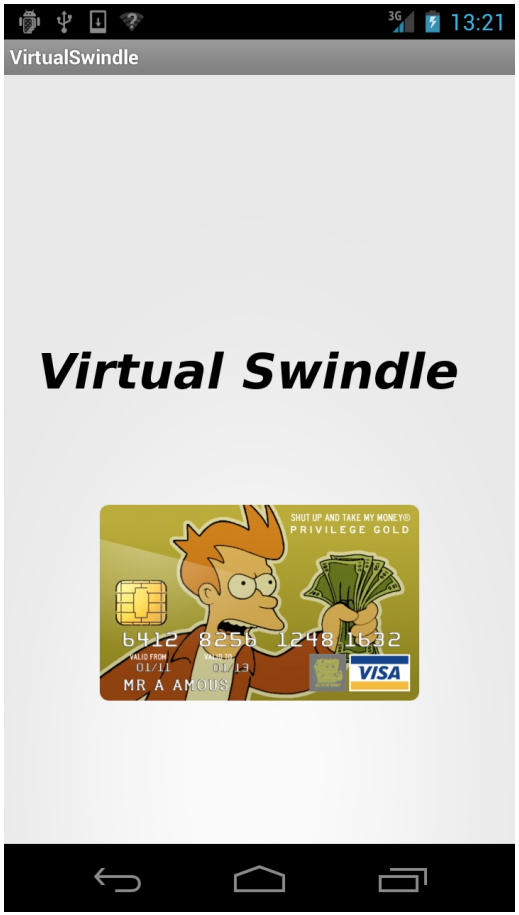
In-App Billing: Purchase Process

- Play Store notifies App that purchase is complete
  - Sends purchase data to App



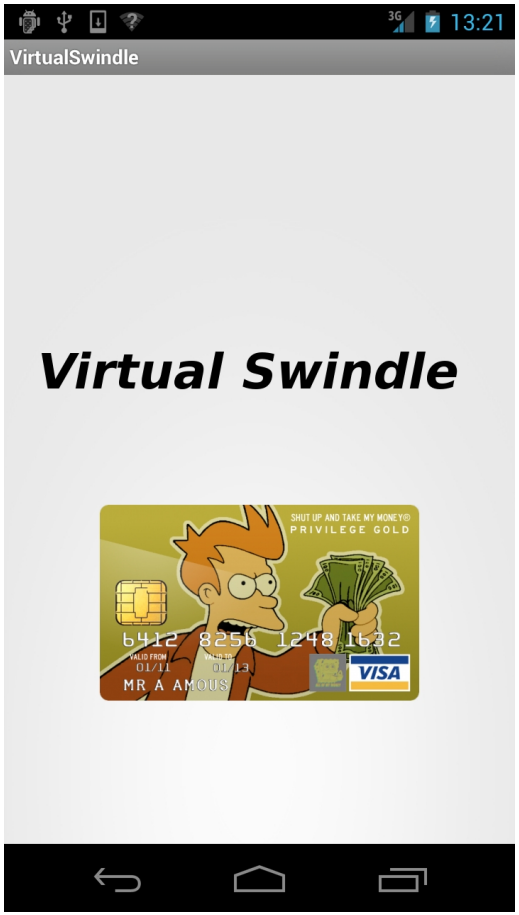
How to VirtualSwindle

- Emulate the Play Store
  - Reply to requests send by App
  - Confirm all actions
- Play Store runs in separate process
  - Communication via Binder RPC
- Communication via: sendBillingRequest()
  - Replace sendBillingRequest() in Play Store process



How to VirtualSwindle

- Emulate the Play Store
  - Reply to requests send by App
  - Confirm all actions
- Play Store runs in separate process
  - Communication via Binder RPC
- Communication via: sendBillingRequest()
  - Replace sendBillingRequest() in Play Store process
- **How to replace sendBillingRequest()?**
  - **Dynamic Dalvik Instrumentation**



Dynamic Dalvik Instrumentation (DDI)

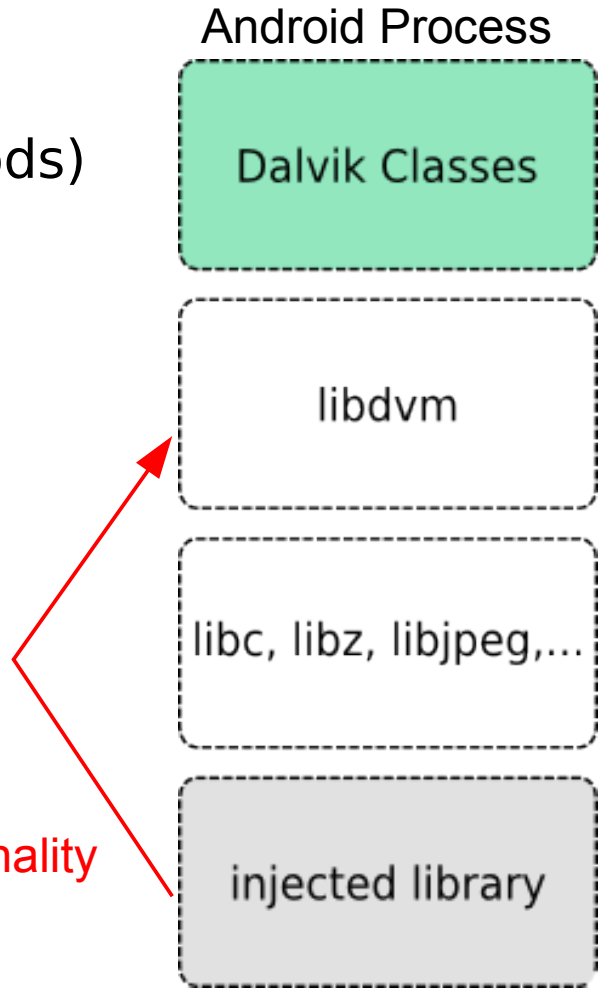
- Framework to instrument Dalvik code at runtime
  - Replace methods
  - Load additional Dalvik code (DEX classes)

- Main idea: convert Dalvik method to JNI method
  - JNI = Java Native Interface (native code)
  - Core functionally of the Dalvik Virtual Machine (DVM)

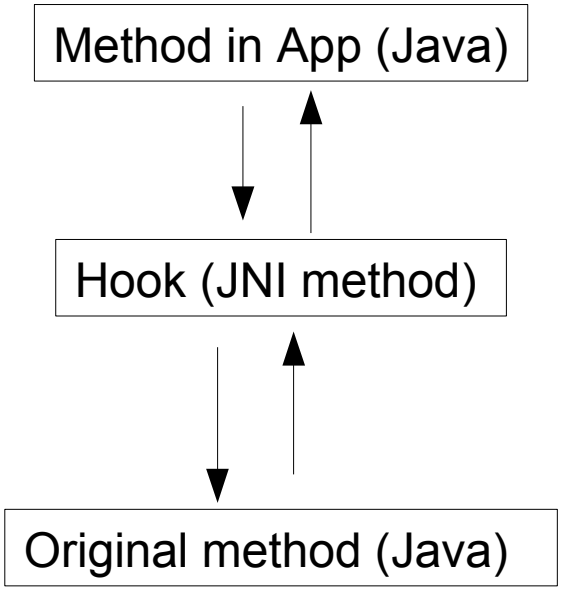
Dalvik Instrumentation: Overview

- Inject 'shared object' into running process
  - Provides the native code (JNI methods)

- Native code interacts with the DVM
  - Resolve symbols from DVM
  - Call DVM functions to:
    - Lookup classes and methods
    - Hook method
    - Call original method

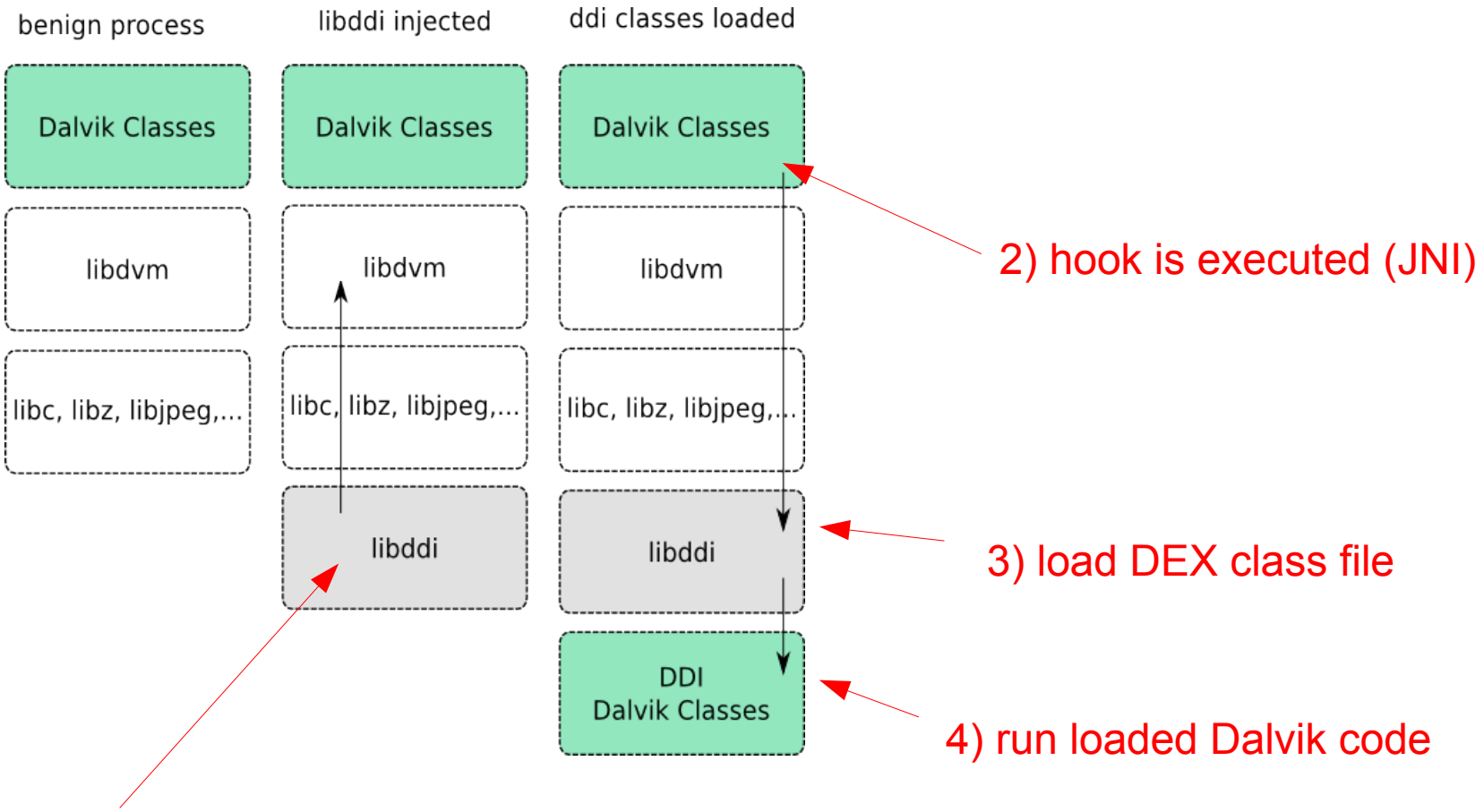


Instrumentation Code Flow (JNI only)

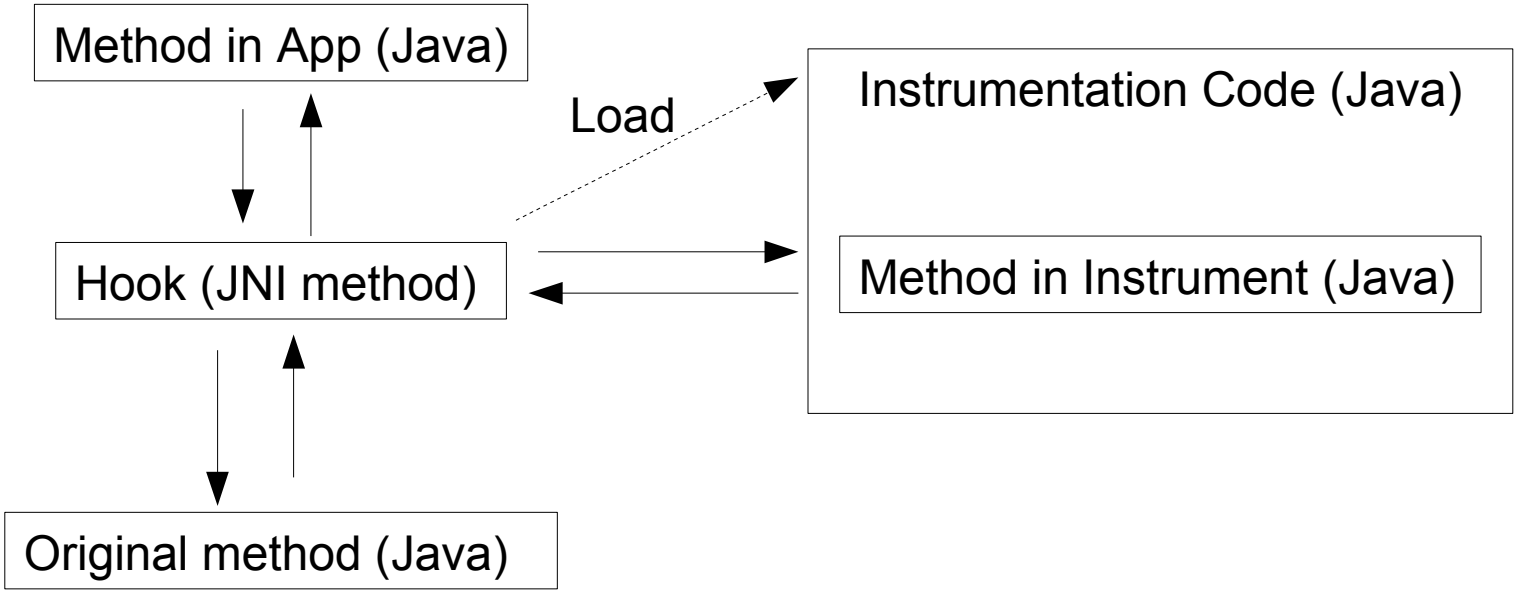


Hook is native code only. JNI can call any Java method.

Load and Run Dalvik Code



Instrumentation Code Flow (with Java code)



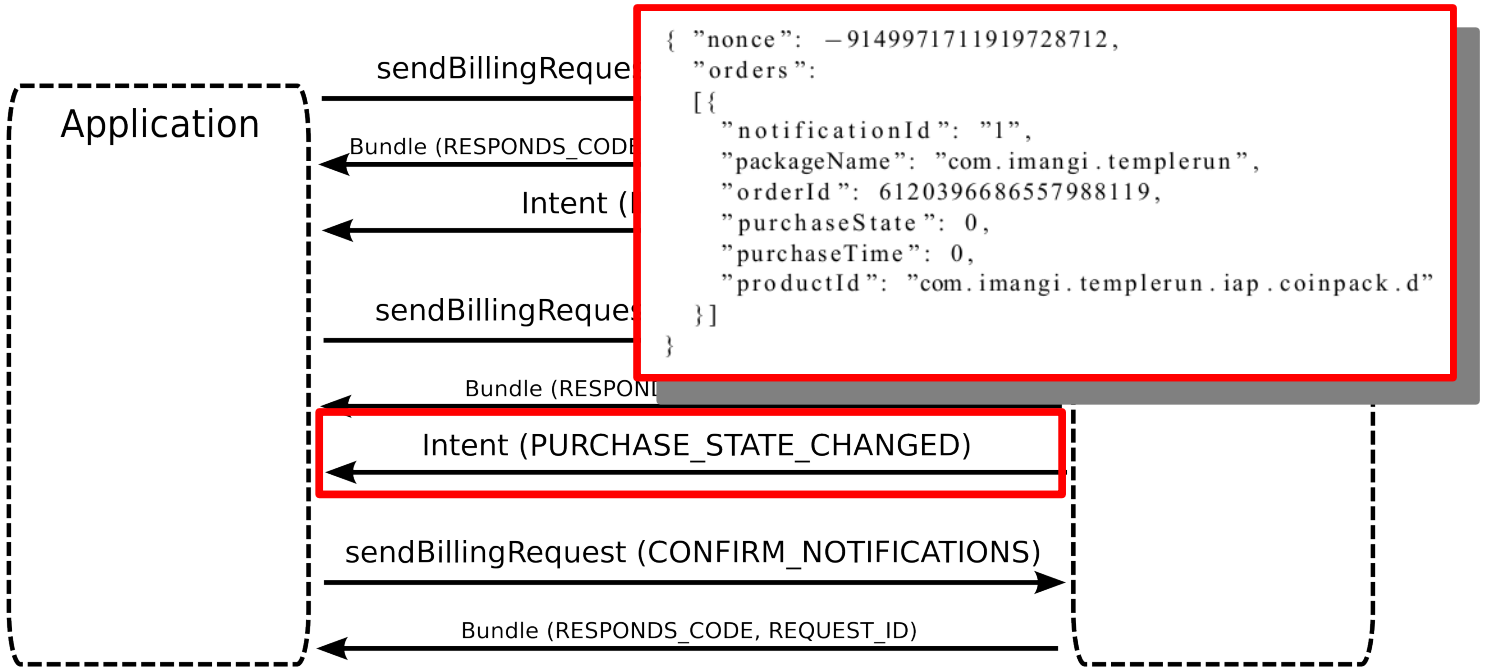
Instrumentation code can also be written in Java.

VirtualSwindle

- Hook sendBillingRequest() using DDI
  - Serves as entry point
- Main logic implemented in Java
  - DEX file loaded via DDI

VirtualSwindle: Purchase Process

- VirtualSwindle notifies App that purchase is complete
  - Creates fake purchase data and sends it to App



VirtualSwindle: Purchase Process

- VirtualSwindle notifies App that purchase is complete
  - Creates fake purchase data and sends it to App

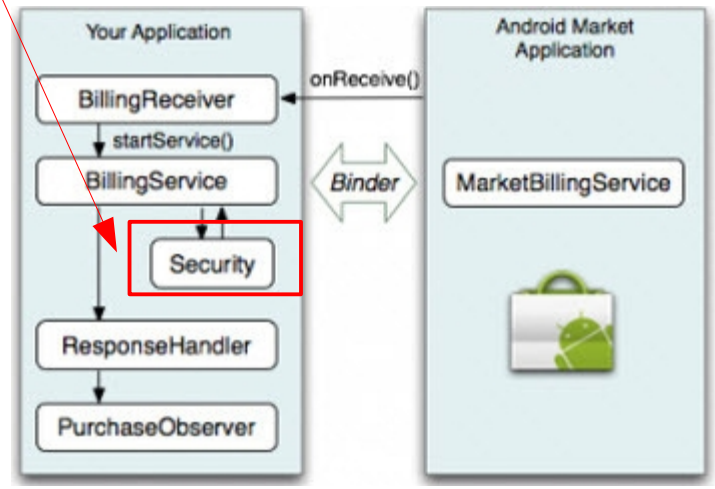
Purchase data is signed by Play Store (private key on Play Store server)
- App checks signature to determine if purchase data is benign
- This is the only security measure for In-App Billing

Bypassing the Signature Check

- Hook and replace: java.security.Signature.verify()
  - Our version always returns “true”

```
boolean java.security.Signature.verify(byte[]) { … }
```

```
int verify(JNIEnv *env, jobject obj, jobject bytearray)
{ return 1; }
```



Bypassing the Signature Check

- Hook and replace: java.security.Signature.verify()
  - Our version always returns “true”

```
boolean java.security.Signature.verify(byte[]) { … }
```

Verify() method is patched globally by instrumenting the zygote process
- zygote is the base VM process, all processes inherit changes made in zygote

Robustness Study

- Selected 85 Apps that support In-App Billing
  - Including many popular Apps
    - Angry Birds, Temple Run

  (more users → more interest in securing App?!)

- Attack Apps using VirtualSwindle
  - Group Apps: cracked and not cracked

- App Analysis (disassemble and inspect)
  - Code reuse (copy-paste Google's example code)
  - Countermeasures
  - Other interesting insights

Application Analysis

- Signature verification present in application?
  - Does App call Signature.verify() in billing code?
- Call to verify() not found but App was cracked
  - Check for Java reflection
- Obfuscation
  - Check if code was obfuscated (mangled symbol names)
- Check for network traffic
  - Check for transaction details in network traffic

Robustness Study: Results

- Analyzed 85 Apps
- 51 Apps (60%) automatically cracked
  - 7 Apps used reflection + obfuscation as defense
  - 1 App did not check signature of payment data
- 34 Apps not cracked
  - 32 Apps implemented server side checks
  - 1 App implemented the signature check in native code
  - 1 App detects failed payment manipulation and blocks
- Reuse of Google's example code (easy target for RE attack)
  - 38 of the vulnerable Apps
  - 15 of non vulnerable Apps

Countermeasures

- Server side signature verification
  - Server component needs to tightly integrated into App otherwise easy to remove
  - This is a lot of additional work

Countermeasures

- Server side signature verification
  - Server component needs to tightly integrated into App otherwise easy to remove
  - This is a lot of additional work

- Our solution: **harden on-device signature check**
  - Force attacker to reverse engineer and patch each individual App

- Package standalone signature verification with App
  - Obfuscate App and signature check code
  - Attacker cannot simply hook “verify()”

Summary

- Generic automated attack can bypassed payment

- Many Apps do not protect their payment code
  - Java reflection and obfuscation do not provide protection

- Super popular applications are vulnerable too
  - Angry Birds, Temple Run, …

Conclusions

- Mobile Apps → Money
  - In-App Billing is a major revenue source for Google and developers
- VirtualSwindle shows that billing code is often not protected
  - 60% of Apps are automatically cracked
  - No reverse engineering and patching of individual App
- Developers need to understand risks
  - Simple code obfuscation does not provide protection
- We show what countermeasures are effective
  - Provide an additional lightweight countermeasure

EOF

Thank you!

Any Questions?