Repackman: Automatic Repackaging of Android Apps

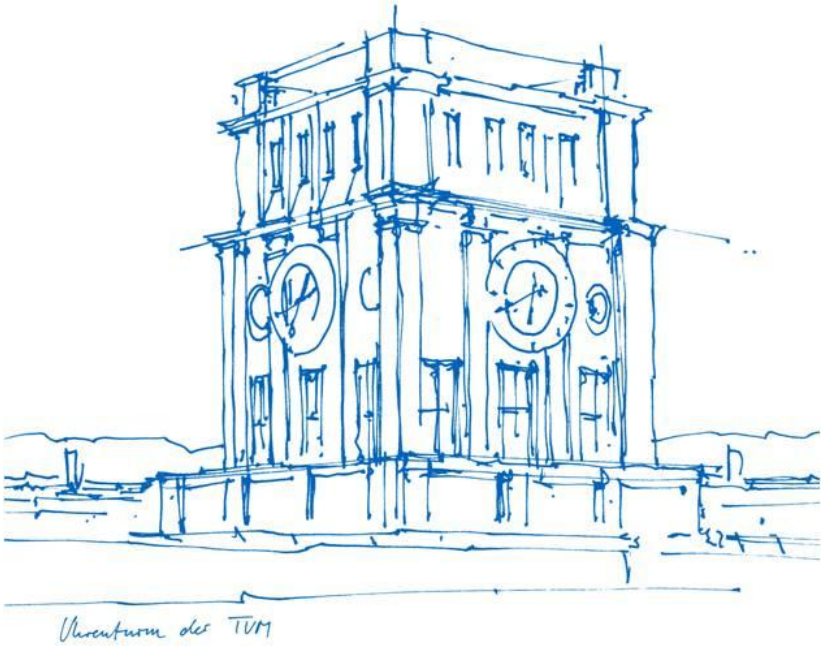
Aleieldin Salem, F. Franziska Paulus, Alexander Pretschner

Technische Universität München

Garching bei München

{salem, paulusf, pretschn @in.tum.de}

Montpellier, 04.09.2018



Abstract

- Repackman = Tool to repackage Android apps with arbitrary (malicious) payloads

Motivation #1

- Repackaging continues to pose a threat
  - Intellectual property
  - Reputational damage
  - Malware distribution

- Proactive vs. Reactive measures
  - Anti-repackaging techniques

- Need to repackage protected apps to evaluate techniques

- Automate repackaging for more comprehensive evaluation?

Motivation #2

- Repackaging continues to pose a threat
  - Intellectual property
  - Reputational damage
  - Malware distribution
- Proactive vs. Reactive measures
  - Repackaging/Malware detection
- Generate malicious, repackaged apps on demand
- Keep up with trends adopted by malware authors

Repackaging Example

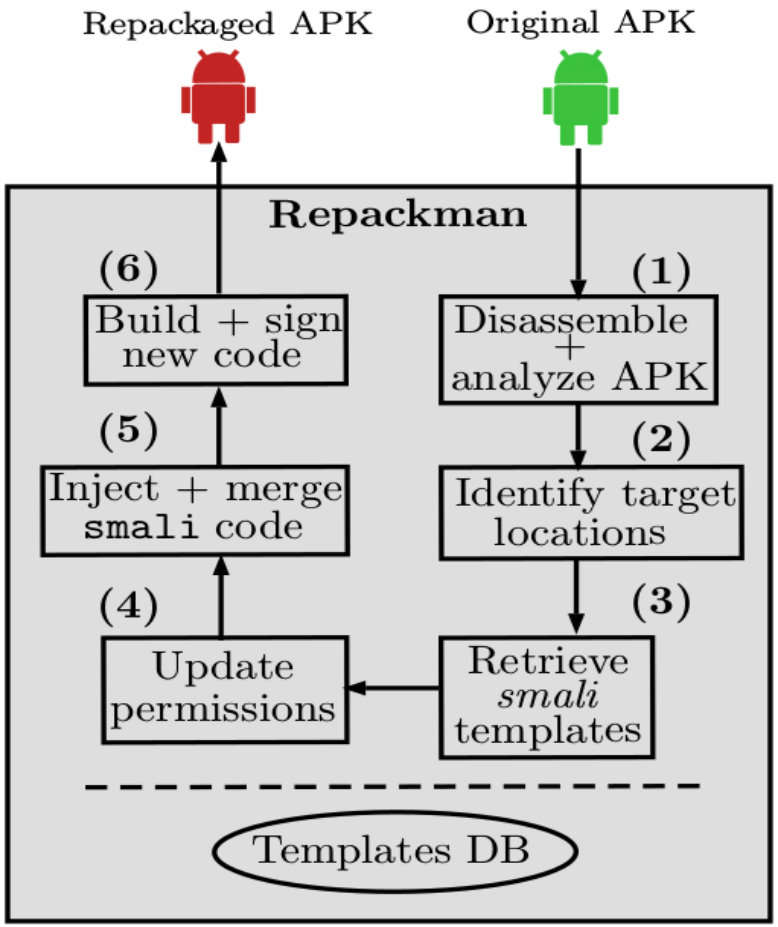
```
.method public sum(Landroid/view/View;)V
   .locals 3
   .prologue
   new-instance v1, Ljava/util/Random;
   invoke-direct {v1}, Ljava/util/Random;-><init>()V
   const/16 v2, 0xa
   invoke-virtual {v1,v2}, Ljava/util/Random;->nextInt(I)I
   move-result v1
   mul-int/lit8 v0, v1, 0x2
   .local v0, "out":I
   return-void
.end method
```

```
invoke-virtual p0, Lcom/test/app/Activity1; ...
...  ->getApplicationContext()Landroid/content/Context;
move-result-object v0
const-string v1, "Injected!!"
const/4 v2, 0x1
invoke-static {v0, v1, v2}, Landroid/widget/Toast;...
...->makeText(...)Landroid/widget/Toast;
move-result-object v0
invoke-virtual {v0}, Landroid/widget/Toast;->show()V
```

```
.method public sum(Landroid/view/View;)V
   .locals 6
   .prologue
   new-instance v1, Ljava/util/Random;
   invoke-direct {v1}, Ljava/util/Random;-><init>()V
   const/16 v2, 0xa
   invoke-virtual {v1,v2}, Ljava/util/Random;->nextInt(I)I
   move-result v1
   invoke-virtual p0, Lcom/test/app/Activity1; ...
   ...  ->getApplicationContext()Landroid/content/Context;
   move-result-object v3
   const-string v4, "Injected!!"
   const/4 v5, 0x1
   invoke-static {v3, v4, v5}, Landroid/widget/Toast;...
   ...->makeText(...)Landroid/widget/Toast;
   move-result-object v3
   invoke-virtual {v3}, Landroid/widget/Toast;->show()V
   mul-int/lit8 v0, v1, 0x2
   .local v0, "out":I
   return-void
.end method
```

Repackman: Overview

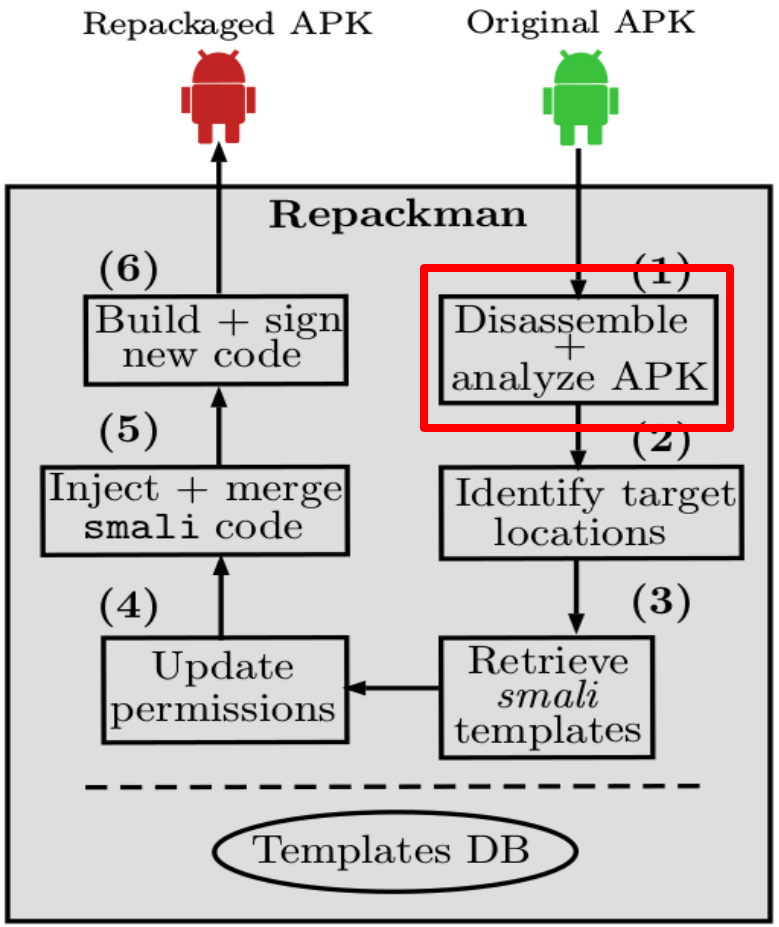
- Written in Python
- Multiple operations
  - Add Template
  - Delete Template
  - List Templates
  - **Repackage**
- Multiple deployment methods
- Support for execution triggers
- Source code: furnished upon request[1]



[1] https://github.com/tum-i22/Repackman

Repackman: Repackaging Process

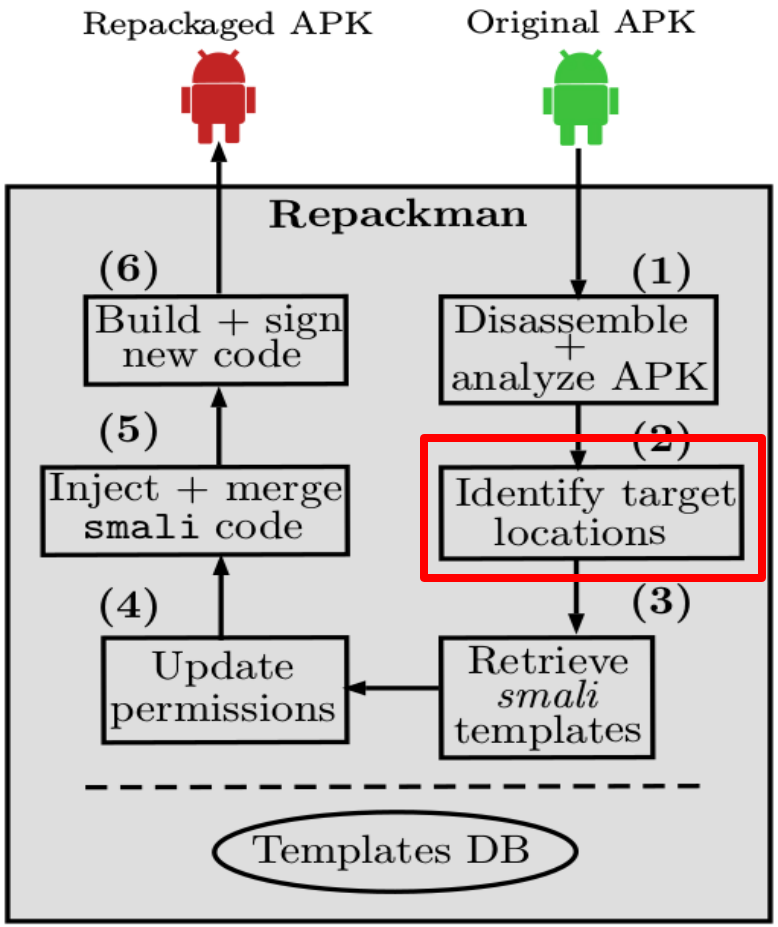
- Disassemble (baksmali) `classes.dex` using Apktool + analyze app using androguard
- Retrieve smali code
- Identify different components of the app (i.e., activities, services, receivers, etc.)



[1] https://github.com/tum-i22/Repackman

Repackman: Repackaging Process

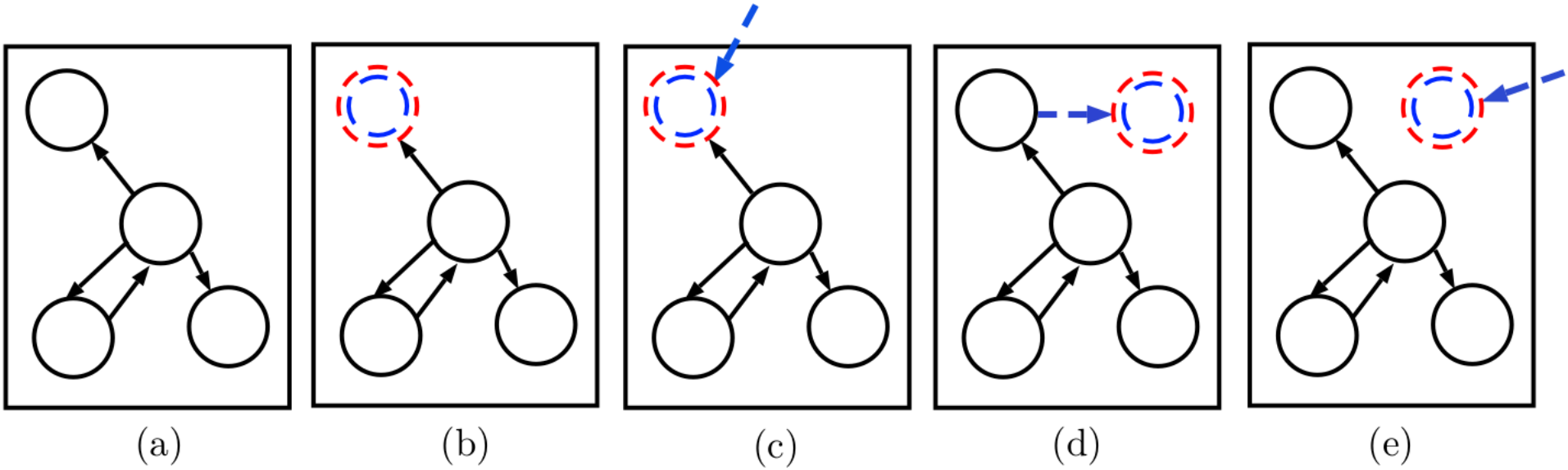
- Where and how to inject the malicious code?
- Deployment methods: specified by user



[1] https://github.com/tum-i22/Repackman

Repackman: Repackaging Process

- Where and how to inject the malicious code?
- Deployment methods: specified by user



Repackman: Repackaging Process

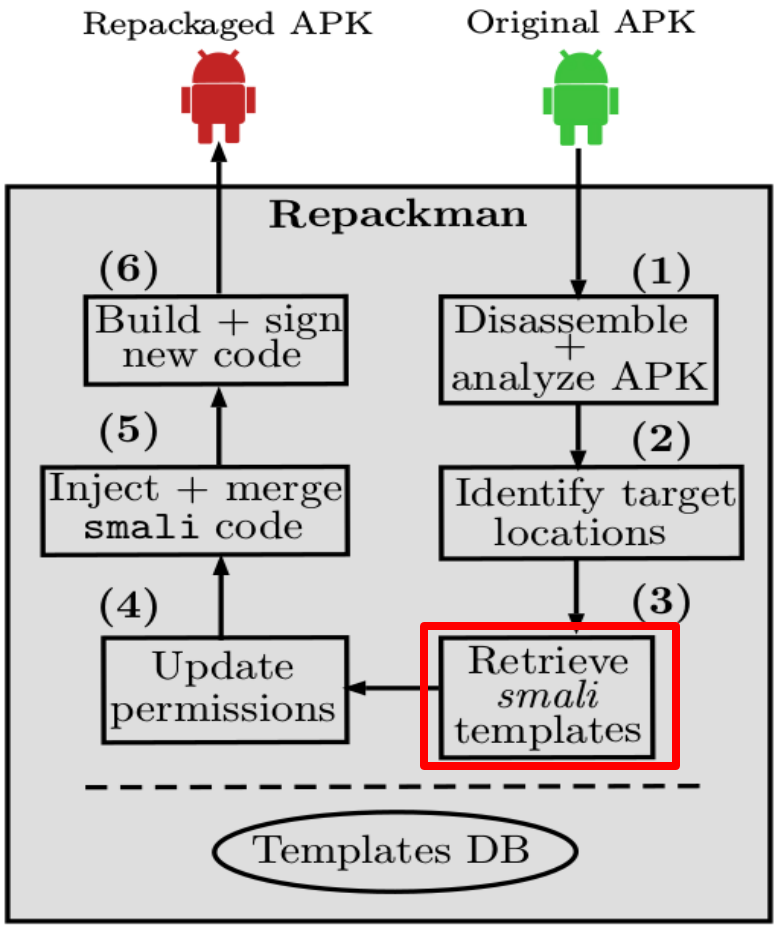
- Where and how to inject the malicious code?

```
usage: repackman.py repack [-h] [-a APK] [-i ID]
                           [-d {activity,service,receiver,random}]
                           [-t TRIGGER] [-v TRIGGERVAL] [-r {True,False}]
                           [-k KEY] [-m {True,False}] [-o OUTPUT]
                           [-p PASSPHRASE]

optional arguments:
  -h, --help            show this help message and exit
  -a APK, --apk APK     path to apk
  -i ID, --id ID        id of payload
  -d {activity,service,receiver,random}, --deploymentmethod {activity,service,receiver,random}
                        What type of component to deploy payload in. If random
                        is selected, a random, possible payload is selected as
                        well.
  -t TRIGGER, --trigger TRIGGER
                        trigger ID
  -v TRIGGERVAL, --triggerval TRIGGERVAL
                        value of trigger
  -r {True,False}, --randomize {True,False}
                        set to false to insert payload or payload call into
                        beginning of main class, set to true to insert into
                        random activity, method and line
  -k KEY, --key KEY     path to key
  -m {True,False}, --forcemethod {True,False}
                        force payload or payload call into separate method
  -o OUTPUT, --output OUTPUT
                        path of repackaged apk, default:
                        ./out/[package_name]_repackaged.apk
  -p PASSPHRASE, --passphrase PASSPHRASE
                        passphrase to sign apk
```

Repackman: Repackaging Process

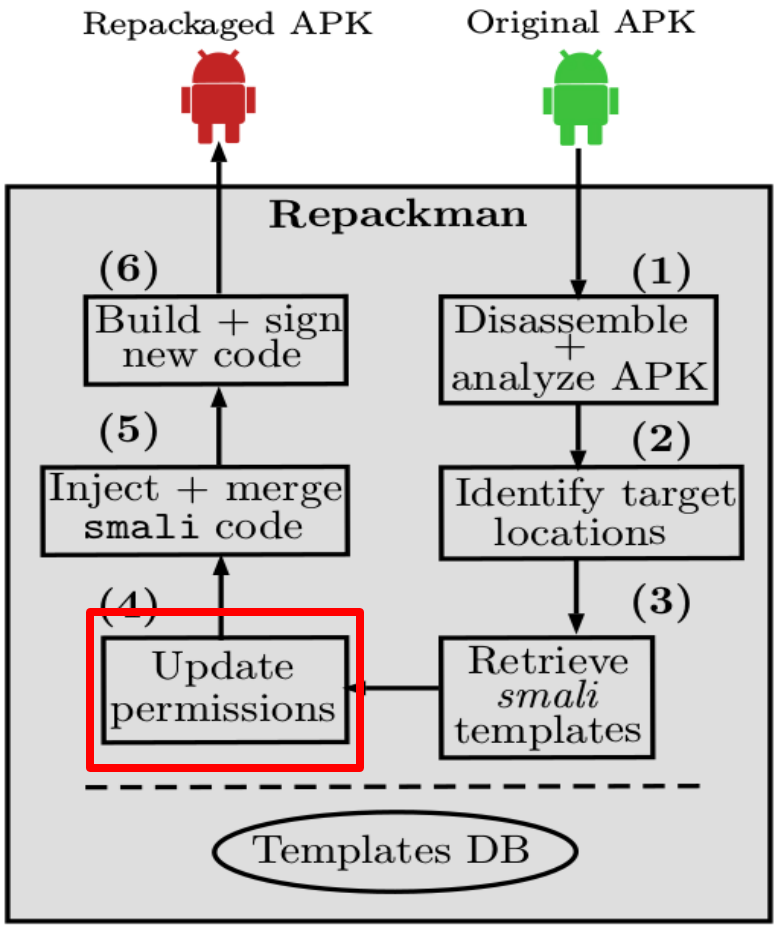
- Load trigger(s) and payload(s)
- Stored as `smali` text files
- Written and added by user to DB



[1] https://github.com/tum-i22/Repackman

Repackman: Repackaging Process

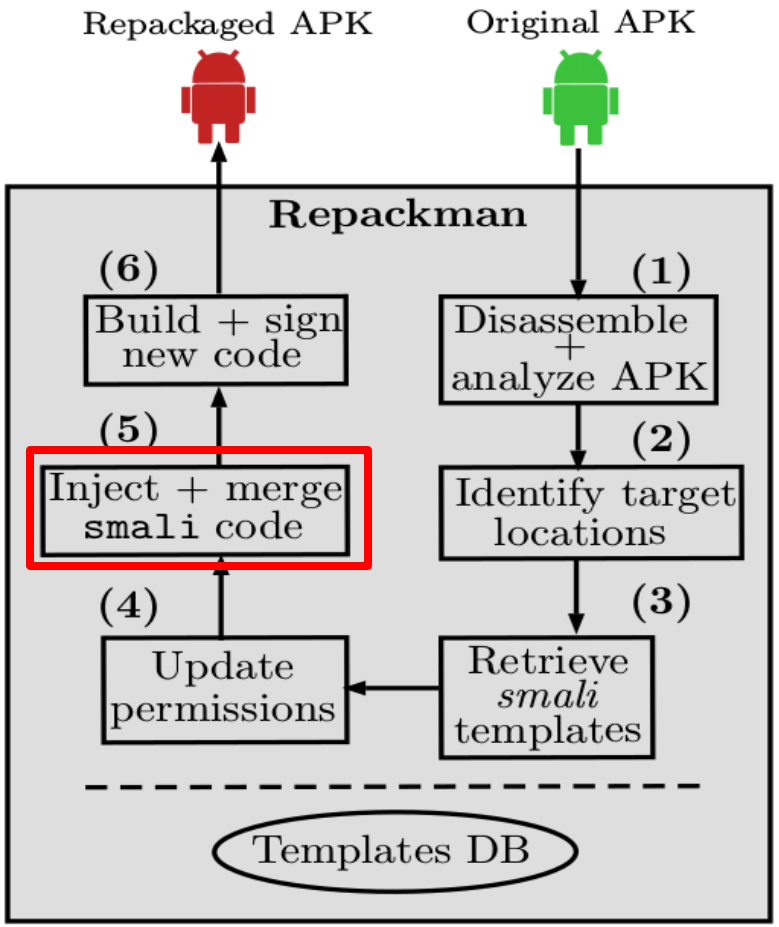
- Add any new components
- Some triggers/payloads need new permissions
- Update `AndroidManifest.xml` file
- Make sure to merge components + permissions



[1] https://github.com/tum-i22/Repackman

Repackman: Repackaging Process

- Merge retrieved templates with original code
- Couple of concerns:
  - Maintain integrity of original code
  - Only 16 registers allowed as variables v0—v15
  - What if we run out of variables?



[1] https://github.com/tum-i22/Repackman

Repackman: Repackaging Process

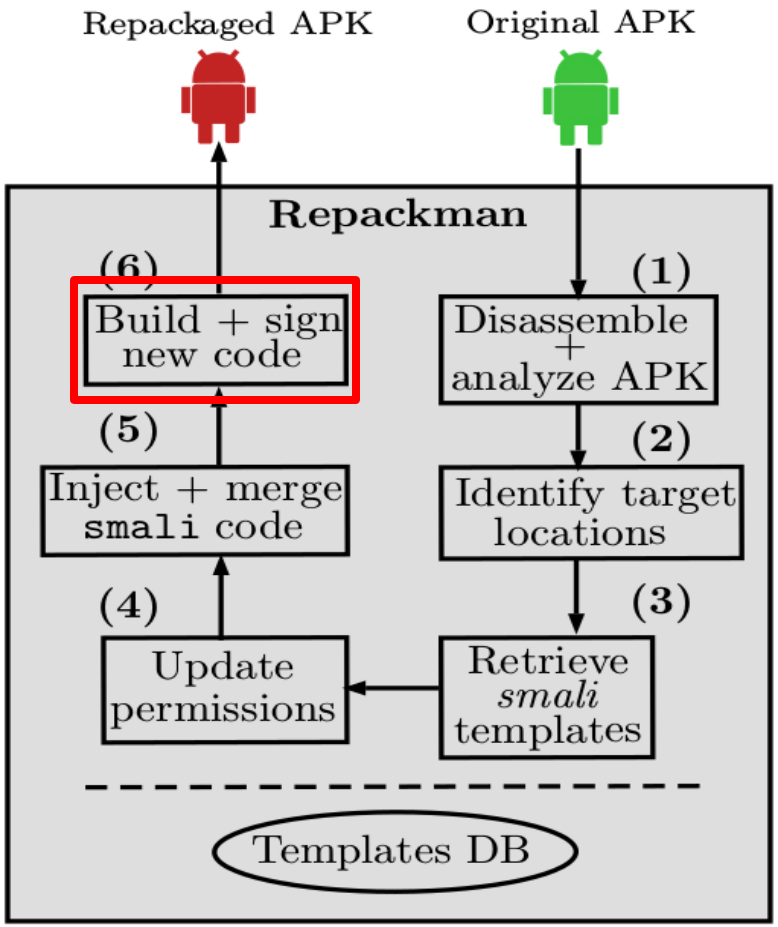
- Where and how to inject the malicious code?

```
usage: repackman.py repack [-h] [-a APK] [-i ID]
                           [-d {activity,service,receiver,random}]
                           [-t TRIGGER] [-v TRIGGERVAL] [-r {True,False}]
                           [-k KEY] [-m {True,False}] [-o OUTPUT]
                           [-p PASSPHRASE]

optional arguments:
  -h, --help            show this help message and exit
  -a APK, --apk APK     path to apk
  -i ID, --id ID        id of payload
  -d {activity,service,receiver,random}, --deploymentmethod {activity,service,receiver,random}
                        What type of component to deploy payload in. If random
                        is selected, a random, possible payload is selected as
                        well.
  -t TRIGGER, --trigger TRIGGER
                        trigger ID
  -v TRIGGERVAL, --triggerval TRIGGERVAL
                        value of trigger
  -r {True,False}, --randomize {True,False}
                        set to false to insert payload or payload call into
                        beginning of main class, set to true to insert into
                        random activity, method and line
  -k KEY, --key KEY     path to key
  -m {True,False}, --forcemethod {True,False}
                        force payload or payload call into separate method
  -o OUTPUT, --output OUTPUT
                        path of repackaged apk, default:
                        ./out/[package_name]_repackaged.apk
  -p PASSPHRASE, --passphrase PASSPHRASE
                        passphrase to sign apk
```

Repackman: Repackaging Process

- Recompile with Apktool
- Sign with your own key
- Voilà!



[1] https://github.com/tum-i22/Repackman

Repackman: Evaluation

- Investigating:
  a) The feasibility and reliability of the repackaging process, and
  b) Any noticeable side effects on the original apps' functionalities and appearance?

- Dataset (97 presumably benign apps):
  - Initially downloaded 150 (Top Free) apps from Google Play
  - Ruled out apps that …
    - require account creation (e.g., Facebook),
    - could not be disassembled via Apktool,
    - crashed on the emulator Genymotion

Repackman: Evaluation

- Experiment 1 (*The feasibility and reliability of the repackaging process*):
  - Repackaged each app using all deployment methods currently supported by the tool (i.e., four repackaged versions / app)
  - Ran apps using Droidutan
  - Recorded:
    - The number of apps that were successfully repackaged
    - The number of apps that did not crash during runtime.

Repackman: Evaluation

- Experiment 1 (*The feasibility and reliability of the repackaging process*):

	Forced Execution	Random Activity	New Service	New Receiver	Average (A+S+R)
Repackaged Successfully	88 (91%)	87 (89%)	92 (95%)	89 (92%)	268 (92%)
Executed Successfully	84 (86%)	87 (89%)	90 (93%)	89 (92%)	266 (91%)

Repackman: Evaluation

- Experiment 2 (*Any noticeable side effects on the original apps…*):
  - Defined in terms of:
    - Size (in KB)
    - Time (in seconds)
    - Difference in appearance (in SSIM)
  - Run repackaged app using same “test case” + take screenshot after each action (e.g., Button tap)

Repackman: Evaluation

- Experiment 2 (*Any noticeable side effects on the original apps…*):

	Forced Execution	Random Activity	New Service	New Receiver	Average
SSIM difference	0.81	0.79	0.77	0.77	0.785
Time difference (seconds)	0.266	0.228	0.21	0.224	0.232
Size difference (KB)	20	20.1	19.6	19.8	19.875

Conclusion

- Implemented Repackman, a tool to automatically repackage Android apps with arbitrary (malicious) payloads.

- Repackman successfully repackaged least 86% of the Android apps we gathered from Google Play with arbitrary payloads

- No noticeable side effects on the user UI experience, app performance, or app size.

Enhancement(s)

- Repackman needs to be continuously updated to incorporate the latest repackaging trends.
- Support the injection of payloads as native libraries developed in C/C++.
- Add new types of triggers including those that trigger payloads upon receiving system notifications (e.g., BOOT_COMPLETED).
- Automate process of template creation
- Multiple triggers/payloads per app
- Add interactive shell to interact with Repackman's functionalities.

Thank You

Any questions?