

From Large Scale Analysis to Dynamic Deobfuscation

The Woes of Android Reverse Engineering

MINEAU Jean-Marie

LALANDE Jean-François, PhD supervisor

VIET TRIEM TONG Valérie, PhD co-supervisor

2025-12-09



CentraleSupélec

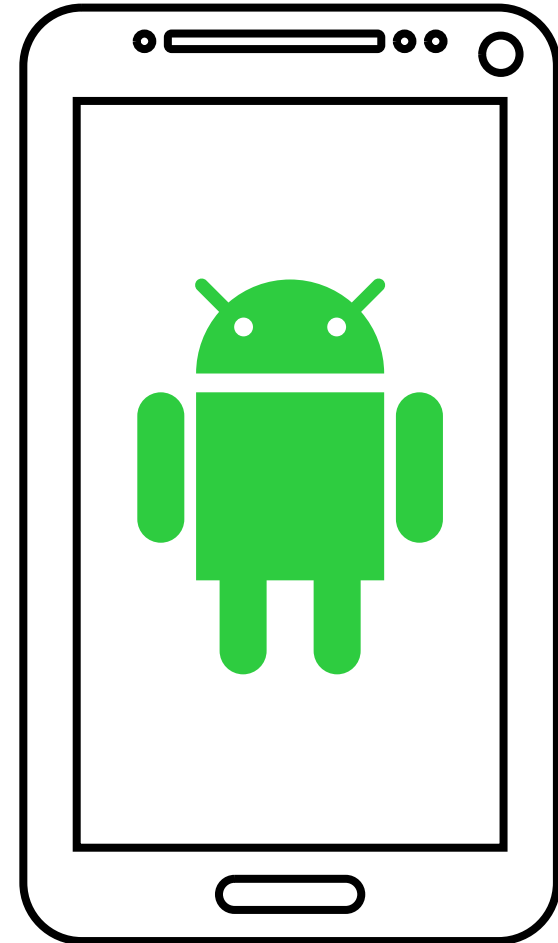




Smartphones are computers

Android = Linux + Android Runtime
(ART)

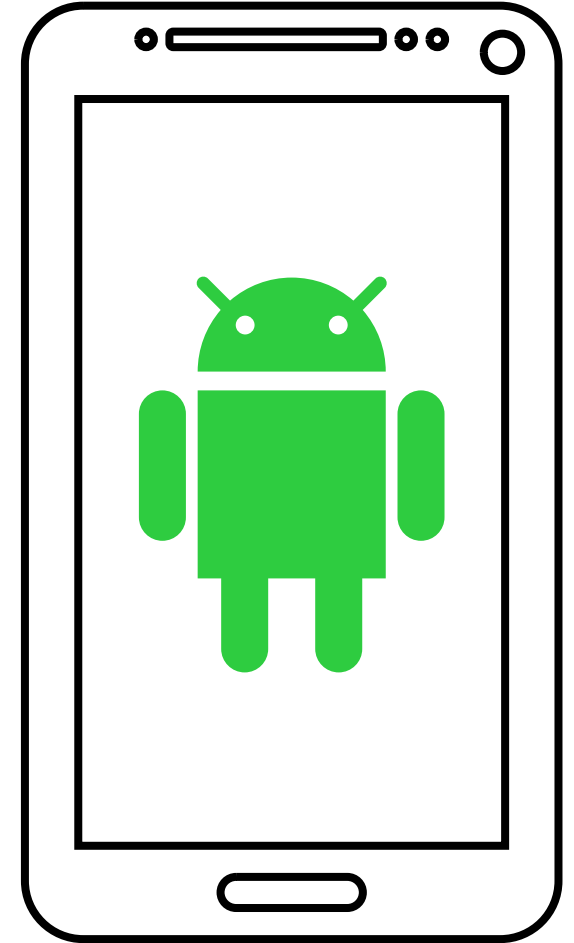
APK = computer program (Java-ish)



Personal Data and PII



Ransomware/Spyware



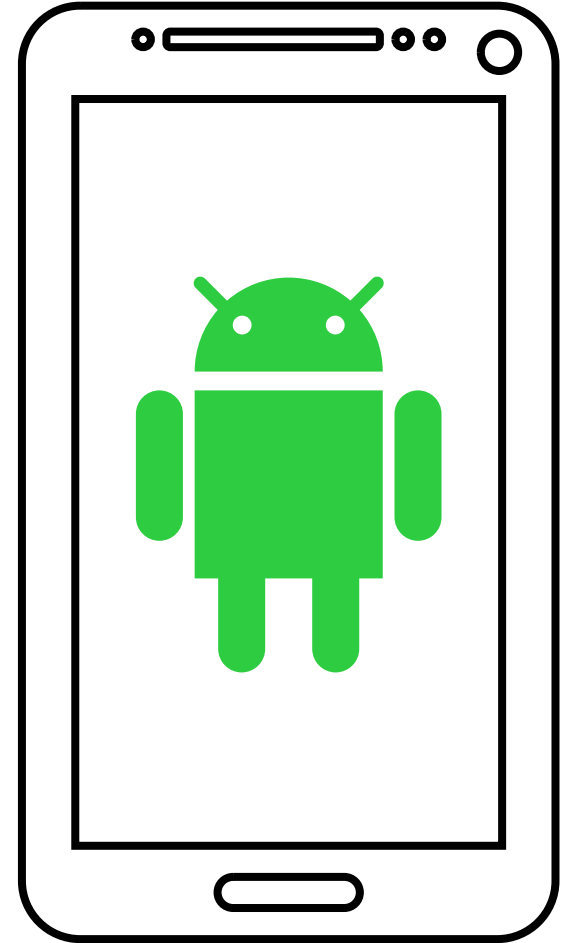
Personal Data and PII

Computing Power



Ransomware/Spyware

Cryptojacker



Personal Data and PII

Computing Power

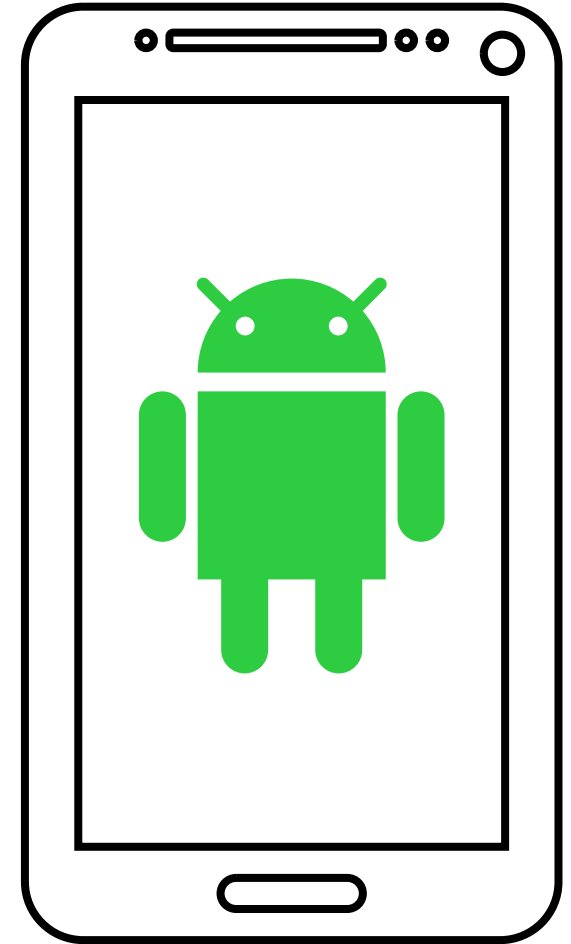
Phone



Ransomware/Spyware

Cryptojacker

Expander (phone billing)



Personal Data and PII

Computing Power

Phone

Mic, Camera,
Geolocalisation

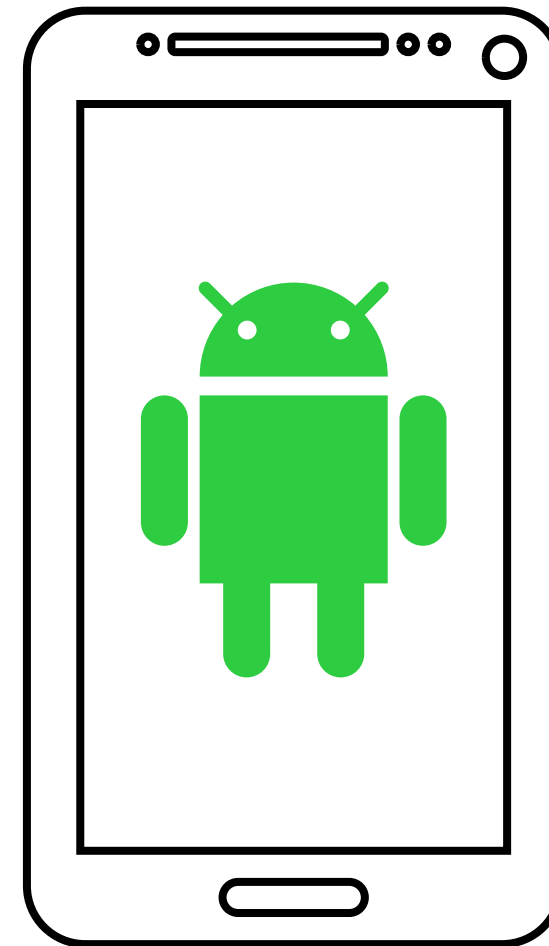


Ransomware/Spyware

Cryptojacker

Expander (phone billing)

Stalkerware



Obfuscation

Applications might use **obfuscation** to either:

- protect their Intellectual Property
- hide malicious behaviour

Obfuscation

Applications might use **obfuscation** to either:

- protect their Intellectual Property
- hide malicious behaviour

We will focus on two techniques:

- **Dynamic Code Loading**
- **Reflection**

Obfuscation — EXAMPLE



```
1 String DEX = "ZGV4CjA [...] EAAABEAwAA";
2 String className = "W5f3 [...] 3sls=";
3 String methodName = "n6WGYJzjDrUvR9cYljlnlw==";
4
5 ClassLoader cl = new InMemoryDexClassLoader(
6     ByteBuffer.wrap(Base64.decode(DEX, 2)),
7     Main.class.getClassLoader()
8 );
9
10 Class<?> loadedClass = this.cl.loadClass(decrypt(className));
11 Object obj = "FooBar";
12 Object ret = loadedClass.getMethod(
13     decrypt(methodName),
14     String.class
15 ).invoke(null, obj);
```

Obfuscation — DYNAMIC CODE LOADING



```
1 String DEX = "ZGV4CjA [...] EAAABEAwAA";
2 String className = "W5f3 [...] 3sls=";
3 String methodName = "n6WGYJzjDrUvR9cYljlnlw==";
4
5 ClassLoader cl = new InMemoryDexClassLoader(
6     ByteBuffer.wrap(Base64.decode(DEX, 2)),
7     Main.class.getClassLoader()
8 );
9
10 Class<?> loadedClass = this.cl.loadClass(decrypt(className));
11 Object obj = "FooBar";
12 Object ret = loadedClass.getMethod(
13     decrypt(methodName),
14     String.class
15 ).invoke(null, obj);
```

Obfuscation — REFLECTION



```
1 String DEX = "ZGV4CjA [...] EAAABEAwAA";
2 String className = "W5f3 [...] 3sls=";
3 String methodName = "n6WGYJzjDrUvR9cYljlnlw==";
4
5 ClassLoader cl = new InMemoryDexClassLoader(
6     ByteBuffer.wrap(Base64.decode(DEX, 2)),
7     Main.class.getClassLoader()
8 );
9
10 Class<?> loadedClass = this.cl.loadClass(decrypt(className));
11 Object obj = "FooBar";
12 Object ret = loadedClass.getMethod(
13     decrypt(methodName),
14     String.class
15 ).invoke(null, obj);
```

Obfuscation — REFLECTION



```
1 String DEX = "ZGV4CjA [...] EAAABEAwAA";
2 String className = "W5f3 [...] 3sls=";
3 String methodName = "n6WGYJzjDrUvR9cYljlnlw==";
4
5 ClassLoader cl = new InMemoryDexClassLoader(
6     ByteBuffer.wrap(Base64.decode(DEX, 2)),
7     Main.class.getClassLoader()
8 );
9
10 Class<?> loadedClass = this.cl.loadClass(decrypt(className));
11 Object obj = "FooBar";
12 Object ret = loadedClass.getMethod(
13     decrypt(methodName),
14     String.class
15 ).invoke(null, obj);
```

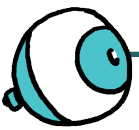
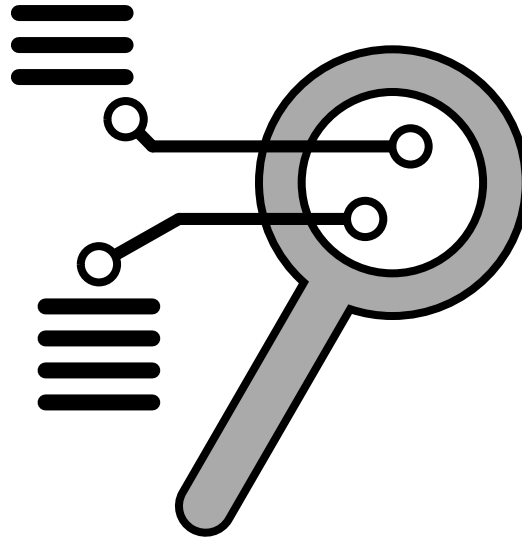
Obfuscation — DEOBFUSCATED

```
1 String DEX = "ZGV4CjA [...] EAAABEAwAA";
2 String className = "W5f3 [...] 3sIs=";
3 String methodName = "n6WGYJzjDrUvR9cYljINlw==";
4
5 ClassLoader cl = new InMemoryDexClassLoader(
6     ByteBuffer.wrap(Base64.decode(DEX, 2)),
7     Main.class.getClassLoader()
8 );
9
10 Class<?> loadedClass =
11     this.cl.loadClass(decrypt(className));
12 Object obj = "FooBar";
13 Object ret = loadedClass.getMethod(
14     decrypt(methodName),
15     String.class
16 ).invoke(null, obj);
```

```
1 public class Foo {
2     public static String
3     bar(String arg) {
4         ...
5         ...
6     }
7 }
```

```
1 String ret =
2     Foo.bar("FooBar");
```

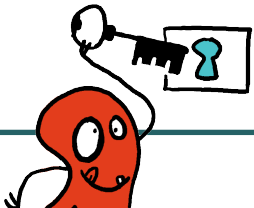
Analysis Methods



Dynamic Analysis

Run the application

Static Analysis

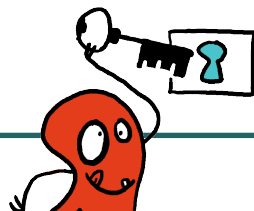


Dynamic Analysis

Run the application

See dynamically loaded bytecode

Static Analysis



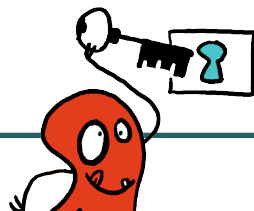
Dynamic Analysis

Run the application

See dynamically loaded bytecode

See reflection calls

Static Analysis



Dynamic Analysis

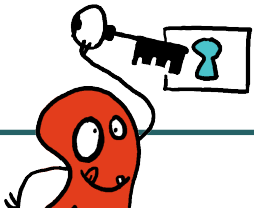
Run the application

See dynamically loaded bytecode

See reflection calls

Limited by code coverage

Static Analysis



Dynamic Analysis

Run the application

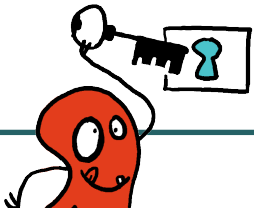
See dynamically loaded bytecode

See reflection calls

Limited by code coverage

Static Analysis

Do **not** run the application



Dynamic Analysis

Run the application

See dynamically loaded bytecode

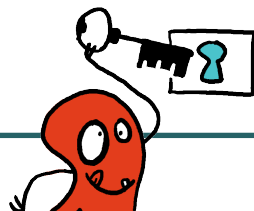
See reflection calls

Limited by code coverage

Static Analysis

Do **not** run the application

Not limited by code coverage



Dynamic Analysis

Run the application

See dynamically loaded bytecode

See reflection calls

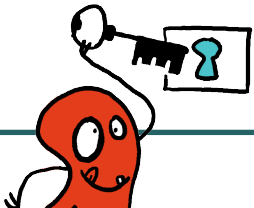
Limited by code coverage

Static Analysis

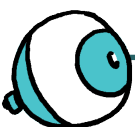
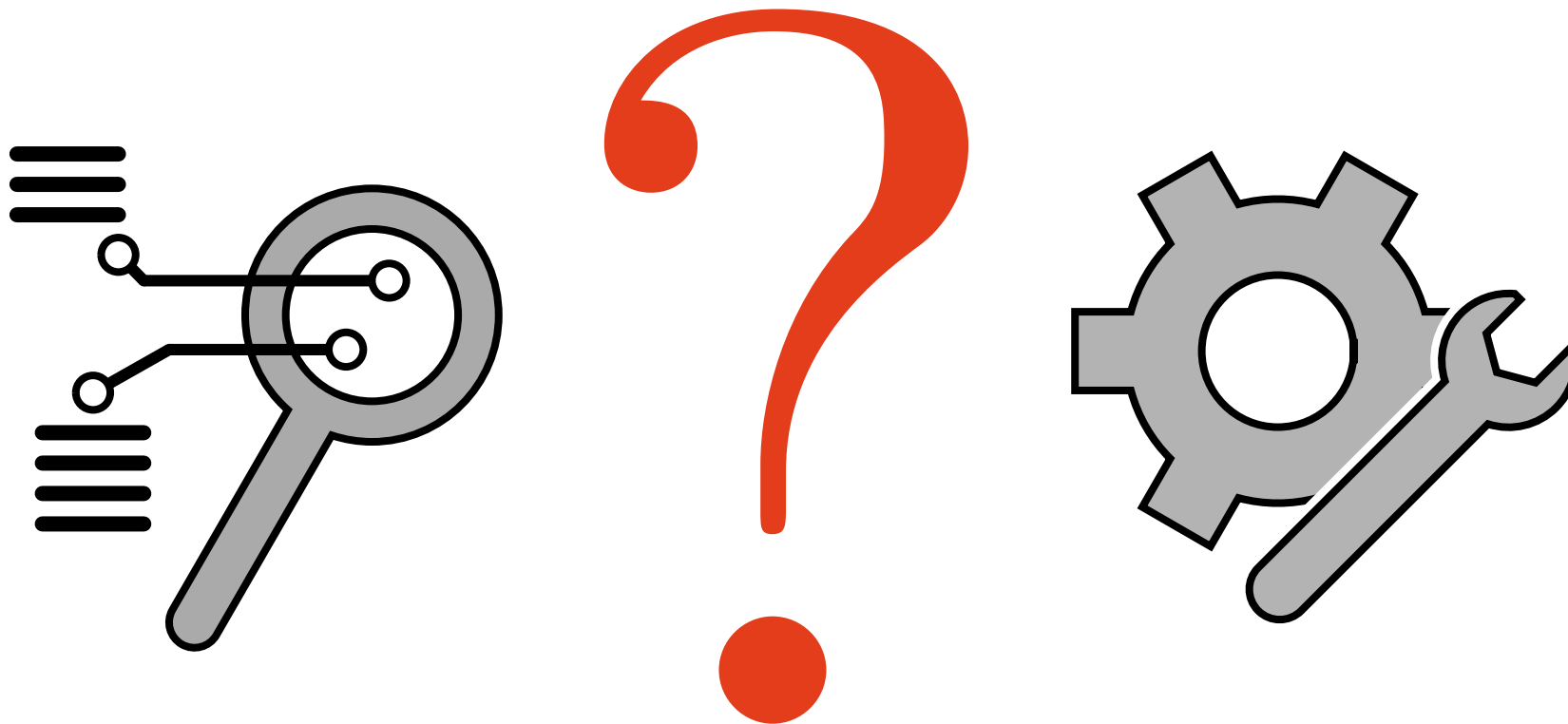
Do **not** run the application

Not limited by code coverage

But only for the **code available**



Which Tools are *really* Working?



Problem Statement 1

Which static analysis tool to use?

To what extent are previously published Android analysis tools still usable today, and what factors impact their reusability?

Problem Statement 1

Which static analysis tool to use?

Are they easy to install?

To what extent are previously published Android analysis tools still usable today, and what factors impact their reusability?

Problem Statement 1

Which static analysis tool to use?

Are they easy to install?

Are they working?

To what extent are previously published Android analysis tools still usable today, and what factors impact their reusability?

Class Loading

```
1  ClassLoader cl = new InMemoryDexClassLoader(  
2      ByteBuffer.wrap(Base64.decode(DEX, 2)),  
3      Main.class.getClassLoader()  
4  );  
5  
6  Class<?> loadedClass = this.cl.loadClass(decrypt(className));
```

Class Loading

```
1 ClassLoader cl = new InMemoryDexClassLoader(  
2     ByteBuffer.wrap(Base64.decode(DEX, 2)),  
3     Main.class.getClassLoader()  
4 );  
5  
6 Class<?> loadedClass = this.cl.loadClass(decrypt(className));
```

```
1 class A {  
2     public static void foo() {  
3         B b = new B();  
4         b.bar();  
5     }  
6 }
```

Where is the class loader?

Problem Statement 2

Used to select classes implementation

What is the default Android class loading algorithm, and does it impact static analysis?

Problem Statement 2

Used to select classes implementation

More complex than it looks

What is the default Android class loading algorithm, and does it impact static analysis?

Problem Statement 2

Used to select classes implementation

More complex than it looks

Doubious documentation

What is the default Android class loading algorithm, and does it impact static analysis?

Problem Statement 2

Used to select classes implementation

More complex than it looks

Doubious documentation

Not studied in the context of Android Static Analysis

What is the default Android class loading algorithm, and does it impact static analysis?

Deobfuscation

Dynamic Analysis

Easier to solve Dynamic Code
Loading and Reflection Calls

Static Analysis

Better code coverage

Problem Statement 3

Dynamic analysis is good against DCL and reflection

Can we use instrumentation to provide dynamic code loading and reflection data collected dynamically to static analysis tools and improve their results?

Problem Statement 3

Dynamic analysis is good against DCL and reflection

Dynamic analysis is limited by code coverage

Can we use instrumentation to provide dynamic code loading and reflection data collected dynamically to static analysis tools and improve their results?

Problem Statement 3

Dynamic analysis is good against DCL and reflection

Dynamic analysis is limited by code coverage

Static analysis is not

Can we use instrumentation to provide dynamic code loading and reflection data collected dynamically to static analysis tools and improve their results?

Problem Statement 3

Dynamic analysis is good against DCL and reflection

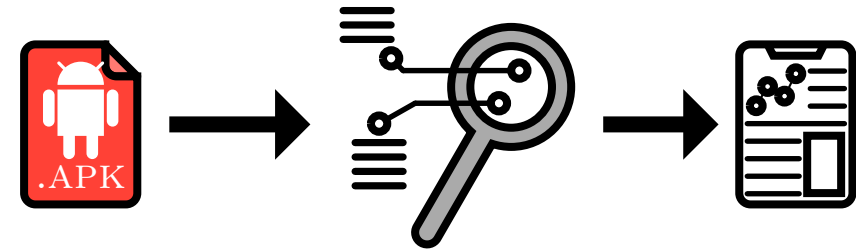
Dynamic analysis is limited by code coverage

Static analysis is not

How to use existing tools without modifying them?

Can we use instrumentation to provide dynamic code loading and reflection data collected dynamically to static analysis tools and improve their results?

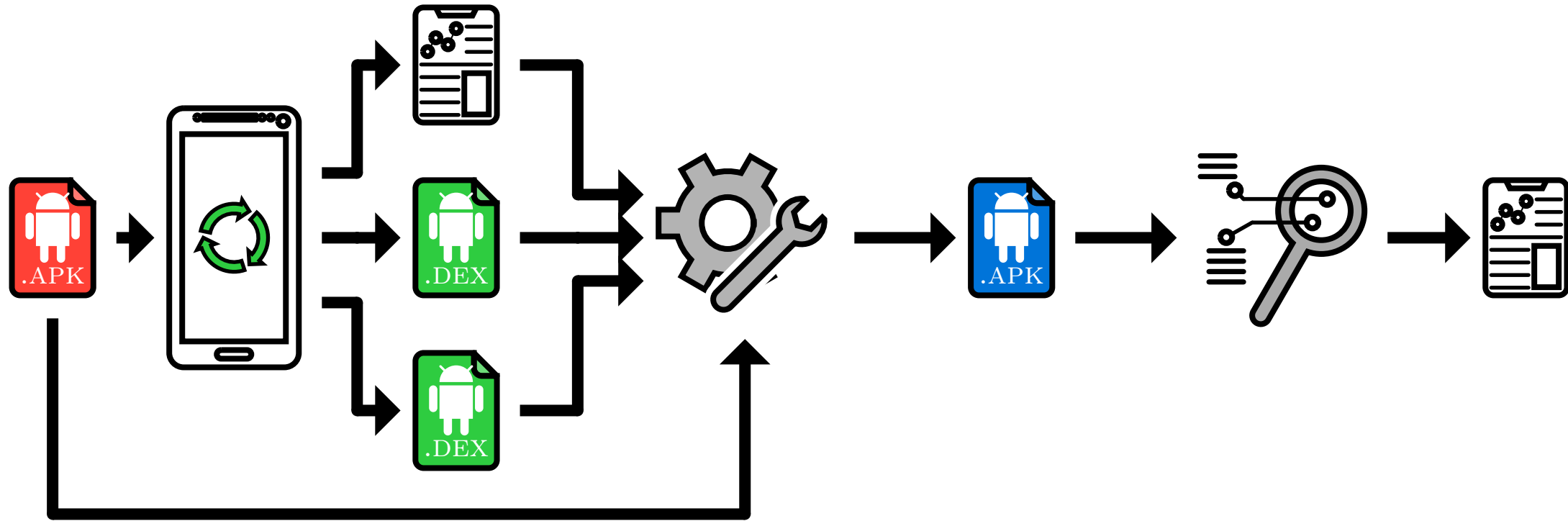
PB 1: Static Analysis
are the tools working?



PB 2: Do Static Tools
match Android Runtime?



PB 3: Improve any Static Tools with dynamic analysis?



Tool Reusability

State of the Art

Li *et al.* (2017):

Systematic literature review for Android static analysis (38 tools)

State of the Art

Li *et al.* (2017):

Systematic literature review for Android static analysis (38 tools)

Lists 31 open-sourced tools

State of the Art

Li *et al.* (2017):

Systematic literature review for Android static analysis (38 tools)

Lists 31 open-sourced tools

Does not test the tools

State of the Art

Li *et al.* (2017):

- Systematic literature review for Android static analysis (38 tools)

- Lists 31 open-sourced tools

- Does not test the tools

Reaves *et al.* (2016):

- Tests 7 Android analysis tools

State of the Art

Li *et al.* (2017):

- Systematic literature review for Android static analysis (38 tools)

- Lists 31 open-sourced tools

- Does not test the tools

Reaves *et al.* (2016):

- Tests 7 Android analysis tools

- Tests analysing 16 real-world applications

State of the Art

Li *et al.* (2017):

- Systematic literature review for Android static analysis (38 tools)

- Lists 31 open-sourced tools

- Does not test the tools

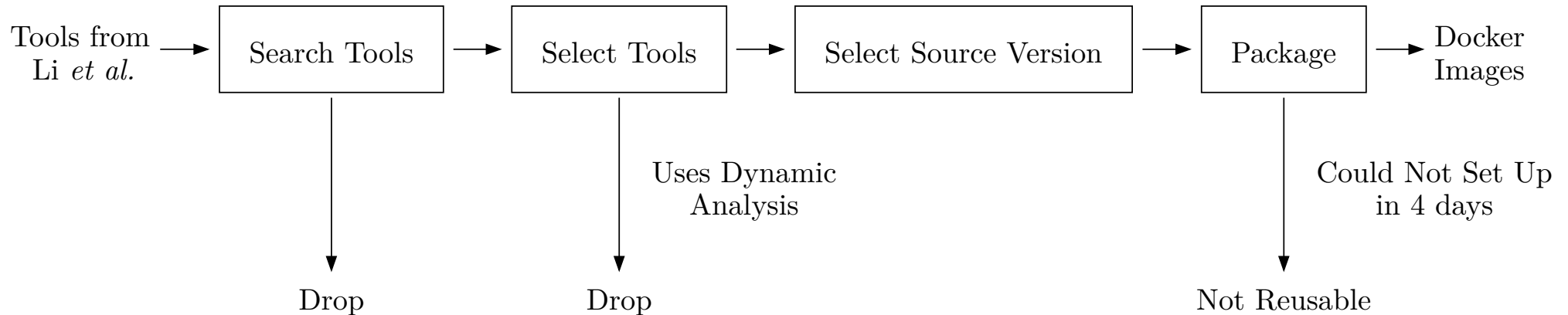
Reaves *et al.* (2016):

- Tests 7 Android analysis tools

- Tests analysing 16 real-world applications

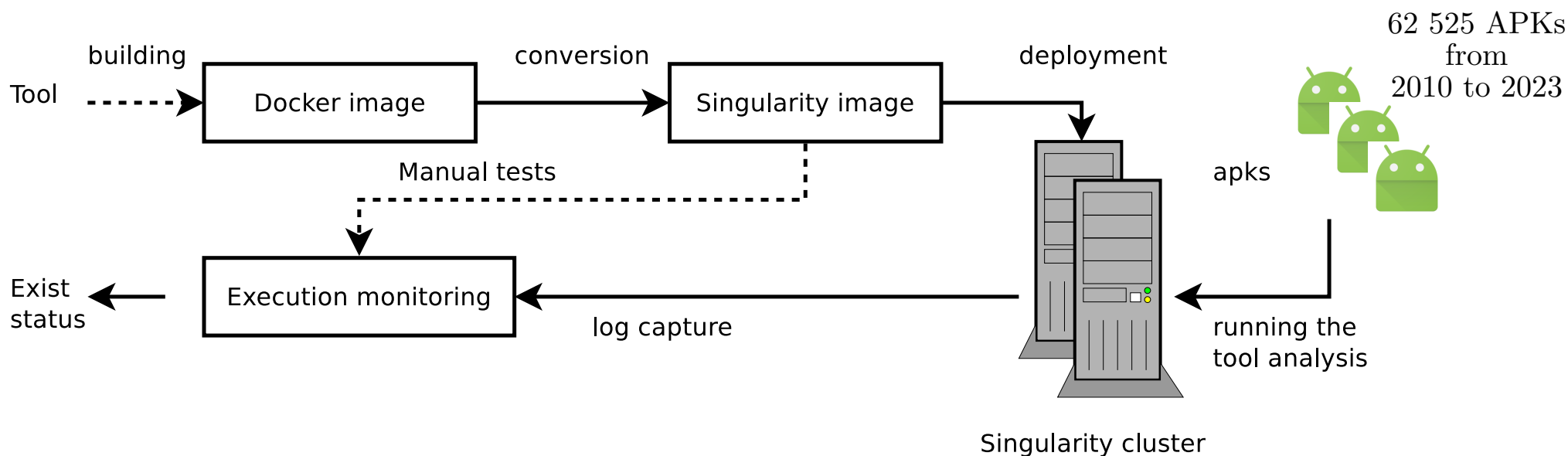
- Raises concerns about reusability and analysis of real-world applications

Methodology: Packaging Static Analysis Tools



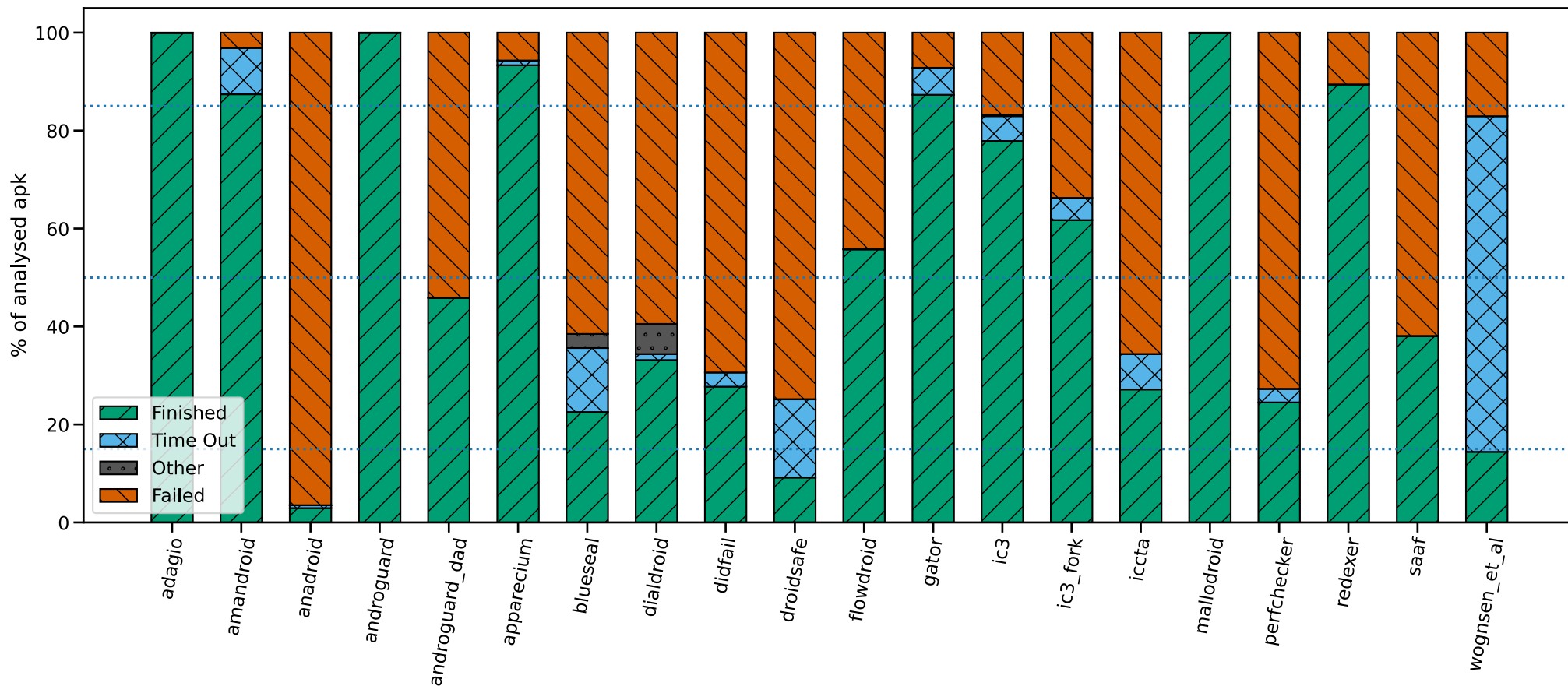
22 tools selected, 2 we could not package

Methodology: RASTA

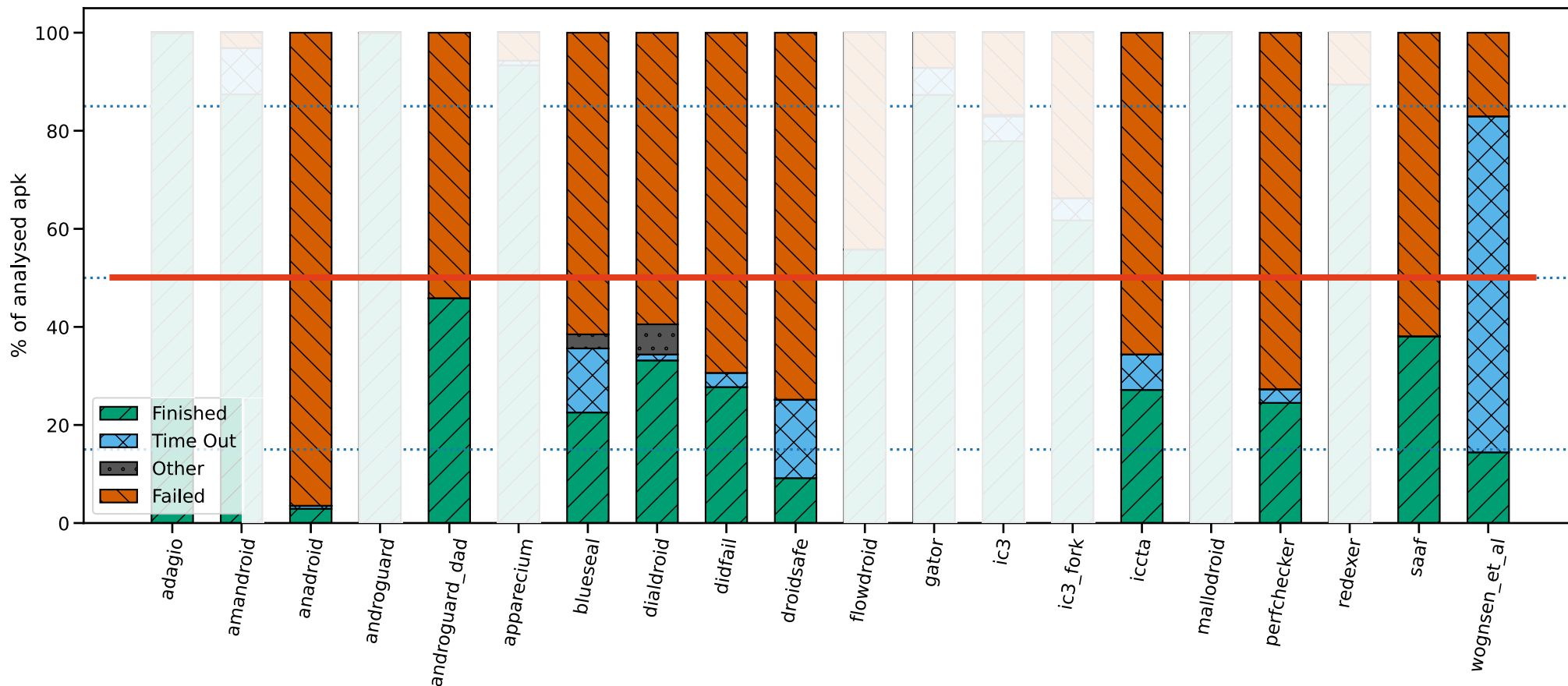


We check if the results **exist** after running a tool

Results



Results



PB1: Conclusion

Over **22** tools, **10** are usable

International Conference on Software and Systems Reuse (ICSR 2024)

PB1: Conclusion

Over **22** tools, **10** are usable

Newer applications are harder to analyse

International Conference on Software and Systems Reuse (ICSR 2024)

PB1: Conclusion

Over **22** tools, **10** are usable

Newer applications are harder to analyse

Applications with **more bytecode** are **harder** to analyse

International Conference on Software and Systems Reuse (ICSR 2024)

PB1: Conclusion

Over **22** tools, **10** are usable

Newer applications are harder to analyse

Applications with **more bytecode** are **harder** to analyse

Applications targetting more recent versions of Android are harder to analyse

International Conference on Software and Systems Reuse (ICSR 2024)

PB1: Conclusion

Over **22** tools, **10** are usable

Newer applications are harder to analyse

Applications with **more bytecode** are **harder** to analyse

Applications targetting more recent versions of Android are harder to analyse

Confirms and **extends Reaves *et al.***

International Conference on Software and Systems Reuse (ICSR 2024)

PB1: Conclusion

Over **22** tools, **10** are usable

Newer applications are harder to analyse

Applications with **more bytecode** are **harder** to analyse

Applications targetting more recent versions of Android are harder to analyse

Confirms and **extends Reaves *et al.***

Docker containers for tools **released**

International Conference on Software and Systems Reuse (ICSR 2024)

Class Shadowing

Class Loading

```
1  ClassLoader cl = new InMemoryDexClassLoader(  
2      ByteBuffer.wrap(Base64.decode(DEX, 2)),  
3      Main.class.getClassLoader()  
4  );  
5  
6  Class<?> loadedClass = this.cl.loadClass(decrypt(className));
```

```
1  class A {  
2      public static void foo() {  
3          B b = new B();  
4          b.bar();  
5      }  
6  }
```

State of the Art

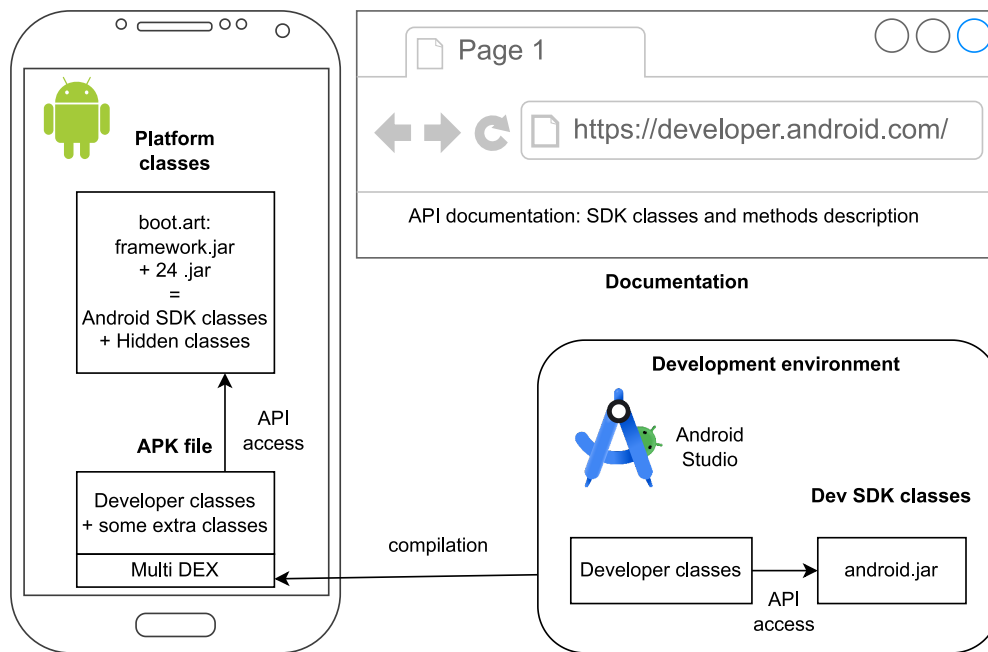
Previous contributions focus on Java runtime (*e.g.* Gong 1998)

State of the Art

Previous contributions focus on Java runtime (*e.g.* Gong 1998)

Android related contributions focus on Dynamic Code Loading (*e.g.* Zhang *et al.* 2015)

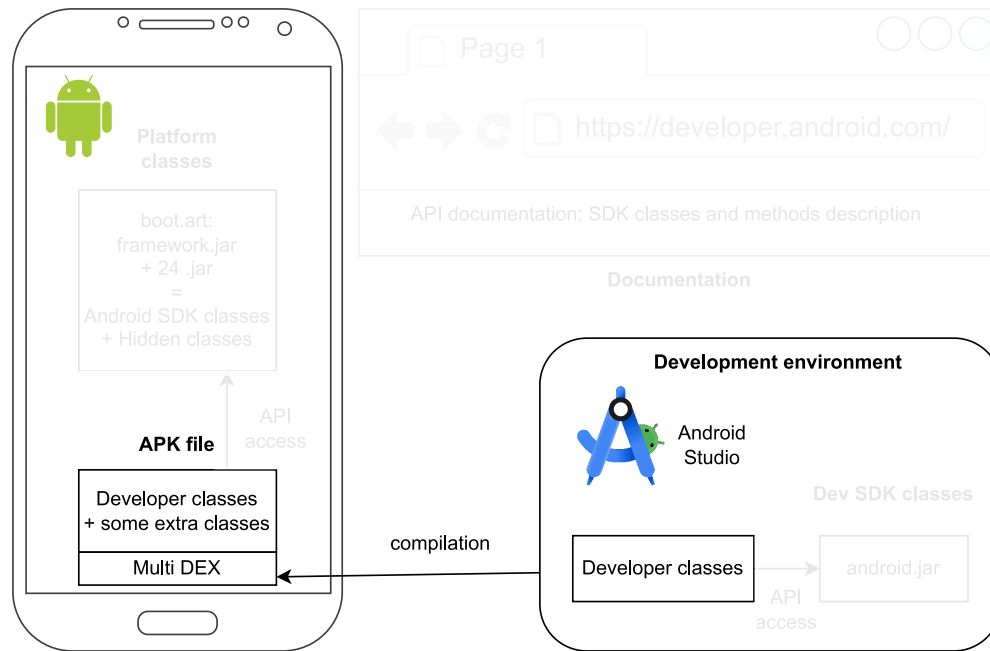
Android Ecosystem



Types of classes:

- APK Classes
- Platform Classes
- SDK Classes
- Hidden APIs

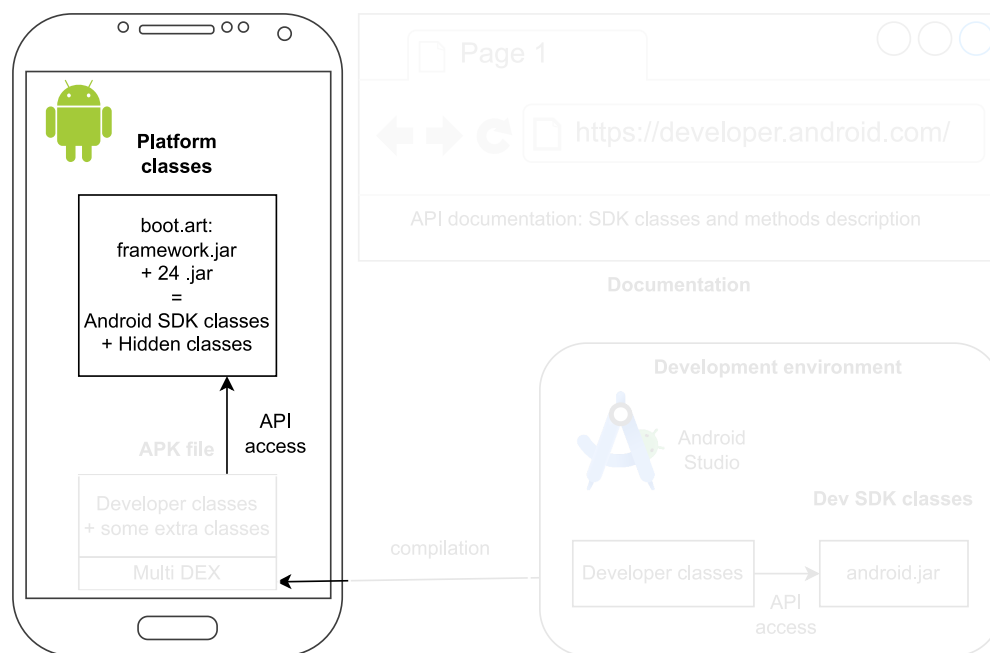
Android Ecosystem



Types of classes:

- **APK Classes**
- Platform Classes
- SDK Classes
- Hidden APIs

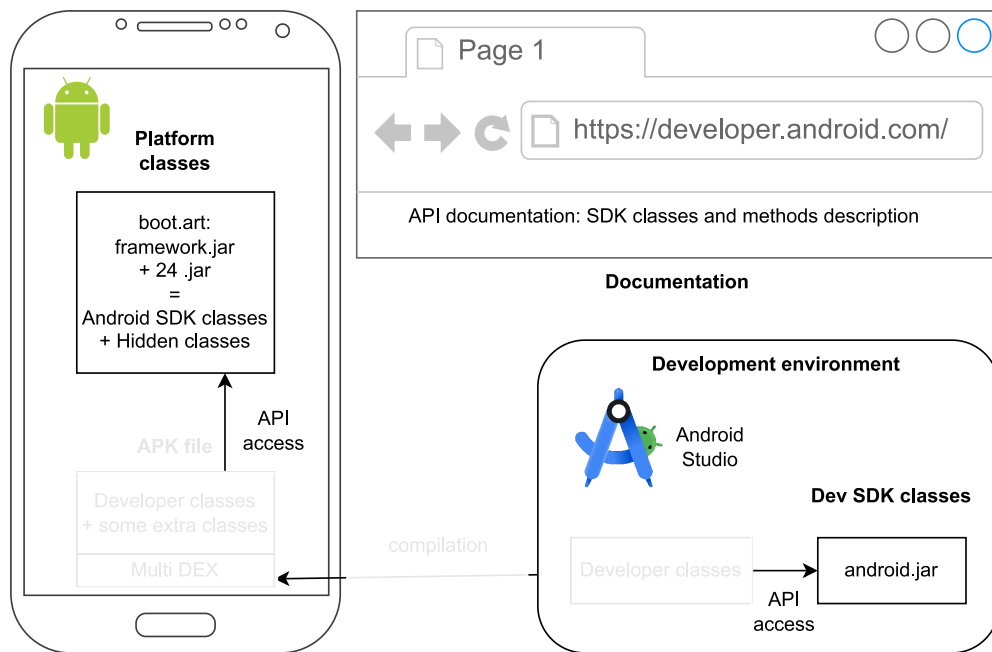
Android Ecosystem



Types of classes:

- APK Classes
- Platform Classes
- SDK Classes
- Hidden APIs

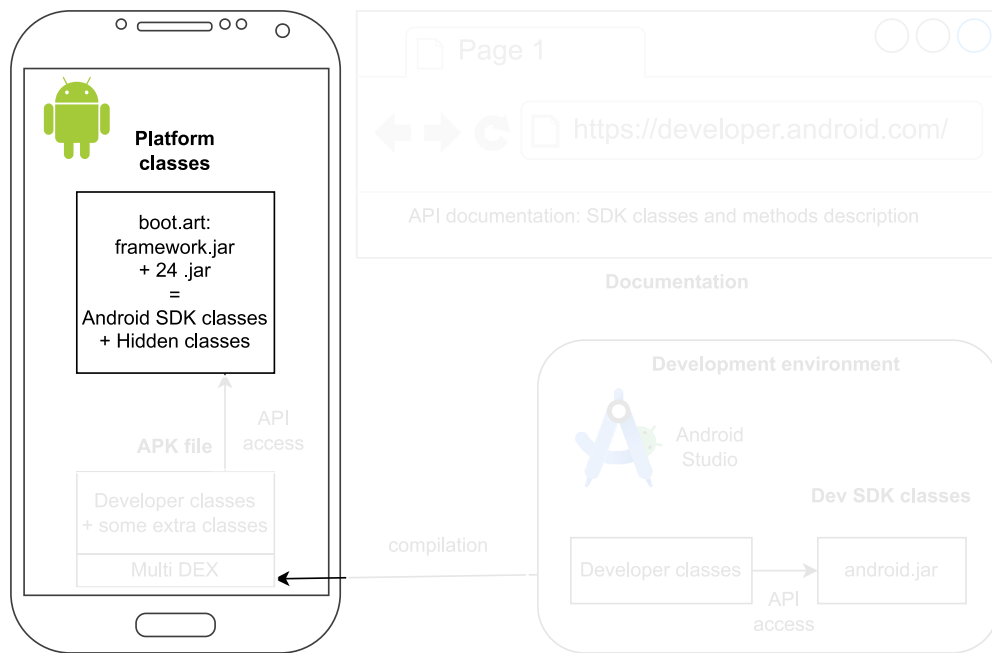
Android Ecosystem



Types of classes:

- APK Classes
- Platform Classes
- SDK Classes
- Hidden APIs

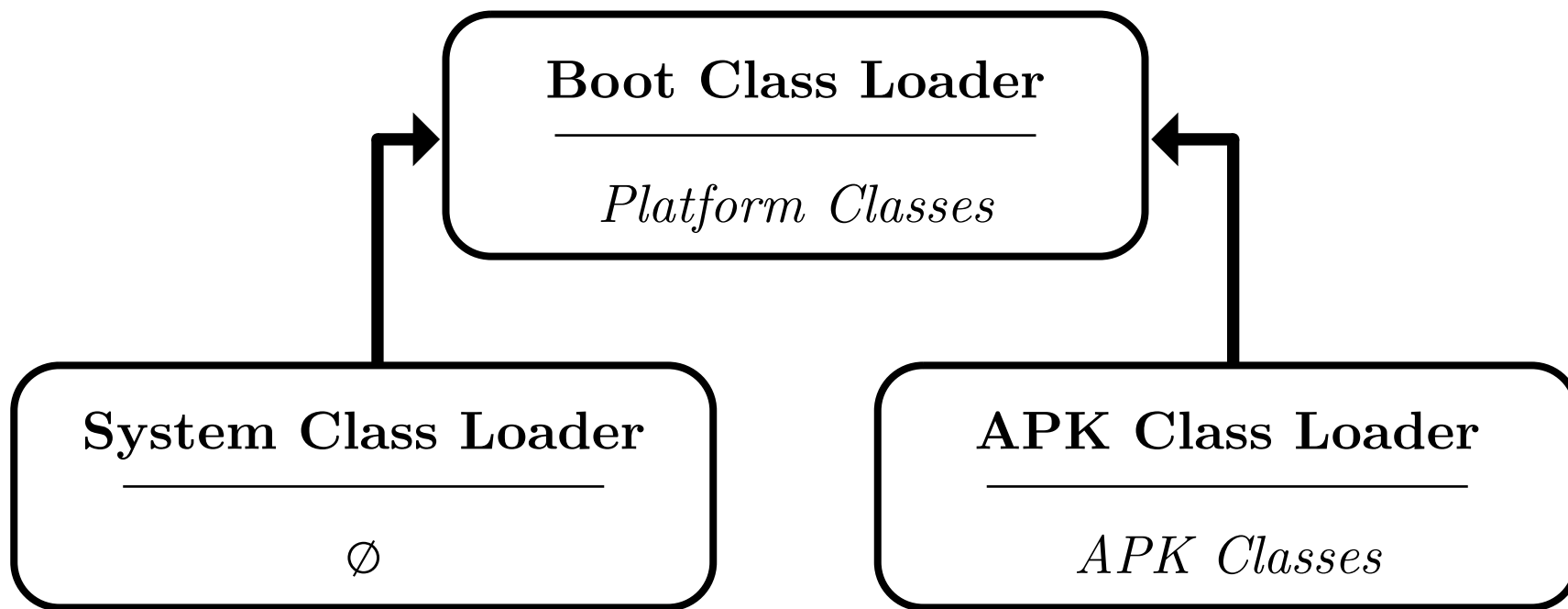
Android Ecosystem



Types of classes:

- APK Classes
- Platform Classes
- SDK Classes
- **Hidden APIs**

Android ClassLoaders



MultiDex

app.apk

AndroidManifest.xml
resources.arsc
META-INF/
res/
classes.dex

MultiDex

app.apk

AndroidManifest.xml

resources.arsc

META-INF/

res/

classes.dex

classes2.dex

classes3.dex

MultiDex

app.apk

AndroidManifest.xml

resources.arsc

META-INF/

res/

classes.dex

classes2.dex

classes3.dex

classes4.dex

classes5.dex

classes6.dex

classes7.dex

classes8.dex

classes9.dex

classes10.dex

classes11.dex

classes12.dex

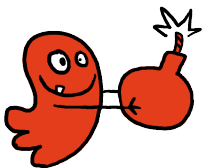
classes13.dex

classes14.dex

classes15.dex

classes16.dex

...



MultiDex

app.apk

```
AndroidManifest.xml
resources.arsc
META-INF/
res/
classes.dex
classes2.dex
classes3.dex
```

```
1 def get_dex_name(index: int):
2     if index == 0:
3         return "classes.dex"
4     else:
5         return
        f"classes{index+1}.dex"
```

Self-Shadowing

```

1 def get_dex_name(index:
  int):
2     if index == 0:
3         return "classes.dex"
4     else:
5         return
        f"classes{index+1}.dex"

```

★ estimation, non-deterministic

Android	Soot	Androguard*
classes.dex		
classes2.dex		
classes3.dex		
classes9.dex	...	
classes10.dex		
classes1.dex		

Self-Shadowing

```

1 def get_dex_name(index:
  int):
2     if index == 0:
3         return "classes.dex"
4     else:
5         return
        f"classes{index+1}.dex"

```

★ estimation, non-deterministic

Android	Soot	Androguard*
classes.dex	classes.dex	
	classes1.dex	
	classes10.dex	
classes2.dex	classes2.dex	
classes3.dex	classes3.dex	
	...	
classes9.dex	classes9.dex	
classes10.dex		
classes1.dex		

Self-Shadowing

```

1 def get_dex_name(index:
  int):
2     if index == 0:
3         return "classes.dex"
4     else:
5         return
        f"classes{index+1}.dex"

```

★ estimation, non-deterministic

Android	Soot	Androguard*
classes.dex	classes.dex	classes10.dex
	classes1.dex	
	classes10.dex	
classes2.dex	classes2.dex	classes9.dex
classes3.dex	classes3.dex	classes8.dex
	...	
classes9.dex	classes9.dex	classes2.dex
classes10.dex		classes1.dex
classes1.dex		classes.dex

Shadow Attacks

```
1 def get_dex_name(index: int):
2     if index == 0:
3         return "classes.dex"
4     else:
5         return f"classes{index+1}.dex"
6 def load_class(class_name: str):
7     if is_platform_class(class_name):
8         return boot_cl.load(class_name)
9     else:
10        index = 0
11        dex_file = get_dex_name(index)
12        while exists_in_apk(dex_file) and \
13            class_name not in classes_of(dex_file):
14            index += 1
15            dex_file = get_dex_name(index)
16        if file_exists_in_apk(dex_file):
17            return load_from_file(dex_file, class_name)
18        else:
19            raise ClassNotFoundError()
```

ANDROID CLASS LOADING ALGORITHM

Shadow Attacks

```

1  def get_dex_name(index: int):
2      if index == 0:
3          return "classes.dex"
4      else:
5          return f"classes{index+1}.dex"
6  def load_class(class_name: str):
7      if is_platform_class(class_name):
8          return boot_cl.load(class_name)
9      else:
10         index = 0
11         dex_file = get_dex_name(index)
12         while exists_in_apk(dex_file) and \
13             class_name not in classes_of(dex_file):
14             index += 1
15             dex_file = get_dex_name(index)
16         if file_exists_in_apk(dex_file):
17             return load_from_file(dex_file, class_name)
18         else:
19             raise ClassNotFoundError()

```

ANDROID CLASS LOADING ALGORITHM

Self Shadowing

Trick the tool into using an APK class instead of another APK class

Shadow Attacks

```

1  def get_dex_name(index: int):
2      if index == 0:
3          return "classes.dex"
4      else:
5          return f"classes{index+1}.dex"
6  def load_class(class_name: str):
7      if is_platform_class(class_name):
8          return boot_cl.load(class_name)
9      else:
10         index = 0
11         dex_file = get_dex_name(index)
12         while exists_in_apk(dex_file) and \
13             class_name not in classes_of(dex_file):
14             index += 1
15             dex_file = get_dex_name(index)
16         if file_exists_in_apk(dex_file):
17             return load_from_file(dex_file, class_name)
18         else:
19             raise ClassNotFoundError()

```

ANDROID CLASS LOADING ALGORITHM

Self Shadowing

Trick the tool into using an APK class instead of another APK class

SDK shadowing

Trick the tool into using an APK class instead of an SDK class

Hidden API shadowing

Trick the tool into using an APK class instead of an hidden API class

Shadow Attacks

```

1  def get_dex_name(index: int):
2      if index == 0:
3          return "classes.dex"
4      else:
5          return f"classes{index+1}.dex"
6  def load_class(class_name: str):
7      if is_platform_class(class_name):
8          return boot_cl.load(class_name)
9      else:
10         index = 0
11         dex_file = get_dex_name(index)
12         while exists_in_apk(dex_file) and \
13             class_name not in classes_of(dex_file):
14             index += 1
15             dex_file = get_dex_name(index)
16         if file_exists_in_apk(dex_file):
17             return load_from_file(dex_file, class_name)
18         else:
19             raise ClassNotFoundError()

```

ANDROID CLASS LOADING ALGORITHM

Self Shadowing

Trick the tool into using an APK class instead of another APK class

SDK shadowing

Trick the tool into using an APK class instead of an SDK class

Hidden API shadowing

Trick the tool into using an APK class instead of an hidden API class

Impact on Tools

Tool	Version	Shadow Attack		
		Self	SDK	Hidden
Jadx	1.5.0	○	●	●
Apktool	2.9.3	○	●	●
Androguard	4.1.2	○	●	●
Flowdroid	2.13.0	●	×	●

●: attack successful

○: successful but producing warning or can be seen by the reverser

×: attack failed

Impact on Tools

Tool	Shadow Attack		
	Self	SDK	Hidden
Jadx	×	●	●
Apktool	○	●	●
Androguard	×	●	●
Flowdroid	×	×	●

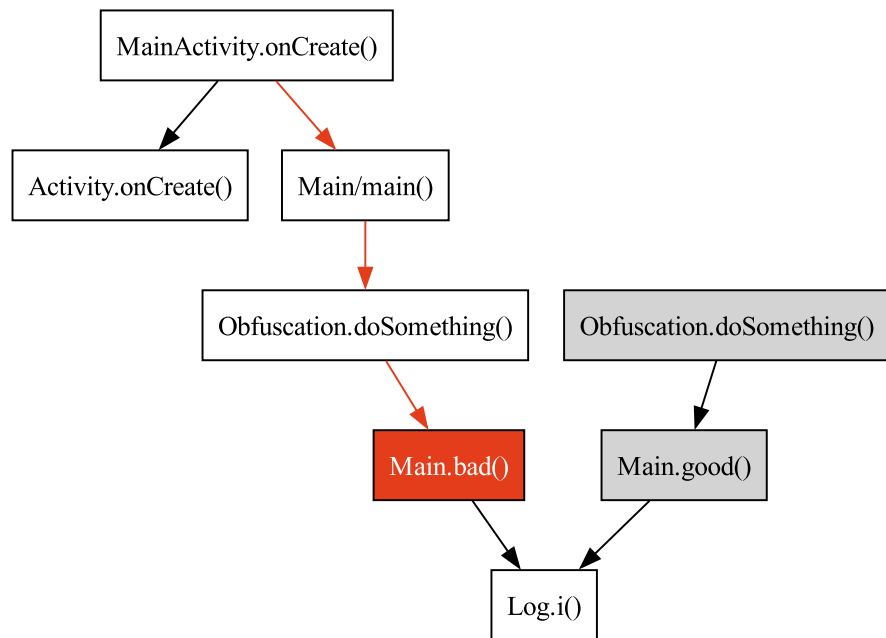
Pull Requests:
[soot/pull/2211](#) (merged)
[jadx/pull/2702](#) (merged)
[androguard/pull/1149](#)

●: attack successful

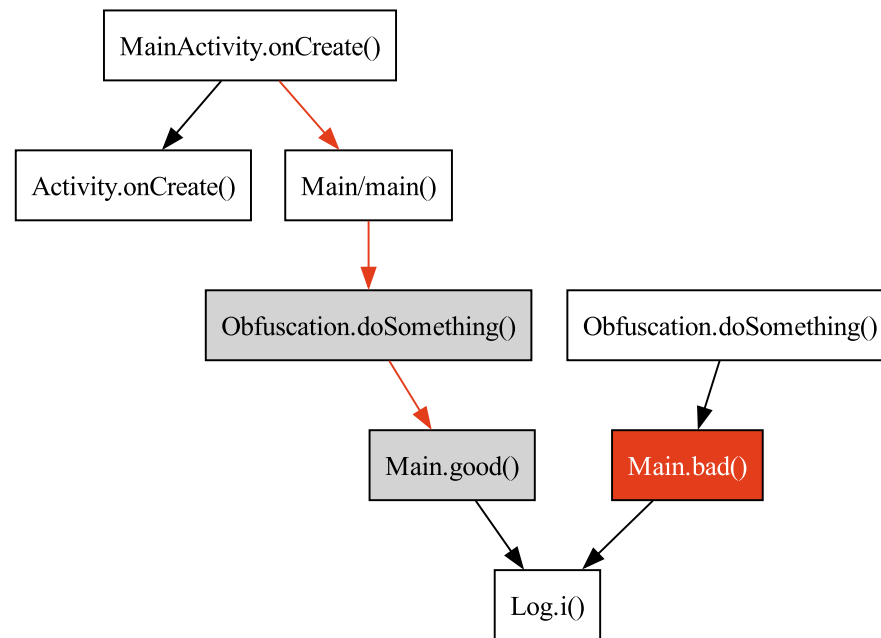
○: successful but producing warning or can be seen by the reverser

×: attack failed

Androguard on Toy Malware



Expected



Computed

In the Wild: 49 975 APKs

	Number of apps			Nb Shadow Classes		Identical Code
		%	% malware	Average	Median	
Self	234	0.47%	5.98%	438.1	18	74.8%
Sdk	11 755	23.52%	0.38%	27.6	5	8.04%
Hidden	1556	3.11%	0.71%	16.1	1	17.42%
Total	12 301	24.61%	0.42%	36.7	6	23.76%

In the Wild: 49 975 APKs

	Number of apps		% malware	Nb Shadow Classes		Identical Code
	%			Average	Median	
Self	234	0.47%	5.98%	438.1	18	74.8%
Sdk	11 755	23.52%	0.38%	27.6	5	8.04%
Hidden	1556	3.11%	0.71%	16.1	1	17.42%
Total	12 301	24.61%	0.42%	36.7	6	23.76%

In the Wild: 49 975 APKs

	Number of apps			Nb Shadow Classes		Identical Code
		%	% malware	Average	Median	
Self	234	0.47%	5.98%	438.1	18	74.8%
Sdk	11 755	23.52%	0.38%	27.6	5	8.04%
Hidden	1556	3.11%	0.71%	16.1	1	17.42%
Total	12 301	24.61%	0.42%	36.7	6	23.76%

PB2: Conclusion

We modeled the class loading algorithm

Digital Threats: Research and Practice, vol. 6 (3), 2025

PB2: Conclusion

We modeled the class loading algorithm

Static Analysis Tools did not

Digital Threats: Research and Practice, vol. 6 (3), 2025

PB2: Conclusion

We modeled the class loading algorithm

Static Analysis Tools did not

We introduced obfuscation techniques based on this model

Digital Threats: Research and Practice, vol. 6 (3), 2025

PB2: Conclusion

We modeled the class loading algorithm

Static Analysis Tools did not

We introduced obfuscation techniques based on this model

We did not find deliberate shadow attacks

Digital Threats: Research and Practice, vol. 6 (3), 2025

PB2: Conclusion

We modeled the class loading algorithm

Static Analysis Tools did not

We introduced obfuscation techniques based on this model

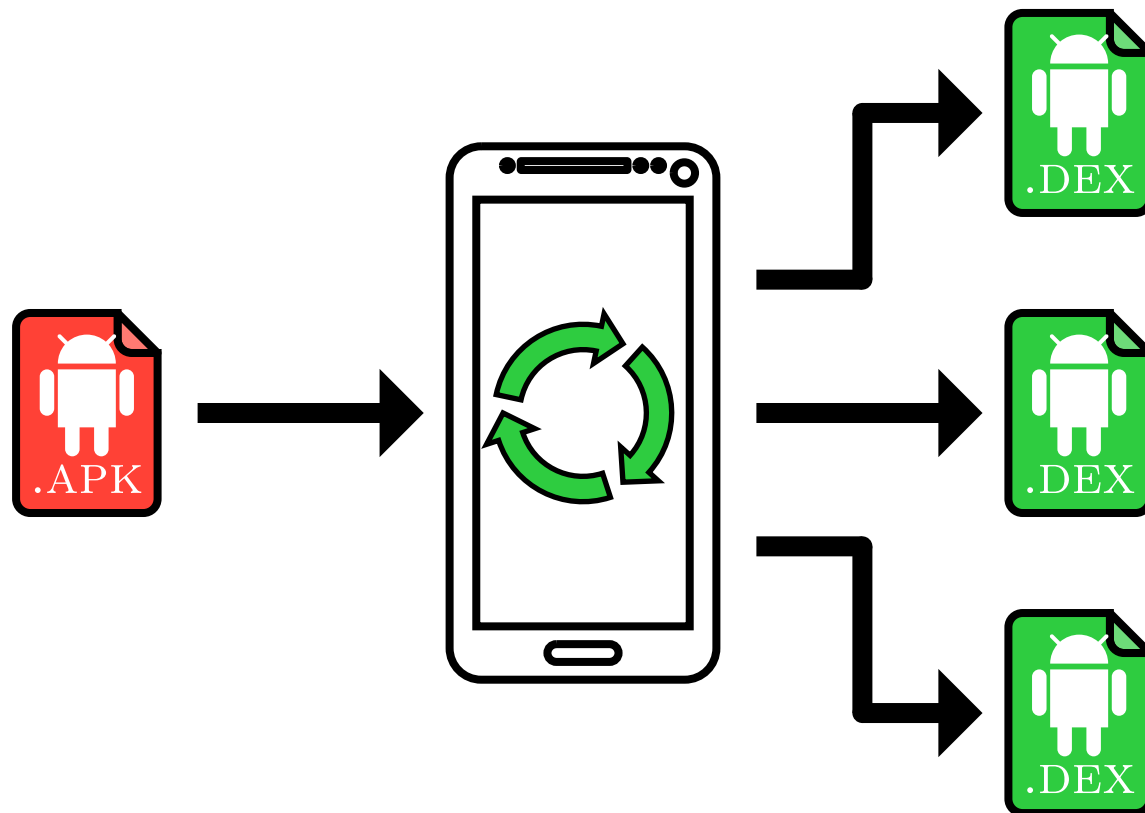
We did not find deliberate shadow attacks

Ambiguous cases exist in the wild

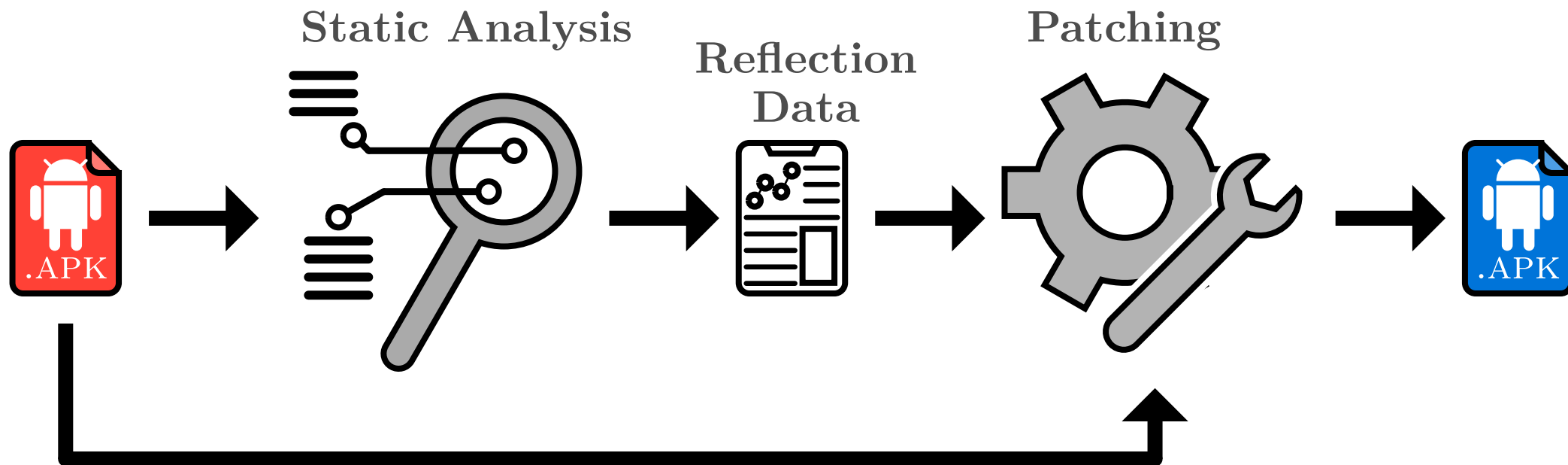
Digital Threats: Research and Practice, vol. 6 (3), 2025

The Application of Theseus

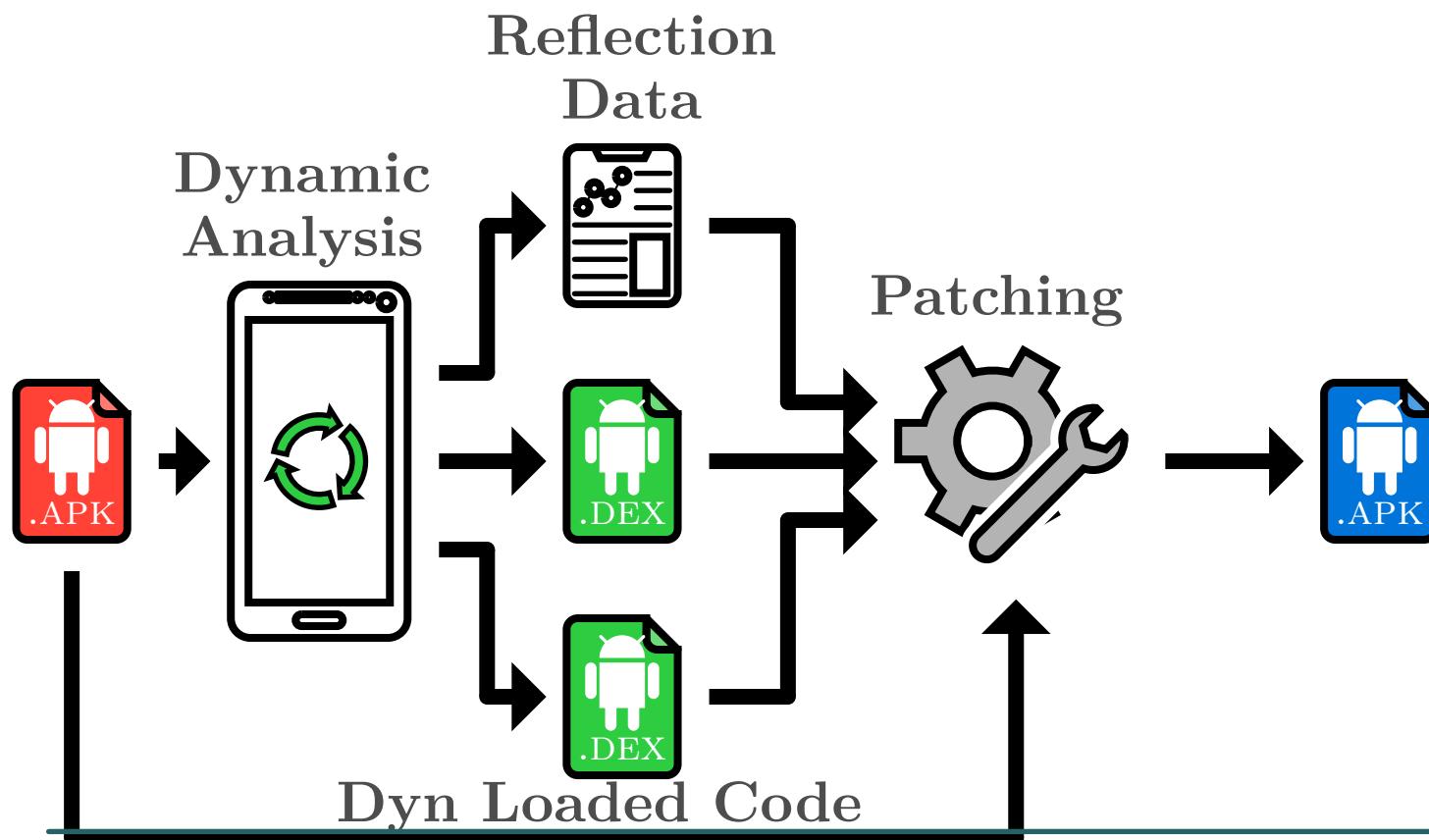
Dexhunter: Zang *et al.* (2015)



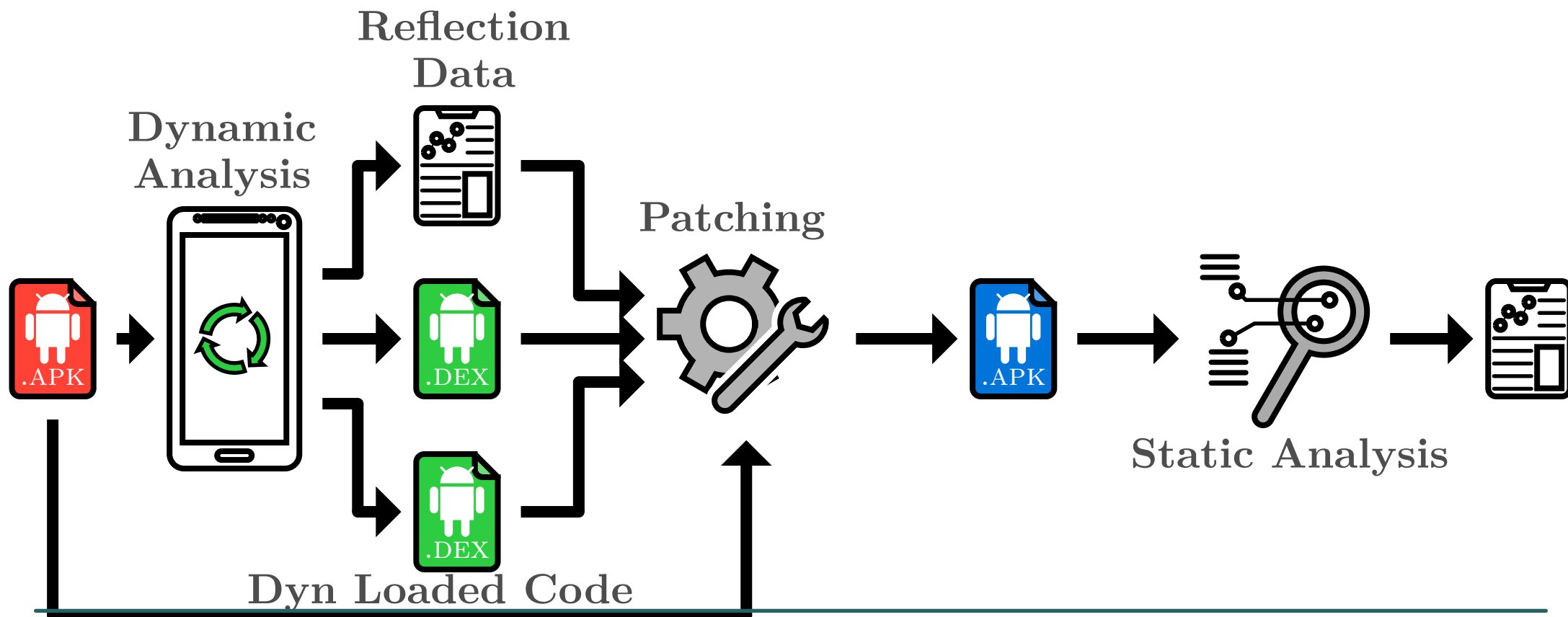
DroidRA: Li *et al.* (2016)



Theseus: Overview



Theseus: Overview



Dynamic Analysis

Frida: intercepts method calls

Dynamic Analysis

Frida: intercepts method calls

Android Emulator: runs on computer/server

Grodd Runner: clicks buttons (Abraham *et al.*, 2015)

Dynamic Analysis

Frida: intercepts method calls

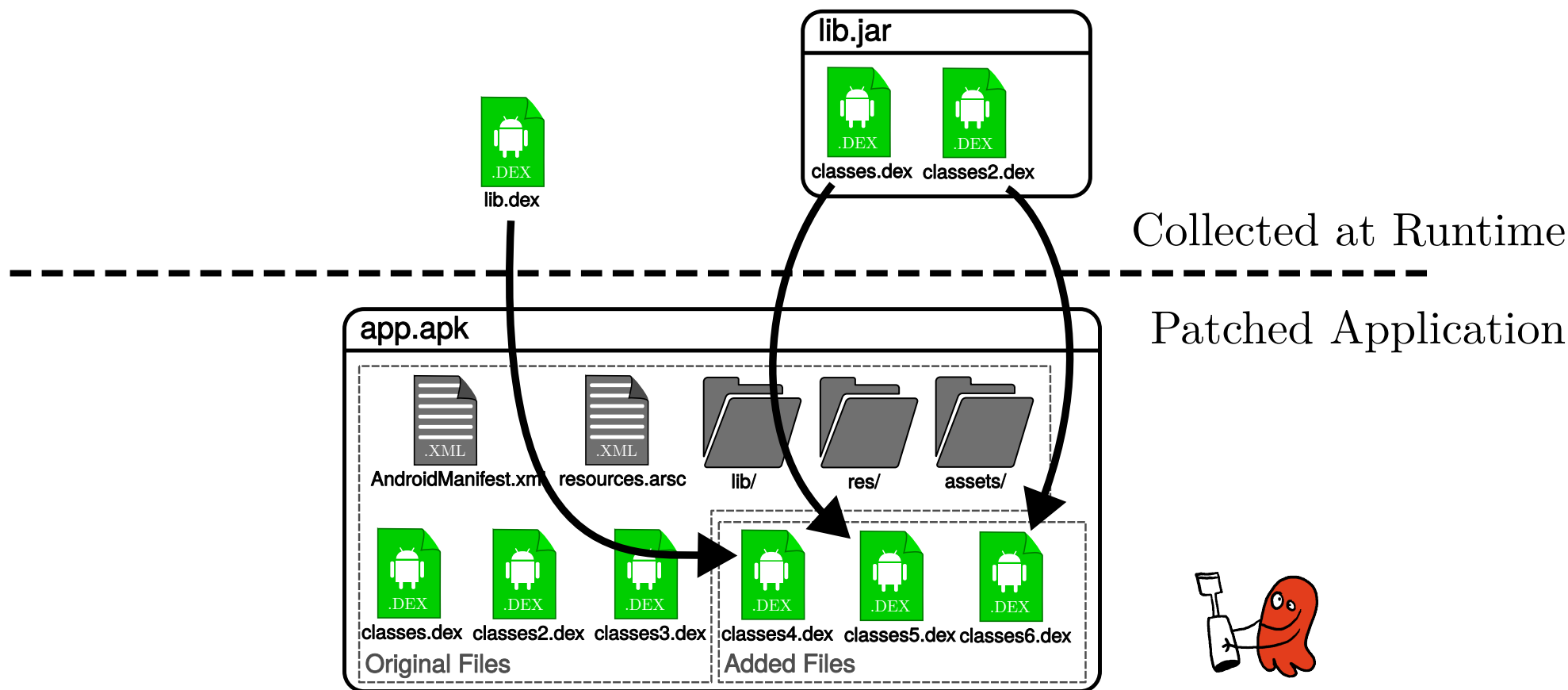
Android Emulator: runs on computer/server

Grodd Runner: clicks buttons (Abraham *et al.*, 2015)

Phone with adb enable: actual hardware

Human: intelligent button clicker

Transformation: Dynamic Code Loading



Reflection Transformation

```
5 Method mth = getMethod();  
6 String retData = (String) mth.invoke(obj, args);
```



```
6 String retData = obj.myMethod((String)args[0]);
```

Reflection Transformation

```
5 Method mth = getMethod();  
6 String retData = (String) mth.invoke(obj, args);
```



```
6 String retData = obj.myMethod((String)args[0]);
```

Reflection Transformation

```
5 Method mth = getMethod();
6 String retData = (String) mth.invoke(obj, args);
```



```
5 Method mth = getMethod();
6 Object objRet;
7 if (T.check_is_reflectee_mymethod_XXXX(mth)) {
8     objRet = (Object)((Reflectee) obj).myMethod((String)args[0]);
9 } else {
10     objRet = mth.invoke(obj, args);
11 }
12 String retData = (String) objRet;
```

Reflection Transformation

```
5 Method mth = getMethod();
6 String retData = (String) mth.invoke(obj, args);
```



```
5 Method mth = getMethod();
6 Object objRet;
7 if (T.check_is_reflectee_mymethod_XXXX(mth)) {
8     objRet = (Object)((Reflectee) obj).myMethod((String)args[0]);
9 } else {
10     objRet = mth.invoke(obj, args);
11 }
12 String retData = (String) objRet;
```

Reflection Transformation

```
5 Method mth = getMethod();
6 String retData = (String) mth.invoke(obj, args);
```



```
5 Method mth = getMethod();
6 Object objRet;
7 if (T.check is reflectee mymethod XXXX(mth)) {
8     objRet = (Object)((Reflectee) obj).myMethod((String)args[0]);
9 } else {
10     objRet = mth.invoke(obj, args);
11 }
12 String retData = (String) objRet;
```

Reflection Transformation

```
5 Method mth = getMethod();
6 String retData = (String) mth.invoke(obj, args);
```



```
5 Method mth = getMethod();
6 Object objRet;
7 if (T.check_is_reflectee_mymethod_XXXX(mth)) {
8     objRet = (Object)((Reflectee) obj).myMethod((String)args[0]);
9 } else {
10     objRet = mth.invoke(obj, args);
11 }
12 String retData = (String) objRet;
```

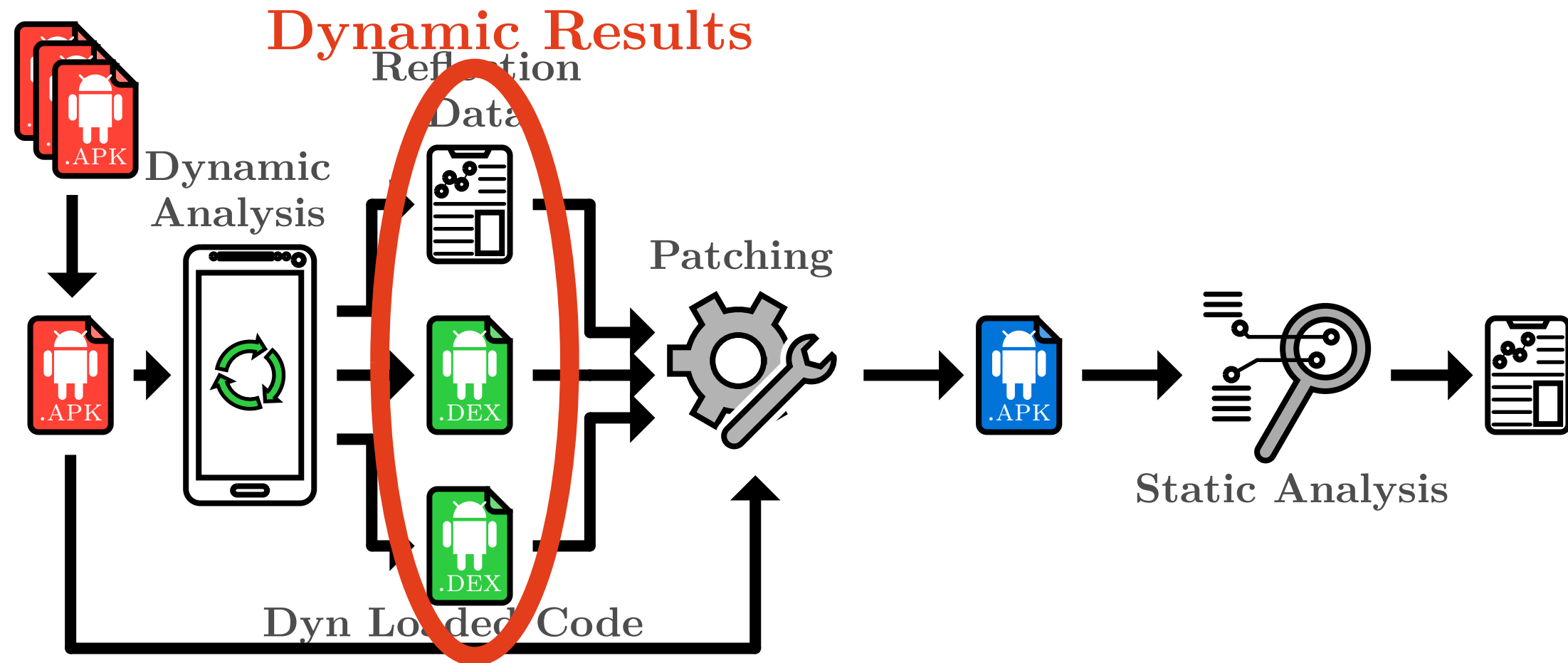
Reflection Transformation

```
5 Method mth = getMethod();
6 String retData = (String) mth.invoke(obj, args);
```



```
5 Method mth = getMethod();
6 Object objRet;
7 if (T.check_is_reflectee_mymethod_XXXX(mth)) {
8     objRet = (Object)((Reflectee) obj).myMethod((String)args[0]);
9 } else {
10     objRet = mth.invoke(obj, args);
11 }
12 String retData = (String) objRet;
```


RASTA



Dynamic Analysis

	nb apk	activities visited	
		0	≥ 1
All	4957	4025	723
With Reflection	3948	3298	650
With Code Loading	598	453	145



Dynamic Analysis

	nb apk	activities visited	
		0	≥ 1
All	4957	4025	723
With Reflection	3948	3298	650
With Code Loading	598	453	145



Dynamic Analysis

	nb apk	activities visited	
		0	≥ 1
All	4957	4025	723
With Reflection	3948	3298	650
With Code Loading	598	453	145

Reflection detected on
79.64% of APKs tested

DLC detected on
12.06% of APKs tested



Collected Bytecode



Nb Occurences	SHA 256	Content	Format
273	bee390afa2...	Lcom/facebook/ads/*	DEX
98	7aae06433c...	Lcom/facebook/ads/*	DEX
70	920e465a87...	Lcom/facebook/ads/*	DEX
31	51dd5ff34a...	Lcom/google/android/ads/*	APK
12	d44cfb6b41...	Lcom/google/android/ads/*	APK
9	be87bb0a50...	Lcom/facebook/ads/*	DEX
9	26fb1a7903...	Lcom/facebook/ads/*	DEX
7	8395a0121e...	Lcom/facebook/ads/*	DEX
7	1eea5584eb...	Lcom/google/android/ads/*	APK
6	94f66aa1ae...	Lcom/appsflyer/internal/*	DEX

...

PB3: Conclusion

We can statically **analyse APKs** with reflection and dynamic code **loading** with our method

PB3: Conclusion

We can statically **analyse APKs with reflection and dynamic code loading** with our method

Our dynamic analysis is questionable

PB3: Conclusion

We can statically **analyse APKs with reflection and dynamic code loading** with our method

Our dynamic analysis is questionable

The dynamically loaded bytecode we intercepted is **mainly telemetry and advertisement** related

PB3: Conclusion

We can statically **analyse APKs with reflection and dynamic code loading** with our method

Our dynamic analysis is questionable

The dynamically loaded bytecode we intercepted is **mainly telemetry and advertisement** related

Software Contributions:

PB3: Conclusion

We can statically **analyse APKs with reflection and dynamic code loading** with our method

Our dynamic analysis is questionable

The dynamically loaded bytecode we intercepted is **mainly telemetry and advertisement** related

Software Contributions:

Androscalpel: rust crate to **parse, modify and generate** bytecode

PB3: Conclusion

We can statically **analyse APKs with reflection and dynamic code loading** with our method

Our dynamic analysis is questionable

The dynamically loaded bytecode we intercepted is **mainly telemetry and advertisement** related

Software Contributions:

Androscalpel: rust crate to **parse, modify and generate** bytecode

Theseus: tool implementing the method presented here

Conclusion

Experimentations

	Experiment	Number of APKs	Time
RASTA	20 static analyses	62 525	2 months
Class Loading	1 static analysis	49 975	1 week
Theseus	1 dynamic analysis	4957	1 week
	patching	4748	2 days
	18 static analyses	8955	2 months

We showed that:

Most Static analysis tools are no longer usable after a few years. The size of the application seems to be the most significant factor.

We showed that:

Most Static analysis tools are no longer usable after a few years. The size of the application seems to be the most significant factor.

Android behaviour is complex and not well known. In the specific case of class loading, we showed that state-of-the-art tools do not match Android, leading to invalid analyses.

We showed that:

Most Static analysis tools are no longer usable after a few years. The size of the application seems to be the most significant factor.

Android behaviour is complex and not well known. In the specific case of class loading, we showed that state-of-the-art tools do not match Android, leading to invalid analyses.

APKs can be augmented with instrumentation to improve further analyses with any other tools.

We showed that:

Most Static analysis tools are no longer usable after a few years. The size of the application seems to be the most significant factor.

Android behaviour is complex and not well known. In the specific case of class loading, we showed that state-of-the-art tools do not match Android, leading to invalid analyses.

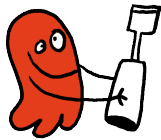
APKs can be augmented with instrumentation to improve further analyses with any other tools.

Also, dynamic analysis is still very much not trivial.

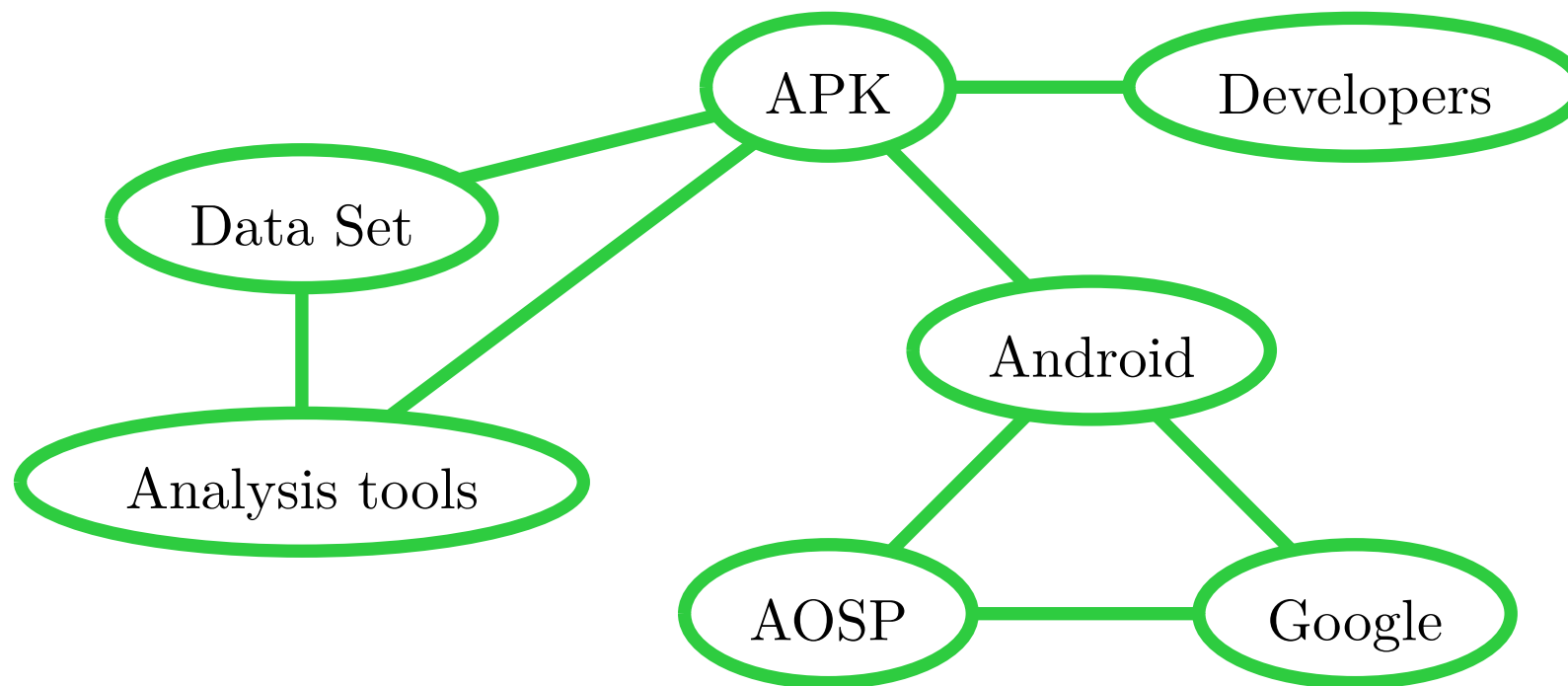
Future Works



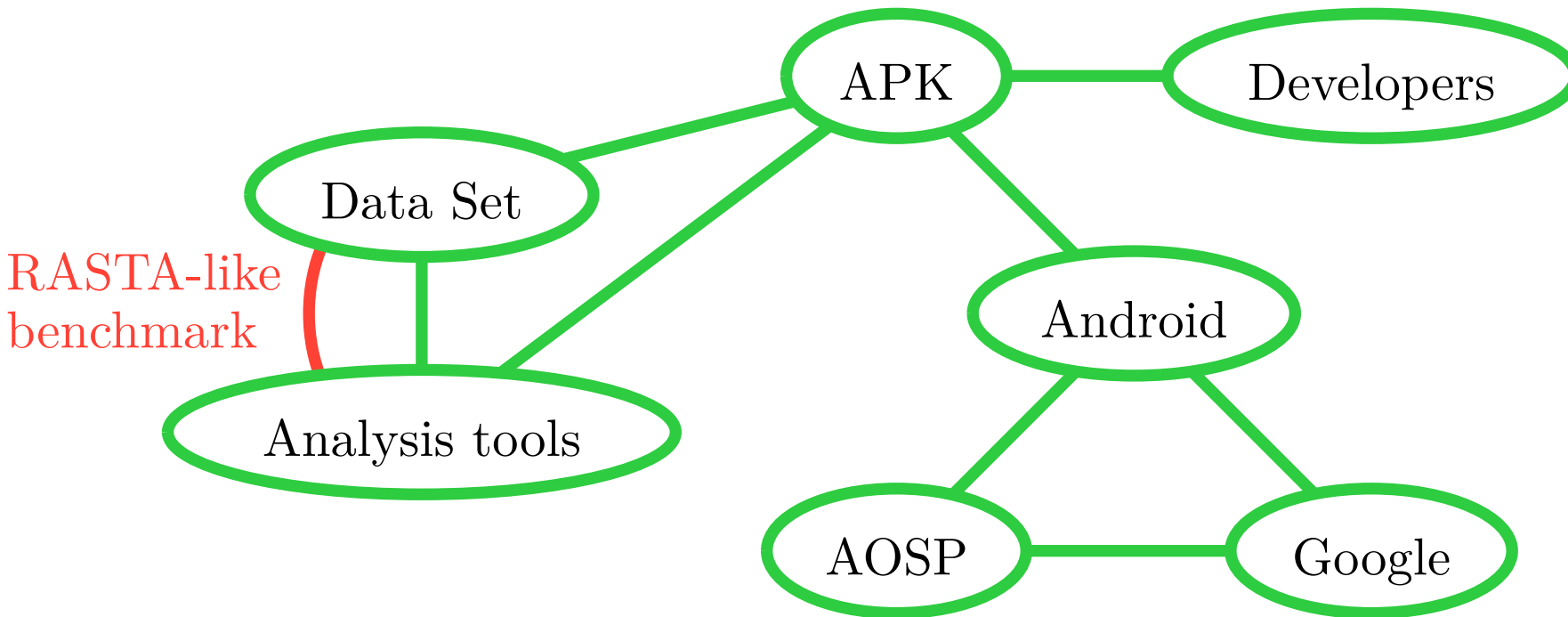
A lot of engineering, preferably spearheaded by Google.



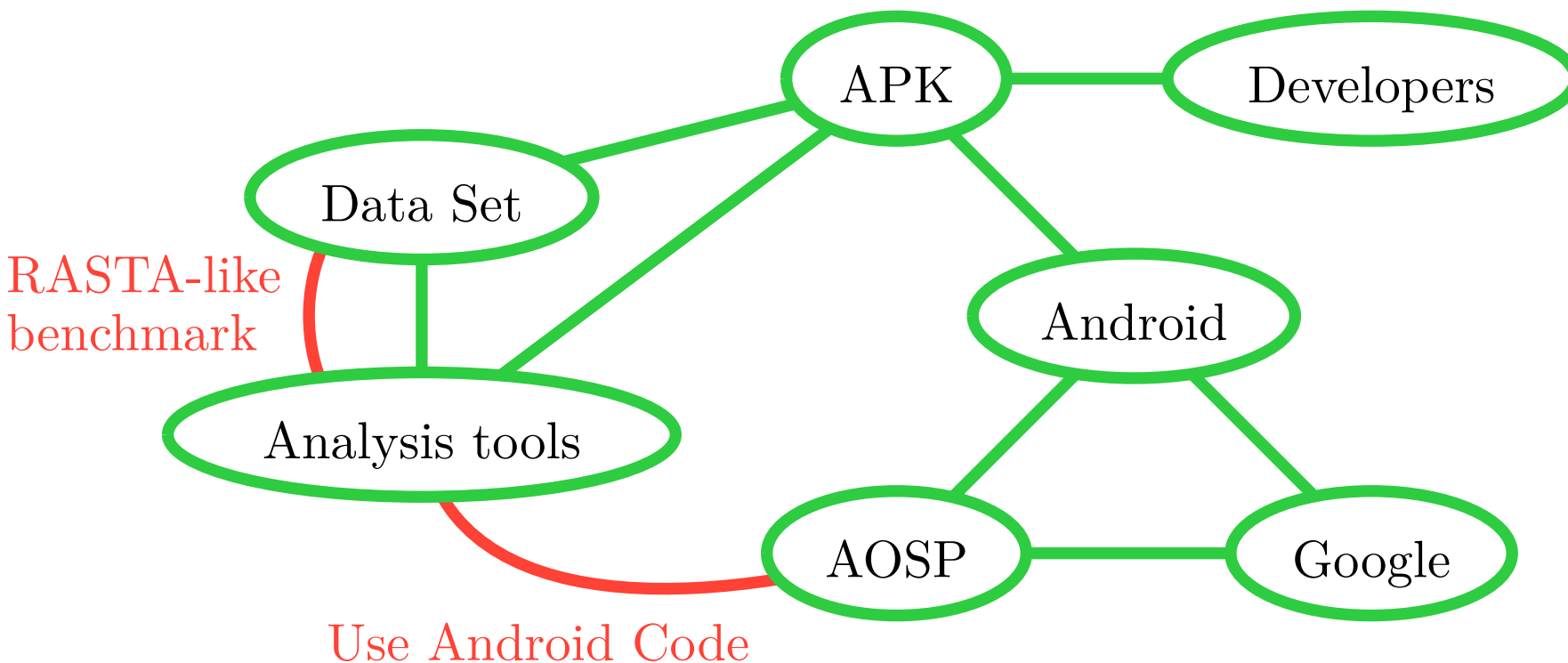
Future Works



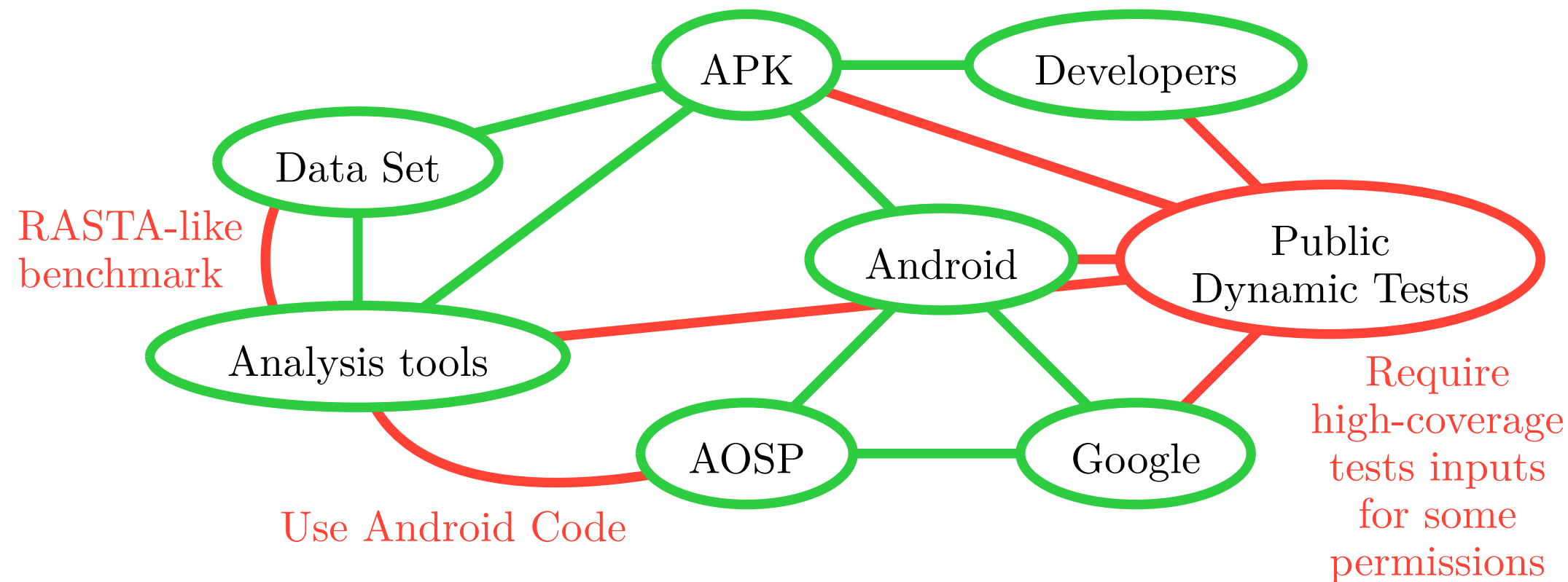
Future Works



Future Works



Future Works



Questions

Bibliography