The Impact of Third-party Code on Android App Security

Erik Derr

CISPA
HELMHOLTZ-ZENTRUM i. G.

Third-party Code – A Double-edged Sword

Eases software development

- Code re-use
- Faster development, less costs

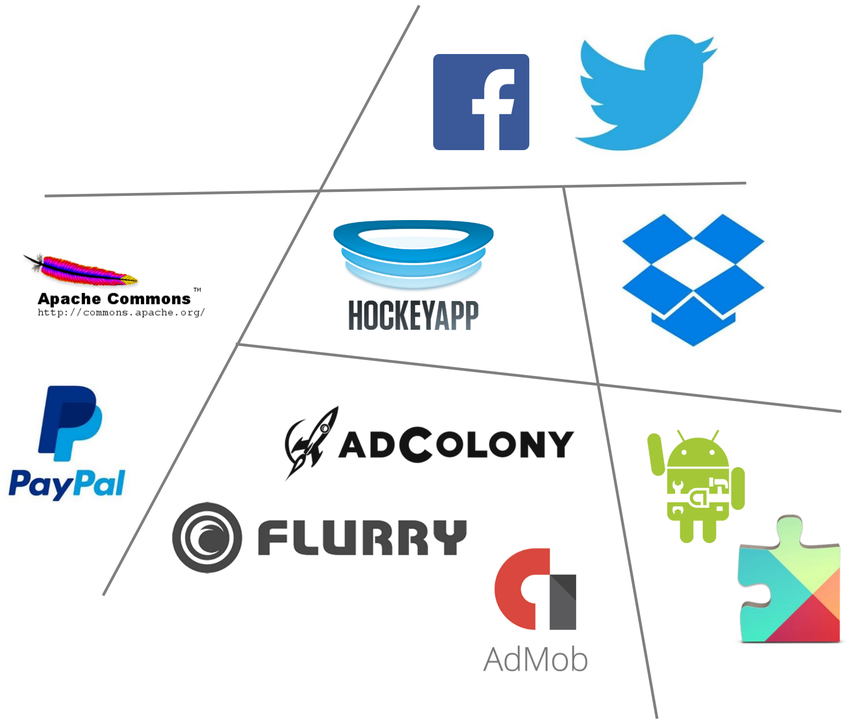
Increases apps‘ attack surface

- Code from different origins
- Trust closed-source components

Risk Estimation

2,000,000,000 components with known vulnerabilities are downloaded / year

Outdated libs have a **3x** higher probability to include vulnerabilities



OWASP Top 10 Security Risks (since 2013)

“***Using components with known vulnerabilities***”

Quantify Security Impact

Measure the status quo of outdated libs in the software ecosystem

Identify apps that use lib versions with known vulnerabilities

Attribute new vulnerabilities to the correct component (library / app code)

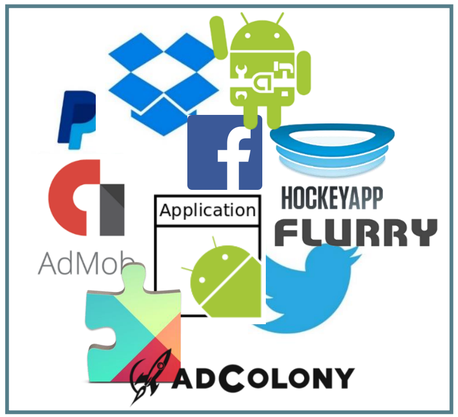
Requires a reliable detection of libraries in app binaries

Detection Challenges on Android

Developer View

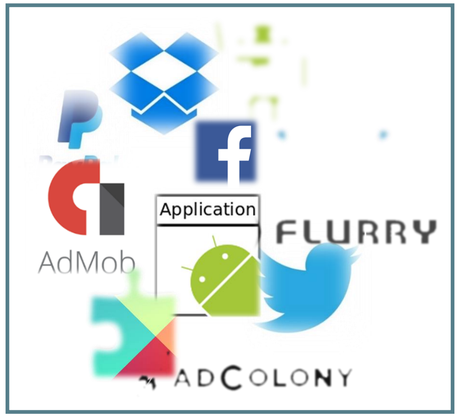
- Explicit declaration
- Libraries / versions known

Compiled App



- Monolithic bytecode
- Same Origin

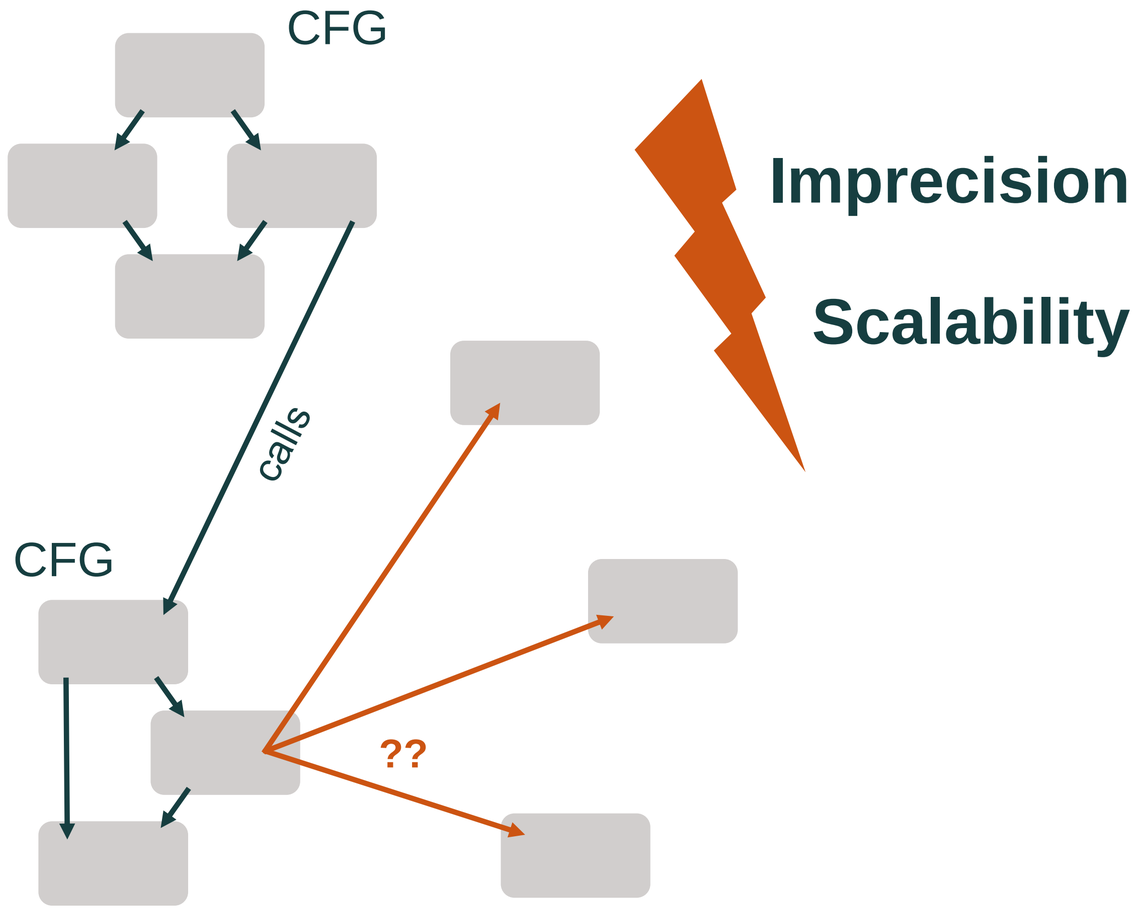
Obfuscated App



- Identifier renaming
- Dead code elimination

Common Analysis Approach

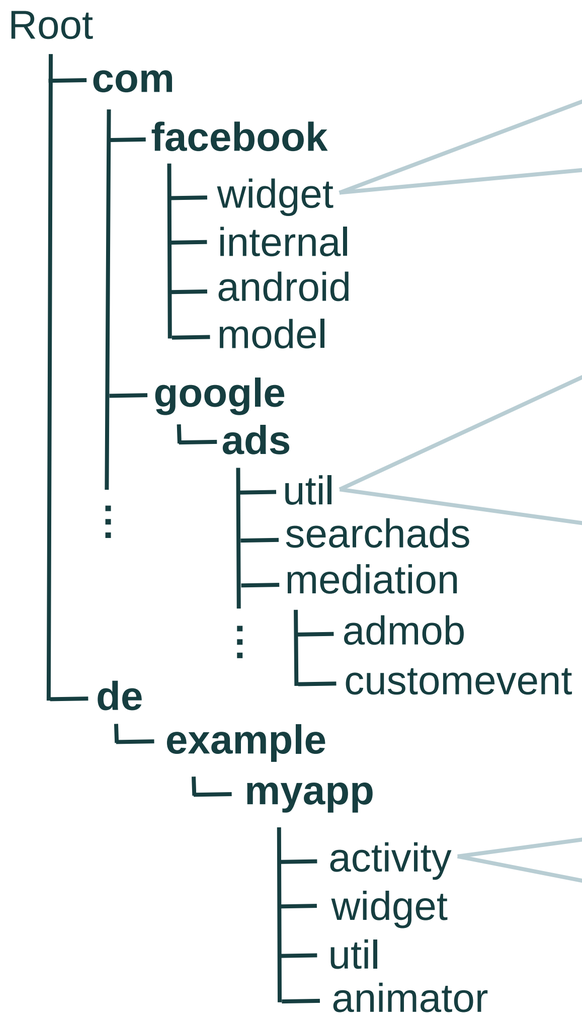
```
public void myTest(int, int, String) {
    code code code
    code code code
    if (..) {
        code code code
        code code code
    } else
        code = doSomething(data);
}

private int doSomething(String) {
    while (true) {
        obfuscatedCall
        obfuscatedCall
        if (..) {
            return data;
        }
    }
    return otherData;
}
```



Static (Byte)Code Analysis

Data Structures (CFG, CG, ..)

Code Structure Detection

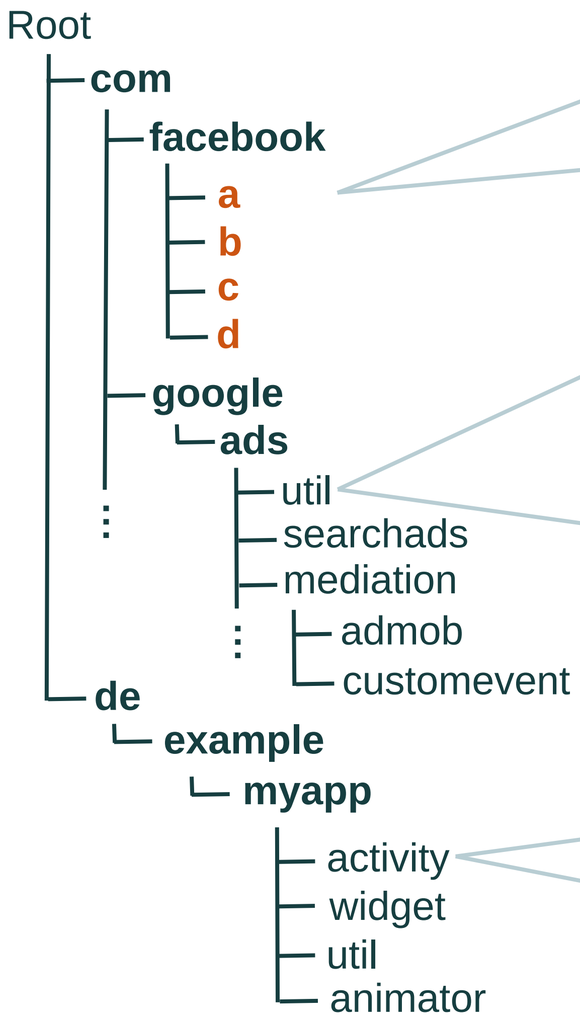



```
package com.facebook.widget;
public class someClass {
    {..}
}
```

```
package com.google.ads.util;

public class myClass {

        public void myTest(int, int, String) {
        }

        private int doSomething(String) {
        }

}
```

```
package de.example.myapp.activity;
public class anotherActivity {
    {..}
}
```

Code Structure Detection




```
package com.facebook.widget;
public class someClass {
    {..}
}
```

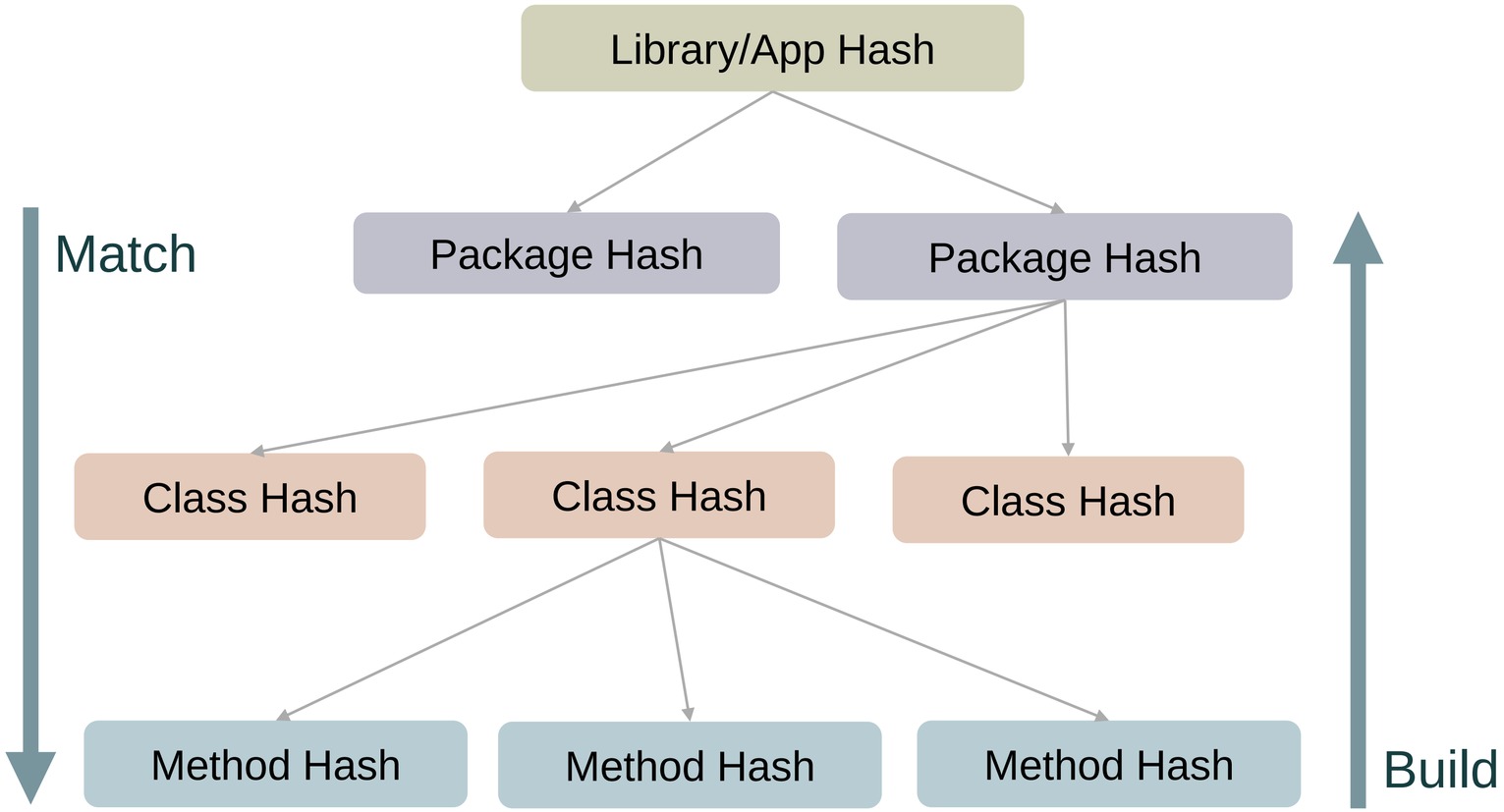
```
package com.google.ads.util;

public class myClass {

        public void myTest(int, int, String) {
        }

        private int doSomething(String) {
        }

}
```

```
package de.example.myapp.activity;
public class anotherActivity {
    {..}
}
```

Profiling Apps & Libraries

Merkle Tree

- Parent hash generated from child hashes
- Efficient integrity checks for large data structures
- Sort hashes for deterministic build order



Method Hashing

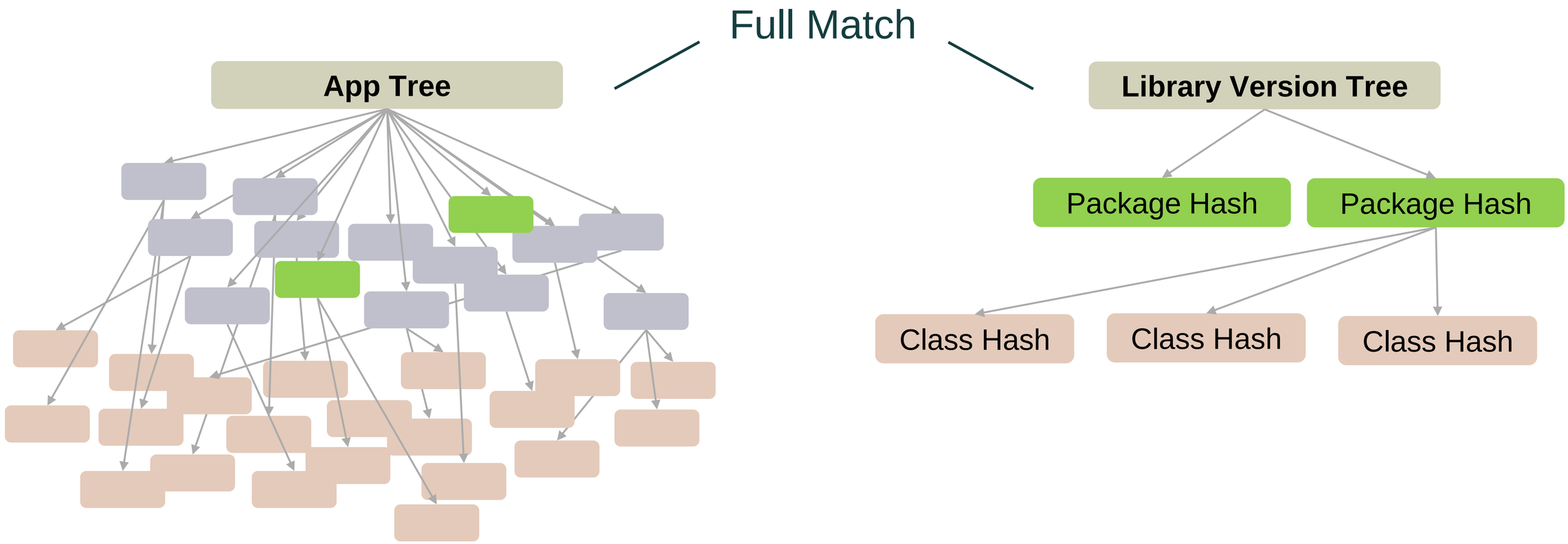
Idea: Replace anything that is prone to identifier renaming

```
signature com.myClass.do(android.content.Context, int, com.Foo) com.Session
descriptor               (android.content.Context, int, com.Foo) com.Session
fuzzy descriptor         (android.content.Context, int, X) X
```

Side-effect: Error introduced at method layer to defeat obfuscation

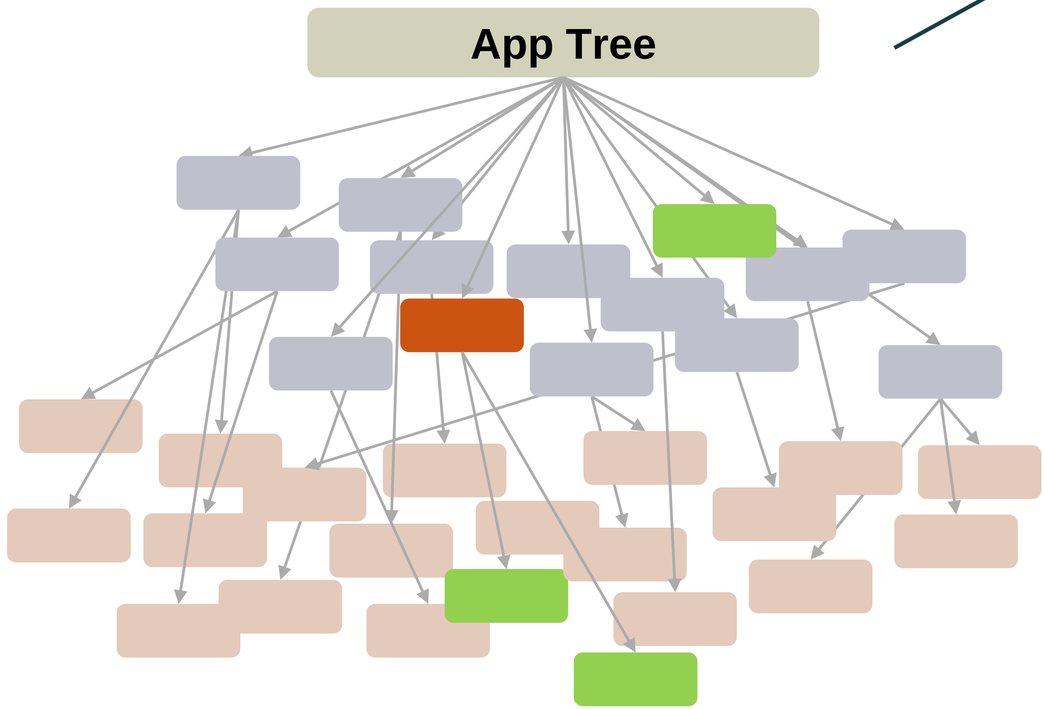
But, error decreases when building the entire tree

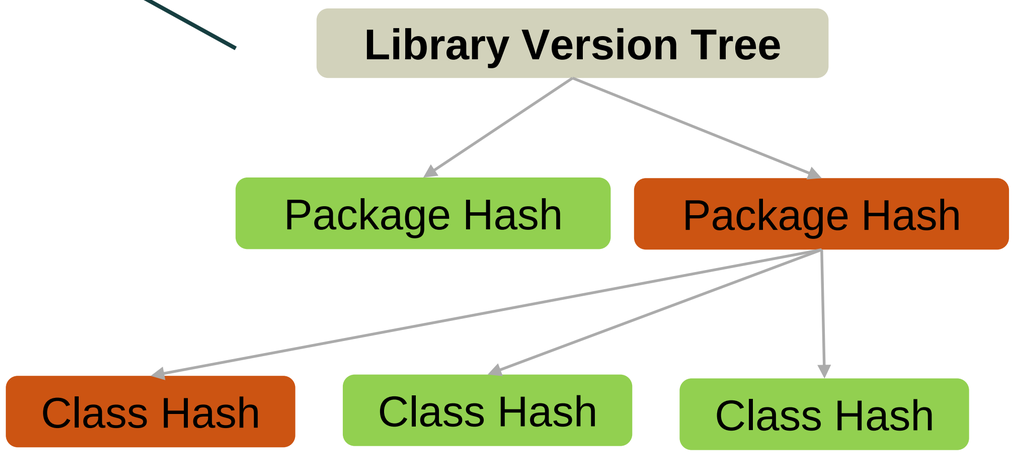
Profile Matching



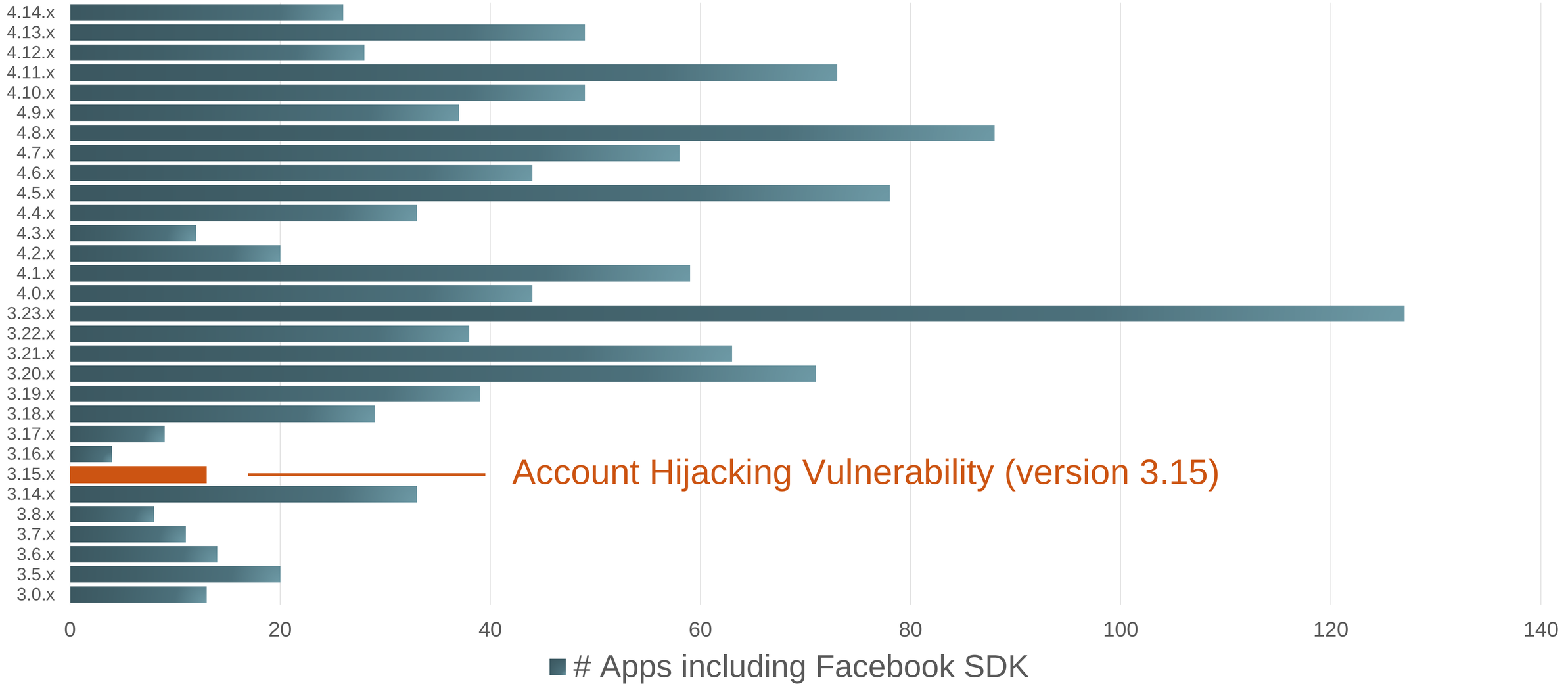
Profile Matching

Partial Match
90% of original library code



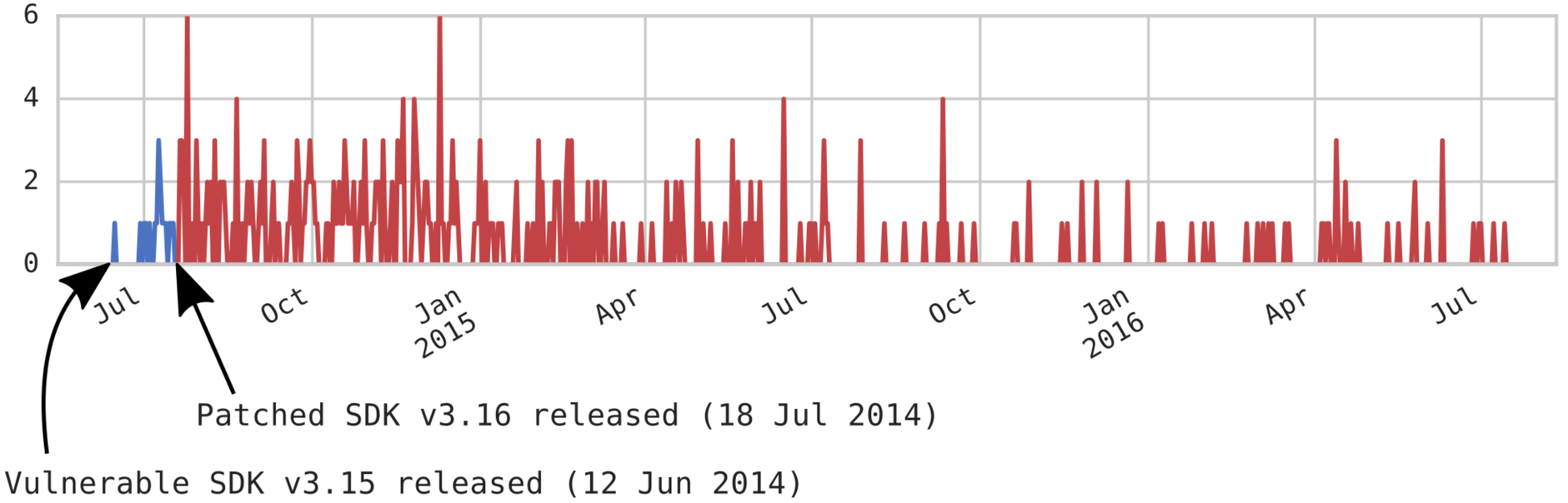


Measuring Library Outdatedness

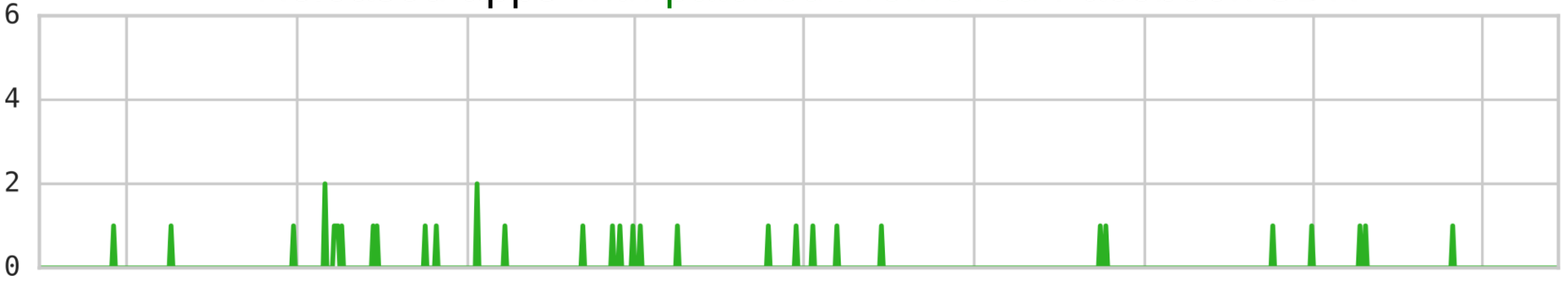


Vulnerability Lifetime

Released apps with vulnerable Facebook SDK (**before** / **after** release of patched SDK)



Released apps with patched / removed Facebook SDK



Call for Action

Android Ecosystem

Need for **Development Tools**

Library Developer
simplify updates

Additional **Platform Support**
(dependency manager)

Market Operator
adopt state-of-the-art analyses

Takeaways

Third-party libraries are a double-edged sword

- Consider potential security risks

Reliable library detection

- Quantify security impact of third-party code
- Raise awareness

No silver bullet

- Requires combined effort to improve status quo more sustainably

LibScout is Open-Source!

https://github.com/reddr/LibScout

- Growing library DB (>230 libs, >4,500 versions)
- Detected more than 20k apps with vulnerable libs