OSSPolice - Identifying Open-Source License Violation and 1-day Security Risk at Large Scale

Ruian Duan, Ashish Bijlani, Meng Xu

Taesoo Kim, Wenke Lee

ACM CCS 2017

Background

- Open Source Software (OSS) is gaining popularity, e.g. GitHub reported 20M users and 57M repos
- Mobile app market grows fast with over 2M apps on Play Store
- Developers reuse OSS as is for lots of benefits
- Legal risks and security risks arise

Risks in OSS use

- OSS licenses have constraints (e.g. GNU GPL requires derivative works to open source)

For now, GNU GPL is an enforceable contract, says US federal judge!

Artifex

Artifex Slaps Palm with PDF Reader Copyright Suit

- 1-day vulnerabilities in stale OSS versions are exploited by hackers

EQUIFAX

Equifax blames open-source software for its record-breaking security breach

CHS

Community Health Systems Breach Possible due to Heartbleed Vulnerability

Goal

- Design a tool, OSSPolice, to analyze Android apps for open-source license violation and 1-day security risk by detecting reuse of OSS and their versions at large scale

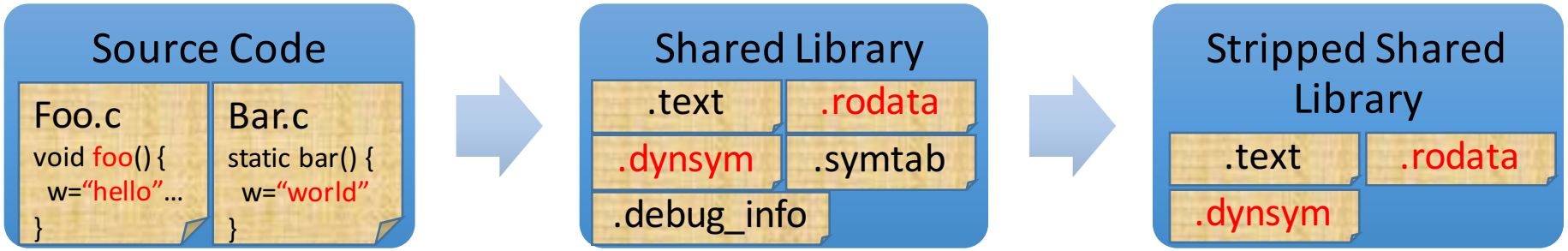
- Requirements
  - Accurate detection for hundreds of thousands of OSS
  - Accurate version pinpointing
  - Efficient resource usage
  - Fast search to support vetting a large number of Android apps

Overview and challenges

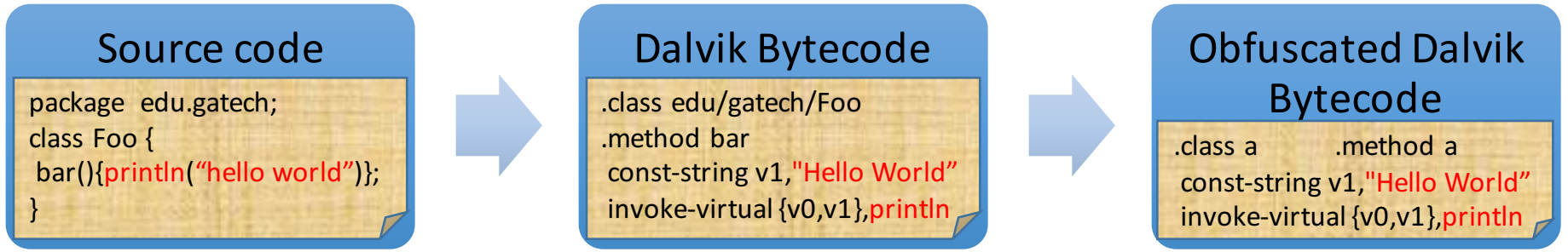
- Feature selection
  - ***Source vs binary***: automatically building source code is hard, due to dependencies, various build configs etc.
- Compare App against OSS
  - ***Fused app binaries***: multiple OSS can be linked or compiled into a single file
  - ***Partial builds*** and ***internal code clones***: not all OSS features are built into libraries and OSS reuses other OSS
- Identify OSS versions
  - ***Cross-match of unique version features***: fused app binaries and internal code clones can confuse the provenance of unique features

Source vs binary

- C/C++ OSS are built into stripped native shared libraries (so files)



- Java OSS are built into obfuscated dalvik executables (dex files)



Feature selection

- C/C++ OSS vs so files
  - String literal
    - Clang-based lexer for OSS and **.rodata** for libraries
  - Exported function
    - Clang-based parser for OSS and **.dynsym** for libraries
- Java OSS vs dex files
  - String constant
  - Normalized class
    - Captures interaction with framework
  - Function centroid
    - Captures intra-procedural control flow

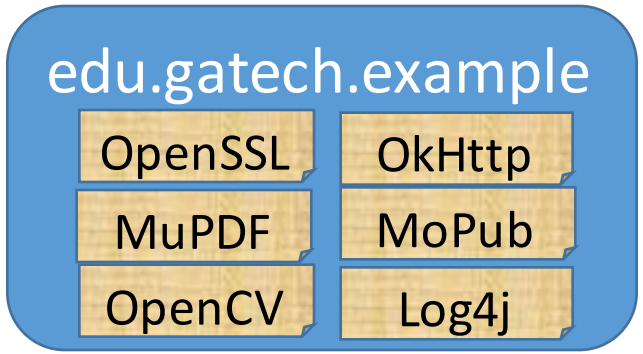
Fast search: no graph based comparison!

Enough or not: 85% shared libraries have ≥50 features!

Uniqueness: 83% of C/C++ and 41% Java OSS versions have unique features

Fused app binaries

- An app uses multiple OSS
  - $\frac{|BIN \cap OSS|}{|BIN|}$
  - $\frac{|OSS \cap BIN|}{|OSS|}$

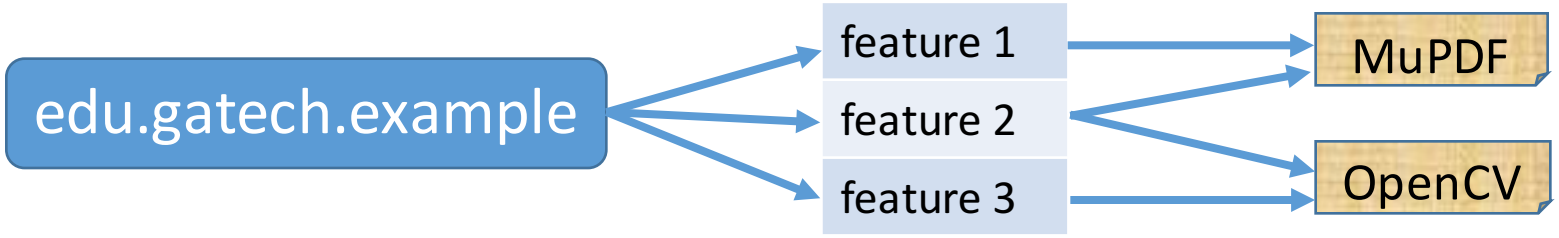


- Iterate N OSS has $O(N)$ time complexity

- Flag all OSS being used at the same time
  - Index OSS and their versions!

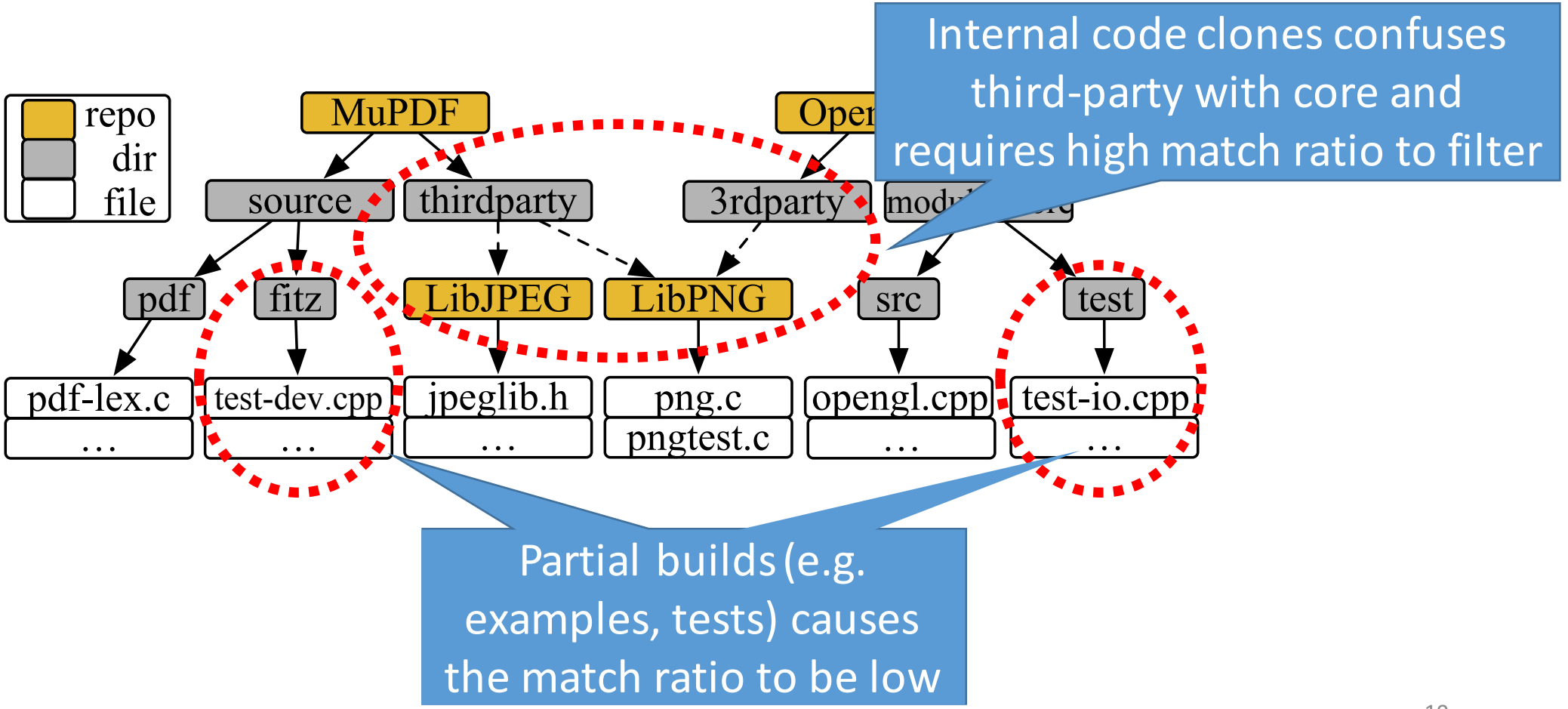
Flat indexing and matching

- Indexing: Maps features to OSS
- Matching: Lookup feature -> OSS mapping to identify OSS reuse



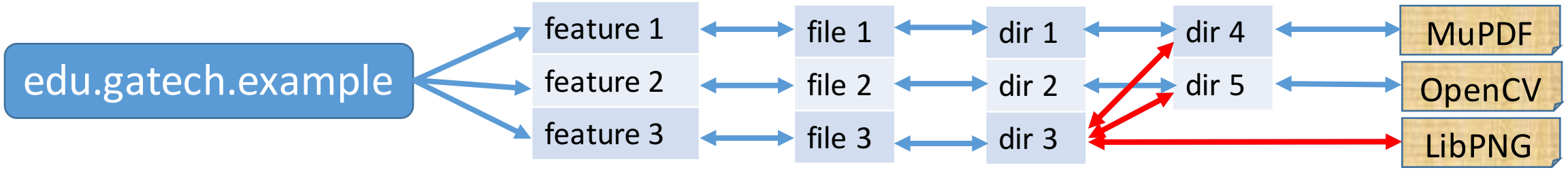
- Flat indexing blow up table to 90G after indexing 7K OSS
- Indexing multiple versions of OSS further adds to the problem
  - Given N OSS with F features and V versions, $O(NFV)$ space complexity

Partial builds and internal code clones



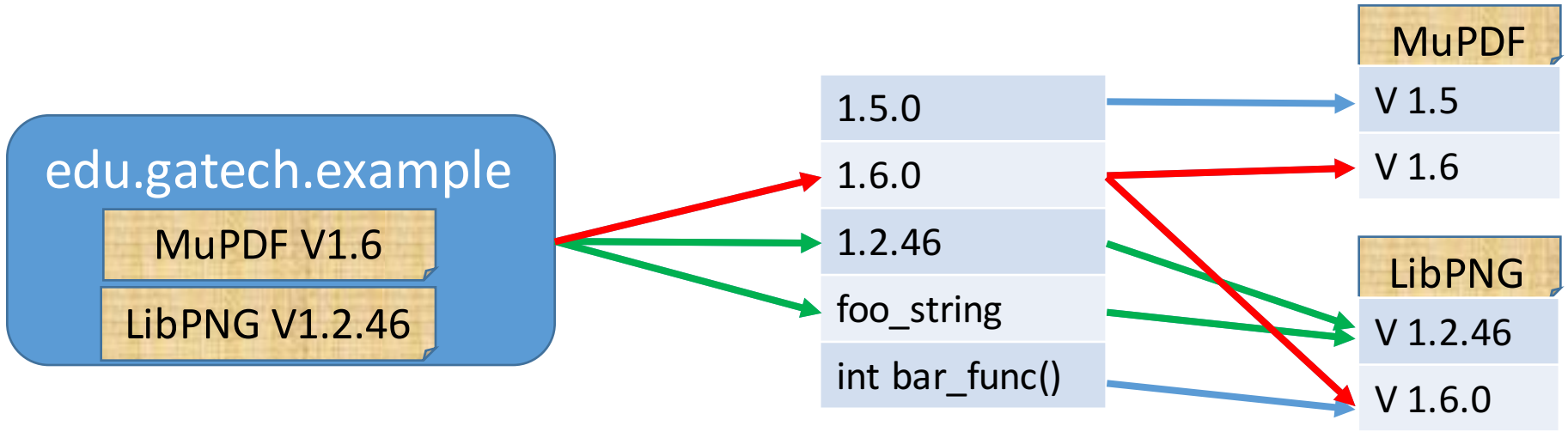
Hierarchical indexing and matching

- Hierarchical Indexing
  - Records source hierarchy to track internal clones
  - Uses Simhash algorithm to generate ids for non-leaf nodes for deduplication
  - Record unique features across versions via separate lists



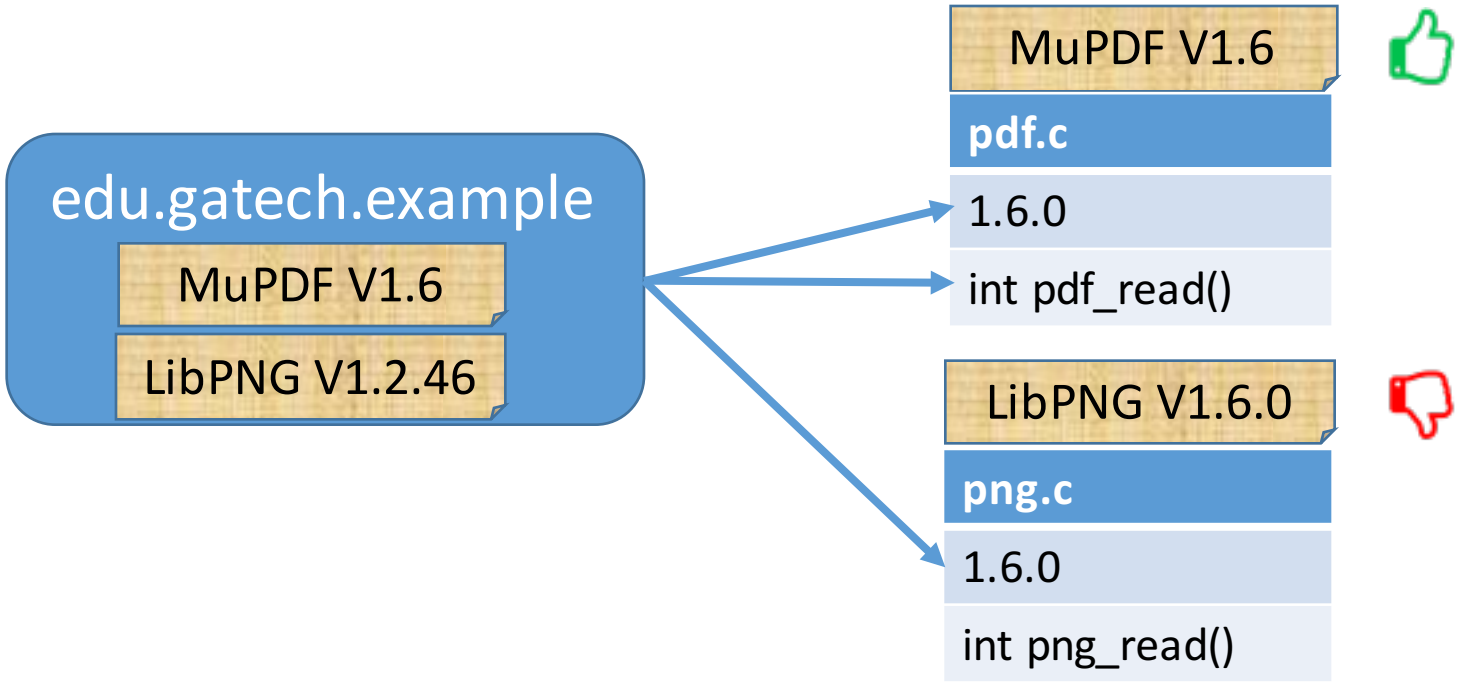
- Hierarchical Matching
  - NormScore (TF-IDF based) to promote unique parts when computing matching ratio of a node
  - **Allow** partial builds by skipping nodes with low ratio
  - **Drop** internal code clones by skipping nodes likely to be third-party

Cross-match of unique version features



Collocation-based filtering

- Leverage collocation information in the indexing table and binaries
- Use NormScore to assign different weights to features

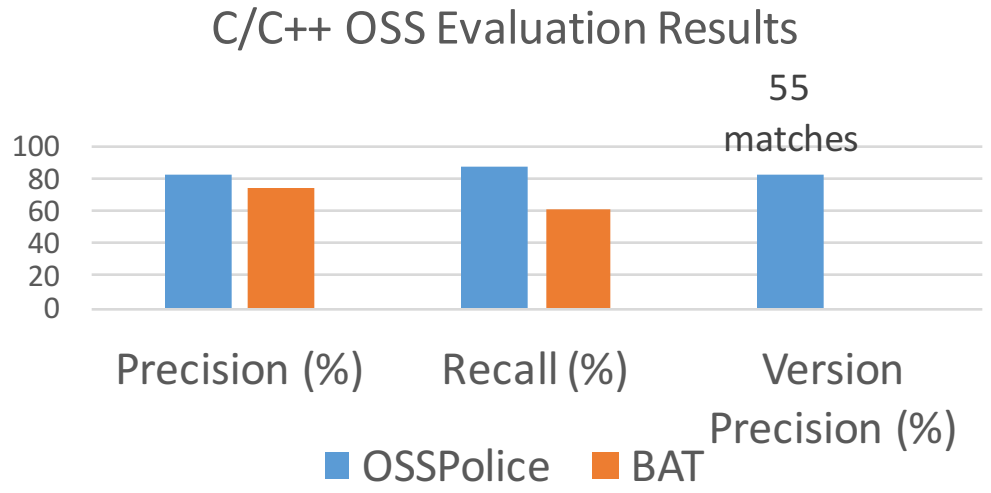


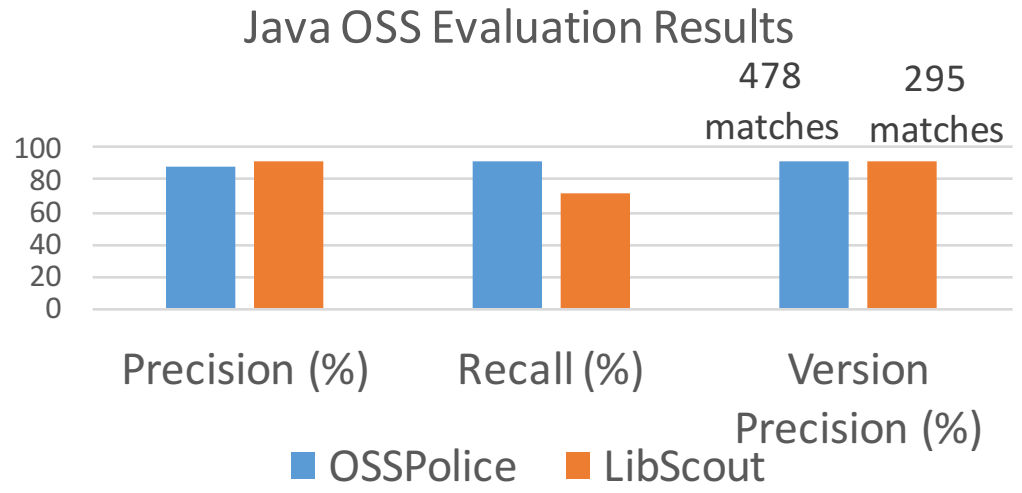
Implementation

- Data Collection
  - Scrapy for crawling of OSS repos
  - PlayDrone for crawling Android apps
- Feature Extraction
  - Clang-based lexer and parser for C/C++ source
  - Pyelftools for native binaries
  - Soot-based parser for Java bytecode and Dex bytecode
- OSS Detection
  - Redis key-value cluster for storing and querying indexing results
  - Celery job scheduler for distributing work to multiple servers

Evaluation

- FDroid Apps
  - 4,469 apps, 579 with native libraries
  - 295 C/C++ OSS uses, 7,055 Java OSS uses
- BAT: internal code clones
- LibScout: partial builds (code removal)



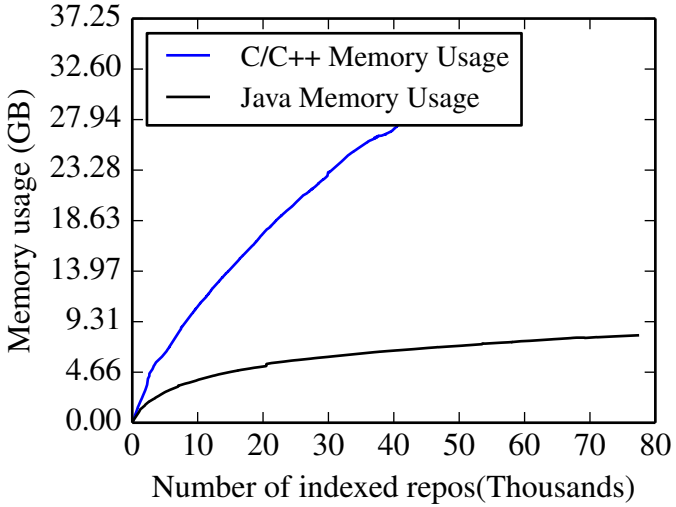


Measurement Dataset

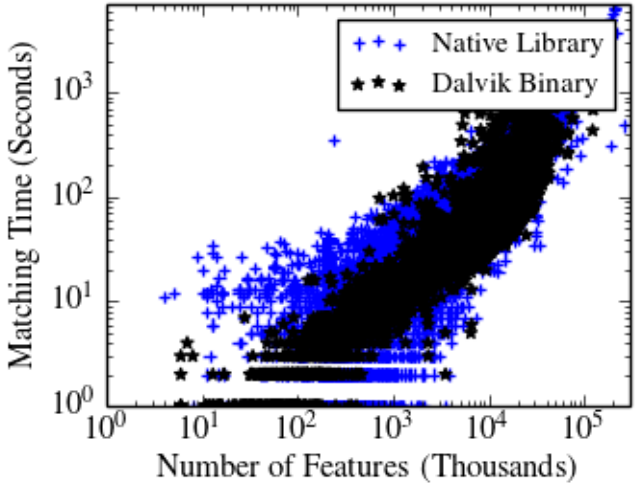
- C/C++ OSS from GitHub
  - 3,119 popular repos and 60,450 OSS versions
  - 29% repos are GPL/AGPL
  - 11% repos are vulnerable with 5,611 severe CVEs ($CVSS \geq 4.0$)
- Java OSS from Maven and JCenter
  - 4,777 popular artifacts, 77,308 artifact versions
  - 2.3% artifacts are GPL/AGPL
  - 1.7% artifacts are vulnerable with 452 severe CVE ids
- Android Apps from Google Play
  - 1.6M apps, 515,812 with native libraries

Performance and Scalability

- Indexing
  - 60,450 C/C++ repos and 77,308 Java repos
  - Time cost is 1000s vs. 40s on average
  - Memory grows sublinearly to 30GB and 9GB

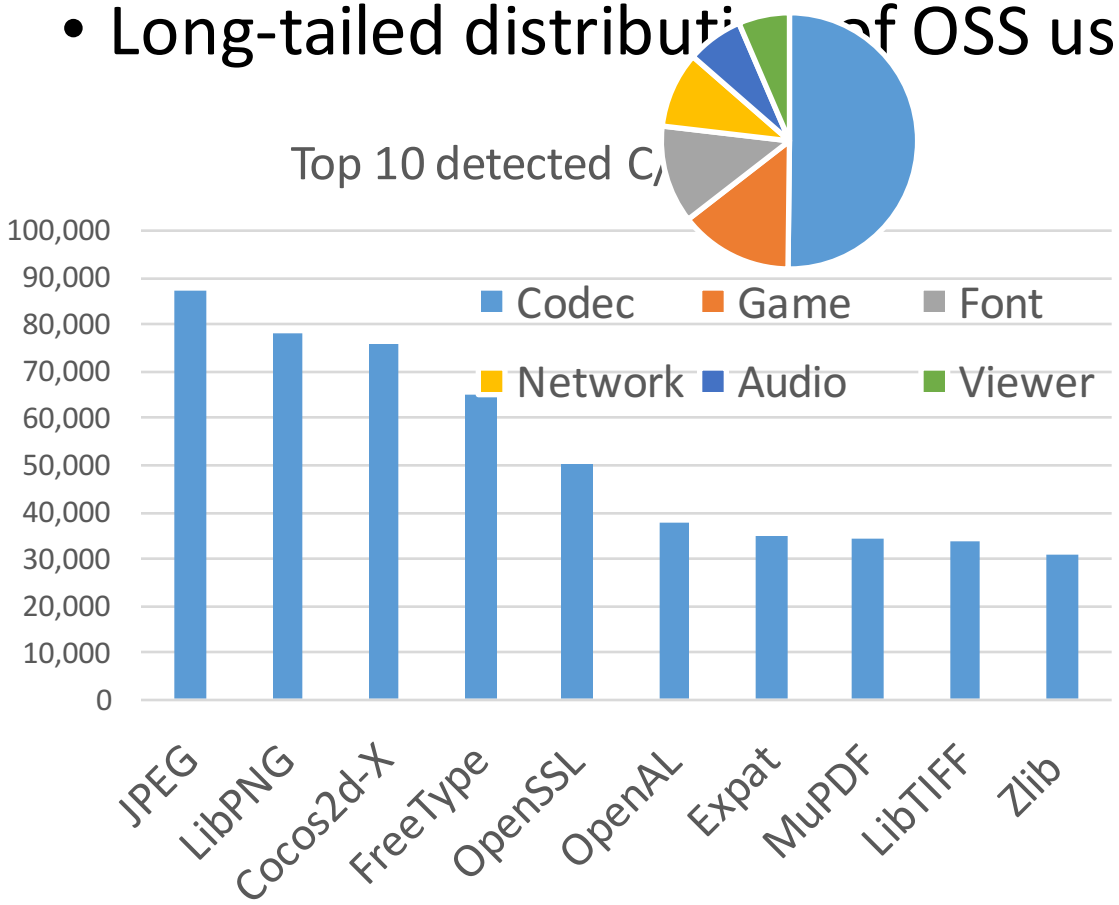


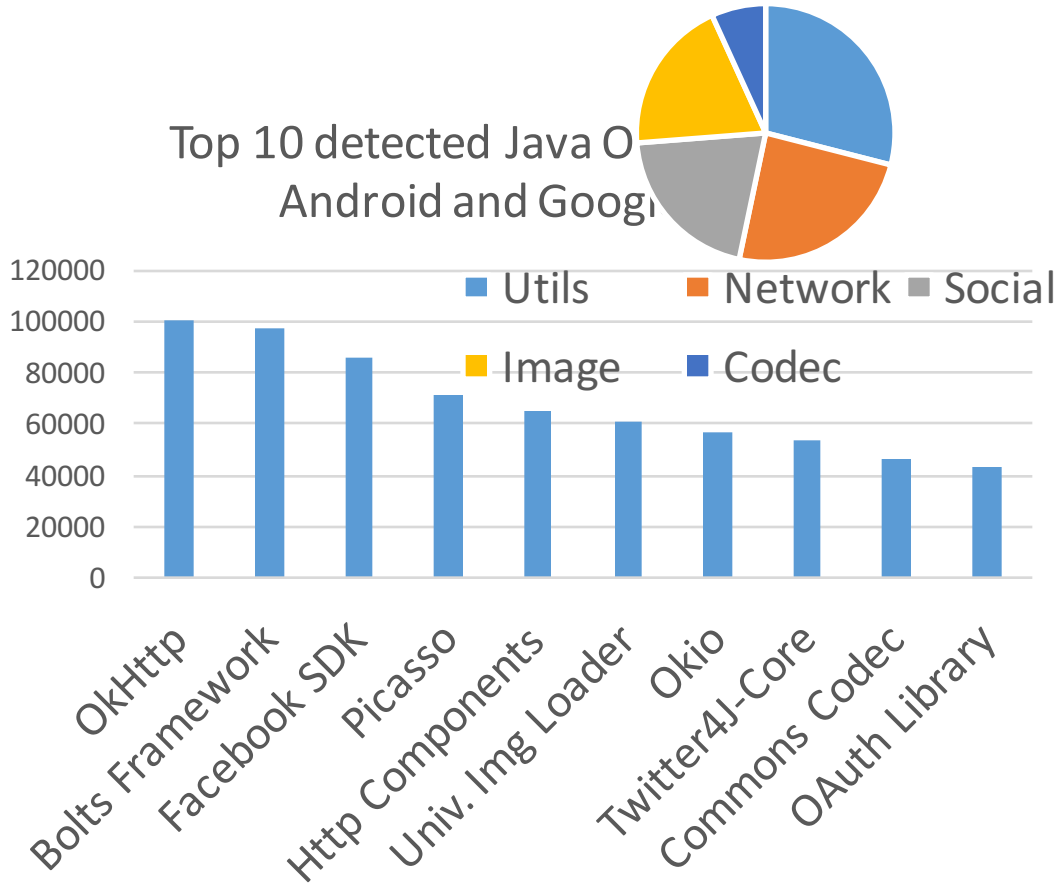
- Matching
  - Sampled 10,000 Google Play apps
  - 80% of dex and so files finish within 100s and 200s



Popular libraries

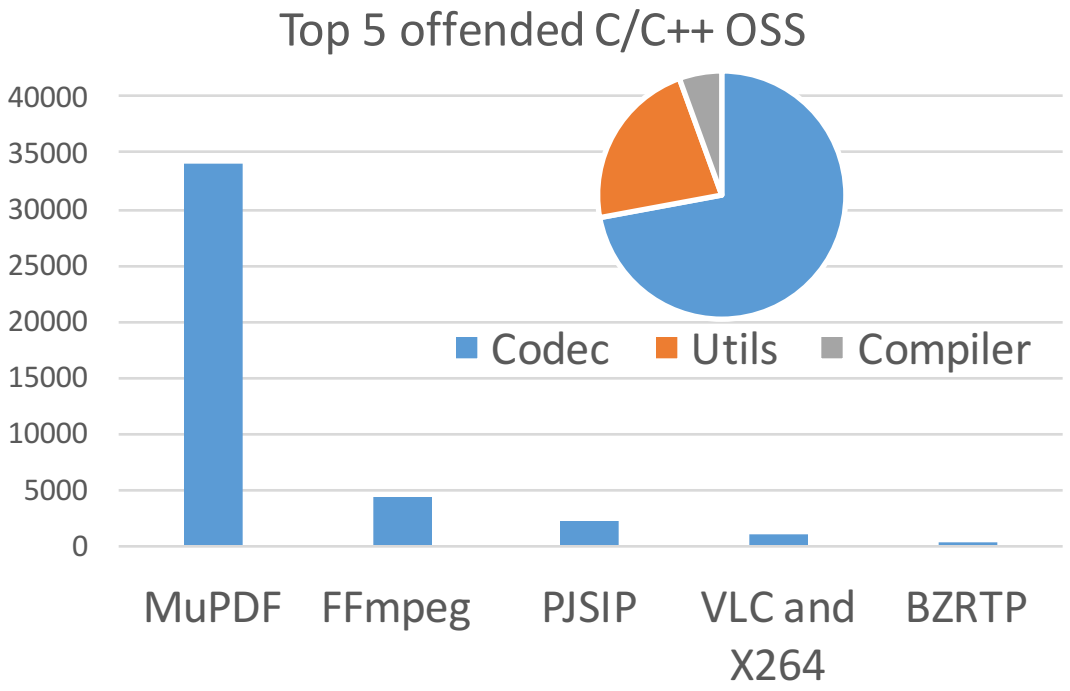
- Long-tailed distributi of OSS uses

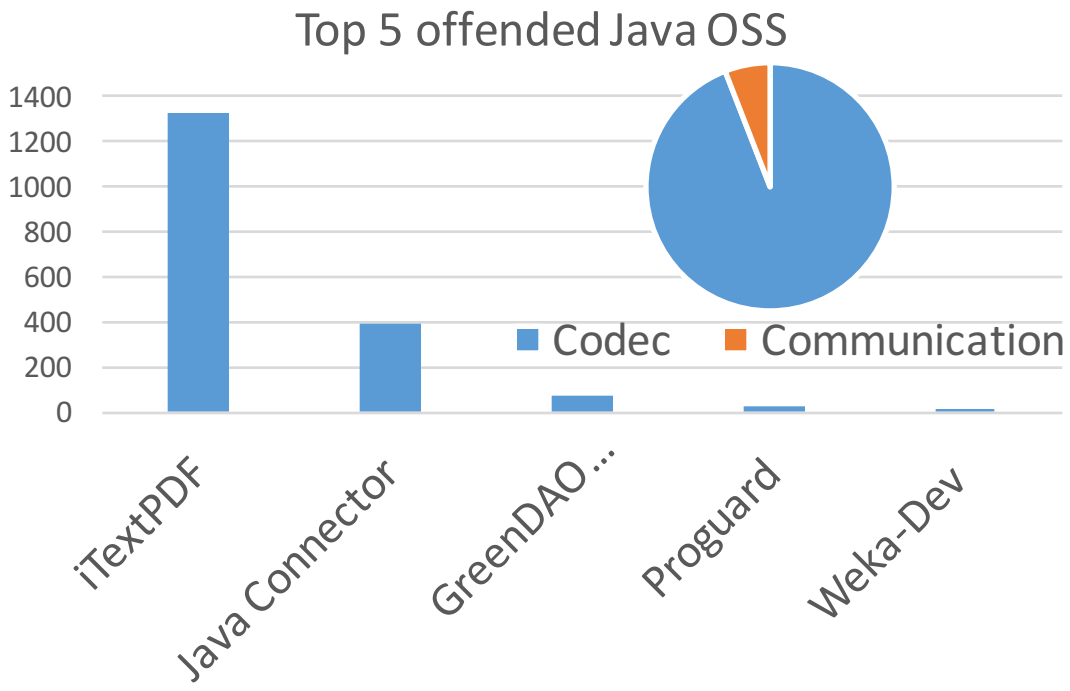




Legal Risks

- More than 40K potential GPL violators
- More violators using C/C++ than Java and encoding libraries dominate



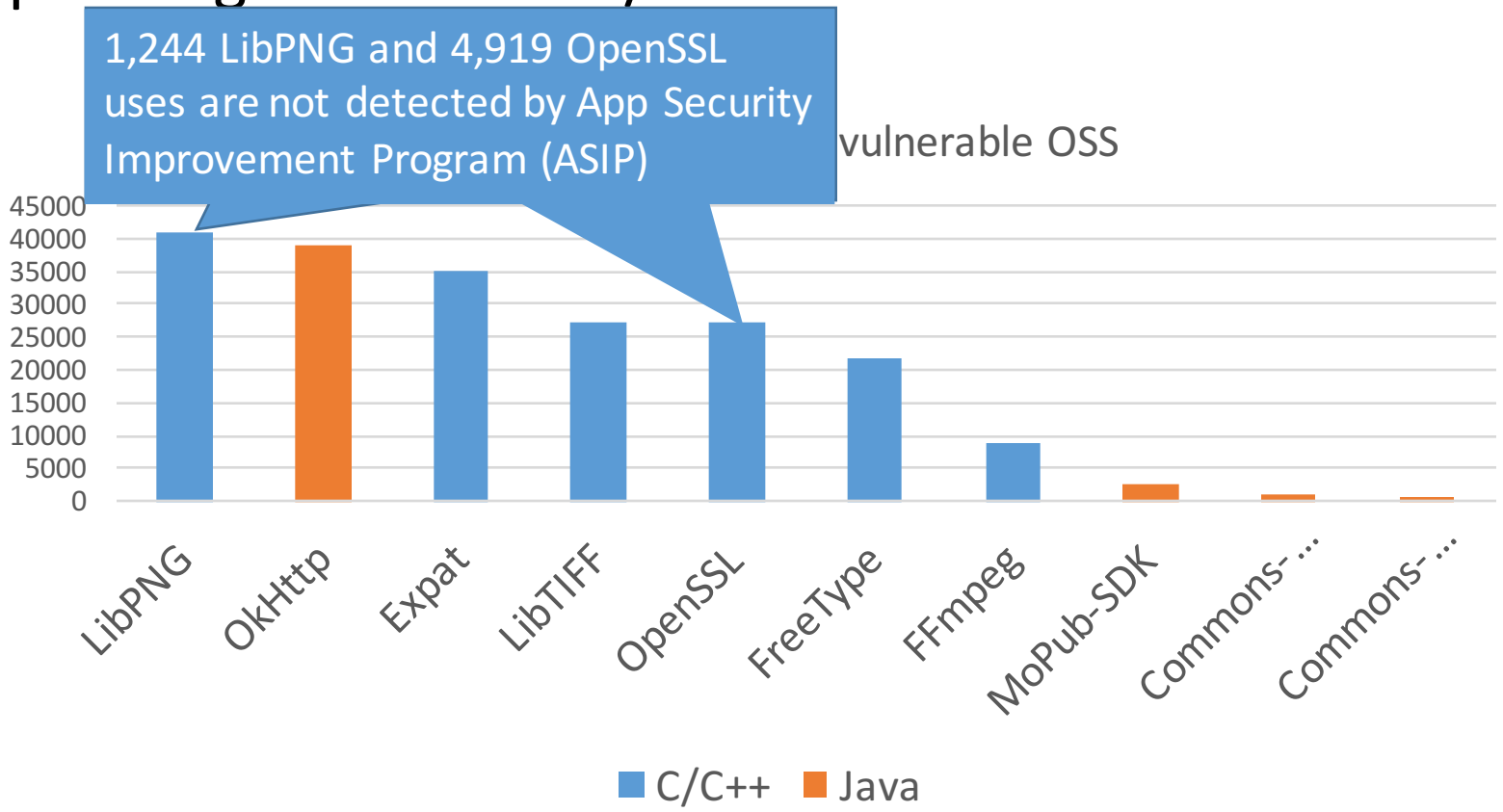


Legal Risks

- Why violating GPL/AGPL?
  - MuPDF and iTextPDF are used due to lack of free alternatives
- OSS developers responses
  - MuPDF got new customers ☺
  - FFmpeg and VideoLAN have interest, but FFmpeg cannot enforce ☺
  - PJSIP not interested due to NDA, iText did not reply ☹
- Awareness of OSS licensing terms
  - None of the app developers provided source code yet ☹

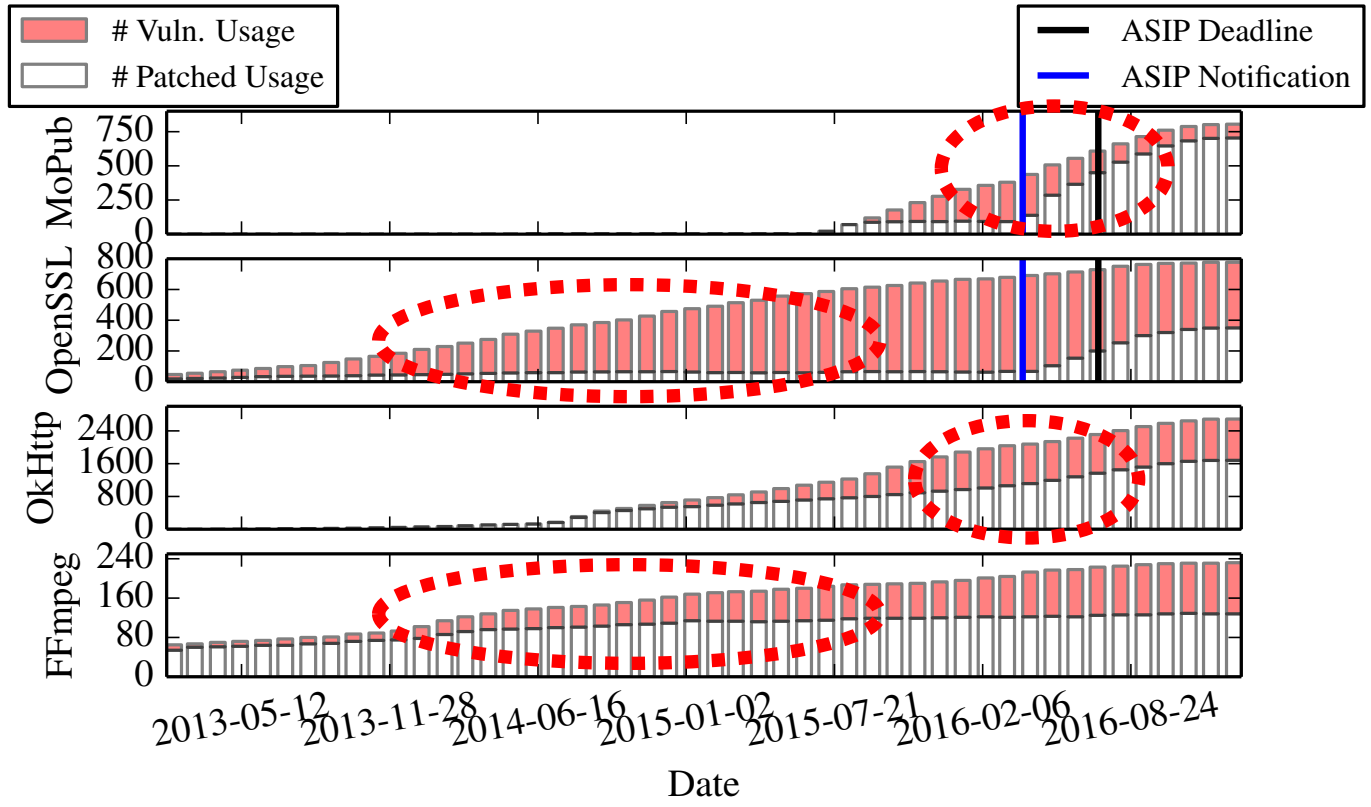
Security Risks

- More than 100K apps using vulnerable OSS versions
- More apps using vulnerable C/C++ OSS than Java



Security Risks

- Which versions of OSS do new app developers choose?
  - Both vulnerable and patched OSS are being used
- When do developers update OSS versions?
  - ASIP mitigates vulnerable OSS usage, but still remains a problem



Timeline of OSS usage for the top 10K apps, 300K app versions

Discussion

- Checking license compliance requires manual efforts
- Obfuscation and optimization
  - String encryption in dex files
  - Function hiding in so files
- Version pinpointing
  - Not all versions can be uniquely identified
- More programming languages (i.e. JS, Python) and platforms (i.e. iOS)

Conclusion

- OSSPolice: an accurate and scalable tool to identify license violations and 1-day security risks
  - Hierarchical indexing and matching scheme
  - Collocation-based unique feature filtering
- A large scale measurement
  - 1.6M free Google Play Store apps
  - 40K cases of potential GPL/AGPL violations and 100K apps using vulnerable OSS
- Interesting insights
  - App developers violate GPL/AGPL due to lack of free alternatives
  - App developers use vulnerable OSS versions despite efforts from Google