ATVHunter: Reliable Version Detection of Third-Party Libraries for Vulnerability Identification in Android Applications

Authors : Xian Zhan, Lingling Fan, Sen Chen, Feng Wu, Tianming Liu, Xiapu Luo, Yang Liu. Published in ICSE 2021

Siva Sai Nagender Vasireddy

September 2021

Third-party libraries (TPLs)

- Facilitate fast development of Android Applications and are widely used.
  - About 57% of Android Apps include ad libraries.
  - More than 60% of the code in an app belongs to TPLs.

- **Security threat** : Exploit the vulnerabilities or inject backdoors in the popular TPLs.
  - Developers may not know how many and which TPLs are used in their apps due to many direct and transitive dependencies.
  - Identifying the exact version of a TPL is challenging. Not all TPL versions are vulnerable.

Workflow of ATVHunter

Takes an Android app as input and automatically identifies the used vulnerable TPL-Vs (if any).

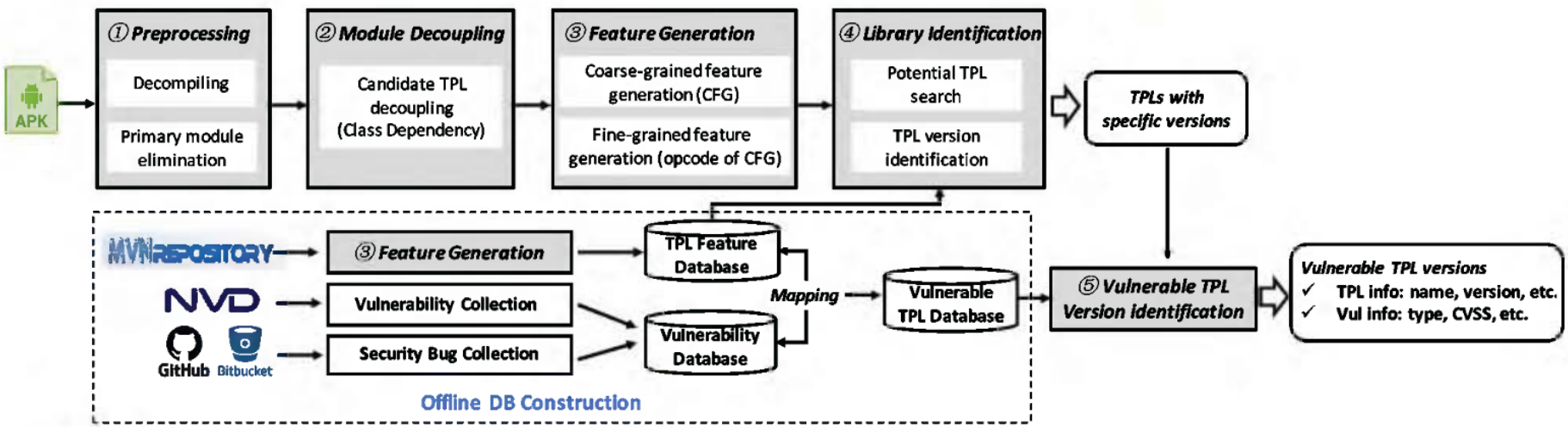


Fig. 1: Workflow of ATVHUNTER

Preprocessing

- Decompile the input app and transform the bytecode into appropriate intermediate representations.
- Delete the primary module in the app. Primary module is the code of the host app.
  - Parse the AndroidManifest.xml fil and get the host app packages.
  - Delete files under the host namespace.

- Potential issues due to deletion of files
  - Part of host code might not be deleted due to code obfuscation or special package names.
    - If the host code and TPLs have no dependencies, it will not affect accuracy of TPL identification.
    - If the undeleted host parts include the TPLs, interference can be eliminated in the comparison stage.
  - TPLs with the same package namespace as the host app might lead to their deletion
    - Might lead to false negatives.

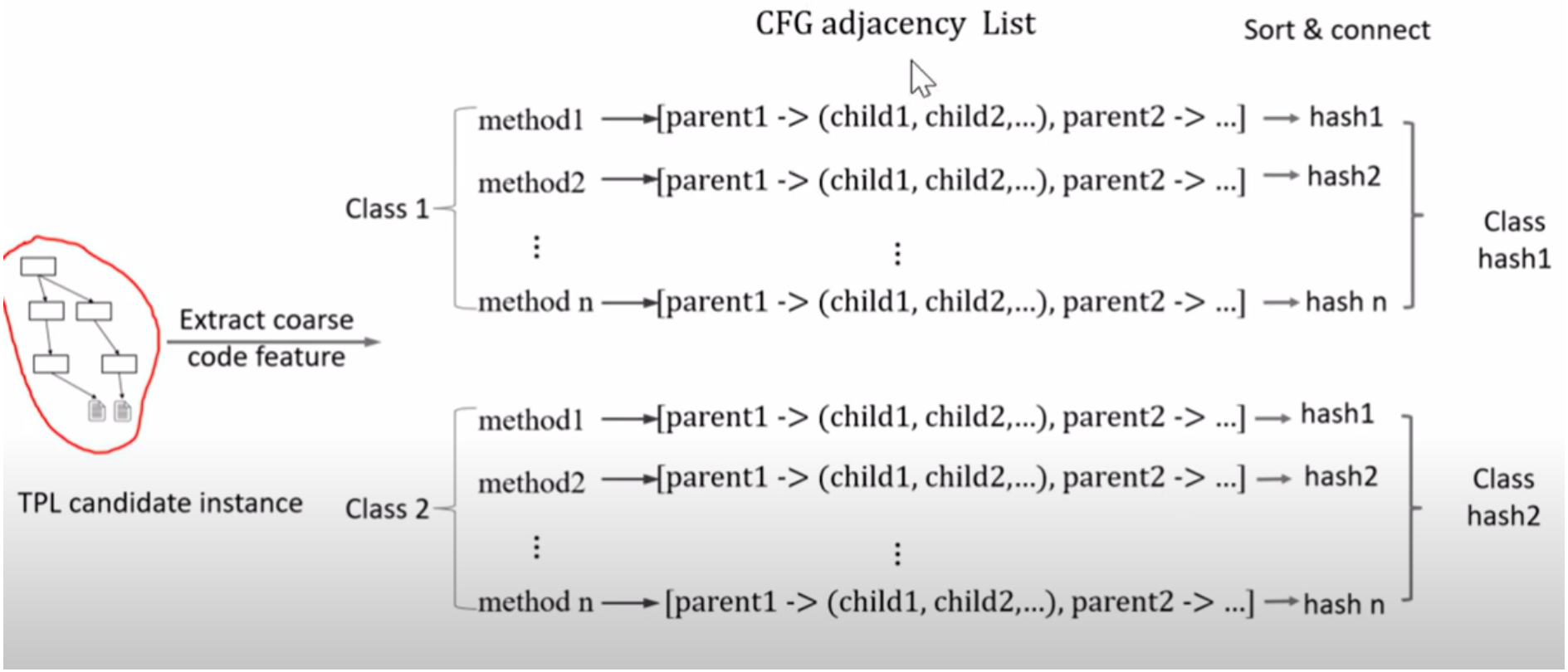
Module Decoupling

- Split up the non-primary module of an app into different independent library candidates.
- Class dependency graph (CDG) is used to split up TPL candidates.
- CDG is resilient to package flattening.
- Class dependency relationships include
  - Class inheritance
  - Method call relationship
  - Field reference relationship
- Related class files in each CDG will be considered as a TPL candidate.

Feature Generation

- Extract coarse-grained and fine-grained features to represent each TPL candidate.
- **Coarse-grained feature extraction**
  - Extract Control Flow Graph (CFG) for each method.
  - Traverse the CFG and assign each node a unique serial number according to the execution order.
    - For a branch node with sequence number *n*, its child with more outgoing edges will be given a sequence number *n+1* and the other child is given *n+2*.
    - If two child nodes have the same outgoing edges, we will give *n+1* to the child node with more statements in the basic block.

Coarse-grained Feature



Fine-grained Feature

- The coarse-grained features are likely to generate the same signature of different versions that have minor changes.

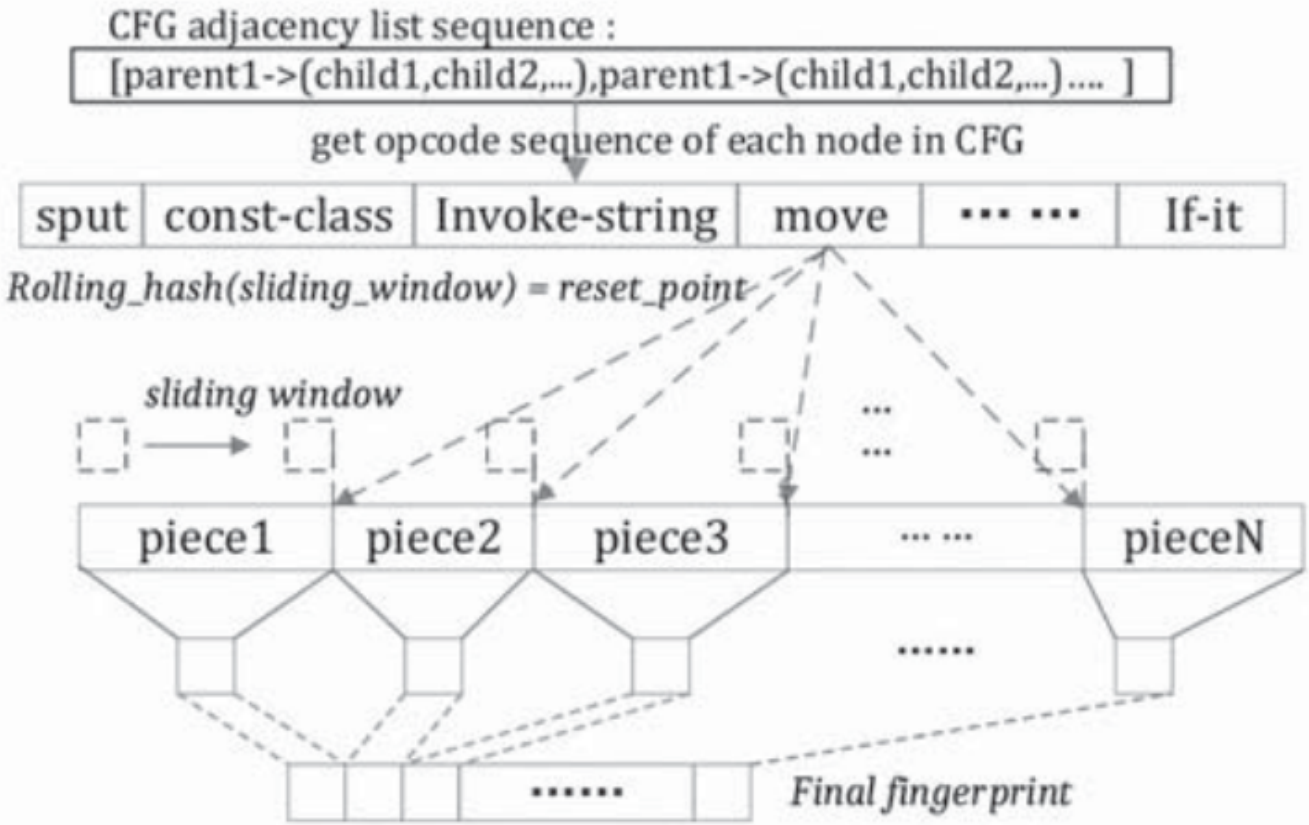
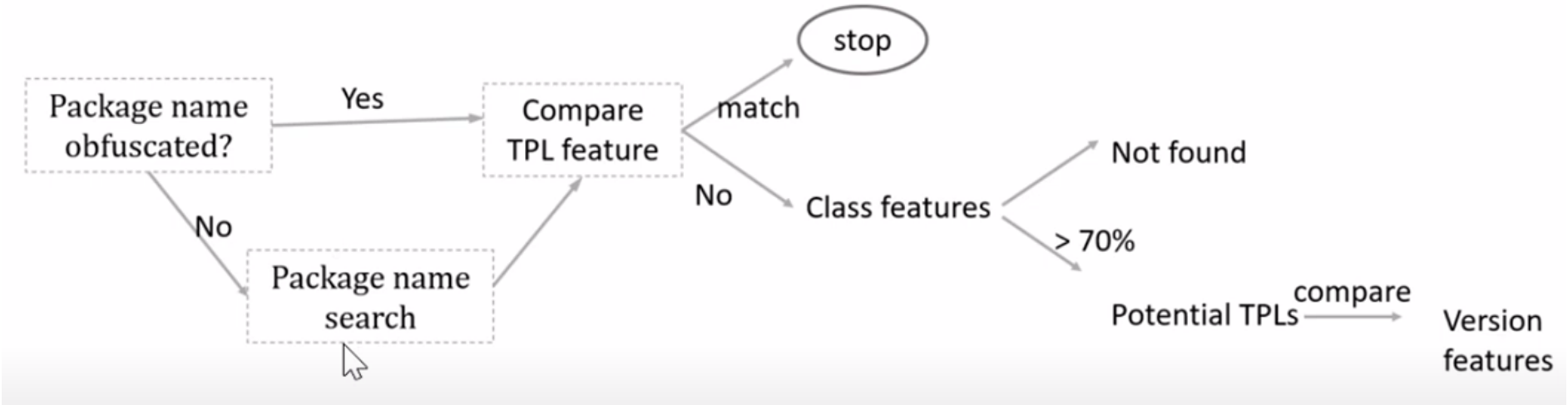


Fig. 3: Fuzzy hashing for method feature generation as the version feature

TPL Database Construction

- Crawled all Java TPLs from Maven Repository to build TPL database.
  - 189,545 unique TPLs with their 3,006,676 versions.
  - Query the vulnerable TPLs frpm the public CVE (Common Vulnerabilities and Exposures) database use *cve-search*.
    - 1,180 CVEs from 957 unique TPLs with 38,243 affected versions.
- Store both coarse-grained and fine-grained features in a MongoDB database.
- Size of the entire database is 300GB.
- It took more than 1 month to collect all of the TPLs and another two months to generate the TPL feature database.

TPL Identification



Method and TPL Similarity Scores

- Method Similarity Score is based on the edit distance between two method fingerprints.

$$MSS(m_a, m_b) = 1 - \frac{d[m_a, m_b]}{max(m, n)} \tag{1}$$

- MSS Threshold is set as 0.85 based on experiments.
- TPL Similarity Score is based on the number of matched methods.

$$TSS(t_1, t_2) = \frac{M_{|t_1 \cap t_2|}}{M_{|t_2|}} \tag{2}$$

- t_1 is a TPL candidate from the test app and t_2 is a TPL from the database for comparison.
- TSS threshold is set at 0.95 based on experimental results.

Implementation

- APKTOOL : Tool to decompile Android apps.
- Androguard : Class dependency relations.
- SOOT : Generate CFG and get the opcode sequence in each basic block of a CFG.
- ssdeep : Implementation of fuzzy hash algorithm.
- 2k+ lines of python code.

Ground-truth Dataset Construction

- Used to compare the performance with state-of-the-art tools.
- Consists of open-source apps from F-DROID as it is difficult to know thw specific TPL-Vs from commercial apps.
- Construction steps
  - Collect the latest versions of 500 open-source apps from F-DROID.
    - Specific TPL information (including the version) is available in the configuration and source code of the apps.
  - Collected apps are from 17 different categories with various sizes.
  - Manually analyze each app and get the in-app TPLs with their specific versions.
    - The number of TPLs in each app ranges from 2 to 37.
    - Download these TPLs with their versions from the Maven repository.
    - Filtered 144 apps out due to incomplete versions of TPLs maintained in the Maven repository.
  - Final ground truth dataset contains 356 apps and 189 unique TPLs with the complete 6.819 version files.

Evaluation

- Comparison with other tools is done using open-source apps collected from F-Droid.
- Effectiveness Evaluation
  - Precision : $\frac{TP}{TP+FP}$
  - Recall : $\frac{TP}{TP+FN}$
  - F1 Score : $\frac{2*Precision*Recall}{Precision+Recall}$

TABLE I: Library and Version Detection Comparison

Tools	Library-level			Version-level		
	Precision	Recall	F1	Precision	Recall	F1
ATVHunter	**98.58%**	**88.79%**	**93.43%**	**90.55%**	**87.16%**	**88.82%**
LibID	98.12%	68.45%	80.64%	68.70%	66.42%	67.54%
LibScout	97.10%	46.65%	63.02%	44.82%	43.50%	44.15%
OSSPoLICE	97.91%	43.39%	60.13%	88.83%	42.25%	57.26%
LibPecker	93.16%	57.82%	71.35%	60.35%	57.67%	58.98%

Efficiency Evaluation

- Comparison of detection time
  - All tools construct their own TPL databases using the same dataset.
  - Detection time is the period cost for finding all TPL-Vs in a test app. Detection time does not include database construction time.
  - Evaluation metrics
    - Q1 : Median of the lower half of the data
    - Mean
    - Median
    - Q3 : Median of the upper half of the data

TABLE II: Comparison Results of Detection Time (per app).

Tool	ATVHunter	LibID	LibScout	OSSPoLICE	LibPecker
Q1	15.92s	51.43s	30s	33.48s	12168s
Mean	66.24s	59616s	83s	2052.34s	16396s
Median	47.78s	9286s	64s	80.42s	16632s
Q3	90.30s	38300s	100s	226.60s	23292s

Obfuscation-resilient capability

- Tested on 100 apps. Used an obfuscation tool, Dasho, to obfuscate these apks with four obfuscation techniques i.e., renaming obfuscation, control flow randomization, package flattening and dead code removal.

TABLE III: Comparison on Code Obfuscation Techniques

Tool	No Obfuscation	Obfuscation			
		Renaming	CFR	PKG FLT	Code RMV
ATVHunter	**99.26%**	**99.26%**	**90.13%**	**99.26%**	75.57%
LibID	12.93%	12.93%	0.03%	1.58%	2.49%
LibScout	88.75%	88.75%	18.24%	17.69%	17.69%
OSSPoLICE	85.62%	85.62%	23.04%	39.52%	48.86%
LibPecker	98.79%	98.79%	86.63%	73.56%	**79.28%**

Renaming: renaming obfuscation; CFR: Control Flow Randomization; PKG FLT: Package Flattening; Code RMV: Dead Code Removal

Large Scale Analysis

- A large scale study has been conducted on Google Play apps to reveal threats of TPL-Vs in the real world.
  - Collected popular apps whose installation range is from 10,000 to 5 billion.
  - Collected 104,446 apps across 33 different categories.
    - 72% (73,110/104,446) of them use TPLs.
- Vulnerable TPL Landscape
  - IF CVSS v3.0 score $>$ 7.0, then the severity of vulnerability is critical.
    - 21.35% of all vulnerabilities in the dataset.
    - Severe vulnerabilities usually include remote code execution, sensitive data leakage, server-side request forgery.
    - 74.95% of these vulnerable TPLs are widely used by other TPLs.
    - The library "org.scala-lang:scala-library" with a severe security risk (CVSS=9.8) that allows local users to write arbitrary class files has been used 24,112 by other TPLs and most of vulnerable versions of this TPL have been used more than 2000 times.

Large Scale Analysis Contd...

- Impact Analysis of Vulnerable TPLs
  - 12.37% (9,050/73,110) of apps include TPL-Vs, involving 53,337 known vulnerabilities and 7,480 security bugs from open-source TPLs.
  - The known vulnerabilities are from 166 different vulnerable TPLs with corresponding 10,362 versions and the security bugs are from 27 vulnerable TPLs with 284 different versions.
  - These vulnerable apps use a total of 58,330 TPLs and approximately 18.2% of them are vulnerable ones.
  - Among the 9,050 vulnerable apps, 329 apps (37.5%) with TPLs contain both vulnerabilities and security bugs.
  - There are 778 apps containing the TPLs with security bugs and each app contains about 2.45 security bugs in their TPLs.
  - Many educational and financial apps use the popular UI library "PrimeFaces" that include severe vulnerability (CVE-2017-1000486).
    - Primefaces 5.x is vulnerable to a weak encryption flaw resulting in remote code execution.