Each folder contains a tool, including the following components: ReadMe.md file, code folder, experiment folder, and result folder. For instance, for the tool DNADroid, its directory's structure is as follows.

DNADroid.
- ReadMe.md (provides general introductions, settings, environments, and additional remarks for DNADroid);
- code (source code folders which contain all source code of DNADroid);
- experiment (code and settings for experiment)
- result (experimental results)

■ Dataset Construction

Based on our research, we found that 13 techniques chose Genome as the dataset. Genome project collected 1,260 malware samples, which cover 49 Andriod malware families. 86% apps of the malware samples are repackaged apps. We did not choose the Genome data as our dataset because this dataset does not provide the corresponding original apps of the repackaged pairs. Hence, if we want to use them, we have to spend a large amount of time in manually verifying the repackaged app pairs.

The goal of our research is to conduct a comparative assessment of different repackaging detection techniques. The dataset comes from a branch of AndroZoo named Repack[1], which is a repository of repackaged Android app pairs. Repacks includes 18,073 apps. It offers 15,297 repackaged pairs, including 2,776 original apps and corresponding 15,297 repackaged apps. These apps have calculated their similarity by similarity measure algorithms and original researchers of AndroZoo have verified its effectiveness and accuracy. The dataset size is larger than that of the Genome malware project (15 times). AndroZoo contains most up-to-date apps. Moreover, AndroZoo is a growing collection of Android apps from different app stores, including Android official market Google play and other alternative app markets. Currently, AndroZoo has collected more than 8 million apps. Each original apps of Repack from AndroZoo are average repackaged 2.168 times. This dataset contains both small-size and large-size apps and covers over 40 distinct repackaged malware families. For more detailed statistics of our dataset, we also conducted a thorough analysis of these apps.

■ Methodology of Implementation

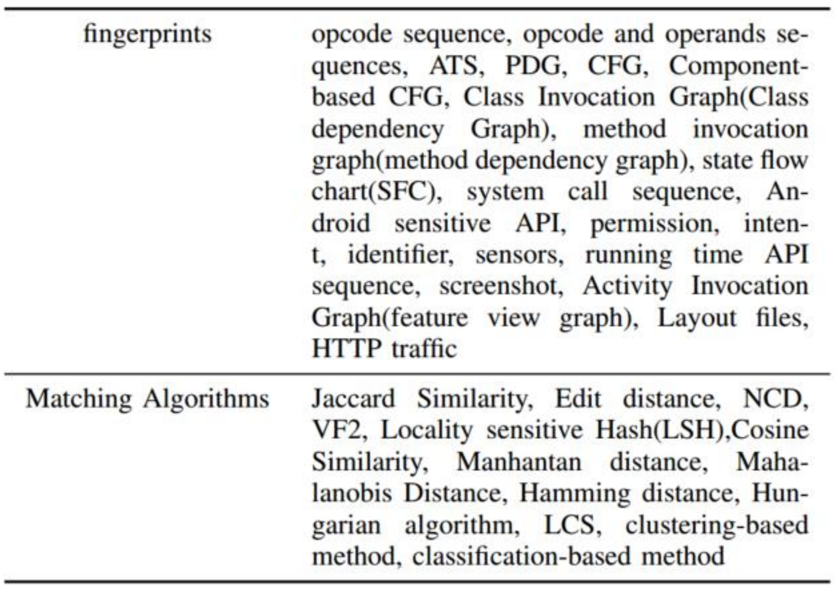
For each repackaging detection system, it can be roughly divided into five steps in Section II Background B. Repackaged APPs Detection: 1) Pre-Processing, 2) Feature extraction, 3) Filtering out uninterested Features, 4)Similarity Detection, and 5)Post-Processing. The primary steps of repackaging detection are feature extraction and similarity detection. We just need to extract the features and combine different matching detection algorithm, in

[1] https://github.com/serval-snt-uni-lu/RePack

this way, we can easily create different systems.

Based on the source representations and matching detection algorithms from the collected papers, we get the following Table I.

Table I Refined fingerprint and matching algorithm



fingerprints	opcode sequence, opcode and operands sequences, ATS, PDG, CFG, Component-based CFG, Class Invocation Graph(Class dependency Graph), method invocation graph(method dependency graph), state flow chart(SFC), system call sequence, Android sensitive API, permission, intent, identifier, sensors, running time API sequence, screenshot, Activity Invocation Graph(feature view graph), Layout files, HTTP traffic
Matching Algorithms	Jaccard Similarity, Edit distance, NCD, VF2, Locality sensitive Hash(LSH),Cosine Similarity, Manhantan distance, Mahalanobis Distance, Hamming distance, Hungarian algorithm, LCS, clustering-based method, classification-based method

Take DroidMOSS as an example. It needs to extract the opcode sequences and use the fuzzy hash to generate the fingerprint Finally, the edit distance algorithm is selected as the Matching Detection algorithm to find the repackaged apps. In this way, we can implement this DroidMOSS.

We thoroughly analyzed the collected papers set and summarized these techniques. We first extracted the common parts/techniques and implemented them, and then we implemented the different techniques one by one. The different techniques' modules of our implementations are shown in the following figures(I-V).

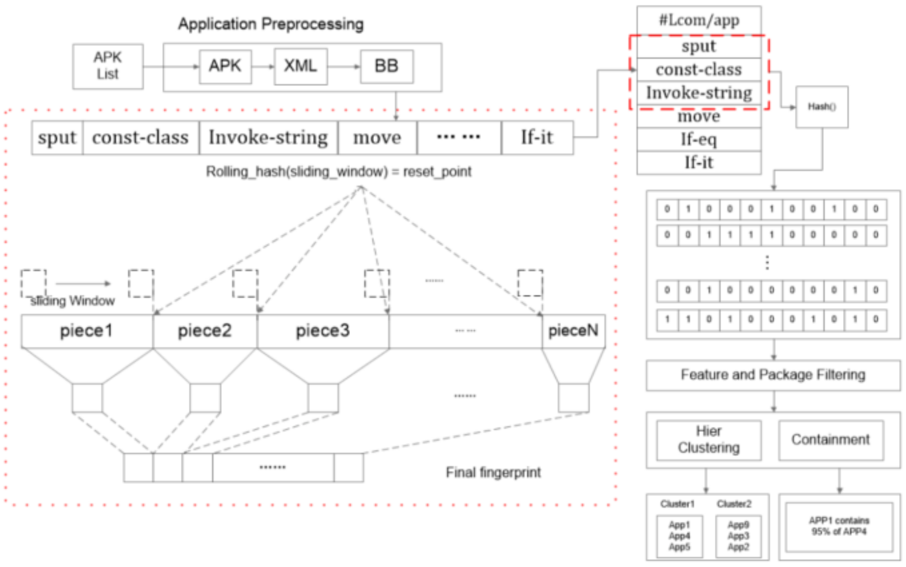


Figure I Basic implementation module of opcode-based Technique

Figure I is the module of opcode-based technique which combines the principle of DroidMOSS and Juxtapp. Both methods depend on opcode sequences as their main feature to detect repackaged apps. The red dotted line circle is the specific process of DroidMOSS and the left part is the technique principle of Juxtapp. Both of them extract the opcode as the fingerprints. We first use dexdump[2] to get the opcode sequence. Then, we use a sliding window to cut the whole opcode sequence into different parts and calculate a hash value for each piece. In this way, we get the final fingerprint for DroidMOSS. The edit distance algorithm is used to find the repackaged app pairs.

We implemented a prototype based on Juxtapp. The system first distills the classes.dex file from the APK file and converts it into XML format. We extract the opcode sequence for the XML file and use the k-grams to cut the opcode sequence into small pieces. The final fingerprint is a bit-vector which can be gotten by using the feature hashing(hashing algorithm is djb2). Agglomerative hierarchical clustering is used to detect repackaged apps.

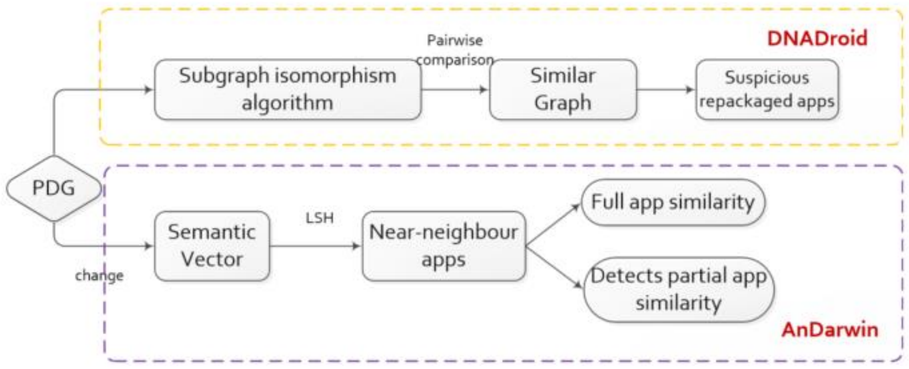


Figure II Basic implementation module of PDG-based Technique

We also use the same method to implement the PDG-based systems. The PDG-based techniques module can be seen from the Figure II. We first extract the PDG and then use

[2] https://github.com/plum-umd/dexdump

different approaches to get the potential repackaged app pairs. DNADroid directly employs the PDGs and VF2 algorithm to calculate the similarities among apps. AnDarwin changes the PDG into the sematic Vector and uses LSH to find repackaged apps.

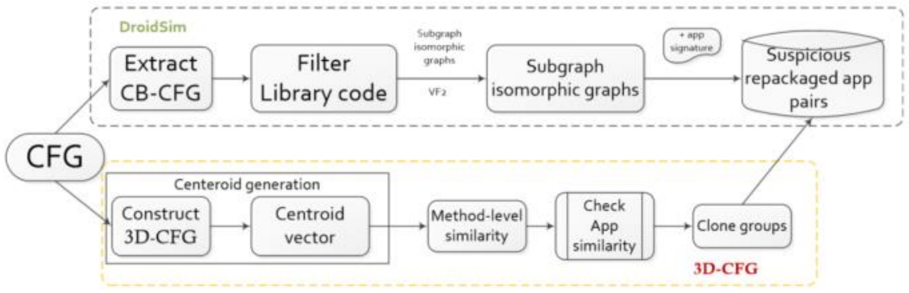


Figure III Basic implementation module of CFG-based Technique

The CFG-based techniques module can be seen from the Figure III. For CFG-based techniques, both of DroidSim and 3D-CFG use the Control Flow Graph as the fingerprint. We employ the static analysis to extract CFG and then implement the remaining parts of these systems.

DroidSim extracts the CFG and uses the VF2 algorithm to get repackaged apps. The 3D-CFG uses the *centroid algorithm* to change CFG into the vector and then checks these apps' similarity.

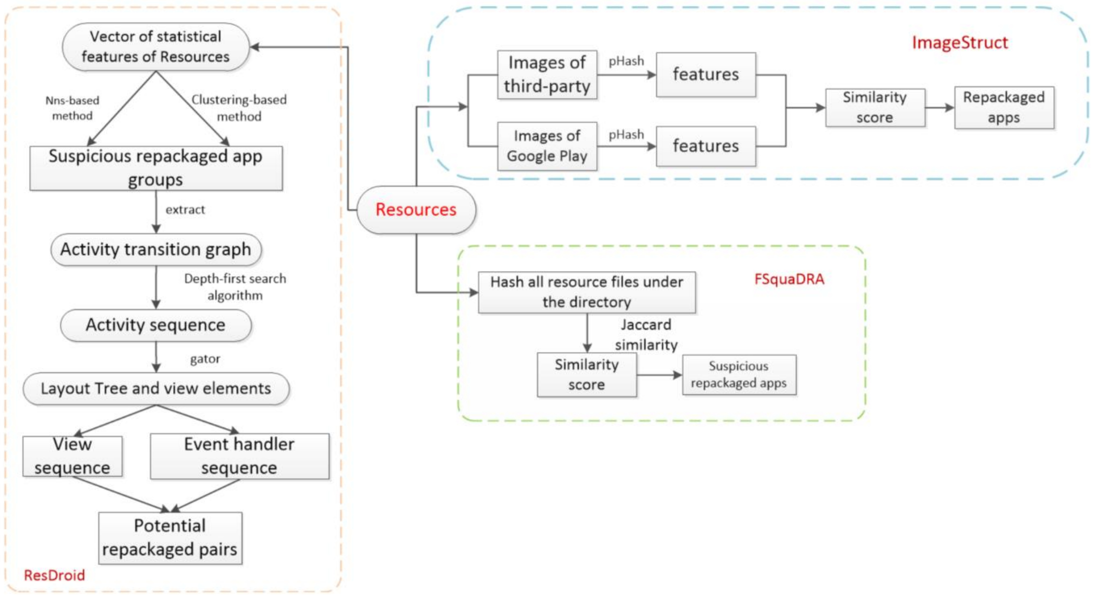


Figure IV Basic implementation module of Resources-based Technique

Figure IV describes the resource-based technique module. Their common part is to use resources to find repackaged app pairs. ImageStruct gets the screenshots of apps and handles them with the *PHash* algorithm and compares their similarity to find repackaged apps. ResDroid first utilizes statistical features of resources to get potential repackaged apps. FsquaDRA hashes all resource under the directory to generate their final fingerprints and calculates the similarity to get the suspicious repackaged apps.

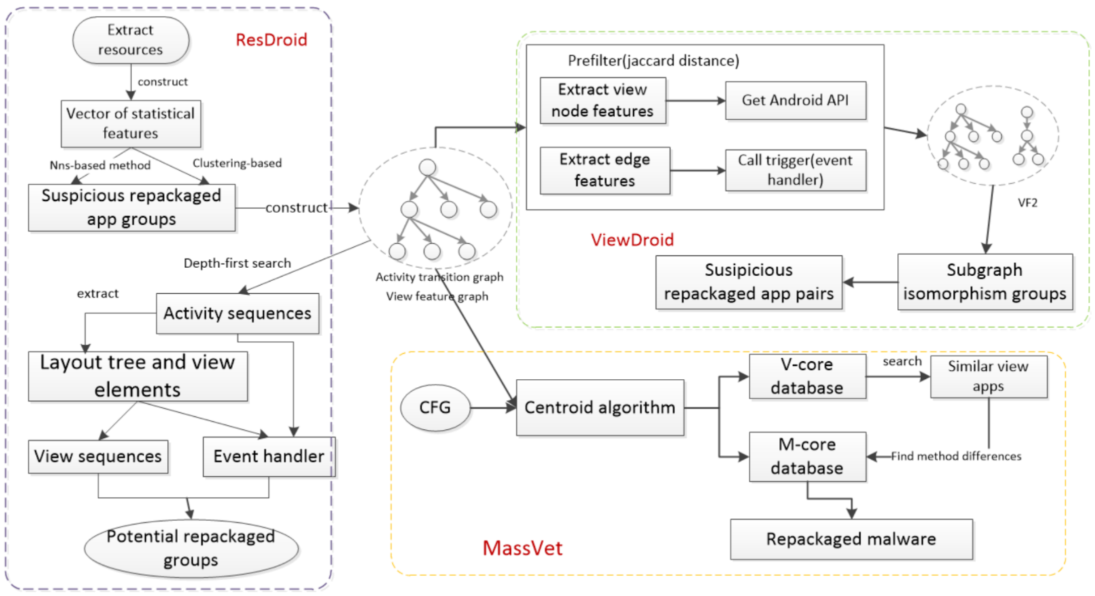


Figure V Basic implementation module of UI-based Technique

Figure V shows the UI-based technique module. We can find that the core part is Activity Transition Graph(ATG), and all systems have used this feature. ViewDroid gets the ATG and sets some rules to reduce the unnecessary comparison. After that, ViewDroid uses the VF2 to get isomorphism groups and find suspicious repackaged app pairs. MassVet changes the ATG and CFG by using centroid algorithm to create the view vector and method vector, respectively.

Each app can be indicated by the view vectors and method vectors. The view vectors are stored in the v-core database and method vectors are stored in an m-core database. MassVet uses view vector to find suspicious repackaged apps and then utilizes method vector to find the changed code. ResDroid traverses the ATG by using Depth-first search algorithm and gets the Activity sequence. After that, it extracts the view elements and event handlers to construct two UI-related sequences. ResDroid adopts LCS algorithm to find potential repackaged app pairs.

In sum, we summarize all these repackaged apps detection systems and then extract all the common parts. We first implement all the common components and different similarity measure algorithms. Through combining these different techniques together, we can construct different systems.