Intelligent Automation & Soft Computing
DOI:10.32604/iasc.2020.013849
Article

Tech Science Press

Feature Point Detection for Repacked Android Apps

M. A. Rahim Khan* **and Manoj Kumar Jain**

Department of Computer Science and Engineering, Lingaya's Vidyapeeth, Faridabad, 121002, India
*Corresponding Author: M. A. Rahim Khan. Email: khan_rahim@rediffmail.com
Received: 24 August 2020; Accepted: 16 October 2020

Abstract: Repacked mobile applications and obfuscation attacks constitute a significant threat to the Android technological ecosystem. A novel method using the Constant Key Point Selection and Limited Binary Pattern Feature (CKPS: LBP) extraction-based Hashing has been proposed to identify repacked Android applications in previous works. Although the approach was efficient in detecting the repacked Android apps, it was not suitable for detecting obfuscation attacks. Additionally, the time complexity needed improvement. This paper presents an optimization technique using Scalable Bivariant Feature Transformation extract optimum feature-points extraction, and the Harris method applied for optimized image hashing. The experiments produced better results than the CKPS: LBP method in terms of execution time. Further, the proposed method is extended to detect obfuscation of malware attacks by detecting the packed executables, which is the initial step in obfuscation attack detection.

Keywords: Repacked malware; phishing; key-point selection; hashing; Harris method; collision factor

1 Introduction

Malicious code writers typically do not want their code to be analyzed by any obfuscation method. The code writers are becoming familiar with the code alteration methods, such as packing and other encryptions, and other techniques that prevent their system from being re-engineered to conceal the malware. Most of the obfuscation methods are using a signature-based approach for the detection of malwares. Hence, the coders have learned that hash evaded using the packing technique for payload the malwares in the compressing layer [1]. The existing familiar methods have widely adopted many static and dynamic methods for recovering the payloads in a packed code, but they are not usually as effective as expected [2]. If the given malware is either packed or encrypted, it is a challenging task to analyze them. Hence, to prevent distracts and generate signatures that detect malwares, the packed code and other executables should be unpacked first.

As there is a significant race between malware writers and anti-malware vendors, the complicated code obfuscation is generally implemented into mobile malwares. The executables undergo techniques, such as polymorphism, packing, and other encryption techniques to prevent the anti-malware from detecting these malwares. Conventional methods are signature-based, need regular updates of the database, and malware detection is purely dependent on the existing known malware database and packed malwares [3]. Among

the available methods, executables' packing is very common, as many open source packers are available in the market. Over 90% of the mobile malwares are known to use packing methods. Moreover, there is clear evidence that one-half of the newly identified malwares are simply repacked variations of existing malwares. Even though executables' packing is very popular in writing malware, it is also found to be applied in the encryption of original executables. Genuine software developers use the packing method to make the resultant *.exe files smaller in size, in terms of a few bytes, enabling faster distribution over the network. The packing technique makes reverse engineering more complicated and, hence, forces hackers to crack the license. Given this fact, there are many packing tools available in the marketplace, which have been developed to protect genuine apps from pirated [4]. The unpacking process's essential step is to identify the *.exe files that are generally called "executables." These methods always use intelligent techniques or rely on databases. There are a couple of significant issues associated that need to be noted. First, mass malicious codes and genuine codes are too complicated to be collected to form a substantial database. Second, the classification takes a significant amount of time to become trained and classified, regardless of whether the app is genuinely packed or malware. As these classifications are based on the existing database, detecting new malwares is difficult [5]. The vital thing to be done here is first to identify the packed and un-packed apps separately.

Also, considering the drawbacks of the time complexity in the CKPS: LBP, and as a way of extending the method for detection of obfuscation attacks, this paper proposes an optimization technique using Scalable Invariant Feature Transformation for the extraction of optimum feature points extraction and the Harris method to optimize image hashing. The local feature points of the image are obtained in an optimal way using the SIFT method, and the usage of the Harris technique reduces the Hamming distances between the layout hashes. The rest of the manuscript is organized in the following manner. Section 2 briefly discusses some of the critical literature in the related domain, Section 3 elaborates on the proposed methodology, and Section 4 describes the implementation. Section 5 discusses the results in detail, and finally, the paper is concluded with the scope for future research.

2 Related Works

Many of the anti-obfuscation methods are classified into signature-based and Heuristic-based techniques. In [6], a detailed survey on the taxonomy of the present works proposed to secure the Android devices is intended. Based on the different stages of the deployment, the authors claim that these taxonomies help identify the common objective among the methods and identify many crucial differences. The essential limitations in the existing literature and the present challenges are also presented. Exeinfo [7] was introduced for the detection of packed executables, and the information on the header is extracted. It displays the entry point, offsets of the files, information about the compilers, and unpacks of input information. In [8], a procedure for the systemization of the anti-malware applications is proposed. A lack of openness in the existing methods was identified, and in order to resolve this issue, an effective method was proposed to detect the code obfuscation attacks. The proposed method was found to be effective for the anti-virus developers to test the resilience of products from a large volume of malwares and focus on the improvements to perform well against the code-transforming attacks. A multi-dimensional hybrid-features extraction [9] was developed to detect the Android malwares, which is based on extracting the numbers of API calls, behaviors of the running condition, and obtaining permissions. This approach combines three hybrid vectors by using machine learning classification algorithms. The hybrid-features extraction approach increased a more significant number of vector scales to detect the malwares but could not detect the obfuscation attacks.

In [10], PEiD tools are introduced where the signature-based packing is commonly used. It is designed for the detection of over 500 different kinds of signatures in packed executables. It is unique when compared

with that of the other signature-based methods. The rate of detection was found to be excellent among the other standard identifiers. In [11], a scheme is proposed that can quantitatively monitor the level at which the obfuscation is hidden in an Android application based on Machine Learning. The authors have used the statistical analysis of the DEX file that is obtained from the Dalvik platform. Another android malware obfuscator (AAMO) classification was used to classify the APIs. The experiments were conducted on the applications selected from Google Play. The results proved the proposed method to be efficient, and it served the purpose of evaluating the obfuscation level without depending on tools. In [12], researchers proposed an approach to protect Android applications from static reverse engineering using Multidex. APK has a DEX file, which uses encryption and adds Stub DEX to protect the apps from reverse engineering (repackaging techniques), and instead, keeps actual classes .dex files; a Dalvik executable file that all Android applications have. Whenever the user starts the application, stub DEX dynamically decrypts the application to verify the integrity of APK. In [13], Hump-and-Dump was discussed, where it follows a different kind of approach from the other literature because it uses pre-defined characteristics of the unpacking by counting the number of loops used in the process of unpacking. When the number of loops is higher than the threshold, and when no larger loops are being used for the threshold period, the loop address is caught and recorded.

In [14], RomaDroid was presented to effectively detect the cloned apps based on the features inherited from other applications. The authors proposed that the app's manifest could be represented as an XML file in a tree-based structure, which can withstand code obfuscation attacks. A string of all the manifest XML files is then compared, between the genuine and repacked apps, using the longest common subsequence (LCS) similarity method. The experiments were performed for both the obfuscated and their clones, using three tools. The results proved that RomaDriod could accurately detect cloned apps, even when code obfuscation has been applied. In [15], eight different methods were presented for detecting the repacked Android apps inside the images. The authors have presented a technique to store images that have malicious content in other apps' resources. After this process, the evaluation of the vulnerability of 10 of the famous Android malwares was done. The results seemed alarming as only one among them could detect the concealment methods in which the obfuscation attack happened. In [16], a framework was proposed for the efficient and dynamic analysis named "EnDroid," to bring out highly précised malware detection based on the multiple kinds of feature behaviors. The features cover the complete system-level traces and some common malicious trends, such as stealing information, subscribing to premium services, etc. The proposed EnDroid uses a novel algorithm for selecting the features for removing noise and other features that are not relevant. The EnDroid was efficient enough to differentiate between genuine and malicious applications. The proposed method also achieved more classification accuracy by using a stacking algorithm for Android malware identification. In [17], AlDroid was introduced for increasing the detection time and accuracy of the code obfuscation attacks. The proposed method used active learning to select only the new and informative apps and, thus, reduced labeling overhead and ensured the systematic and practical process of obfuscation attack detection. The experimental results indicate that the proposed active learning methodology outperforms other solutions provided in the literature, which were heuristic-based. The proposed method also produced high true-positive rates and low false-positive rates.

In [18], a new paradigm called the RIDG (reductive instructions dependent graphs) was proposed to be independent of the platform and stable with all obfuscation attacks. The authors have also introduced a four-tier schema based on RIDG for identifying the similarity between a repacked app and an original app. The experiments were carried out with 100 different apps, the anti-obfuscation attack was evaluated, and the proposed framework showed 98% accuracy.

3 Proposed Methodology

The proposed methodology is shown in Fig. 1 for detecting repackaged Android apps. Feature points are the local patterns of the image that differs from the intermediate neighbors as per some predefined characters, such as the corners, blobs, and salient points. The local features are then extracted, the feature points are identified through the wavelet similarity, and the hashes are generated through the adaptive methods that are based on the probability of the feature distribution. This method aims to solve some issues, the first, as the feature points detection are very critical in the image hashing for extracting optimized features, The Scale Bivariant Feature Transformation is proposed for the detection of optimized feature points, which could reduce the time complexity, and the introduction of Harris method, which ensures the selection of stable key feature points, which are less exposed to obfuscation attacks.

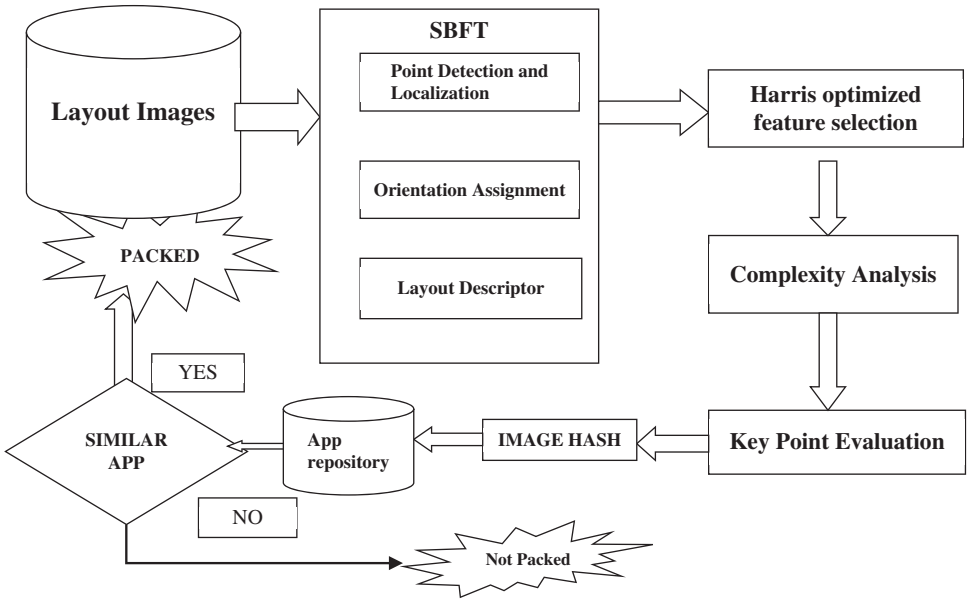

Figure 1: Proposed methodology

3.1 Optimized Local Feature Points

The local features include the parameters, such as the blob and the corners, widely used in detecting manipulations and detecting manipulations and retrieval. There are many advantages to using the local features, as the invariances are kept under the geometric transformations [19]. However, the optimization issues against classical attacks, specifically noising and blurring, are limited. The scalable invariant feature transform (SIFT) was relatively optimal among the different local feature detection techniques. Here, the scalable bivariant feature transformation (SBFT) was introduced to consider the tradeoffs between the optimization, distinctiveness, and efficiency. In the proposed methodology, SBFT is presented along with the Harris method for improving optimization against obfuscation attacks through robust image processing. The optimum features extracted are then used for the process of generating hashes.

3.2 Scalable Bi-Variant Feature Transformation

The proposed SBFT is primarily comprised of three levels, namely:

1. Scale-bivariant point detections,
2. Localization and orientation and,
3. Assignment and descriptors.

3.2.1 Scale Bivariant Point Identification and Localization

The local feature points are detected as candidates of the scale bivariant through the process of searching the extremes locally as a series of differences in the Gaussian DoGs of the given images in the given space of scaling. The construction of the DoG is performed as follows:

Image $I(x;\ y)$ is initially convolved as a series in the Gaussian kernel represented as $G(x,y)$ that are incremented continuously.

$$L(x,y,\eta)\ =\ G(\ x,y,\alpha) * i(x,y) \tag{1}$$

The two-dimensional Gaussian function is given as:

$$G(x,\ y,\ \alpha) = \frac{1}{2\pi\alpha^2} e^{-\frac{(x^2+y^2)}{2\alpha^2}} \tag{2}$$

Then, a DoG is produced considering the Gaussian values and the nearby scales of η:

$$D(x,y,\alpha) = L(x,y,k\alpha) - L(x,y,\alpha) \tag{3}$$

Given the series of DoGs, the local maxima and minima are then detected as a candidate of all the keypoints in comparison with every pixel to all its neighbors in different regions in the current and adjustable scales. The final location points are then localized to sub-pixel the accuracy by introducing a 3-D function, which is quadratic for selecting candidates to determine the interpolated positions of the maxima that reject some of the candidates that have low contrast. Further, the Hamming matrix is then computed at the given location and to scale every candidate key point. Those that have a more significant curvature are then rejected to eliminate the edge's response.

The must-have identifier attributes towards the detection of blob-like structures that are inside the given image, as they provide close proximity to the Laplacian scales of the Gaussian functions. Here, the $\nabla^2(x,y)$ is termed, and it is the Laplacian operator that is commonly used for the detection of edges and the corners of the image. In general, the 2-D Laplacian and Gaussian operator is defined by:

$$\nabla^2 G\ \ = -\frac{1}{\pi\alpha^4}\left[1 - \frac{x^2+y^2}{2\alpha^2}\right] e^{-\frac{(x^2+y^2)}{2\alpha^2}} \tag{3.1}$$

The difference in the Gaussian is the proximity of the concerned Laplacian in the Gaussian proximity of the Laplacian for the Gaussian function and are isotropic as the $\nabla^2 G$ is rotation invariant. In these, the DoG gives better optimization in the case of geometrical transformations.

3.2.2 Orientation Assignment

This step is crucial as the concerned descriptor shall be represented as a relative measure to its orientation and rise for rotational invariance. The orientation peak calculates it in the histograms constructed from the gradients in the keypoints that are detected along with the neighbors. The orientation consisted of 36 bins, which give 360 degrees of coverage in the orientation, and are weighted by the magnitude in the corresponding gradient with a Gaussian circular layout that is used for the reinforcement of weights of the gradients at the center of the neighborhood, and which improves the optimization against the noise.

3.2.3 Layout Descriptor

Given the position, the scale, and the orientation of every keypoint, the corresponding layout descriptor is generated inside the local region of the respective layout in the scaling space. The local neighbors of the keypoints are then divided into sub-regions and are relative to the orientation. Within each of the sub-regions, the magnitude of the gradient and the orientation are calculated. Inside the sub-regions, the magnitude of the

gradient and the orientation are calculated, and then the magnitudes are assigned weight through the Gaussian circular window for the orientation histogram in an octal direction. Hence, each of the keypoints has descriptors with 128 dimensions. The SBFT descriptor has been shown to provide the expected distinctiveness for matching the points and optimizing the image obfuscation attacks and other geometric transforms, which are commonly done in injecting malware into the original app. Although other methods for improving the SBFT descriptors are based on the DoG [20]. As we are dealing with the visual similarity of the layouts in an Android application for identifying a packed app from unpacked, the main objective is to ensure the optimization of keypoints in the image hashing. It is achieved by using the SBFT technique.

3.3 Optimized Keypoints Detection Using Harris Criteria

Further, to achieve a robust hashing, the optimized extraction of features is more important. Although DoG detection gives an excellent performance, some hindrances, such as noises and collision issues, affect the similarity measure between the genuine and packed app's layouts. This method also decreases the number of false negatives. To extract the optimal features, it is better to select the more stable keypoints in different distortions. It is found that the Harris method can provide more stability in the detection with high accuracy. Hence, it is proposed to incorporate the Harris condition for selecting the more stable SBFT keypoints for hashing. The Harris detection is based on the autocorrelation matrix representing the gradient's distribution locally on the selective keypoints. For a certain image $I(x, y)$, the auto-correlation matrices M in a given point (x, y) shall be represented as:

$$|\mathrm{M}| = \sum_{x}^{y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \tag{4}$$

where, $w(x,\ y)$ is a given window for the determination of accumulated region, and the gradients are represented as x, y axis. The Gaussian kernel is used for the weighted layouts to make the isotropic matrix. In general, if both the Eigenvalues are 1 and 2 of the matrix M, an alternative condition for evaluating the corner is:

$$H = \beta_1\ \beta_2\ - k(\eta_1 + \eta_2\ ) = \ Det(M) - k\ trace(M)^2 \tag{5}$$

Here, β denotes the coefficient with a value between 0.04 and 0.15, which is set as the threshold range. Given the set of SIFT keypoints $P\ =\ P(x;\ y; N)$, where x and y represent the coordinates and N denotes the number of scalable parameters, the Harris response $H_{\mathrm{i}}(x;\ y)$ is computed where 'I' denotes the standard deviations of the Gaussian windows that are used for the computation of correlation matrix M and the threshold is set for selecting the optimized SBFT point as:

$$T\ =\ \sum_{1}^{N} H_i\ (x,y) \tag{6}$$

where, H_i is a parameter control optimized selection of keypoints. Empirically, it takes the values between 0.1 and 0.5. In this proposed method, the value is taken as 0.5 as it defies the additive noises. The reason behind choosing such a threshold is that it helps keep track of the local points that are more scalable.

3.4 Detection Evaluation

To further illustrate the Harris method's effect in optimizing SBFT keypoints, an optimization function F is defined for evaluating the performance in SIFT and the proposed SBFT-Harris detection method. Of P

denotes the set of keypoints obtained through SIFT and Q is the keypoints obtained through the SBFT-Harris method, then function F of optimization is defined as:

$$F = \frac{|P \cap Q|}{|P \cup Q|} \tag{7}$$

Here, $|P|$ denotes the set cardinality, which is defined as the measure of unique elements of the set. When the F is about to reach 1.0, it means that the keypoints have been extracted from the layout image hashes of the genuine app and the packed app. F's value is used as the criteria for measuring the stability of the keypoints detected in different layouts.

4 Experimental Analysis

4.1 Complexity

The concept of complexity was proposed for complementing the entropy and for extracting more quantity of information. The complexity denoted by $C(X)$ is the length of the shortest layout, which represents X and is terminated. In any complexity problem, the function of complexity is defined as:

$$C(X) = Min\ (X) \tag{8}$$

However, this concept has a severe issue, which cannot be computed as the identification of the optimum algorithm, which makes the shortest distance from a given X string of input, is not feasible. The comparative algorithm can be used in the same way as the concept of complexity:

$$Compress(X) = (X') \tag{9}$$

Here, X' represents the compressed bit string of the image layout hash X. Different compression techniques are used for the reduced size in input and the best smallest output. Hence, the complexity can be measured as the input that uses compression for the classification. Almost all of the packed *.exe files are either compressed or encrypted, so to classify a packed *.exe, the point at which high complexity is attained must be detected. However, the length of the file before and after compressing will be lower in the case of a packed file versus an unpacked file. For the experimental purpose, the apps were taken from five different third-party Android markets, and the numbers of packed apps were detected. Tab. 1 shows the total number of apps taken from the market, and the corresponding obfuscation attacked shown in the Fig. 2.

Table 1: Results for the packed app detection

Android market	Total apps	Packed apps
Google play	9589	648
APK mirror	7564	429
APK pure	6587	312
Aptoide	4210	125
F-Droid	2500	78

4.2 Evaluation of Optimized Keypoints

Investigation of the benefits of having an optimized selection of keypoints against the obfuscation malwares for the specific purpose of identifying the content was made. Since it is highly unlikely to obtain the exact key-points sets detected in both genuine and packed app layouts, the Hamming distance

is calculated with Harris criteria, which is then used to calculate the similarity between the app's layout hashes. The image hashes of the five layouts in the Twitter app were considered for the evaluation, and the average Hamming distance using the Harris approach was calculated. In the proposed method, the vectors represent the scalable points' coordinates that have a high rank. It is also noted that the average Hamming distance between the keypoints, which are detected using the Harris method, which was those from the normalized Hamming method, were computed using the CPK method. The proposed SBFT–Harris method found to be more optimal, even when investigated under the packed Android apps. Although the proposed method is similar to SIFT, the advantage is that the coordinates can be used for the detection of scalable and optimized keypoints directly from the layout hashes of apps, and these kinds of keypoints provide more optimization in the key point selection, which is vital in the detection of packed apps. Tab. 2 presents the values of the Mean Hamming Distance (MHD) obtained for the five layouts using the two methods previously discussed.

As seen in Fig. 3, there is a significant variation in the hamming distances calculated using the CKP method and the proposed method with Harris Technique. The proposed method has less hamming distance, which assures more efficiency in the similarity calculation of original and packed apps with obfuscation attack programs.

Tab. 3 shows the time complexity (TC) and depicted in the Fig. 4 of the CKPS method for the layouts considered, after using scalable invariant feature extraction and optimized feature point selection using the Harris method. The results show the time complexity significantly reduced in the latter, which addresses the objective. It has been calculated that the time is taken to decide whether the app is a packed one was reduced to 0.81 s compared to 0.91 s in the case of CKPS.

From the resulting analysis, it is proven that the proposed scale-invariant points detection and localization, and the optimized keypoints detection, using the Harris criterion, resulted in time complexity reduction and also is helpful in the identification of obfuscation attacks through the detection of packed apps.

4.3 Evaluation of Accuracy

The proposed method was tested for its accuracy by obtaining the True Positive, True Negative, False Positive, and False Negative for the proposed method using Harrison optimization and SIFT (H: SIFT) with that of the previous work CKPS: LBP, along with other state-of-the-art methods discussed earlier. The results are tabulated in Tab. 4.

Fig. 5 shows a comparative graph of the parameters represented in Tab. 4. It can be seen that the proposed method offers the best values for True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN). Based on the values obtained in Tab. 4, the True Positive Rate and False Negative Rate were calculated and presented in Tab. 5. Fig. 6 shows a graphical representation of the TPR and FNR of the different methods. It can be noted that the proposed method has a 41% higher TPR and 38% lesser FNR than the other methods, which proves the efficiency of the proposed method.

Tab. 6 details the precision, recall, and F-measures obtained from the Tab. 4 parameters, wherein it is noted that the proposed method has a higher value for precision and recall than the other methods discussed, which increases the F-score value, thus proving the consistency. Fig. 7 presents a graphical depiction of the results.

The accuracy consistency was measured between the previous method CKPS: LBP and the proposed methodology. The number of layouts was obtained from 10–10000. The accuracy levels are recorded at each layout and are logged in Tab. 7.

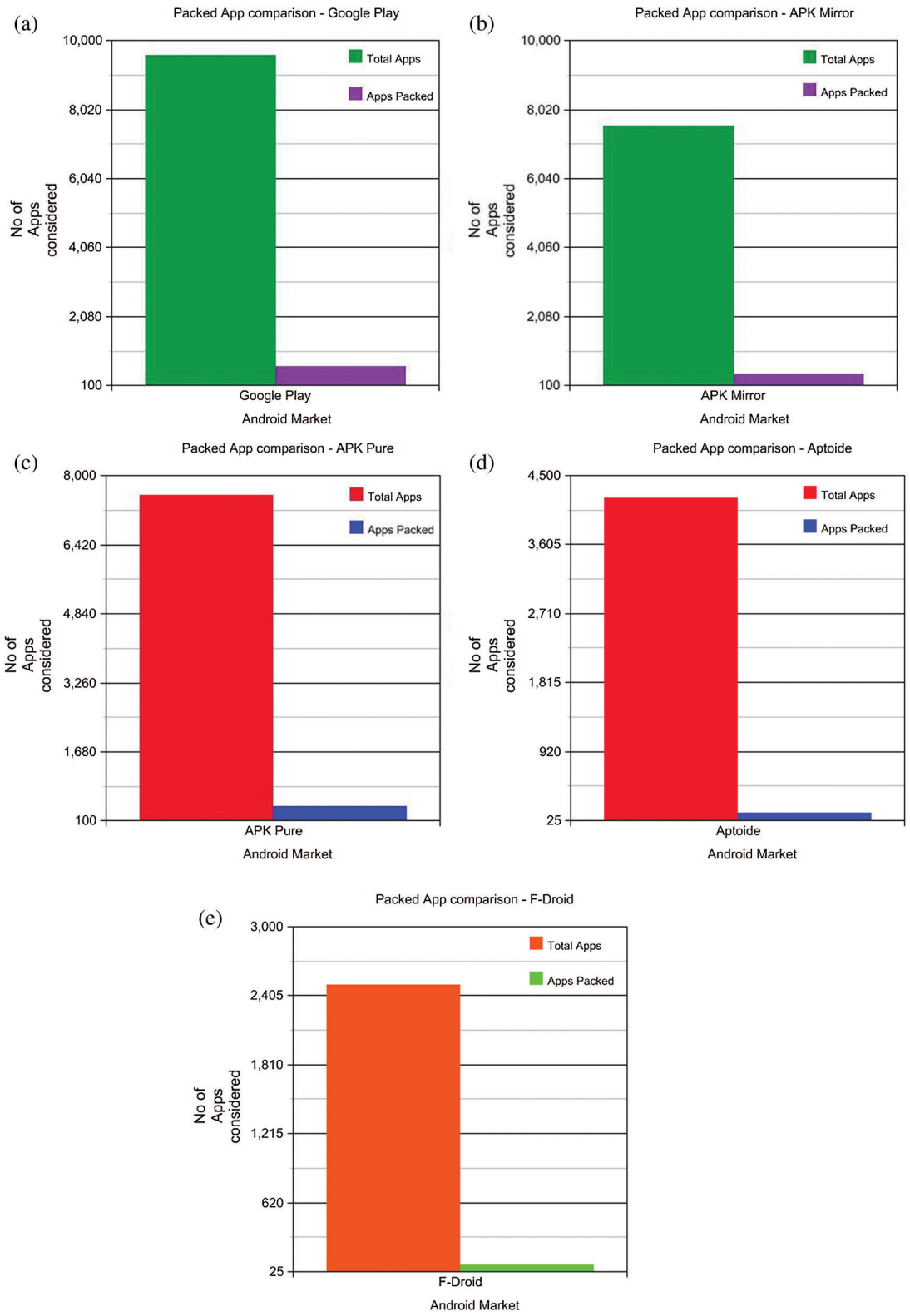

Figure 2: (a) Comparative study of the number of packed apps-google play. (b) Comparative study of the number of packed apps-APK mirror. (c) Comparative study of the number of packed apps-APK-pure. (d) Comparative study of the number of packed apps-Aptoid. (e) Comparative study of the number of packed apps F-Droid

Table 2: Comparison of mean hamming distance of the layout hashes

Layout	MHD-Harris method	MHD-CKP
Layout 1	0.268	0.387
Layout 2	0.202	0.324
Layout 3	0.358	0.589
Layout 4	0.412	0.810
Layout 5	0.514	0.754

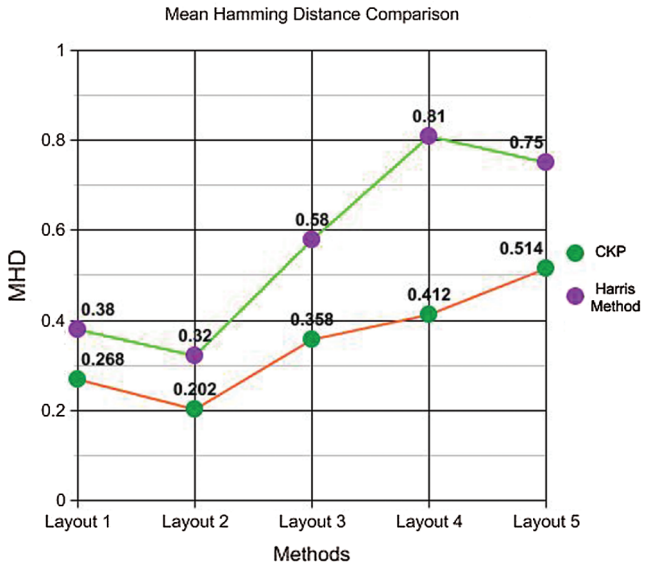

Figure 3: Mean hamming distance comparison

Table 3: Comparison of time complexity

Layouts	TC-CKP	TC-Harrison method
Layout 1	0.212	0.182
Layout 2	0.356	0.244
Layout 3	0.415	0.301
Layout 4	0.281	0.194
Layout 5	0.541	0.314

In Tab. 7, it can be seen that the accuracy of the proposed optimization, using the Harris method and SIFT, provides not only more accurate but also maintains the consistency irrespective of the number of layouts obtained. Fig. 8 details a steady decrease in the accuracy level with CKPS: LBP when the number of layouts is increased. However, when H: SIFT optimization is implemented, high and consistent accuracy is maintained, even when the numbers of the layout are increased, proving the accuracy and the consistency of the proposed technique.

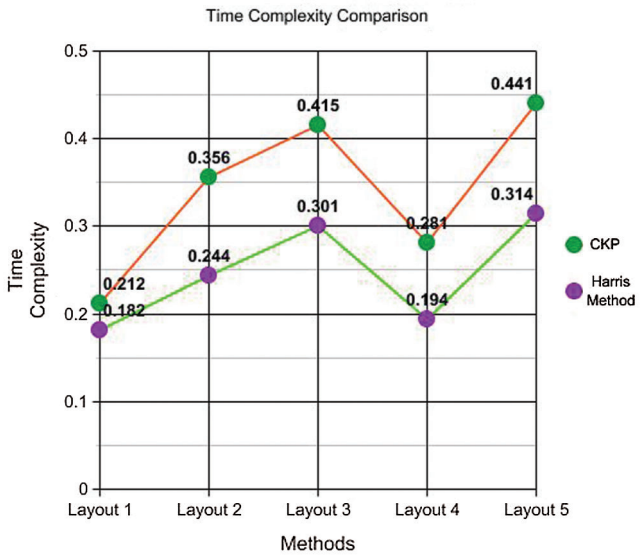

Figure 4: Plot of time complexity

Table 4: Comparison of TP, TN, FP, and FN

Method	TP	TN	FP	FN
OmniPack	0.84	0.38	0.24	0.81
PolyUnpack	0.81	0.31	0.19	0.86
EnDroid	0.90	0.24	0.49	0.84
RomaDroid	0.78	0.29	0.17	0.76
Aldroid	0.93	0.41	0.37	0.83
CKPS:LBP	0.95	0.21	0.24	0.96
H:SIFT	0.97	0.14	0.12	0.98

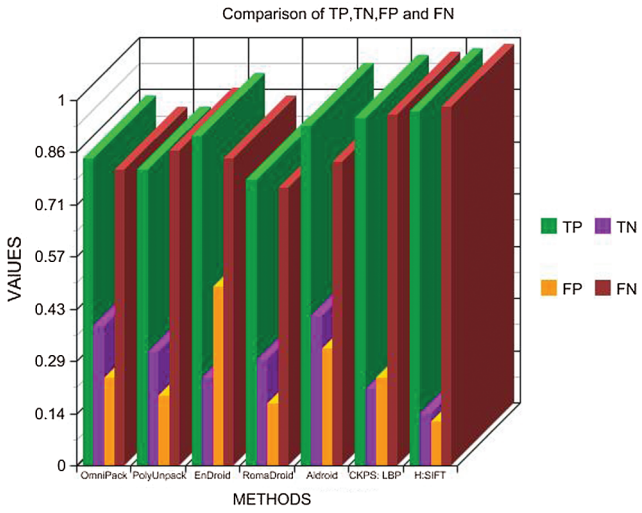

Figure 5: Comparison of TP, TN, FP, and FN

Table 5: Comparison of TPR and FNR

Method	TPR	FNR
OmniPack	0.24	0.62
PolyUnpack	0.39	0.51
EnDroid	0.26	0.30
RomaDroid	0.58	0.42
Aldroid	0.61	0.37
CKPS:LBP	0.92	0.21
H:SIFT	0.96	0.16

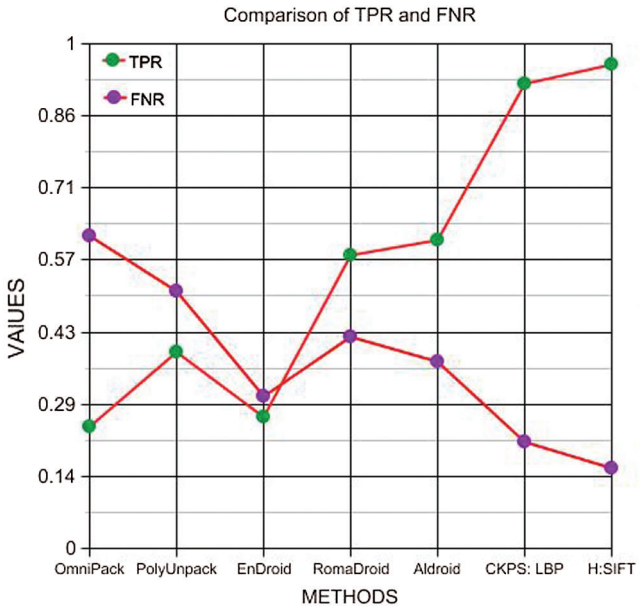

Figure 6: Comparison of TPR and FNR

Table 6: Precision, recall, and F-score comparison

Methods	Precision	Recall	F-Score
OmniPack	89.2	87.4	88.4
PolyUnpack	78.2	89.3	79.3
EnDroid	91.1	89.7	89.8
RomaDroid	89.5	84.5	87.4
Aldroid	81.5	79.5	80.4
CKPS:LBP	96.5	94.5	95.4
H:SIFT	98.4	97.3	98.4

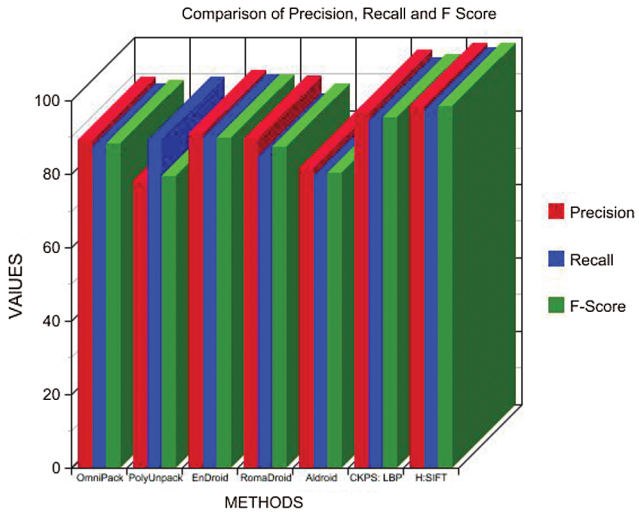

Figure 7: Comparison of precision, recall, and F-measure

Table 7: Accuracy comparison

Layout	Accuracy CKPS: LBP	Accuracy H: SIFT
10	96.2	98.4
20	96.1	98.4
30	96.1	98.4
100	95.9	98.4
200	95.1	98.2
300	94.8	98.2
1000	93.5	98.2
2000	93.1	98.1
3000	93.0	98.1
7000	92.8	98.0
10000	92.0	98.0

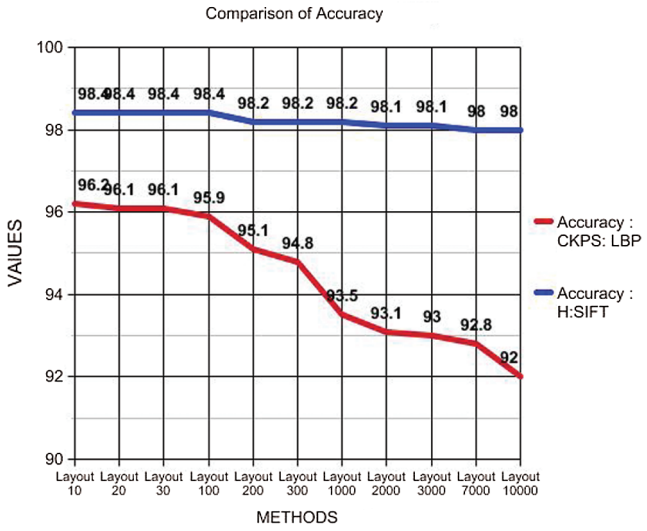

Figure 8: Accuracy plot

5 Conclusion

This paper presents an optimization technique using scalable invariant feature transformation to extract optimum feature-points extraction and apply the Harris method to optimize image hashing. The experiments produced better results than the CKPS: LBP in terms of time complexity. Additionally, the proposed method is extended to detect obfuscation of malware attacks by detecting packed executables, which is the initial step in obfuscation attack detection. The results were impressive, and, on average, 1592 packed apps out of a total of 35942 apps could be detected with which the layout hashes compared. The detection time was a minimum of 0.81 s. The proposed optimization method proves to be efficient in terms of high precision, recall, and F-score. Further, a high accuracy of 98% was maintained consistently, even when the number of layouts increased. As future research intends to extend the proposed methodology to other domains to study the feasibility.

Funding Statement: The authors received no specific funding for this study.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] J. Garcia, M. Hammad and S. Malek, "Lightweight, obfuscation-resilient detection and family identification of Android malware," *ACM Transactions on Software Engineering and Methodology*, vol. 26, no. 3, pp. 1–29, 2018.

[2] M. Chua and V. Balachandran, "Effectiveness of Android obfuscation on evading anti-malware," in *Proc. of the Eighth ACM Conf. on Data and Application Security and Privacy*, Tempe AZ USA, pp. 143–145, 2018.

[3] C. Yuan, S. Wei, C. Zhou, J. Guo and H. Xiang, "Scalable and obfuscation-resilient Android app repackaging detection based on behavior birthmark," in *2017 24th Asia-Pacific Software Engineering Conf.*, Nanjing, China, pp. 476–485, 2017.

[4] Y. Wang, H. Wu, H. Zhang and A. Rountev, "Orlis: Obfuscation-resilient library detection for Android," in *Proc. of the 5th Int. Conf. on Mobile Software Engineering and Systems-MOBILESoft 18*, Gothenburg, Sweden, pp. 13–23, 2018.

[5] X. Yang, L. Zhang, C. Ma, Z. Liu and P. Peng, "Android control flow obfuscation based on dynamic entry points modification," in *2019 22nd Int. Conf. on Control Systems and Computer Science*, Bucharest, Romania, pp. 296–303, 2019.

[6] D. J. Tan, T. W. Chua and V. L. Thing, "Securing Android: A survey, taxonomy, and challenges," *ACM Computing Surveys*, vol. 47, no. 4, pp. 1–45, 2015.

[7] https://www.oreilly.com/library/view/learning-malware-analysis/9781788392501/5d32bae2-f024-4493-ad8c-5b5db3e537c5.xhtml.

[8] T. Cho, H. Kim and J. H. Yi, "Security assessment of code obfuscation based on dynamic monitoring in Android things," *IEEE Access*, vol. 5, pp. 6361–6371, 2017.

[9] Y. Li, G. Xu, H. Xian, L. Rao and J. Shi, "Novel Android malware detection method based on multi-dimensional hybrid features extraction and analysis," *Intelligent Automation and Soft Computing*, vol. 25, pp. 637–647, 2019.

[10] D. Shin, C. Im, H. Jeong, S. Kim and D. Won, "The new signature generation method based on an unpacking algorithm and procedure for a packer detection," *International Journal of Advanced Science and Technology*, vol. 27, pp. 59–78, 2011.

[11] M. D. Preda and F. Maggi, "Testing Android malware detectors against code obfuscation: A systematization of knowledge and unified methodology," *Journal of Computer Virology and Hacking Techniques*, vol. 13, no. 3, pp. 209–232, 2017.

[12] K. Lim, N. Kim, Y. Jeong, S. J. Cho, S. Han *et al.,* "Protecting Android applications with multiple DEX files against static reverse engineering attacks," *Intelligent Automation and Soft Computing*, vol. 25, pp. 143–153, 2019.

[13] G. Jeong, E. Choo, J. Lee, M. Bat-Erdene and H. Lee, "Generic unpacking using entropy analysis," in *2010 5th Int. Conf. on Malicious and Unwanted Software*, Nancy, Lorraine, pp. 98–105, 2010.

[14] B. Kim, K. Lim, S. J. Cho and M. Park, "RomaDroid: A robust and efficient technique for detecting Android app clones using a tree structure and components of each app's manifest file," *IEEE Access*, vol. 7, pp. 72182–72196, 2019.

[15] S. Badhani and S. K. Muttoo, "Evading Android anti-malware by hiding malicious application inside images," *International Journal of System Assurance Engineering and Management*, vol. 9, no. 2, pp. 482–493, 2017.

[16] J. Zhao, X. Mo and Q. Zheng, "A novel method of Android malware detection based on ensemble learning algorithm," in *Proc. of 2018 the 8th Int. Workshop on Computer Science and Engineering*, Bangkok, pp. 531–538, 2018.

[17] N. Nissim, R. Moskovitch, O. Barad, L. Rokach and Y. Elovici, "ALDROID: Efficient update of Android anti-virus software using designated active learning methods," *Knowledge and Information Systems*, vol. 49, no. 3, pp. 795–833, 2016.

[18] X. Zhang, J. Pang and X. Liu, "Common program similarity metric method for anti-obfuscation," *IEEE Access*, vol. 6, pp. 47557–47565, 2018.

[19] W. Miao and X. W. Peng, "WLIB-SIFT: A distinctive local image feature descriptor," in *2019 IEEE 2nd Int. Conf. on Information Communication and Signal Processing*, Weihai, China, pp. 379–383, 2019.

[20] J. Zhao, H. Liu, Y. Feng, S. Yuan and W. Cai, "BE-SIFT: A more brief and efficient SIFT image matching algorithm for computer vision," in *2015 IEEE Int. Conf. on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing*, Liverpool, pp. 568–574, 2015.