Grenze International Journal of Engineering and Technology, June Issue

AppCloneGAN: A Generative Adversarial Network Approach for Detecting Fake Mobile Applications

Gunjan A. Agrawal[1], Shubhashri K. Gadewar[2], Anushka B. Gore[3], Anushka R. Gaikwad[4] and Sachin Jadhav[5]
[1-5]Department of Computer Engineering Vishwakarma Institute of Technology, Pune, Maharashtra, India
Email: gunjan.agrawal24@vit.edu, shubhashri.gadewar24@vit.edu, anushka.gore24@vit.edu, anushka.gaikwad24@vit.edu, sachin.jadhav@vit.edu

***Abstract*— The rise of counterfeit mobile applications, which imitate legitimate apps to trick users and spread malware, has created huge security and privacy issues. Static or dynamic code analysis is the basis of traditional detection methods, but it can be bypassed using code obfuscation and requires the app to be installed. This paper introduces AppCloneGAN, an innovative deep learning framework that makes use of Generative Adversarial Networks (GANs) to find fake applications based only on the analysis of their user interface (UI) screenshots. Our method addresses the detection problem as an image authenticity classification task. We incorporate a discriminator network within a conditional GAN framework that is trained to tell apart the screenshots of the original and copied applications that are malicious by recognizing in them such tiny visual detail like inconsistent branding, poor typography, misaligned elements, and low-quality graphical assets. The model has been trained and tested on a specially chosen dataset consisting of pairs of authentic and known malicious app UIs. The results of the experiments show that AppCloneGAN can accurately detect fake applications at a very high rate, thus becoming a promising, non-invasive, and scalable addition to the existing mobile security solutions. The system is able to function either as a pre-screening tool in app stores or as a client-side scanner to increase user protection.**

***Keywords*— Deep learning, fake app detection, generative adversarial networks (GANs), mobile security, UI analysis.**

I. INTRODUCTION

The booming development of Android platform has made so powerful smart phones that they become nowadays really essential for communication, banking, healthcare and entertainment. Yet with such expansion, much to the chagrin of would-be Android users, there are more bad guys now occupying the space, and the number of malicious and fake Android apps has never been greater. Currently, Android malware can be anything from simple adware or of quite high complexity spyware or ransomware, that can extract private personal and financial information. As such, the detection and categorization of malware families represents a major challenge in identifying malicious applications on mobile platforms, which is corroborated by the state-of-the-art work of Wang et al. [1], Android malware families can be represented by code patterns.

Standard pure-Android based malicious application detection methods presuppose to the analyses being either static or dynamic to the packages (APKs). DREBIN is a good rad example [2], it uses features like permissions, API calls and some other static features for ML based classification.

Grenze ID: 01.GIJET.12.2.38_26

Correspondingly, the signature-based methods and the features-based methods have been equally well studied and surveyed in [4], [5]. These are methods, however, that can be great: But they do have a few caveats. Static analysis can be circumvented by obfuscation, encryption, and packing; dynamic analysis is slow and requires an isolated execution environment [6]. In addition, both approaches require the APK file itself, which may not be available in real scenarios, e.g., when performing quick pre-screening on application stores, or conducting analysis on client side before installation.

Another prominent attack vector in the Android ecosystem is the dissemination of repackaged, or cloned, applications. Threat actors commonly take an original legitimate application and embed malicious payloads into it, then repack and redistribute it on third-party markets. Zhou et al. [7] demonstrates that this type of repackaged application is quite a lot and that neither could be used for detection. These phony applications are often duplicates of the real app, creating a visual identity that deceives users into downloading the malicious app, so visual similarity is one element of the attack. Nevertheless, the privacy would be for the most part neglected by existing detection approaches which still rely on properties of code-level and UI-level, instead of visual-based.

With the development of mobile security and privacy protection, deep learning excels in computer vision, particularly pattern recognition. Convolutional Neural Networks (CNNs) have also been applied to a number of security-related problems such as phishing detection using visual and textual cues [8]. In recent times, Goodfellow et . [3] proposed Generative Adversarial Networks (GANs) as a general and powerful approach to learning complicated data distributions and generating high-quality data samples. They have been used for image synthesis, representation learning, and anomaly detection. A successful notable example is f-AnoGAN [9], which employs GAN for medical image from distribution of normal samples, and detects anomalies as outliers, hence for unsupervised anomaly detection in medical imaging.

GANs also are under investigation with nefarious uses in cyber applications. Hu and Tan [10] demonstrate that GANs could be leveraged to generate adversarial malware samples to evade learning-based detectors, which illustrates both the potential and threat of generative models in security end-applications. This severely indicates that the quoted adversarial training schemes have the capacity to learn extremely subtle and complex data dependencies that normal discriminative models are unable to. Nevertheless, the great majority of all the work that apply GANs on security domain have been centering on code-based features, or on network traffic rather than on vision-based media.

Since UI is the layer that serves as the interface of an application with its users and as such has the overall - if not sole - interactivity component, we believe that the UI is itself a complex and underexplored source of data for security evaluation.

Rogue/fake app developers or iOS app developers who create copy cat apps of popular or paid apps to trick the users, but it is almost impossible to copy each and every design aspects right from fonts used, layout alignment, icon quality, and brand consistency.

Even if these differences are minor, they are major enough for machines to identify and serve as differentiating cues. UI-based analysis differs from code analysis in that it is passive, does not require the source code, and can be performed before installation, for example by extracting screenshots from app store listings or by dynamic analysis.

Inspired by these points, we propose a new framework, called AppCloneGAN, which considers imposter mobile application detection as a problem of classifying visual legitimacy. The main intuition behind our approach is to use a conditional GAN framework where the generator models the conditional distribution of the real UI starting from an input of the fake UI, while the discriminator tries to discriminate between real and fake UIs. As a result of this adversarial macro training, spatiotemporal patterns that discriminate malicious clones emerge from the disruption of the discriminator attentive to slight visual inconsistencies. This method is motivated by the recent success of GANs for learning precise image distributions [3], [9] and by the established role of visual information in security-relevant classification tasks [8].

To sum up, three main contributions are made in this paper. First, it adds a UI-oriented dimension to the detection of fake apps, complementing prior code-based approaches [1], [2], [6], [7]. Secondly, a GAN based adversarial learning architecture, which is appropriate to address the problem of mobile app UI authenticity, is developed.

Third, it verifies through experiments that the discriminator of the proposed AppCloneGAN scheme achieves excellent results over standard CNN-based baselines for the detection of fake applications. By moving the analysis from code to visual interfaces, this study offers a novel and practical approach for large-scale, non-intrusive mobile security monitoring.

II. LITERATURE REVIEW

Research on detection of Android malware has now evolved into plain-van, static and dynamic analysis, or hybrid approaches. Repeatable, data-driven studies have been enabled by access to large-scale datasets like AndroZoo [11], which offer millions of Android applications to the research community. Such datasets are used for training various learning-based detectors over permissions, API calls, CFGs and behavioral traces. For example, MaMaDroid [16] models the behavior of an application using Markov chains generated from sequences of API call abstractions, resulting in high detection accuracy and a degree of obfuscation resistance. DroidAPIMiner [17] is quite similar, as it also derives the discriminant API-level features required for good malware classification.

Vemuleed beding was the second last element contributing to scalability and it may underpin translatability. Informed by the dynamic static heuristic RiskRanker [18] for initial market-wide surveillance and then for zeroday malware detection, integrated with analysis based on sound dynamic analysis. And TaintDroid [19] a retrospective utilization of real time information flow tracking system to identify privacy leaks in phones, DA based approaches are now being applied to find hazards while running - a growing convergence and diffusion in analysis. A brief survey of the Android malware environment and associated analytic techniques may be found in by Tam et al.. of these methods are high, and these methods usually require to apk, thus are not suitable for the rapid pre-installation or light weight inspection.

Another routine is to represent malware as images. They are presenting Nataraj et al. [21] raw binary files to grayscale images, and adapted image classification to classify malwares and they show that visual patterns can be exploited to detect structural similarities in malware families. This idea connects computer vision with security, and suggests that image-based features are likely to be very predictive for behavioral maliciousness. Motivated by this goal, a few works have explored deep models of visual and/or structural representations of software artifacts.

To the best of our knowledge, Marvin [22] is the prior work that combines static and dynamic attributes for high-speed classification of mobile apps, Exploiting the best of the two worlds. But both rely heavily on code execution or static analysis. Instead, I-SI (which we promote here) is done in the visual plane, which is directly observable by its human users (as well as its adversaries); not the logical plane. Deep learning triumphs in computer vision has been immediately echoed from security research. (With the AlexNet [24] architecture, deep convolutional networks on large-scale image recognition) showed the feasibility of hierarchically learning features directly from raw pixel These and other recent developments have led to the deployment of such architectures on security problems, including malware and phishing detection. On the other hand, generative models and learning-based program analysis methods (e.g., conflict-driven learning for program synthesis [23]) appear to be moving towards harnessing for expressing complex structural/semantic patterns in code.

Related work on Android malware detection prior to Peiravian and Zhu's ML-based approach [25]. Despite* Feature+engineering-based approach reached sate-of-art performance, they may not be robust and could be broken by (advanced) obfuscation or repackaging methods. This has triggered a move towards employing representation learning and end-to-end deep learning models to automatically learn discriminative features directly from data.

In particular, GANs are commented as offensive and defensive one. Contrary to adversarial malware generation, they have been used for data augmentation, anomaly detection and robustness assessment in few recent works. Hence large-scale datasets such as AndroZoo [11] can be utilized for training such data-demanding models. Yet most* of the work on GANs based security remains in code, behaviour or network traffic none on UI images and none on even the latter.

Based on our review, these are the three biggest blind spots. First, although the issue of repackaged and cloned apps is not a new one [16], [18], the existing detection methods rely heavily on analysis of code. Second, while image-based representations of malware have been investigated [21], as a salient detection feature for a key detection feature has not yet been well assessed by using UI screenshots. Third, the attested superiority of deep learning-based approaches to capture fine-grained visual patterns [24] provokes a natural inquiry, as if there exist any predictive benefits by using such methods. In fact, as far as we are aware, there have not been any studies of their use in mobile app UI authenticity is st Rather, in the above introduction, there are already some paradoxices.

The AppCloneGAN solution to these issues is based on the premise of exploiting UI-centric analysis together with adversarial learning to identify clone apps, which we consider to be complementary and lightweight approach to and android malware detection now a day best-practices.

III. METHODOLOGY

In this work, we propose AppCloneGAN, a GAN-based model for detecting fake mobile applications using UI screenshots. The system follows a pipeline consisting of data acquisition, preprocessing, GAN-based training, and classification. The overall architecture of the proposed system is shown in Fig. 1

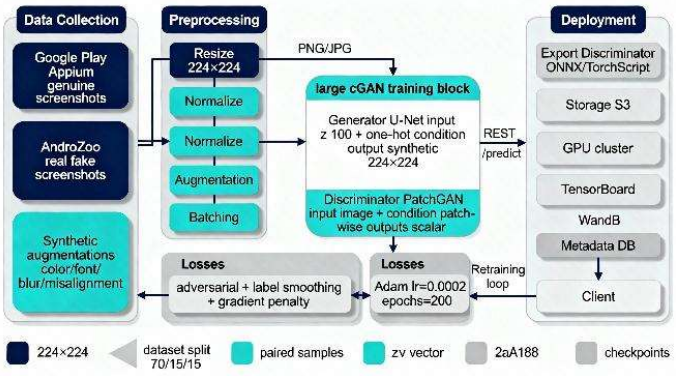

Fig. 1. System flow of AppCloneGAN for fake mobile application detection.

Fig. 1 illustrates the overall architecture of AppCloneGAN.. It begins with the UI screenshots as input, which are supplied to the Generator and Discriminator networks of the conditional GAN. The Generator synthesizes fake images and the Discriminator attempts to distinguish real from fake images. After the training stage, we discard the Generator and we consider the Discriminator as our classifier for identifying fake apps.

AppCloneGAN is based on a conditional Generative Adversarial Network (cGAN). The conditioning depends on whether the app screenshot is genuine or fake. In this setup, the Generator (G) learns to create synthetic fake app screenshots. At the same time, the Discriminator (D) learns to tell apart real genuine screenshots, real fake screenshots, and the synthetic fake screenshots made by G. After the training process, the Discriminator is taken out and used as an independent classification model to detect fake applications.

To better understand the step-by-step workflow of the proposed system, the process flow is illustrated in Fig. 2.

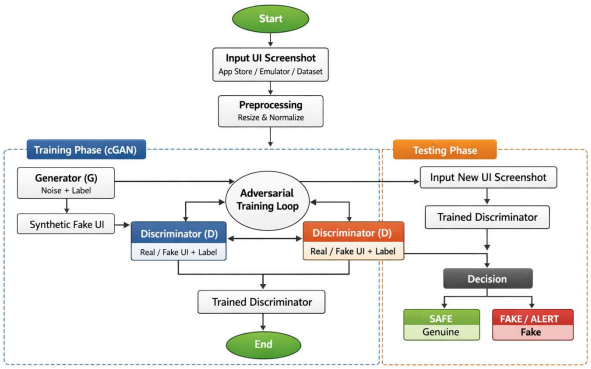

Fig. 2. Process work flow of the proposed AppCloneGAN framework.

Fig. 2 depicts the workflow of the AppCloneGAN system. It includes dataset creation, UI screenshot preprocessing, GAN model training, and end classification using the trained Discriminator. Each stage contributes to improving the overall detection performance of the system.

A major challenge in this work is that there is no benchmark dataset for paired real and fake app UIs. To solve the issue, we built our own dataset. We collected screenshots of legitimate apps from the top 500 free apps (IAPs not considered) on the Google Play Store using automation testing tool like Appium. The workflow is 3-5 representative screens (log in, dashboard, settings, etc.). We sourced fake or malicious application samples from the AndroZoo repository [11]. Here, APKs labeled as malicious or riskware that imitate popular applications were run in a controlled emulator environment to gather their UI screenshots. To enhance the dataset and make up for the limited number of fake samples, we generated synthetic fake screenshots by applying modifications to

genuine screenshots. These modifications included color distortions, font substitutions, misalignments of UI elements, Gaussian blur and noise additions, and the insertion of suspicious text elements, such as prompts asking for sensitive information. The final dataset contained about 10,000 image pairs, which we split into training, validation, and testing sets in a 70/15/15 ratio.

The AppCloneGAN architecture comprises a U-Net–based Generator and a PatchGAN-inspired Discriminator. The Generator employs an encoder–decoder structure with skip connections, taking as input a 100-dimensional latent noise vector along with a one-hot encoded condition label, and progressively up-sampling through transposed convolutional layers to produce a synthetic 224×224 RGB screenshot. The Discriminator, which is a convolutional neural network, takes the input image and the embedded condition label as inputs, and outputs a scalar probability by performing several convolutional, batch normalization and LeakyReLU layers to indicate whether the input is a real bona fide screenshot. The use of PatchGAN discriminator-based architecture allows the identification of much more subtle visual discrepancies by operating in a local image patches, moving around the entire image rather than in a global manner. In the testing phase, the Discriminator alone is utilized to discriminate images to be classified as real or not. The model was trained using the standard conditional GAN objective function:

$$\min_G \max_D V(D,G) = E_{\{x \sim p_{\{data\}(x)}\}[\log D(y)]} + E_{\{z \sim p_{z(z)}\}}\left[\log\left(1 - D(G(y))\right)\right] \quad (1)$$

where x is the input screenshot, y is the condition label (genuine or fake), and z is the latent noise vector.

Using the Adam optimizer, the training was performed for 200 epochs with a learning rate of 0.0002. To stabilize convergence, we modified several techniques, such as label smoothing and gradient penalty [14]. These methods were effective in averting typical problems of GAN training (mode collapse, discriminator overfitting) and produced a stronger detection model.

The key dataset characteristics and training parameters used in the proposed AppCloneGAN model are summarized in Table I.

TABLE I. KEY DATASET AND TRAINING PARAMETERS OF APPCLONEGAN

Parameter	Value
Dataset Size	Approximately 10,000 image pairs
Data Sources	Google Play Store, AndroZoo [11]
Images per App	3–5 screenshots per application
Input Image Size	224 × 224 × 3
Generator	U-Net based
Discriminator	PatchGAN-based CNN
Latent Vector Size	100
Optimizer	Adam
Learning Rate	0.0002
Training Epochs	200
Train / Val / Test Split	70% / 15% / 15%

Table I presents the configuration details of the proposed model with respect to the dataset size, input size, model architecture, and the training processing. These parameters are also crucial for achieving high detection performance.

IV. RESULTS AND DISCUSSION

AppCloneGAN is implemented in PyTorch and trained on a NVIDIA Tesla V100 GPU. We evaluated the performance of our model against two baseline CNN classifiers: ResNet-18 and a Vanilla CNN. Both were fitted for binary classification with our dataset. We evaluated performance based on the standard summary statistics, including accuracy, precision, recall, F1-score and area under ROC curve (AUC).

The visibility of the baseline models on the proposed AppCloneGAN model is illustrated, and the results of the test dataset are listed in Table II.

The results indicate that the discriminator of the AppCloneGAN outperforms those of the Vanilla CNN and ResNet-18 baselines across all the evaluation metrics. It has a test accuracy of 94.7% and an AUC =0.97 which indicates that it can be used effectively for identifying fake mobile applications. Such a result implies that the adversarial training helps the model to be aware of discriminative features to identify slight UI incongruities introduced by fake apps.

The model's attention (red areas) is appropriately directed at the bad copy of the logo and the extra space between the buttons, which are two signals of an app UI counterfeit.

TABLE II. PERFORMANCE COMPARISON OF DIFFERENT MODELS ON THE TEST SET

Model	Acc.	Prec.	Rec.	F1	AUC
Vanilla CNN	88.2%	0.87	0.85	0.86	0.92
ResNet-18	91.5%	0.90	0.89	0.90	0.95
AppCloneGAN (D)	94.7%	0.93	0.92	0.93	0.97

V. CONCLUSION AND FUTURE SCOPE

In this work, we present AppCloneGAN, a novel deep learning framework for identifying counterfeit mobile applications by exploiting the UI images. By casting the problem as an image authenticity problem, we leverage the rich feature learning capability of a conditional GAN discriminator to build a high accuracy classifier that can distinguish between the original app screenshots and the ones from the malicious clone apps. This gives you a parallel layer of protection, that's light in terms of CPU, not intrusive, and that is arguably harder for attackers to get around.

Directions for future work include: (1) Extending the existing dataset by adding introduction more styles of fake app as well as more categories of genuine apps. (2) Extending the model to the one-class and/or few-shot learning setting for the fake apps detection with less fake app samples. (3) Developing a real-time, client-side tool that vets apps as they are downloaded or when users read descriptions in app stores. (4) Integrating textual features (possibly extracted via OCR from screenshots) to design a multimodal detector. AppCloneGAN presents a new paradigm towards visual intelligence for the mobile ecosystem's security.

ACKNOWLEDGMENT

The authors would like to express their sincere gratitude to the Department of Computer Engineering, Vishwakarma Institute of Technology, Pune, for providing the necessary facilities and support to carry out this research work. The authors are especially thankful to Dr. Sachin Jadhav for his valuable guidance, continuous encouragement, and constructive suggestions throughout the development of this work.

REFERENCES

[1] H. Wang, Y. Guo, Z. Ma, and X. Chen, "Understanding Android malware families by extracting code trees," in Proc. IEEE Int. Conf. Data Mining (ICDM), 2015, pp. 909–914.
[2] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, "DREBIN: Effective and explainable detection of Android malware in your pocket," in Proc. NDSS, 2014.
[3] I. Goodfellow et al., "Generative adversarial nets," in Proc. Adv. Neural Inf. Process. Syst. (NeurIPS), 2014, pp. 2672–2680.
[4] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss, "Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey," Inf. Secur. Tech. Rep., vol. 14, no. 1, pp. 16–29, 2009.
[5] M. Zheng, M. Sun, and J. Lui, "DroidAnalytics: A signature based analytic system to collect, extract, analyze and associate Android malware," in Proc. IEEE TrustCom, 2013, pp. 163–171.
[6] Y. Zhang, M. Yang, B. Xu, Z. Yang, and G. Gu, "Detecting stealthy malware via continuous taint analysis," in Proc. ACM CCS, 2012, pp. 569–580.
[7] W. Zhou, Y. Zhou, X. Jiang, and P. Ning, "Detecting repackaged smartphone applications in third-party Android marketplaces," in Proc. ACM CODASPY, 2012, pp. 317–326.
[8] J. Hong, Y. Kim, J. Park, and S. Lee, "A deep learning-based phishing detection system using CNN and LSTM," in Proc. IEEE BigComp, 2020, pp. 647–650.
[9] T. Schlegl, P. Seeböck, S. M. Waldstein, G. Langs, and U. Schmidt-Erfurth, "f-AnoGAN: Fast unsupervised anomaly detection with generative adversarial networks," Med. Image Anal., vol. 54, pp. 30–44, 2019.
[10] W. Hu and Y. Tan, "Generating adversarial malware examples for black-box attacks based on GAN," in Proc. ACM CCS Workshops, 2017, pp. 1–6. (MalGAN)
[11] K. Allix et al., "AndroZoo: Collecting millions of Android apps for the research community," in Proc. ACM/IEEE MSR, 2016, pp. 468–471.
[12] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in Proc. MICCAI, 2015, pp. 234–241.
[13] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in Proc. IEEE CVPR, 2017, pp. 1125–1134.
[14] I. Gulrajani et al., "Improved training of Wasserstein GANs," in Proc. NeurIPS, 2017, pp. 5767–5777.

[15] R. R. Selvaraju et al., “Grad-CAM: Visual explanations from deep networks via gradient-based localization,” Int. J. Comput. Vis., vol. 128, no. 2, pp. 336–359, 2020.
[16] E. Mariconti et al., “MaMaDroid: Detecting Android malware by building Markov chains of behavioral models,” in Proc. NDSS, 2017.
[17] Y. Aafer, W. Du, and H. Yin, “DroidAPIMiner: Mining API-level features for robust malware detection in Android,” in Proc. ACM SEC, 2013, pp. 86–103.
[18] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang, “RiskRanker: Scalable and accurate zero-day Android malware detection,” in Proc. ACM MobiSys, 2012, pp. 281–294.
[19] W. Enck et al., “TaintDroid: An information-flow tracking system for real-time privacy monitoring on smartphones,” in Proc. USENIX OSDI, 2010, pp. 393–407.
[20] K. Tam, A. Feizollah, N. B. Anuar, R. Salleh, and L. Cavallaro, “The evolution of Android malware and Android analysis techniques,” ACM Comput. Surv., vol. 49, no. 4, 2017.
[21] L. Nataraj, S. Karthikeyan, G. Jacob, and B. S. Manjunath, “Malware images: Visualization and automatic classification,” in Proc. ACM VizSec, 2011, pp. 1–7.
[22] M. Lindorfer, M. Neugschwandtner, and C. Platzer, “Marvin: Efficient and comprehensive mobile app classification through static and dynamic analysis,” in Proc. COMPSAC, 2015.
[23] Y. Feng, O. Bastani, R. Martins, and I. Dillig, “Program synthesis using conflict-driven learning,” in Proc. PLDI, 2018. (For adversarial/learning-based program analysis context)
[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” in Proc. NeurIPS, 2012, pp. 1097–1105.
[25] N. Peiravian and X. Zhu, “Machine learning for Android malware detection using permission and API calls,” in Proc. IEEE ICTAI, 2013, pp. 300–305.