T.C.

KIRIKKALE ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

DOKTORA TEZİ

ANDROID KÖTÜCÜL YAZILIMLARINDAN KORUMA SİSTEMLERİNİN DEĞERLENDİRİLMESİ VE GÖRÜNTÜ İŞLEME ALGORİTMALARINI YAPAY ZEKA TEKNİKLERİ İLE MELEZLEŞTİREREK YENİ BİR ALGILAMA YAKLAŞIMININ GELİŞTİRİLMESİ

HALİT BAKIR

ŞUBAT 2021

Bilgisayar Mühendisliği Anabilim Dalında Halit BAKIR tarafından hazırlanan "Android Kötücül Yazılımlarından Koruma Sistemlerinin Değerlendirilmesi Ve Görüntü İşleme Algoritmalarını Yapay Zekâ Teknikleri İle Melezleştirerek Yeni Bir Algılama Yaklaşımının Geliştirilmesi" adlı Doktora Tezinin Anabilim Dalı standartlarına uygun olduğunu onaylarım.

Doç.Dr. Atilla ERGÜZEN
Anabilim Dalı Başkanı

Bu tezi okuduğumu ve tezin Doktora Tezi olarak bütün gereklilikleri yerine getirdiğini onaylarım.

Doç.Dr. Halil Murat ÜNVER
Danışman

Jüri Üyeleri

Başkan	:	Prof. Dr. Necaattin BARIŞÇI	_____
Üye (Danışman)	:	Doç. Dr. Halil Murat ÜNVER	_____
Üye	:	Doç. Dr. Atilla ERGÜZEN	_____
Üye	:	Doç. Dr. Bülent Gürsel EMİROĞLU	_____
Üye	:	Dr. Öğr. Üyesi. Zafer CİVELEK	_____

……/…../…….

Bu tez ile Kırıkkale Üniversitesi Fen Bilimleri Enstitüsü Yönetim Kurulu Doktora derecesini onaylamıştır.

Prof.Dr. Recep ÇALIN
Fen Bilimleri Enstitüsü Müdürü


Aileme

ÖZET

ANDROID KÖTÜCÜL YAZILIMLARINDAN KORUMA SİSTEMLERİNİN DEĞERLENDİRİLMESİ VE GÖRÜNTÜ İŞLEME ALGORİTMALARINI YAPAY ZEKâ TEKNİKLERİ İLE MELEZLEŞTİREREK YENİ BİR ALGILAMA YAKLAŞIMININ GELİŞTİRİLMESİ

BAKIR, Halit[1]
Kırıkkale Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı, Doktora tezi
Danışman: Doç. Dr. Halil Murat ÜNVER
Şubat 2021, 191 sayfa

Cep telefonlarının yaygınlaşması, onları günlük hayatımızın vazgeçilmez bir parçası haline getirmiştir. Bu cihazlar, çoğu insan tarafından, bankacılık uygulamalarına erişmek veya internet alışverişi yapmak gibi önemli hizmetlerden faydalanmak için kullanılmaktadır. Android tabanlı cihazlar, kullanılan akıllı telefonlar arasında her zaman ilk sırada yer almış ve son kullanıcılar tarafından en çok tercih edilen aygıtlar olmuşlardır. Android tabanlı akıllı telefonların yaygın kullanılmaları, onları kötü amaçlı yazılım geliştiriciler için önemli bir hedef haline getirmiştir. Bu nedenle, günlük yayınlanan çoğu kötü amaçlı yazılımın üstesinden gelmek için literatürde, çok sayıda model önerilip geliştirilmiştir. Bu çalışmada, birden fazla bölümden oluşan kapsamlı bir uygulama gerçekleştirilmiştir.

Çalışmanın ilk bölümünde, iki yüzden fazla eseri kapsayan sistematik bir inceleme yapılarak Android kötü amaçlı yazılım tespit alanı için tüm araştırma trendlerini kapsayan yeni bir taksonomi ve Android kötücül yazılım algılama alanında kullanılabilen yöntemleri anlatan yeni bir grafiksel inceleme yöntemi önerilmiştir.

[1] Khaled BAKOUR: Yazarın çifte vatandaşlığından dolayı adı iki farklı şekilde yazılabilir.

Çalışmanın ikinci bölümü, bazı iyi bilinen ticari anti virüs sistemlerinin, kamuflaj tekniklerine karşı savunma sağlamlığını test etmeyi amaçlamaktadır. Bu amaçla, iyi bilinen kötü amaçlı yazılım veri kümelerine dayalı olarak birden çok kamufle edilmiş kötü amaçlı yazılım veri kümesi oluşturmak için saldırılar önerilip uygulanmıştır. İlk olarak, iyi huylu izin, ardından iyi huylu izinler-kod olmak üzere iki enjeksiyon saldırısı önerilmiştir. Ayrıca, yüksek derecede kamuflaj elde etmek ve anti virüs sistemler tarafından tespit edilmesinden kaçınmak için önerilen bu saldırılar, bazı gizleme teknikleriyle melezleştirilmiştir. Gerçekleştirilen literatür taramasında görüldüğü kadarıyla, bu tip enjeksiyon saldırıları ve bu saldırılarla gizleme tekniklerinin önerilen şekilde melezleştirilmesi Android işletim sistem alanında ilk kez bu çalışmada kullanılmıştır.

Çalışmanın üçüncü bölümünde, kötü amaçlı uygulamaları tespit etmek ve Android uygulamaları sınıflandırmak için yeni bir genel algılama yöntemi önerilmiştir. Bu bölüm kapsamında, görüntü işleme tekniklerine, makine öğrenme algoritmalarına ve derin öğrenme tekniklerine dayalı birden fazla model önerilip geliştirilmiştir. Önerilen yöntem, Android uygulamalarının içeriğindeki bazı dosyaları gri tonlamalı görüntülere dönüştürmeye, oluşturulan görüntülerden görüntü tabanlı özellikleri ayıklamaya ve makine öğrenme modellerini eğitmek için ayıklanan özellikleri kullanmaya dayanmaktadır. Bu amaç ile, önerilen modelleri eğitmek için ÖBÖD (SIFT), HSÖ (SURF), KAZE ve YDB (ORB) olmak üzere dört tane görüntü tabanlı lokal; renk histogramı, Hu anları ve Haralick doku olmak üzere üç tane görüntü tabanlı global özellik çıkarılmıştır. Görsel Sözcük Çantası (Bag Of Visual Words), ayıklanan lokal özelliklerin tanımlayıcılarından bir özellik vektörü oluşturmak için kullanılmıştır. VisDroid adlı yeni bir model önerilmiş, geliştirilmiş, ayıklanan görüntü tabanlı özellikler kullanılarak eğitilmiş ve Android uygulamaları iyi huylu veya kötücül olarak sınıflandırılmıştır. Ardından bu model kötücül olarak algılanan yazılımları türlerine göre sınıflandırmak için kullanılmıştır. Ayrıca, DeepVisDroid adlı görüntü özelliklerine ve derin öğrenme tekniklerine dayalı 1 boyutlu evrişimli katman tabanlı yeni bir model önerilip ayıklanan görüntü tabanlı özellikler kullanılarak eğitilmiştir. Önerilen DeepVisDroid modeli, Android uygulamaları iyi huylu veya kötücül olarak sınıflandırmak için kullanılmıştır. Geliştirilen VisDroid ve

DeepVisDroid modellerinde kullanılan yöntemler Android kötücül yazılımları tespit etmek için yine ilk kez bu çalışmada kullanılmıştır.

Daha sonra, önerilen modellerin sonuçları, klasik derin öğrenme modelleriyle karşılaştırmak için iki boyutlu klasik evrişimli sinir ağ modeli önerilmiştir. Son olarak, önerilen modeller, iyi bilinen derin öğrenme modeller ile karşılaştırmak için ResNet ve Inception V3 olmak üzere iki iyi bilinen derin öğrenme modeli, oluşturulan görüntü veri kümeleri kullanılarak test edilmiştir. Önerilen yöntemin, tipik bir hesaplama süresi ile %98'den daha fazla doğruluğa ulaşan bir sınıflandırma elde ettiği gözlemlenmiştir. Bu yöntemin, yaptığı sınıflandırma doğruluğu ve hesaplama süresi açısından bazı son teknoloji modellerle karşılaştırılmış ve kıyaslanan modellerin çoğundan daha iyi performans gösterdiği karşılaştırılmalı olarak sunulmuştur.

Anahtar kelimeleri: Android Kötücül Yazılım Tespit Sistemleri, Enjeksiyon Saldırıları, Gizleme Teknikleri, Görüntü Tabanlı Özellikler, Görsel Kelime Çantası, Görselleştirme Tabanlı Kötücül Yazılım Sınıflandırılması.

ABSTRACT

EVALUATING THE ROBUSTNESS OF ANDROID ANTI-MALWARE SYSTEMS AND DEVELOPING A NOVEL DETECTION APPROACH BASED ON HYBRIDIZING IMAGE PROCESSING ALGORITHMS WITH ARTIFICIAL INTELLIGENCE TECHNIQUES

BAKIR, Halit[2]

Kırıkkale University

Graduate School of Natural and Applied Sciences

Department of Computer Engineering, Ph. D. Thesis

Supervisor: Assoc. Prof. Dr. Halil Murat ÜNVER

February, 191 pages

The proliferation of mobile phones makes them an indispensable part of our daily lives. Most of the people are using these devices to access important services such as accessing banking applications or making e-shopping. Android-based devices have always ranked first among the smartphones used and favoured by the largest number of the end-users. The widespread use of Android-based smartphones made them an important target for malicious applications' developers. So, a large number of frameworks have been proposed to tackle the huge number of daily published malware. In this work, a comprehensive study composed of multiple sections has been conducted.

In the first section of this work, a systematic review of more than two hundred works has been conducted. To this end, a new taxonomy for the Android malware detection

[2] Khaled BAKOUR: The author's name can be written in two different ways due to his dual citizenship.

research trend has been proposed. Also, a novel graphical review method has been suggested.

The second section of the work aims at testing the defence robustness of some well-known commercial anti-malware systems against camouflage techniques. To this end, multiple attacks have been proposed and applied to create multiple camouflaged malware datasets based on well-known malware datasets. First of all, we proposed two injection attacks namely, benign permissions injection attack and benign permissions-code injection attack. Furthermore, the proposed attacks have been hybridized with some obfuscation techniques to obtain a high degree of camouflage and avoiding anti-malware systems' detection. To the best of our knowledge, this type of injection attacks and the proposed hybridization of these attacks with obfuscation techniques have been used for the first time in the Android operating system domain.

In the third section of the work, a novel generic detection method has been proposed for detecting malicious applications and classifying android applications. Multiple models have been proposed and developed in the scope of this section based on image processing techniques, machine learning algorithms and deep learning techniques. The proposed method is based on converting some files from the Android applications' contents into grayscale images, extracting image-based features from constructed images and using the extracted features for training machine learning models. To this end, four image-based local features namely, SIFT, SURF, KAZE and ORB, and three image-based global features namely, colour histogram, Hu moments and Haralick texture have been extracted to train the proposed models. The bag of visual word representation (BOVW) has been used for constructing one feature vector from the extracted local features' descriptors. A novel model called VisDroid has been proposed, developed, trained using the extracted image-based features and used for classifying the Android application into benign or malicious and classifying the malicious applications into their families. Furthermore, an image features and deep learning techniques-based novel 1D-convolutional layer-based model called DeepVisDroid has been proposed, trained using the extracted image-based features and used for classifying the Android

application into benign or malicious. Moreover, two classical convolutional neural network models have been proposed in order to compare the results of the proposed models with classical deep learning models. Also, two well-known deep learning models namely, ResNet and Inception V3 have been tested using the constructed image datasets and compared with the proposed models. The proposed method obtained a very high classification accuracy reached more than 98% with a typical computational time. The results of the proposed models have been compared with some state-of-the-art models in term of classification accuracy and computational time. The results of the proposed models outperformed the most of compared state-of-art models in term of both classification accuracy and computational time.

Keywords: Android Anti-malware Systems, Injection Attacks, Obfuscation Techniques, Visualization-based Malware Classification, Image-based Features, Bag of Visual Words.

TEŞEKKÜR

İlk olarak, bu tez ve araştırmayı tamamlamak için bana güç ve cesaret veren yüce Allah'ıma binlerce kez şükrediyorum.

Bu doktora tez çalışmamı hayatımdaki her aşamada bana destek veren ve bu günlere gelmemde üzerimde çok emeği olan ve maddi manevi desteğini üzerimden hiç eksik etmeyen rahmetli sevgili babam Ahmad BAKOUR'un aziz hatırasına ithaf ederim.

Tezimin hazırlanması esnasında hiçbir yardımı esirgemeyen Sayın Doç. Dr. Halil Murat ÜNVER'e, tez çalışmalarım esnasında, bilimsel konularda yardımlarını aldığım Sayın Prof. Dr. Necaattin BARIŞÇI'ya ve Sayın Doç. Dr. Bülent Gürsel EMİROĞLU'na teşekkür ederim.

Bu günlere gelmemde üzerimde emeği olan bütün öğretmenlerime ve hocalarıma teşekkür ederim.

Maddi manevi desteklerini üzerimden hiç eksik etmeyen her konuda beni destekleyen Sevgili annem Jamileh MARJAN'a, kıymetli kardeşlerim Nayif ve Husam BAKOUR'a ve tüm diğer kardeşlerime sonsuz teşekkür ederim.

Tez sürecince anlayış ve sabır gösteren ve çalışmalarımı aksatmamam için sayısız fedakârlıkta bulunan ve her aşamada destek verip beni güçlendiren eşim Razan GHANEM'e ve oğlum Ahmet'e teşekkürlerimi sunarım.

Son olarak, tezin Türkçesi konusunda büyük fedakarlıklarla bana yardımcı olan mesai arkadaşlarım Kırgaz A.Ş. Bilgi İşlem Sorumlusu Fatih ELİDOLU ve Yüksek. Endüstri Mühendisi Sevde AKAY'a teşekkür ederim.

Halit BAKIR (Khaled BAKOUR)

İÇİNDEKİLER DİZİNİ

ŞEKİLLER DİZİNİ

ŞEKİL **Sayfa**

ÇİZELGELER DİZİNİ

Çizelge **Sayfa**

ALGORİTMALAR DİZİNİ

Algoritma **Sayfa**

KISALTMALAR DİZİNİ

ÖBÖD (SIFT)	Ölçekten Bağımsız Öznitelik Dönüşümü (Scale-Invariant Feature Transform)
HSÖ (SURF)	Hızlandırılmış Sağlam Öznitelikler (Speeded Up Robust Features)
YDB (ORB)	Yönlendirilmiş FAST ve Döndürülmüş BRIEF (Oriented FAST and Rotated BRIEF)
HSTÖ (FAST)	Hızlandırılmış Segment Testinden Öznitelikler (Features from Accelerated Segment Test)
İSBTÖ (BRIEF)	İkili Sağlam Bağımsız Temel Öznitelikler (Binary Robust Independent Elementary Features)
GF (DoG)	Gauss Farkı (Difference of Gaussian)
GSÇ (BOVW)	Görsel Sözcük Çantası (Bag of Visual Words)
APK	Android Paketi (Android Package)
DEX	Dalvik Yürütülebilir (Dalvik EXecutable)
ESA (CNN)	Evrişimsel Sinir Ağı (Convolutional Neural Network)

SİMGELER DİZİNİ

H	Renk Histogram vektörü.
$I(x, y)$	Görüntüdeki (x, y) konumundaki pikselin yoğunluğu.
μ_{ij}	Görüntüdeki merkez moment.
$G(x, y, \sigma)$	σ Ölçeği için (x, y) konumundaki LoG.
$\mathcal{H}(x, y)$	Hessian matrisi.
$L_{xx}(x, \sigma)$	x noktasında görüntünün Gaussian ikinci dereceden türevi.

1. GİRİŞ

Akıllı telefonlar, günlük yaşamımızda birçok önemli işlemi gerçekleştirmeye imkân sağlayan en önemli cihazlardan biri haline gelmiştir. Bu nedenle, akıllı telefon teknolojisinin büyümesine ve hızlı gelişimine ayak uydurmak için hem resmi hem de üçüncü taraf uygulama mağazalarında birçok uygulama geliştirilmiş ve sunulmuştur. Sonuç olarak, akıllı telefonların talebi zaman içinde önemli ölçüde artmıştır. Gartner'ın dünya çapında akıllı telefon satışlarıyla ilgili 2017 raporuna göre, küresel akıllı telefon satışı, 2017 yılının ikinci çeyreğinde 2016 yılının aynı dönemine göre %6.7 artışla 366.2 milyon adete ulaşmıştır [1]. Ayrıca Gartner'ın 2018'de yayınlanan raporuna göre, akıllı telefonların küresel satışları 2018'in ilk çeyreğinde büyüyerek 2017'nin aynı dönemine göre yüzde 1,3 artışla 455 milyon adete ulaşmış olup akıllı telefon satışlarının yüzde 86,1'ini Android tabanlı akıllı telefonların oluşturduğu belirtilmiştir [2]. Ayrıca, Android işletim sisteminin 2017 yılında toplam pazarın %86'sını yakalayarak 2016'ya göre %1,1 artışla liderliğini genişlettiğini belirtilmiştir [3]. Android işletim sistemi popülaritesinin arkasındaki en önemli nedenler şu şekilde özetlenebilmektedir; 1) Android işletim sistemi, Linux işletim sisteminin çekirdeğine dayanmaktadır, bu da mevcut donanımın en geniş miktarına uyabileceği ve üreticilerin işini kolaylaştıracağı anlamına gelmektedir. 2) Android, tamamen açık kaynaklı bir sistemdir; bu da herkesin kaynak kodunu inceleyebileceği, değiştirebileceği ve geliştirebileceği anlamına gelmektedir. 3) Google, Android uygulama geliştiricilerinin çalışmasını kolaylaştırmak için birçok araç sağlamaktadır. Buna ek olarak, Android işletim sistemi, uygulamalarını resmi ve üçüncü taraf uygulama pazarlarından indirmeye esneklik sağlamaktadır. Android resmi mağazası (Google Play) ilk olarak Ekim 2008'de Android Market adı altında başlatılmış olup Statista web sitesinin istatistik analizlerine göre Google Play uygulama mağazasındaki mevcut uygulama sayısı Mart 2018'de 3.3 milyona ulaşmıştır [4]. Ayrıca, 2017 yılının dördüncü çeyreğinin üçüncü çeyreğine göre uygulama sayısında % 8,84'lük bir artışı temsil ettiği belirtilmiştir [5]. Ayrıca, Ağustos 2010 ile Mayıs 2016 arasında Google Play uygulama mağazasından indirilen uygulama sayısının 65 milyar uygulamaya ulaştığı iddia edilmiştir [6]. Öte yandan, Android'in

yaygınlaşması ve açık kaynak yapısından dolayı, kötü amaçlı yazılım geliştiricileri için önemli bir hedef haline gelmiştir. 2015'teki Pulse Secure mobil tehdit raporuna göre, Android işletim sistemini hedefleyen yaklaşık bir milyon benzersiz kötü amaçlı uygulama 2014'te 2013'e göre %391 artışla tespit edilmiştir [7]. Ayrıca, aynı raporda, Android işletim sistemini hedefleyen kötü amaçlı yazılım örnek sayısı, akıllı telefon işletim sistemlerini hedefleyen kötü amaçlı yazılım örnek sayısının %97'sine ulaştığı ve Android işletim sisteminin kötü amaçlı yazılım sayısı açısından ilk akıllı telefon işletim sistemi olarak sıralandığı belirtilmiştir. Ayrıca, Symantec'in 2016 internet güvenliği tehdit raporuna göre, 2015 yılında yayınlanan Android kötü amaçlı yazılım ailesinin sayısı 2014'teki %20'lik büyümeye kıyasla %6 artışa ulaşmıştır [8]. Ayrıca, Android kötü amaçlı yazılım geliştiricilerinin statik analiz ve dinamik analiz çerçevelerini atlamak için gizleme ve şaşırtma teknikleri gibi bazı yöntemler kullanmaya başladığı belirtilmiştir. Ayrıca, G-Data CyberDefense şirketi, Android'i hedefleyen 750.000 kötü amaçlı yazılımı 2017 yılın ilk çeyreğinde keşfetmiş olup, toplam üç milyondan fazla yeni kötü amaçlı yazılım örneğini aynı yılda tespit ettiğini iddia etmiştir [9]. Ayrıca G-Data güvenlik uzmanları, 2018'in ikinci çeyreğinde, her 7 saniyede bir, yeni çıkan kötü amaçlı yazılım türü olduğunu ve Android işletim sistemini hedefleyen siber suçların oldukça arttığını belirtmişlerdir [10]. Ayrıca G-Data İnternet güvenlik merkezi, yayınlanan mobil kötücül yazılım örnek sayısının yaklaşık yüzde 40 arttığını açıklamış ve onların analistleri de 2018'in üçüncü çeyreğinin sonunda Android tabanlı mobil cihazları hedefleyen yaklaşık 3,2 milyon kötücül yazılım örneği tespit ettiklerini belirtmişlerdir. Bu her gün 12.000 yeni Android kötücül uygulaması yayınlandığı anlamına gelmektedir [11]. McAfee mobil araştırma ekibi, Google Play'deki 144 uygulamanın içinde barınan, Grabos isimli yeni bir kötü amaçlı etkinlik tespit etmişlerdir. [12]. Grabos başlangıçta "Aristotle Music Audio Player 2017" adlı ücretsiz bir ses çalar olarak tanıtılan bir Android uygulamasında bulunmuştur. McAfee mobil araştırma merkezi, Eylül 2017'de Google'ı Grabos hakkında bilgilendirmiş, Google ise, bildirilen uygulamayı derhal kaldırdığını doğrulamıştır. McAfee, daha sonra Google Play'de Grabos ailesinden 143 adet daha zararlı uygulama bulmuştur. Ayrıca, 2018'in ilk çeyreğindeki McAfee mobil tehdit raporuna göre [13], Google Play mağazasında bulunan kötü amaçlı yazılım ailelerinin sayısı, 2017'de %30 artmıştır, bu da resmi Android uygulama mağazasının bile kullanıcılar için riskli olduğu anlamına gelmektedir. Ayrıca,

McAfee Labs'ın mobil araştırma ekibi, 2018 yılında Google Play'de yayınlanan en az 15 uygulamadan oluşan yeni bir faturalandırma dolandırıcılık kampanyası keşfettiklerini belirtmişlerdir [14]. Ayrıca, SophosLabs'a [15] göre Ağustos 2017'de, casus yazılımları, bankacılık botları ve agresif reklam yazılımları dahil olmak üzere en az beş farklı tür kötü amaçlı uygulama, Google Play'de bulunmuş olup, binlerce uygulamanın bu kötü amaçlı yükleri içerip milyonlarca kullanıcıyı etkilediğini belirtmişlerdir. Trend Micro, Google Play pazarında yayınlanan ve 85 sahte uygulama olarak gizlenen bir reklam yazılım ailesi tespit ettiğini ve dünya çapında dokuz milyon kez indirildiği belirtmişlerdir [16]. Netice olarak, Android cihazları hedef alan kötücül uygulamaları tespit etmek için sağlam savunma sistemleri geliştirmeye acil ihtiyaç olduğu görülmektedir.

1.1. Cep Telefonlarının En Önemli Güvenlik Tehditleri

1.1.1. Fidye Yazılımı (Ransomware)

Para kazanmak amacıyla kullanıcının kendi cihaz kaynaklarına erişmesi engelleyen her türlü kötü amaçlı yazılımı ifade etmektedir. Genellikle bu saldırı, dosyaları şifreleyerek veya depolama ortamını kilitleyerek ve ele geçirilen kaynakları serbest bırakmak için para talep ederek gerçekleştirilmektedir. Fidye yazılımı, Windows platformuna karşı çok yaygın bir tehdit olarak bilinmekteydi, ancak son zamanlarda, akıllı telefonlarda depolanan değerli veriler nedeniyle, bu tür cihazları hedeflemeye başlamıştır. Bu nedenle, Android fidye yazılımı kötü amaçlı yazılım geliştiricilerinin ilgisini çekmiştir. Android.Lockdroid.E, 2014 yılında Symantec tarafından keşfedilen Android fidye yazılımı örneğidir [17]. Android.Lockdroid.E, akıllı telefondaki ekranı kilitledikten sonra bir fidye talebi görüntülemektedir. Android/DoubleLocker.A, 2017'de ESET araştırmacıları tarafından keşfedilen başka bir android fidye yazılımı örneğidir [18]. Double Locker.A, kurbanın kaynaklarına erişimi önlemek için verileri şifreleyip cihazın PIN'i değiştirmektedir.

1.1.2. Köklenme Kötücül Yazılımı (Rooting Malware)

Köklenme kötü Amaçlı Yazılımı, son birkaç yılda Android kullanıcılarının karşılaştığı en büyük tehdit olarak bilinmektedir. Bu tür kötü amaçlı yazılımlar, neredeyse her şeyi yapmalarına izin veren ayrıcalıklı kullanıcıların haklarını elde etmek için sistem güvenlik açıklarından yararlanmaktadır. Ayrıca, bu tür kötü amaçlı yazılımlar, Sistemden kaldırılmaya karşı kendisini korumak için sistem klasörlerine modüller yükleyebilmektedir. Ztorg, SecureList ekibi tarafından 2016 yılında Trojan.AndroidOS.Ztorg.ad [19] olarak tespit edilen köklenme kötü amaçlı yazılımın örneğidir. SecureList ekibi, Google Play'de Pokémon GO rehberi olarak adlandırılan virüslü bir uygulamayı keşfettikleri ni söylemişlerdir. Keşfedilen uygulama, birkaç hafta boyunca Google Play'de bulunmuş olup, 500.000'den fazla defa indirildiği belirtilmiştir. Bundan sonra, Google Play Store'da bazı benzer virüslü uygulamaları bulunmuştur. Keşfedilen uygulamalardan birinin, bir milyondan fazla yüklendiği belirtilmiştir.

Trojan.AndroidOS.Dvmap.a, SecueList ekibi tarafından Nisan 2017'de keşfedilen bu tür kötücül yazılımların bir başka örneğidir [20]. Dvmap, 50.000'den fazla indirildikten sonra Google Play Store'da keşfedilmiştir. Bu kötücül yazılım, kötü amaçlı kodunu, sistemin çalışma zamanı kitaplıklarına enjekte etmek için kök ayrıcalığı kazanmaktadır.

1.1.3. SMS Truva Atları

Bu tür Truva atları, virüs bulaşmış mobil cihazlardan özel tarifeli numaralara metin mesajları göndermek için kullanılmaktadır. Trojan.SMS.Agent, Trojan.SMS.FakeInst ve Trojan.SMS.Rufraud, bu tür kötücül yazılım ailesinin örnekleridir. 2010 yılında keşfedilen AndroidOS.FakePlayer.A, ilk keşfedilen Android SMS Truva atı olarak bilinmektedir. FakePlayer.A, önceden belirlenmiş numaralara [21] özel tarifeli (Premium-rate) SMS mesajları göndermeye çalışan bir Truva atıdır.

1.1.4. WAP-Faturalama Truva Atı (WAP-Billing Trojan)

WAP faturalandırması, maliyeti doğrudan kullanıcının cep telefon faturasına yansıtan bir mobil ödeme şeklidir. Bu teknik özel tarifeli SMS mesajlarına benzer, ancak bu durumda kötücül uygulaması, WAP faturalandırma mekanizması içeren bir web sayfasındaki belirli bir düğmeye tıklamaya bağlıdır. Bu tür Truva atları, 2017 SecureList'in BT tehdit gelişimi raporunda [22] ilk 20 mobil kötücül yazılım listesinde sunulmuştur. Genel olarak, WAP faturalandırma Truva atları, Wi-Fi bağlantısı kapatmak, internet bağlantısı açmak ve WAP faturalandırma mekanizması içeren sayfaya yeniden yönlendiren bir URL açmak gibi benzer davranışlara sahiptir. Genellikle bu Truva atları, WAP faturalandırma mekanizmasını içeren bir web sayfasını otomatik olarak yükleyip, JavaScript (JS) dosyaları kullanan düğmelere tıklamaktadır. Bundan sonra, Truva atı, mobil şebeke operatöründen gelen ve abonelik hakkında bilgi içeren SMS mesajlarını silmektedir.

1.1.5. Reklam Yazılımı (Adware)

Reklam görüntüleyen herhangi bir yazılım veya yazılım parçasını içermektedir. ANDROIDOS_ROTTENSYS, Trend MICRO'ya göre 2016'dan bu yana yaklaşık 5 milyon Android cihazı etkileyen bu tür Truva atlarının bir örneğidir [23]. ANDROIDOS_ROTTENSYS'i kurduktan sonra, BotNet'in bir parçası olmak için komut ve kontrol (C&C) sunucusuna bağlantı kurmaktadır. Ayrıca, ANDROIDOS_ROTTENSYS Truva atı ilk kurulduğunda, dinamik algılama tekniklerinden kaçınmak için belli bir süre içinde C&C sunucusuna bağlanmamaktadır.

1.1.6. Bankacılık Truva Atları

Bankacılık Truva atları, en tehlikeli kötü amaçlı uygulamalardan biridir. Bu tür Truva atları, kullanıcı bilgileri çalmak ve bir C&C sunucusuna göndermek için e-dolandırıcılık tekniklerine dayanmaktadır. Kaspersky Lab, yalnızca 2016 yılında 77.000'den fazla mobil bankacılık Truva atı, keşfedildiğini belirtmiştir [24]. Asacub, Kaspersky Lab araştırmacıları tarafından 2015 yılında keşfedilen bu tür Truva

atlarına bir örnektir [25]. Asacub'ın orijinal sürümü, gelen tüm SMS mesajlarını çalma ve bunları kötü niyetli bir sunucuya yükleme yeteneğine sahiptir. Asacub'ın keşfedilen son sürümü, kullanıcının kimlik bilgilerini çalıp, uzak bir C&C sunucusuna göndermek için sahte banka oturum açma sayfalarını kullanmaktadır. Svpeng, bankacılık Truva atlarına başka bir örnektir. Svpeng, kullanıcı bilgilerini çalma ve özel numaralara SMS mesajları gönderme yeteneğine sahiptir. Banker.AndroidOS.Svpeng.ae, 2017 yılında Kasper Lab tarafından keşfedilen Svpeng'in yeni bir sürümüdür [26]. Yeni sürüm, tuş vuruşlarını dinleyen ve toplanan metinleri uzak bir sunucuya gönderen bir keylogger içermektedir.

1.1.7. Truva At Casusu (Trojan-Spy)

SecureList'e göre, Trojan-Spy, kullanıcı eylemlerini, gizlice izlemek ve bir C&C sunucusuna göndermek için kullanılmaktadır. Trojan-Spy, tuş vuruşlarını dinleyebilme, ekran görüntülerini alabilme, çalışan uygulamaların listesini alabilmek gibi bir sürü yeteneğe sahiptir. Android/Tramp.A, Android platformu hedefleyen bu Truva atlarına bir örnektir. Tramp.A, cihaz ile ilgili bilgileri uzaktaki bir sunucuya gizlice iletmektedir. Tramp.A Sistemlere yüklendiğinde, kullanıcı tarafından fark edilmemek için, hiçbir kısa yol oluşturmamaktadır [27]. Bu sebepten Tramp.A'nın, cihazdaki varlığının fark edilmesi zorlaşmaktadır [28].

1.1.8. Botnet

Bu tür kötü amaçlı yazılımlar, virüslü cihazlarda bir Bot oluşturup, bunları bir Bot ağının parçası haline getirmektedir. Bir cihaza bu tür kötü amaçlı uygulamalar yüklendiğinde, olabildiğince veriyi toplayıp, C&C sunucusuna göndermektedir. Anserverbot.A, Android platformunu hedefleyen, Botnet kötü amaçlı yazılımına bir örnektir.

1.2. Android İşletim Sistemine Giriş

Android, Google tarafından yönetilen, Android Açık Kaynak Projesi (AOSP) tarafından geliştirilen, Linux çekirdeğine dayanan, ücretsiz bir açık kaynaklı işletim

sistemidir. Google, 2005 yılında ana geliştiricilerden, Android sistemini satın almış, ancak Android'in resmi duyurusu 2007'de yapılmış ve ilk Android cihazı 2008'de piyasaya çıkmıştır.

1.2.1. Android Uygulaması

Genellikle, Android uygulaması Java programlama dili kullanılarak yazılmakta, onun dışında bazı yerel kodu eklenebilmektedir. Daha sonra, uygulama Dalvik bayt koduna dönüştürülen ve .dex (Dalvik EXecutable) veya .odex (Optimize Edilmiş Dalvik Executable) dosyalarında saklanan Java bayt koduna derlenmektedir. Sonunda, uygulama, kaynak kodunu (.dex dosyaları), kaynakları, varlıkları ve bildirim dosyasını içeren bir APK arşivine sıkıştırılmaktadır. Android uygulamasında tanımlanabilecek dört tür bileşen vardır: etkinlikler (Activities), hizmetler (Services), yayın alıcılar (Broadcast Receivers) ve içerik sağlayıcıları (Content Providers). Etkinlikler, Android uygulamasında kullanıcı arabirimini sağlayan bölümdür. Hizmetler, herhangi bir nedenle uygulamanın arka planda çalışmasını sağlamak için genel amaçlı bir giriş noktasıdır. Yayın alıcıları, uygulamanın işletim sisteminden veya başka bir uygulamadan belirli bir olay almasına izin veren bir giriş noktasıdır. İçerik sağlayıcı, uygulama ve SQLite veri tabanı arasındaki bağlantıyı yöneten bir veri tabanı yönetim sistemi görevi görmekte, ayrıca, içerik sağlayıcı uygulamalar arasında paylaşılan verileri yönetmek için kullanılmaktadır. Android APK arşivi aşağıdaki dosya ve klasörleri içermektedir:

1.2.1.1. AndroidManifest.xml

Android uygulamasındaki en önemli dosyalardan biri olup herhangi bir uygulamayı çalıştırıldığında bu dosyayı Android işletim sistemi tarafından okunup ona dayanarak uygulama nasıl çalışacağını belirlenmektedir**.**

1.2.1.2. Classes.dex

Dex kodu, dosya başlığı, dize tablosu, yerel değişken listesi, sınıf tanımlama tablosu, yöntem listesi gibi birden fazla yapı içeren Android uygulamaları için optimize edilmiş bir bayt kodudur.

1.2.1.3. Resources.arsc

Bu dosya uygulamanın kaynaklarını ikili formatında içermektedir.

1.2.1.4. Lib/ klasör

Bu klasör yerel kod (Native Code) kütüphaneleri içermektedir.

1.2.1.5. Assets/ klasör

Genel olarak, müzik, görüntüler ve PDF dosyaları gibi varlıklar bu klasöre yerleştirilip AssetManager kullanılarak erişilmektedir.

1.2.1.6. Res / klasör

Uygulamanın simgeler, resimler vb. Kaynakları bu dizine yerleştirilmektedir.

1.2.1.7. META-INF/ klasör

Uygulamanın sertifikasını içermekte ve MANIFEST.MF, *.SF ve *.RSA olmak üzere üç temel dosyadan oluşmaktadır.

1.2.2. Android Güvenlik Mekanizmaları

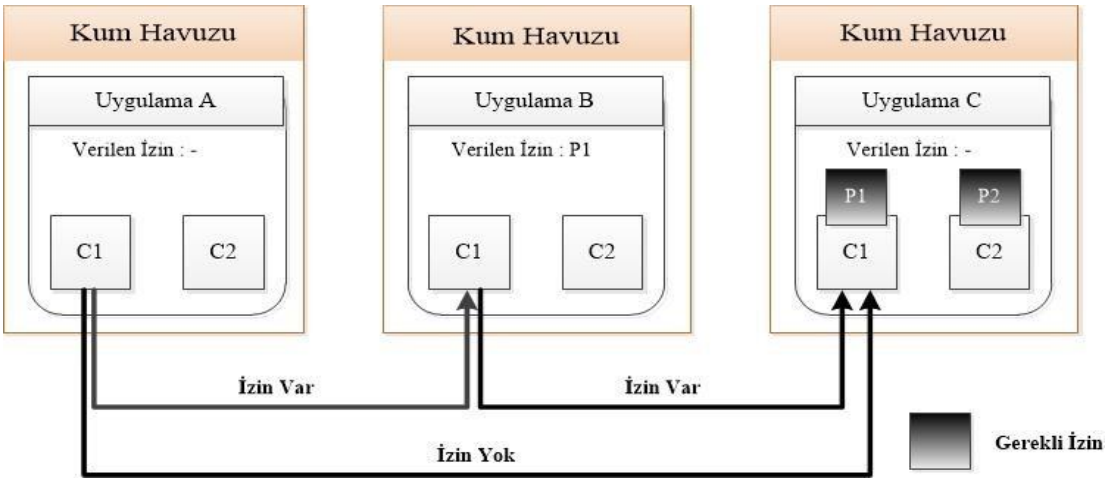
Android sistemi, izin çerçevesi, kum havuzu (Sandboxing) ve uygulama imzalama olmak üzere üç koruma mekanizmasına dayanmaktadır.

1.2.2.1. İzin Çerçevesi

Bu yöntem, Google tarafından sistem kaynakları korumak için bir mekanizma olarak tasarlanmıştır; böylece programın, doğru izinlere sahip olmadığı sürece sisteme veya başka bir uygulamaya ait herhangi bir korumalı kaynağa erişmesini engellemektedir. Verilen izinler, uygulamanın sanal alanına ait tüm bileşenler tarafından devralınmaktadır. Dolayısıyla, bir uygulamanın belirli bir sistem kaynağına erişmek istemesi durumunda, AndroidManifest.xml dosyasında uygun izin bildirilmesi gerekmektedir.

1.2.2.2. Kum Havuzu (Sandbox)

Kum havuzu, belirli bir izni olmadığı sürece uygulamaları birbirinden yalıtmak ve herhangi bir uygulamanın diğer uygulamaların kaynaklarına erişimini engellemek için kullanılan bir tekniktir. Başka bir deyişle, her Android uygulaması sanal makine (VM) örneği içinde yürütülmekte, böylece uygulamaları birbirinden izole etmek için bu örneklerin her birisi benzersiz bir kullanıcı kimliği altında yürütülmektedir. Uygulamalar diğer uygulamaların kaynaklarına yalnızca IPC (İşlemler Arası İletişim) bağlayıcı mekanizmasını kullanarak erişebilmektedir. Şekil 1.1'de gösterilen ayrıcalık yükseltme (Privilege escalation) olarak adlandırılan bir saldırıyı kullanarak kum havuzu mekanizması atlanabilmektedir. Ayrıcalık yükseltme saldırısı şu şekilde çalışmaktadır: A, B ve C olmak üzere üç uygulama olduğu varsayalım ve A uygulaması C uygulamasındaki C_1 bileşenine erişmek istemekte, ancak C_1, P_1 diye bir izin gerektirmektedir. A uygulaması P_1 iznine sahip olmadığından C_1 bileşenine doğrudan erişememektedir. Ancak A uygulaması, herhangi bir izin gerektirmeyen ve P_1 iznine sahip B diye bir uygulamaya erişebilir ise A uygulaması, gerekli bileşen (yani C_1 bileşeni) B uygulaması aracılığıyla dolaylı olarak erişebilecektir.



Şekil 1.1. Ayrıcalık yükselme saldırı senaryosu

1.2.2.3. Uygulama İmzalaması

Android yazılım geliştiricileri, kendi uygulamaları kendi anahtarlarını kullanarak imzalamalıdır. Bu yöntemi kullanma amacı kötü amaçlı uygulamalardan koruma mekanizması sağlanması değil, aynı kuruluş tarafından geliştirilen uygulamalara belli bir seviyeye kadar güvenin sağlanmasıdır. Aynı anahtarla imzalanan uygulamalar aynı kum havuzu (SandBox) içinde çalışabilmektedir.

Daha önce Google, uygulamayı yalıtılmış bir ortamda çalıştırarak ve davranışlarını inceleyerek uygulamaları dinamik olarak analiz eden Bouncer adlı bir çerçeve kullanmaktaydı. Google kısa süre önce, son kullanıcı tarafından yüklendikten sonra bile uygulamaları tarayan ve her zaman açık bir araç olan Play Protect adlı yeni bir çerçeve tanıtmıştır. Ayrıca, Play Protect, üçüncü taraf pazarlardan indirilen uygulamaları dahi tarayabilmektedir. İlave olarak, bu çerçevenin her gün 50 milyardan fazla uygulamayı tarayabildiğini ve doğrulayabileceği belirtilmiştir [29].

2. LİTERATÜR TARAMASI

Android kötü amaçlı yazılım analiz alanında yürütülen 200'den fazla makale analiz ederek önceki eserler için yeni kapsamlı bir taksonomi önerilmiştir [30].

2.1. Önerilen Taksonomi

Önerilen taksonomi, analiz edilen çalışmaların hedeflerine ve eserlerde çözmeye çalışılan sorunlarına dayanmaktadır. Ayrıca, ele alınan eserlerde kullanılan özellikler için kapsamlı bir taksonomi önerilmiştir. Ek olarak, kullanılan analiz yöntemlerinin zafiyetleri ayrıntılı bir şekilde sunulmuş ve tartışılmıştır. Şekil 2.1 önerilen taksonomiyi göstermektedir. İncelenen çalışmaları, davranış analizi çerçeveleri, dinamik analiz ortamları, politika zorlama çerçeveleri, paketleme/paketten çıkarma araçları ve kullanıcı ara yüzü tetikleme araçları olmak üzere beş ana eğilime bölünmüştür.

2.1.1. Davranış Analizi Çerçeveleri

Bu araştırma trend, kötü amaçlı yazılımların davranışlarını analiz etmek ve sınıflandırmak için kullanılabilen çerçeveleri önermeyi amaçlayan tüm çalışmaları kapsamaktadır. Bu tür işler, iki kritere göre sınıflandırılmıştır. Birinci kriter, herhangi bir kötü amaçlı yazılım analiz modelinde kullanılan üç iyi bilinen analiz aşamasının (yani ön işleme aşaması, özellik çıkarma aşaması ve algılama aşaması) her birinde kullanılan teknik yöntemlerdir. İkinci kriter, incelenen çalışmada, analiz yönteminin karşılaştığı zorluklara karşı önerilen tedbirlerdir.

2.1.1.1. Analiz Aşamalarında Kullanılan Tekniklere Dayan Taksonomi

Genel olarak, Android kötü amaçlı yazılım algılama sürecini, ön işleme, özellik ayıklama, özellik seçim ve algılama olmak üzere dört aşamaya ayrılmıştır. İncelenen çalışmaları, bu aşamalarda kullanılan tekniklere göre sınıflandırılmıştır. Bu aşamaları, aşağıdaki bölümlerde ayrıntılı olarak tartışılacaktır.

I. Ön-İşleme Aşamasında Kullanılan Metodolojilere Dayan Taksonomi

Bu aşamada, uygulamaların davranışları tanımlayan kalıpları oluşturmak için kullanılacak özellikleri çıkarmak üzere Android uygulama veri seti uygun bir formata çevrilmektedir. Bu aşama, kötü amaçlı yazılım algılama sürecinde çok önemlidir, çünkü ayıklanacak özelliklerin doğası ve gücü belirlemekte, bu da kullanılan sınıflandırıcı gücüne yansımaktadır. Genel olarak, bu aşamada kullanılan dört ana yöntem vardır: görselleştirmeye dayalı analiz, statik analiz, dinamik analiz ve melez analiz. Melez analiz yönteminde, uygulama öncelikle statik olarak analiz edip uygun statik özellikleri ayıklanmakta, ardından uygulama dinamik özellikleri ayıklanmak için dinamik olarak analiz edilmektedir. Bu dört yöntem ve her birinin bazı örnekleri aşağıdaki bölümlerde tartışılacaktır.

1) Görselleştirmeye Dayalı Analiz

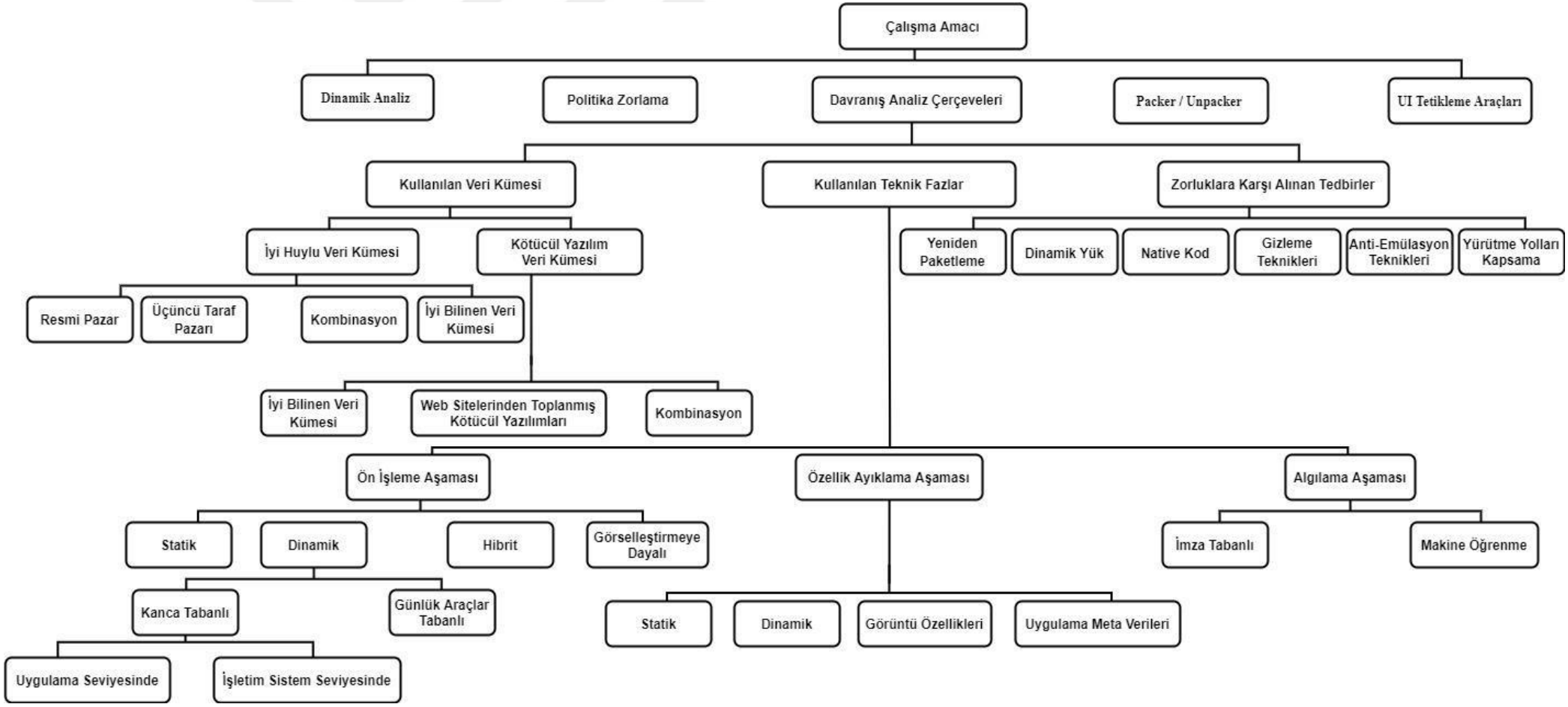
Görüntü işleme teknikleri, masaüstü bilgisayarları hedef alan kötü amaçlı yazılımları tespit etme alanında yaygın olarak kullanılmasına rağmen ve bu teknolojileri, etkinliğini kanıtlamış olmasına rağmen, akıllı telefonları hedef alan kötü amaçlı yazılımlar tespit etme konusunda çok sınırlı bir şekilde kullanılmıştır. Bununla birlikte, bu yöntem önceki bazı çalışmalarda kullanılmıştır, örneğin Huang ve arkadaşları, uygulama kaynak kodunu, bir RGB görüntüsüne dönüştürmeye ve uygulama sınıfı tahmin etmek için derin öğrenme teknikleri kullanmaya dayanan bir çerçeveyi önermişlerdir [31]. Bu amaçla, uygulama geri derlenmiş, DEX kodunu çıkarılmış ve bayt kod olarak temsil edilmiştir. Bundan sonra, talimatların on altılı gösterimi üç bölüme bölünmüş ve her bir bölüm renk kanallarından birisi (yani R, G ve B) temsil ederek uygulamanın kaynak kodunu RGB görüntülere dönüştürülmüştür. Daha sonra, evrişimli sinir ağı, uygulamaların sınıfları tahmin etmek için bir modeli olarak kullanılmıştır. Huang ve Wen, Classes.dex, AndroidManifest.xml, Resources.arsc ve CERT.RSA olmak üzere dört dosyanın gri tonlamalı görüntülere dönüştürülmesini önermişlerdir [32]. Bundan sonra, oluşturulan görüntülerden GIST özelliği ayıklanıp Android uygulamaları kötü niyetli veya iyi niyetli olarak sınıflandırmak için rastgele karar ormanlar sınıflandırıcısına girdi olarak kullanılmıştır. Karimi ve Moattar tarafından, APK arşivi açılmış, bayt

kodu çıkartılmış, işlenmiş ve gri tonlamalı görüntülere dönüştürülmüştür [33]. Bu amaç ile, 2 uzunluğuna sahip İşlem Kod Dizileri (Opcode Sequences) uygulama kaynak kodundan oluşturulmuştur. Ardından, eğitim veri set örneklerindeki sıklığına göre oluşturulan kod dizilerine ağırlık değerleri atanmış ve ağırlık değerleri, uygulama temsil eden görüntüsünde bir piksel olarak kabul edilmiştir. Bundan sonra, görüntü gürültüsü azaltmak ve algılama doğruluğu artırmak için Latent Dirichlet Allocation (LDA) algoritması kullanılarak en iyi dizileri seçip görüntü boyutları azaltılmıştır. Sonunda, optimize edilmiş pikselleri tek bir vektörde istiflenmiş ve Ransomware uygulamaları tespit etmek için imza olarak kullanılmıştır. Yen ve arkadaşlarının çalışmasında, uygulama kaynak kodu çıkartılmış ve komutları, önemlerine göre gruplara ayrıştırılmış, ardından, oluşturulduğu komut grupları, Simhash ve Djb2 hash fonksiyonları kullanılarak sayısallaştırılmıştır [34]. Elde edilen hash değerleri bir görüntüye dönüştürülmüştür. Son olarak, android uygulamaları iyi huylu veya kötücül olarak sınıflandırmak için bir evrişimli sinir ağı kullanılmıştır.

2) Statik Analiz

Düşük hesaplama süresi, uygulama kolaylığı ve bir dereceye kadar etkililiği sayesinde birçok araştırmacı tarafından en çok kullanılan ve tercih edilen yöntem olarak kabul edilmektedir. Bu yöntemde, uygulama kaynak kodu bir emülatörde veya gerçek cihazda çalıştırmaksızın analiz edilmektedir. Bu amaçla, öncelikle kaynak kod içindeki kullanılan sınıfları, Manifest dosyası, meta-veri bilgileri ve medya dosyaları elde etmek için APK arşivi içerikleri çıkarılmaktadır. Bu aşamada, uygulama kaynak kod formatı, işlenmesi kolay olmayan olan Dex bayt kodudur, bu nedenle, işlenmesi daha kolay hale getirilmek için Java veya Smali koduna geri derlenebilmektedir. Bu adımda, APK arşivleri açabilen ve uygulamaların neredeyse orijinal içeriğinin aynısını çıkartabilen açık kaynaklı tersine mühendislik aracı olan Apktool [35] gibi birden fazla aracı kullanılabilmektedir. Ayrıca, daha önce de belirtildiği gibi, DEX kod dosyaları, Dex bayt kodu ve Java arasında orta bir temsil olan Smali koduna dönüştürülebilmektedir. Bu nedenle, Smali kod gösterimi, koda dayalı özellikleri ayıklamak için önceki araştırmalarda yoğun bir şekilde kullanılmıştır. Statik analiz yöntem kullanan bazı ele alınan çalışmaları listelenecek

ve geri kalan çalışmalar tek ve kapsamlı bir tabloda listelenecektir. Gurulian ve arkadaşları, uygulama yeniden paketlenmesi tespit etmek için bir yöntem önermişlerdir [36]. Önerilen yöntem, orijinal uygulamanın popülerliğinden yararlanmak için adı ve simgesi gibi orijinal uygulamanın bazı verisi, yeniden paketlenmiş uygulama geliştirici tarafından değiştirilmeme gerçeğine dayanmaktadır. Uygulamadan statik özellikleri ayıklanan araç ve bir imza veritabanı ile karşılaştırarak uygulama hakkında karar veren araç olmak üzer önerilen çerçeve iki aracı içermektedir. Zhu ve arkadaşlarının çalışmasında, kullanılan izinleri, hassas API'leri ve izin oranı gibi bazı statik özellikleri önerilen statik analize dayalı çerçeveyi eğitmek ve test etmek için kullanılmıştır [37]. Ardından, ayıklanan özelliklerin ön işleme tabi tutulması için temel bileşen analizi (TBA) algoritması benimsenmiş ve android uygulamaları kötü amaçlı veya zararsız olarak sınıflandırmak için topluluk Rotation Forest RF sınıflandırıcısı kullanılmıştır. Ayrıca, elde edilen sonuçlar, aynı deneysel koşullar altında Destek Vektör Makinesi (SVM) modelinin sonuçlarıyla karşılaştırılmıştır. Chen ve arkadaşlardaki çalışmada, kötücül yazılım tespit yöntemlerinin karşılaştığı çeşitli zorlukları gözden geçirilmiş ve geleneksel makine öğrenme sınıflandırıcılarının çözemediği bazı saldırıları tartışılmıştır [38]. Bu hususlara dayanarak, sınıflandırıcılarının algılama yeteneğini düşürmek için veri setlerini zehirleyebilecek üç tür saldırı sunulmuş ve test edilmiştir. Bu saldırıları gidermek için, yanlış negatifleri önemli ölçüde azaltan ve algılama doğruluğu artıran KUAFUDET adlı bir algılama sistemi önerilmiştir. Wang ve arkadaşlarındaki, önerilen modeli eğitmek ve test etmek için istenilen izinleri, filtrelenmiş Intent'leri, kısıtlanmış API çağrıları, donanım özellikleri, ve şüpheli API çağrıları dahil olmak üzere birçok statik analiz özelliği kullanılmıştır [39]. Kirubavathi ve Anitha, Android Botnet uygulamaları algılayabilen statik analize dayalı bir yöntem önermişlerdir [40]. İlk olarak, Botnet'in kötü niyetli faaliyetleri tanımlayabilen benzersiz desenleri oluşturmak için istenilen ve kullanılan izinleri, iyi huylu ve bot uygulamaları içeren bir veri kümesinden ayıklanmıştır. Sonunda, oluşturulan desenleri kullanılarak bazı makine öğrenime sınıflandırıcıları eğitilip android uygulamaları iyi huylu veya Bot olarak sınıflandırmak için kullanılmıştır.



Şekil 2.1.Önerilen taksonomi.

Tao ve arkadaşlarındaki çalışmada, izinle ilişkili API'ler ve rastgele orman sınıflandırıcı kullanan ve MalPat olarak adlandırılan otomatik bir kötücül yazılım algılama sistemi önerilmiştir [41]. Önerilen model oluşturmak için, her uygulamadaki API'leri ayıklanmış ve kötü amaçlı yazılımları, zararsız uygulamalardan ayırmak amacıyla kullanılabilecek benzersiz desenleri oluşturmak için izin-API korelasyonları kullanılmıştır.

3) Dinamik Analiz

Statik analiz teknikleri, hızı, uygulama kolaylığı ve hesaplama süresinin düşük olması nedeniyle birçok araştırmacı tarafından tercih edilmektedir. Ancak, statik analiz teknikleri, kod gizleme teknikleri ele alamama veya kötü amaçlı içeriğin dinamik yüklenmesi gibi birçok zayıflığına maruz kalmaktadır. Ön işleme aşamasında kullanılabilecek ikinci yöntem dinamik analizdir. Bu yöntemde, android uygulamaları, izole bir ortamda yürütülmekte ve uygulamaların davranışı hakkında mümkün olduğunca fazla bilgi toplamak için uygulamaların normal kullanımını simüle edilmektedir. Bu amaçla, uygulamanın normal çalışmasını simüle etmek ve davranışı hakkında bilgi toplamak için hem android uygulaması hem de kullanıcı ara yüzü olay oluşturma aracı, emülatöre veya gerçek bir cihaza kurulmaktadır. Bu çalışmada, dinamik analiz tabanlı çerçeveleri, uygulamanın davranışını izlemek için kullanılan tekniklere göre kanca (hook) tabanlı çerçeveleri ve günlük araçları tabanlı çerçeveleri olmak üzere iki tür olarak ayrılmıştır.

Günlük araçları tabanlı çerçevelerde, uygulamayı gerçek bir cihazda veya emülatörde çalıştırılıp, iyi bilinen günlük araçları kullanarak uygulamadaki davranışı izlenmektedir. Örneğin, Somarriba ve Zurutuza, android kötücül yazılımları tespit etmek için bulut tabanlı bir dinamik analiz çerçevesi önermişlerdir [42]. Önerilen çerçeve, bazı açık kaynak araçları kullanarak Android uygulamalarının yürütme zaman davranışını izlemeye ve analiz etmeye dayanmaktadır. Wei ve arkadaşları, uygulamanın ağ davranışlarını izlemeye dayanarak android uygulamaları iyi huylu veya kötücül yazılım olarak sınıflandırmak için bir yöntem önermişlerdir [43]. Uygulamanın giden ağ trafiği, analiz ortamında Tcpdump çalıştırılarak izlenmiş olup iyi huylu uygulamalar kötü niyetli uygulamalarından ayırt etmek amacıyla bir

makine öğrenme sınıflandırıcısı benimsenmiştir. Kurniawan ve arkadaşlarındaki çalışmada, ağ trafiği, pil tüketimi ve pil sıcaklığı gibi bazı kaynak tüketim özellikleri, bazı kayıt araçları kullanılarak izlenmiştir [44]. Ardından, kötü amaçlı yazılım uygulamaları, iyi huylu uygulamalardan ayırmak için birden fazla makine öğrenme algoritması benimsenmiştir. Alzaylaee ve arkadaşlarındaki çalışmada, makine öğrenme teknikleri kullanarak android kötü amaçlı yazılımı tespit etmek için dinamik analize dayalı bir çerçeve önerilmiştir [45]. Bu amaçla, dinamik özellikleri otomatik olarak ayıklanan bir araç uygulanmıştır, ardından, emülatör tabanlı ve telefon tabanlı kötücül yazılım tespit arasında karşılaştırma yapmak için birçok deneyi yapılmıştır. Gerçek telefondan ve emülatörden ayıklanan özellikleri, birden fazla makine öğrenme sınıflandırıcısı eğitmek için kullanılmış olup elde edilen sonuçları birbiriyle karşılaştırılmıştır.

İkinci kullanılan dinamik analiz metodolojisi, enstrümantasyon tabanlı veya kanca tabanlı çerçeveler olarak bilinmektedir. Bu yöntemde, uygulamadaki etkinlikleri, yürütme sırasında kaydetmek için uygulama kaynak kod içine izleme noktaları (kancaları) yerleştirilmektedir. Uygulama düzeyinde enstrümantasyon ve işletim sistem düzeyinde enstrümantasyon olmak üzere önceki çalışmalarda enstrümantasyon tabanlı sistemleri uygulamaya yönelik iki eğilim bulunmaktadır.

Uygulama düzeyindeki enstrümantasyon tabanlı yönteminde, uygulamayı geri derlenmekte ve kaynak kodunu, uygulama davranışı günlüğe kaydedebilen bazı kod parçaları (kancaları) eklenerek değiştirilmekte ve uygulama yeniden derlenmektedir. Gömülü kancalar, yürütme sırasında uygulama davranışını izleyip veri akışı gibi önemli davranışsal bilgiler için bir günlük kaydetmektedir. Örneğin, Shuaifu ve arkadaşlarındaki çalışmasında, şüpheli API'leri izleyerek Android uygulamalarının kötü niyetli davranışlarını analiz etmek için uygulama düzeyinde bir enstrümantasyon yöntemi önerilmiştir [46]. Önerilen yöntem, işletim sistemi düzeyinde herhangi bir değişiklik gerektirmediğinden, Android işletim sisteminin tüm sürümleriyle uyumludur. Ali-Gombe ve arkadaşları, Android uygulamaların şüpheli davranışlarını tespit etmek için AspectDroid adlandırılan ve işletim sistem çalışma zamanı ve işletim sistem sürümü bağımsız çalışabilen melez bir sistemi önermişlerdir [47]. Veri akış analizi, kaynak kötü kullanım tespiti ve şüpheli davranış

analitiği elde etmek için bir enstrümantasyon motoru tasarlanmıştır. Statik aşamada, uygulamaları, tersine mühendislikten geçirilmiş, özel günlük kaydı ve diğer analitik işlevleri gerçekleştirmek için uygulamanın orijinal koduna bazı kodları enjekte edilmiş ve uygulama yeniden derlenmiştir. Enstrümante edilmiş uygulamanın çalışma zaman olayları izlemek ve günlüğe kaydetmek için uygulamayı dinamik olarak yürütülmüştür.

İşletim sistemi seviyesindeki enstrümantasyon yönteminde ise, uygulama yürütme sırasında uygulama davranışını günlüğe kaydedebilmek üzere işletim sistemi, onun kaynak koduna izleme noktaları eklenerek değiştirilmektedir. Örneğin, Rastogi ve arkadaşları, Android işletim sisteminin kaynak kodu değiştirilip API düzeyinde izlemek için kancaları yerleştirmişlerdir [48]. Ayrıca, çekirdek düzeyinde izleme yapmak için çekirdek düzeyinde değişiklikler yapmışlardır. Enck ve arkadaşları, yüklü uygulamalardaki hassas veri akışını izlemek için TaintDroid olarak adlandırılan ve kancaya dayanan bir veri analiz çerçevesi geliştirmişlerdir [49]. Önerilen sistemin temel amacı, sistemden çıkan hassas bilgileri tespit etmek ve analiz etmektir. Önerilen sistemde değişken düzeyi, mesaj düzeyi, yöntem düzeyi ve dosya düzeyi olmak üzere dört veri akış düzeyi izlenmiştir. Yan ve Yin, işletim sistem seviyedeki kancaları ve Java seviyedeki kancaları olmak üzere iki seviyeli kancaları yerleştiren ve DroidScope olarak adlandırılan bir enstrümantasyon aracı geliştirmişlerdir [50]. Ayrıca, önerilen araç, Dalvik bayt kodunun yanı sıra yerel talimatların dinamik analizini sağlayabilmiştir.

4) Melez Analiz (Hibrit Analiz)

Bu yöntemde, daha derin bir analiz elde etmek için hem statik hem de dinamik analizi birleştirilmektedir. Genel olarak, uygulamaların kaynak kodundan statik özellikleri ayıklamak için tersine mühendislikten geçirilmekte, bundan sonra dinamik özellikleri ayıklamak için uygulamaları, yalıtılmış bir ortamda, (bir öykünücü veya gerçek bir cihazda) çalıştırılmaktadır. Melez yöntemdeki uygulama karmaşıklığına rağmen, bu yöntem en derin ve en kapsamlı analiz yöntem olarak kabul edilmektedir. Melez analiz yöntem daha önce çalışmalarda diğer analiz yöntemlere göre daha az kullanılsa da bu yöntem incelenen makaleler arasında bir dizi çalışmada

kullanılmıştır. Örneğin, Kabakus ve Dogru, hem statik hem de dinamik analiz tekniklerinin avantajlarından yararlanmak için "mad4a" adlı melez bir Android kötü amaçlı yazılım analiz yaklaşımı önermişlerdir [51]. Statik analiz aşamasında, uygulamanın Manifest dosyasında bulunan izinleri ayıklanmış ve ayıklanan izinleri, Java kaynak kodundaki karşılık gelen API çağrılarıyla eşleştirilmiştir. Dinamik analiz aşamasında ise, kötü amaçlı yazılımlar ve iyi huylu uygulamalar bir emülatöre yüklenmiş ve uygulamanın normal kullanımını simüle etmek için MonkeyRunner aracı kullanılarak 500 farklı UI olayı oluşturulmuştur. Yuan ve arkadaşlarındaki çalışmasında, iyi huylu ve kötü niyetli Android uygulamaları ayırt etmek için DroidDetector adlı ve derin öğrenmeye dayalı melez analiz bir çerçeveyi önerilmiştir [52]. Statik ve dinamik analiz kullanılarak toplam 192 ikili özelliği ayıklanıp derin öğrenme modeline girdi olarak kullanılmıştır. Derin Öğrenme modelinin Android kötü amaçlı yazılımları tespit etme yeteneğini doğrulamak için bir dizi deneyi yapılmıştır. Elde edilen sonuçlar, Naive Bayes, C4.5, Logistic Regression, SVM ve Multi-Layer Perceptron gibi bazı geleneksel makine öğrenme algoritmalar sonuçlarıyla karşılaştırılmıştır. Chen ve arkadaşlarındaki çalışmasında, gerekli izinleri ve hassas API çağrıları gibi statik özelliklerin yanı sıra ağ etkinliği, dosya sistem erişimi ve işletim sistemiyle etkileşimi gibi bazı dinamik özelliklere dayalı olarak Android kötü amaçlı yazılımı tespit etmek için melez analiz tabanlı bir çerçeveyi önerilmiştir [53]. Ayıklanan statik ve dinamik özellikleri, destek Vektör Makineleri (SVM), Karar Ağacı (C4.5), Yapay Sinir Ağları (MLP), Naive Bayes [54], K-En Yakın Komşular ve Torbalama gibi bazı makine öğrenme algoritmaları eğitmek ve test etmek için kullanılmıştır. Mas'ud ve arkadaşları, melez bir Android kötü amaçlı yazılım analiz çerçeveyi önermişlerdir [55]. Uygulamadan istenilen izinleri ve API yöntemleri, uygulama kaynağından statik olarak ayıklanmış ve sistem çağrıları ve ağ trafiği dinamik olarak izlenmiştir. Ayıklanan statik ve dinamik özellikleri, Android uygulamaları sınıflandırmak amacıyla kullanılan davranış kalıpları oluşturmak için kullanılmıştır.

II. Özellik Ayıklama Aşamasına Göre Taksonomi

Özellik ayıklama aşamasında, sınıflandırma modeli eğitmek için uygun özellikleri çıkarılmaktadır. Ayıklanan özellik türü ve niteliği, ön işleme aşamasında kullanılan

analiz yöntemine göre değişiklik göstermektedir. Bu çalışmada, kullanılabilecek özellikleri, statik özellikler, dinamik özellikler, melez özellikler ve görüntü tabanlı özellikleri olmak üzere dört türe ayrılmıştır. Şekil 2.2, incelenen çalışmalarda kullanılan özellikler için önerilen taksonomisini göstermektedir.

1) Statik Özellikleri

Genel olarak statik özellikler, hassas talimatları, değişken adları, yöntemler, sınıflar, paketler, dizeler, kod bağlamı ve sırası, kontrol akışı ve veri akışı gibi önemli bilgileri ayıklamak için uygulama kaynak kodunu ayrıştırılmasına bağlıdır. Bu çalışmada, statik özellikleri aşağıdaki şekilde sınıflandırılmıştır:

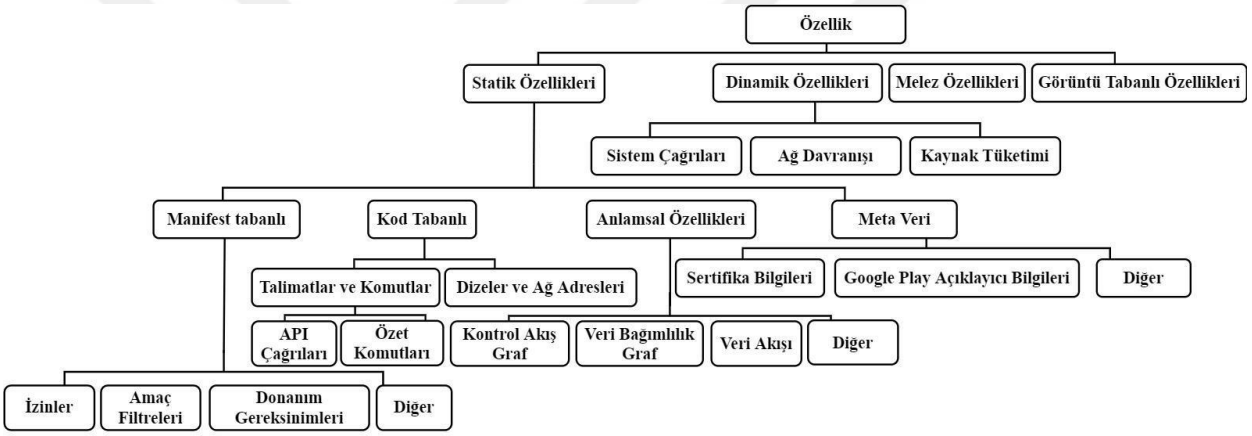
A. Manifest Tabanlı Özellikleri

Manifest dosya, Android uygulamanın en önemli dosyalarından biridir ve uygulamanın nasıl yürütüleceğini belirleyen bir denetleyici veya yol haritası olarak kabul edilmektedir. Ayrıca, faaliyetler, hizmetler ve içerik sağlayıcılar gibi uygulama bileşenleri, kod içinde kullanılmadan önce bu dosyada tanımlanmalıdır. Ek olarak, bu dosyada eylemler, amaçlar, amaç filtreleri, paket adı, kategori gibi uygulama davranışını açıklayan birçok meta-bilgileri içermektedir. En önemlisi, bu dosya, uygulamanın düzgün çalışması için önemli izinleri içermektedir. Bu dosya android uygulamaların çalışmaları için çok önemli olduğundan dolayı, özellikleri ayıklamak amacıyla incelenen çalışmalarda yaygın bir şekilde kullanılmıştır. Manifest dosyasından ayıklanan ve önceki çalışmalarda önerilen modelleri eğitmek için kullanılan özelikleri aşağıdaki gibi özetlenebilmektedir.

- **İzinler**

Bu özellik incelenen eserlerin çoğunda kullanılmıştır. Örneğin, Talha ve arkadaşları, izin tabanlı statik analiz Android kötü amaçlı yazılım tespit sistemi önermişlerdir [56]. Önerilen sistem üç bileşenden oluşmakta: İlk bileşen, uygulamalardaki davranış parmak izileri ve analiz sonuçları içeren bir imza veri tabanıdır. İkinci bileşen, son kullanıcılar tarafından analiz talepleri sağlamak için kullanılan bir Android

istemcisidir. Üçüncü bileşen ise hem imza veri tabanı hem de Android telefon müşterisi ile iletişim kurmak ve tüm analiz sürecini yönetmek için kullanılan merkezi bir sunucudur. Lojistik regresyon, bir uygulama, kötücül veya zararsız olarak sınıflandırmak için kullanılmış olup %88 bir algılama doğruluk oranı elde edilmiştir. Samra ve Ghanem gerçekleştirdiği çalışmada [57], uygulamanın izinleri ve diğer bazı meta-verileri özellik olarak kullanılmış ve K-ortalama algoritması, 18.147 iyi huylu Android uygulama içeren bir veri seti, iş uygulaması veya araç uygulamasına kümelemek için benimsenmiştir. Ayrıca, Arp ve arkadaşlarındaki çalışmasında [54], uygulamada kullanılan izinleri, API çağrıları ve ağ adresleri gibi birden çok özelliği ayıklamak için bir statik analiz yöntemi kullanılmış, ardından ayıklanan özellikleri, bir vektöre gömülüp uygulama sınıflandırmasında desenler olarak kullanılmıştır.



Şekil 2.2. Önerilen özellik taksonomisi.

- **Amaç Filtreleri (Intent Filters)**

Amaç filtresi, eylemler, veriler ve amaç kategorileri dahil olmak üzere uygulama niyetlerinin tam ayrıntılarını açıklayabildiğinden, uygulamaların şüpheli davranışlarını tespit etmek için kullanılabilen güçlü bir özelliktir. Örneğin, amaç filtresi genellikle kötü amaçlı yazılım uygulamaları tarafından kötü amaçlı etkinlik başlatmak için BOOT_COMPLETED gibi olayları algılamak için kullanılmaktadır. Bu tür özellikleri, önceki çalışmalarda yaygın olarak kullanılmıştır, örneğin, Feizollah ve arkadaşlar tarfından gerçekleştirilen çalışmada [58], Android kötü amaçlı yazılım tespit etme konusunda, Intent'lerin (Explicit ve Implicit) etkinliğini

incelenmiştir. Bu amaçla AndroDialysis adlı statik analiz tabanlı bir araç sunulmuştur. Önerilen araç, uygulamalardan amaçları (Intentler), amaç filtreleri ve izinleri ayıklamıştır, ardından, ayıklanan özellikleri kullanılarak birden çok deney gerçekleştirilmiştir. Kötü amaçlı yazılım tespitinde Intentlerinin, izinlerinden daha etkili olduğunu sonucuna varılmıştır.

- **Donanım Gereksinimleri (Hardware Requirements)**

Uygulamanın donanım erişim istekleri (Manifest dosyasında belli izinleri istenerek yansıtılmakta), Arp ve arkadaşlarındaki çalışmasında [54] ve Zhang ve arkadaşlarındaki çalışmasında [59] gibi bazı çalışmalarda uygulamaların kötü amaçlı davranışlarını tespit etmek için bir özellik olarak kullanılmıştır.

- **Diğer Manifest Tabanlı Özellikleri**

Bazı çalışmalarda, faaliyetler, hizmetler, paket adı ve amaçlar gibi diğer bazı Manifest dosyanın bilgileri kullanılmıştır. Örneğin, Moghaddam ve Abbaspour tarafından gerçeleştirilen çalışmada iyi huylu ve kötü amaçlı yazılım uygulamaları arasında ayrım yapmak için bazı kod tabanlı özelliklerle birlikte faaliyet sayısı, hizmet sayısı ve alıcı sayısı kullanılmıştır [60].

B. Kod Tabanlı Özellikleri

Bu özellik türü ister Java ister yerel kod olsun, uygulamanın kaynak kodundan ayıklanan tüm özellik türlerini ifade etmektedir. Uygulama kaynak kodundan ayıklanan özellikler, daha derin bir uygulama davranış analiz yapmak için çok önemlidir. İncelenen çalışmalarda, kod tabanlı özellikleri, Opcode, bayt kodu, Smali kodu ve Java kodu dahil olmak üzere çeşitli kaynak kod temsillerinden ayıklanmıştır. Önceki çalışmalarda kullanılan kod tabanlı özellikleri aşağıdaki gibi sınıflandırılacaktır.

- **Talimatlara ve Komutlara Dayalı Özellikler**

Bu tür özellikte, uygulamadaki kaynak kodu, uygulamanın olası şüpheli davranışını tanımlayabilecek belirli talimatları ayıklamak için ayrıştırılmaktadır. İncelenen çalışmalarda aşağıdaki talimatlar ve komutlara dayalı özellikleri kullanılmıştır.

API çağrıları: Android işletim sistem, işletim sistem kaynaklarına veya cihazdaki donanım bilişenlerine erişmek için geliştiriciler tarafından kullanılabilecek çok çeşitli API'leri sağlamaktadır. API çağrılar, kötü amaçlı yazılım algılama alanında en yaygın kullanılan özelliklerinden biridir. Genel olarak, bu özellikler kullanıldığında, şüpheli API'lerin bir listesi hazırlanıp uygulamanın kaynak kodunda kullanılan API'leri ile karşılaştırılmaktadır. API çağrıları, daha önce birçok çalışmada kullanılmıştır, örneğin, Wu ve arkadaşları, DroidMat adlı bir statik analiz sistemi sunmuşlardır [61]. Önerilen çerçeve, kötü amaçlı Android uygulamaları tespit etmek için, uygulamalarda kullanılan izinler, amaçlar ve API çağrılar gibi birçok özellikleri kullanılmıştır. Martín ve arkadaşları, kötücül yazılımları ve iyi huylu yazılımları ayırt etmek için MOCDroid adlı statik analiz tabanlı bir çerçeveyi önermişlerdir [62]. Üçüncü taraf API çağrı kombinasyonlarından (Import Terms) anlamsal bir desen çıkarılmış ve kötü amaçlı yazılımın veya iyi huylu yazılımın ilgili davranışlarını tutan iki alt model oluşturulmuştur. Bir aday uygulama, iyi huylu yazılım veya kötü amaçlı yazılım ile daha iyi uyuşup uyuşmadığını ölçmek için bu iki alt modele göre değerlendirilmiştir. Diğer bazı çalışmalarda, API çağrıları, API'daki paket düzey bilgi ve API parametrelerle birlikte kullanılmıştır. Örneğin, Aafer ve arkadaşları, uygulama davranışı hakkında bilgi elde etmek için bayt kod içindeki API seviye bilgisine dayanan bir yöntem önermişlerdir [63]. Önerilen yöntem, sık kullanılan kritik API çağrılarına, API paket düzeyindeki bilgilerine ve API parametrelerine odaklanmıştır. Diğer bazı çalışmaları, API'lerin, istenilen izinlerle eşleştirilmesine dayanmaktadır. Örneğin, Felt ve arkadaşları, API çağrıları ayıklanıp gerekli izinlerle eşleştirerek aşırı ayrıcalıklı izinleri tespit edebilen Stowaway adlı bir araç önermişlerdir [64].

Özel talimatları: Kötü amaçlı yazılım geliştirici tarafından sıklıkla kullanılan bazı özel talimatları veya kitaplık çağrıları, kötü amaçlı yazılım tespit etmek için önceki

çalışmalarda kullanılmıştır. Örneğin, DexClassLoader, kötü amaçlı içeriği yükleyip uygulama yürütme zamanında yürütmek için saldırgan tarafından kullanılabilen bir kitaplıktır. Ayrıca, Crypto API, uygulamadaki dizeleri veya diğer içerikleri şifrelemek için kullanılabilen bir kitaplıktır. DexClassLoader ve Crypto kitaplıkları, Sen ve arkadaşlarındaki çalışmada kullanılmıştır [65]. Ayrıca, Martín ve arkadaşlar, Import terimleri gibi diğer bazı özellikleri kullanmışlardır [62]. Ayrıca, yöntem çağrıları ve işlev argümanları ve komutları, Milosevic ve arkadaşlarının tarafından kullanılmıştır [66]. Su, Chmod ve Exec gibi bazı tehlikeli Linux komutları, Chen ve arkadaşlarının çalışması [38] ve Yerima ve arkadaşının çalışması [67] gibi bazı statik çerçevelerde, uygulamadaki kötü niyetli davranışları ortaya çıkarmak için özellikler olarak kullanılmıştır. Ayrıca, Yerima ve arkadaşındaki çalışmasında, uygulama davranışı ortaya çıkartabilecek gömülü Dex, Jar, So veya ELF dosyalarının varlığını tespit etmek için uygulama kaynak kodunu kontrol edilmiştir [67].

- **Dizeler ve Ağ Adresleri**

Genel olarak kötü amaçlı uygulamalar, kurbandan toplanabilecek verileri göndermek ve saldırgandan komutlar almak için bir komut ve kontrol (C&C) sunucusuna bağlanmaktadır. Bu amaçla, sunucunun adresi kod içine yerleştirilmekte, dolayısıyla, Zhang ve arkadaşlarındaki çalışmasında [68] ve Arp ve arkadaşlarındaki çalışmasında [54] olduğu gibi uygulama davranışı tanımlamak için herhangi bir IP veya DNS adresini bulmak için kaynak kodu ayrıştırılabilmektedir. Ayrıca, uygulama kaynak koddaki dizeleri, uygulama davranışına bir gösterge verebilmektedir, bu nedenle, Park ve arkadaşları, Lee ve arkadaşları ve Palumbo ve arkadaşlarındaki çalışmalarda [69-71] olduğu gibi önceki birçok çalışmada, kaynak koddaki dizeleri kullanılmıştır.

C. Anlamsal Özellikler

Anlamsal özelliklere sahip bazı özellikleri veya anlamsal bir şekilde görüntülenen diğer özelliklerin herhangi bir kombinasyonunu anlamsal özellikler olarak kategorize edilmiştir. Başka bir deyişle, anlamsal özellikler, kod tabanlı özellikleri, Manifest tabanlı özellikleri ve hatta semantik bir şekilde birleştirilen veya temsil edilen

uygulama açıklamasına dayalı özellikleri içermektedir. Ele alınan çalışmalarda kullanılan en önemli anlamsal özellikleri aşağıdaki gibi özetlenebilmektedir:

- **Kontrol Akış Çizelgesi (Control Flow Graph CFG)**

En popüler kullanılan uygulama davranış analiz yöntemlerinden biridir. Bu yöntemde, uygulamadaki kaynak kodu, yönlendirilmiş bir graf olarak temsil edilmekte, böylece düğümler, talimatları veya kod blokları temsil eder iken, kenarları, iki düğüm arasındaki kontrol akışını temsil etmektedir (talimatlar arasındaki yürütme yolunu temsil etmektedir). Böylece, CFG, uygulamanın tüm yürütme senaryolarını analiz etmek için tüm olası yürütme yolları temsil eden yönlendirilmiş bir graftır. Kwon ve arkadaşlarındaki çalışmasında [72], bilinmeyen kötü amaçlı yazılım çeşitleri tespit etmek amacıyla API çağrı grafları oluşturmak amacıyla anlamsal imzaları oluşturmak için aynı kontrol akış grafı kavramı kullanılmıştır. Ayrıca, Alam ve arkadaşlarındaki çalışmasında [73], hem bayt koddaki hem de yerel koddaki kötü niyetli davranışları tespit etmede kullanılabilecek anlamsal imzaları oluşturmak için yerel koda dayalı olarak kontrol akış grafları oluşturulmuştur.

- **Veri Bağımlılık Çizelgesi (Data Dependency Graph DDG)**

DDG, programdaki prosedürler arasındaki veri akışları temsil eden bir program analiz yapısıdır [74]. DDG, düğümlerinin uygulamadaki talimatları ve kenarların uygulamadaki talimatlar arasındaki veri bağımlılığı temsil edecek şekilde yönlendirilmiş bir graftır. Veri bağımlılığı, veri akış analizi ile elde edilmekte; böylece, n_2 ve n_1 iki düğüm ise, n_2, n_1 tarafından tanımlanan bir değişken kullanıyorsa n_1 ile n_2 arasında bir bağlantı çizilmektedir ($n_1 \rightarrow n_2$). Bu tür özellikleri önceki bazı çalışmalarda kullanılmıştır, örneğin, Elish ve arkadaşları, veri akış analiz yöntemi, kullanıcı girdilerine ve API çağrılarına dayanarak bir veri bağımlılığı grafı oluşturmak için kullanmışlardır [75]. Ayrıca, Intent tabanlı uygulamalar arası veya bileşenler arasındaki olayları ortaya çıkarmak için akış kontrol analiz yöntemi kullanılmıştır.

- **Bozukluk Akışı (Taint Flow)**

Bu tür analizde, bir API tarafından üretilen hassas verileri, kaynaktan hedefe kadar izlenmektedir. Veriyi üreten API'ye kaynak adı verilirken, üretilen verileri, ağa, dosyaya veya başka bir hedefe gönderen API'ye havuz adı verilmektedir. Kaynak-havuz veri akış dizeleri, uygulamalarda hassas verilerin yayılmasını temsil etmek için oluşturulmaktadır. Dolayısıyla, bu dizeleri, uygulama davranışını tanımlamak için bir model olarak kullanılabilmektedir. Bu tür özellikleri, daha önceki bazı çalışmalarda kullanılmıştır. Örneğin, Brown ve arkadaşları, kaynak-havuz API takibine dayalı bilgi sızıntısı tespit eden bir çerçeveyi önermişdir [76]. Ayrıca, Junaid ve arkadaşları, uygulamaların potansiyel kötü niyetli davranışlarını tespit etmek için, hassas bilgileri kaynak yöntemden havuz yönteme kadar izlenmesine dayanan akış analiz tabanlı bir çerçeveyi önermişlerdir [77].

D. Uygulama Meta Veriye Dayalı Özellikleri

Uygulamanın açıklamasından çıkarılan özellikleri veya uygulamaya eklenen herhangi bir bilgiyi, meta veri özellikleri olarak sınıflandırılmıştır, bu tür özellikleri aşağıdaki gibi özetlenbilemektedir:

- **Uygulamadaki Sertifika Bilgileri**

Bu tür özellikleri, uygulamadaki imzası, sertifikası ve uygulamayı imzalamak için kullanılan anahtarı içeren META-INF klasörünün içeriğine dayanan herhangi özelliği içermektedir. Bu tarz bilgileri, uygulamalarındaki geliştiricileri karşılaştırmak için bazı çalışmalarda, kullanılmıştır. Örneğin, Gurulian ve arkadaşlarındaki çalışmasında, geliştirici imzası, bazı diğer açıklayıcı bilgilerle birlikte, uygulama yeniden paketlenmesi tespit etmek için kullanılmıştır [36].

- **Play Store Tanımlayıcı Bilgileri**

Bu tür özellikler, fiyat, indirme sayısı ve kullanıcı derecelendirmesi gibi uygulama mağazasından çıkarılabilen tüm uygulamaya özgü değerlendirme bilgileri

içermektedir. Örneğin, Sanz ve arkadaşlarındaki çalışmasında, program derecelendirme bilgileri bazı diğer açıklayıcı bilgilerle birlikte, Android uygulamaları kategorize etmek için kullanılmıştır [78].

2) Dinamik Özellikleri

Bu tür özellikler, sistem çağrıları, ağ etkinliği, dosya sistem kullanımı gibi uygulama yürütme sırasında toplanabilecek tüm özellikleri içermektedir. Bu özelliklerinden en önemlileri ayrıntılı olarak aşağıdaki bölümlerde açıklanacaktır:

A. Sistem Çağrıları

Bir uygulamanın bazı görevleri yerine getirmek için özel sistem çağrıları kullanarak işletim sisteme bağlanması gerektiğinden en çok kullanılan dinamik özelliklerden biridir. Sistem çağrısı, uygulamadaki davranışı analiz etmede kullanılabilecek bir günlük dosyada saklanıp izlenebilmektedir. Daha önceki bazı çalışmalarda, uygulamaların davranışını anlamsal bir şekilde yansıtan kalıpları oluşturmak için sistem çağrı dizisi veya sistem çağrılar frekansını kullanılmıştır. Örneğin, Tan ve arkadaşlarındaki çalışmasında, hassas API dizisi ve kullanılan API çağrı sayısı, uygulama yürütülmesi sırasında çıkarılmış, ardından android uygulamaları, kötü amaçlı veya zararsız olarak sınıflandırmak için geliştirilmiş bir Naive Bayes sınıflandırma modeli kullanılmıştır [79]. Diğer bazı çalışmalarda, sistem çağrısı yerine sistem hizmet çağrı dizileri kullanılmıştır, çünkü, sistem çağrıları, parametre bilgileri kaçıran ve tam uygulama davranışı açığa çıkartamayan fonksiyon adından oluşmaktadır. Örneğin, Wang ve arkadaşları, sistem hizmet çağrılarının birlikte oluşma matrisine dayalı olarak android kötü amaçlı yazılımları tespit etmek için dinamik bir analiz çerçeveyi önermişlerdir [80]. İlk olarak, sistem hizmet çağrı dizisinin birlikte oluşum matrisini elde etmek için çalışan Android uygulamalarının hizmet ara yüzü çağrı bilgileri çıkarılmıştır. Elde edilen matris, birden çok makine öğrenme sınıflandırıcısı eğitmek üzere kullanılan özellik vektörleri oluşturmak için normalleştirilmiştir.

B. Ağ Davranışı

Genel olarak, tüm kötü amaçlı uygulamalar, toplanan verileri göndermek, uzak sunucusundan komut almak veya herhangi bir etkinliği gerçekleştirmek için ağa bağlanmaktadır. Bu nedenle, uygulama yürütülme sırasında üretilen ağ trafiği, uygulamanın davranışına iyi bir gösterge sağlamaktadır, dolayısıyla, bu özellik, önceki birçok çalışmada kullanılmıştır. Örneğin, Chang ve arkadaşlarındaki çalışmada [81] ve Wei ve arkadaşlarındaki çalışmada [43], kötü amaçlı yazılım ile iyi huylu yazılım arasında ayrım yapmak için ağ etkinlikleri, bazı diğer dinamik özellikleri ile birlikte kullanılmıştır.

C. Kaynak Tüketimi

Genellikle, cep telefonları, pil, işlemci ve bellek gibi kaynaklar açısından sınırlıdır. Bu nedenle, kaynak tüketimi, önceki çalışmaların bazılarında kötü amaçlı uygulamaları tespit etmek için bir özellik olarak kullanılmıştır. Genel olarak, kötü niyetli uygulamalar, iyi huylu uygulamalardan daha fazla kaynak tükettiği için bu tür çalışmalar, iyi huylu ve kötü niyetli uygulamalar arasındaki kaynak tüketim farkının analiz edilmesine dayanmaktadır. Bu kaynak tüketim farkı, kötücül uygulamaların çoğunun arka planda bir görev gerçekleştirdiğinden veya CPU, bellek, Bluetooth ve kablosuz cihazlar gibi donanım kaynaklarına eriştiğinden doalyı oluşmaktadır [44]. Ayrıca, Amos ve arkadaşları, birçok makine öğrenme sınıflandırıcısı eğitmek ve test etmek için bu tür özellikleri, bazı diğer özelliklerle birlikte kullanmışlardır [82].

3) Melez Tespit Yaklaşım Tabanlı Özellikleri

Adından da anlaşılacağı gibi, bu tür özellikleri, daha derin bir analiz elde etmek ve uygulamaların potansiyel kötü niyetli davranışlarını tespit etmek için statik özelliklerin ve dinamik özelliklerin bir kombinasyonunu içermektedir.

4) Görüntü Tabanlı Özellikleri:

Bu tür bir özellikleri ister gri tonlamalı ister RGB olsun görüntülerden ayıklanan tüm özellikleri içermektedir. Daha önce bahsedildiği gibi, kötü amaçlı yazılım kaynağının bir görüntüye dönüştürülmesine dayanan analiz yöntemleri, önceki çalışmada sınırlı bir şekilde kullanılmıştır. Bu tür çalışmalarda izlenen iki eğilim vardır, birinci eğilimde, uygulamaları sınıflandırılmak için uygulamalardaki kaynağı görüntülere dönüştürüp, görüntü tabanlı özellikleri ayıklanmaktadır, ardından, geleneksel makine öğrenme algoritmaları, ayıklanan görüntü tabanlı özelliklerine dayanarak eğitilmektedir. İkinci eğilimde, android uygulamaları, görüntü dosyalarına dönüştürülmektedir, oluşturulan görüntü dosyaları, derin öğrenme modellerine girdi olarak kullanılmaktadır. Örneğin, Yang ve arkadaşları, GIST özelliği görüntülerden ayıklayıp Rastgele karar ormanlar sınıflandırıcısı eğitmek için kullanmışlardır [32]. Ayrıca, Kumar ve arkadaşları, APK dosyalarının çeşitli görüntü formatları (Gri Tonlama, RGB, CMYK ve HSL) olarak görselleştirilmesine dayanan bir yöntem geliştirmişlerdir [83]. Bundan sonra, karar ağacı [84], Rastgele Ormanlar (RF) ve K-en yakın komşu (KNN) olmak üzere birden fazla makine öğrenme algoritması eğitmek ve test etmek için her görüntüden GIST özelliği ayıklanmıştır. Karimi ve arkadaşlarındaki çalışmasında [33], görüntüdeki her bir piksel için ağırlıklar verilmiş, en iyi piksel dizisi seçilmiş ve kötü niyetli uygulamaları tespit etmek için imza olarak kullanılmıştır. Geri kalan çalışmalarda, derin öğrenme teknikleri kullanılmıştır, örneğin, Yang ve arkaşındaki çalışmada [31] ve Yen ve arkaşındaki çalışmada [34], oluşturulan görüntü veri seti bir evrişimli sinir ağı eğitmek için kullanılmıştır.

III. Algılama Aşaması

Özellik çıkarma aşamasından sonra, uygulamaların kötü niyetli davranışlarını tespit etmemek için kullanılabilecek desenleri, çıkartılan özelliklere dayanarak oluşturulmaktadır. İncelenen çalışmaların çoğunda, bu desenler, ikili vektörler olarak temsil edilmiştir; böylece 0, özelliğin kullanılmadığı durumu temsil ederken, (1) özelliğin kullanıldığı durumu temsil etmektedir. Tespit yöntemine göre etiketli veya etiketsiz veri seti kullanılabilmektedir. Genel olarak bu çalışmada ele alınan

araştırmalarda imza tabanlı ve makine öğrenme teknikler tabanlı olmak üzere iki algılama yaklaşımı kullanılmıştır:

1) İmza Tabanlı Yaklaşım

Bu yöntem, en yaygın geleneksel kötü amaçlı yazılım algılama yöntemlerinden biridir. Genel olarak, bu yöntemde, her uygulama için bir desen veya belirli bir kötü amaçlı yazılım aile davranışını tanımlayan bir desen dizisi oluşturularak bir imza veri tabanı oluşturulmaktadır. Bundan sonra, herhangi bir uygulamanın davranışını incelemek için, uygulamadaki deseni çıkartılıp imza veri tabanında depolanan desenlerle karşılaştırılmaktadır. Uygulama deseni, herhangi bir kötü niyetli imza ile eşleştirilmesi durumunda, uygulamanın kötü niyetli bir davranış içerdiğine karar verilmektedir. Bu yöntem, bilgisayarları hedefleyen kötü amaçlı uygulamaları tespit etmek için yaygın olarak kullanılmıştır, örneğin Bakour ve arkadaşlarındaki çalışmasında, bilgisayarları hedefleyen kötü amaçlı yazılımları tespit etmek üzere bir imza veri tabanı oluşturmak için genetik algoritmasına ve Tabu arama algoritmasına dayanan melez bir model önerilmiştir [85]. Ayrıca, bu yöntem Android kötü amaçlı yazılım algılama alanında kullanılmıştır, örneğin, Shen ve arkadaşları, Android uygulamalarının gerçek davranışını yansıtmak için kaynak koddaki API'ler ve sınıflar temelinde oluşturulan topoloji grafına bağlı olarak desen veri tabanı oluşturmuşlardır [86]. Ardından, topoloji grafları çıkarılıp imza veri tabanı ile eşleştirilerek Android uygulamaları analiz edilmiştir, böylece incelenen uygulama, veri tabanındaki imzalarından herhangi birine monomorfik olan bir alt graf içeriyorsa, her düğümde kullanılan API seti karşılaştırılmıştır. API benzerliği belirli bir eşiğe ulaştığı ise, uygulama kötü amaçlı olarak değerlendirilmiştir. Ayrıca, Faruki ve arkadaşları, yeniden paketlenmiş kötü amaçlı yazılım çeşitleri tespit etmek için istatistiksel özelliklere dayalı bir imza yaklaşımı önermişlerdir [87]. Önerilen yöntemde, değişken uzunlukta imza veri tabanı oluşturmak için benzerlik özet karma şeması (SDHash şeması) kullanılarak oluşturulan istatistiksel özellikleri kullanılmıştır. Grace ve arkadaşlarındaki çalışmasında [88], android uygulamaları, yüksek riskli, orta riskli ve düşük riskli olarak sıralamak için kod yolu tabanlı bir imza veri tabanı oluşturulmuştur. Crussell ve arkadaşları, uygulamalar arasındaki kod benzerliği tespit etmek için anlamsal kod tabanlı bir imza veri tabanı oluşturmak için

Program Bağımlılık Graf (PDG) kullanmışlardır [89]. Ayrıca, Alam ve arkadaşları, uygulama yerel kodundaki şüpheli davranışları tespit etmek için Açıklamalı Kontrol Akışı Grafiğine (ACFG) dayalı olarak semantik tabanlı bir imza veri tabanı önermişlerdir [73]. Yang ve arkadaşlarındaki çalışmasında, kötü amaçlı uygulamaları, karakterize etmek için hem uygulama açıklaması hem de hassas veri akış bilgileri dikkate alan bir yöntem önerilmiştir [90]. Kötü amaçlı yazılımın karakterize edilmesini iyileştirmek için önerilen yöntem, uygulama kategorine özgü hassas veri akış imzalarının madenciliğine dayanmaktadır. Uygulama kategorine özgü imzalar, belirli bir kategorideki uygulamalarda görülen her veri akış desen için bilgi kazanım oranı (Information Gain Ratio) hesaplanarak oluşturulmuştur.

İmza tabanlı tespit yaklaşımı, yalnızca bilinen kötü amaçlı yazılım türlerini tespit etme yeteneği ve bilinmeyen kötü amaçlı yazılımları, çok biçimli kötü amaçlı yazılımları (Polymorphic Malware) veya sıfırıncı gün saldırılarını tespit edememe gibi bazı zayıflıklara maruz kalmaktadır.

2) Makine Öğrenme Tekniklerine Dayalı Yaklaşım

Makine öğrenme ve veri madenciliği algoritmaları, kötü amaçlı uygulamaları tespit etme alanına dahil edilmiş ve bu algoritmaların etkinliğini kanıtlanmıştır. Bu çalışmada incelenen eserlerin çoğu denetimli öğrenme algoritmaları kullanmış olup küçük bir kısmı k-ortalama algoritması gibi bazı denetimsiz kümeleme algoritmaları kullanılmıştır. Uzaydaki kısıtlamalar nedeniyle, incelenen eserlerde kullanılan algoritmalardan bazıları kısaca listelenecektir.

A. Kullanılan Sınıflandırma Algoritmaları

Sınıflandırma algoritmaları, denetimli öğrenmeye dayanmaktadır, böylece, veri setinden bir kısmı eğitim için kullanılırken, diğer kısmı test için kullanılmaktadır. Ayrıca, bu tip algoritmalarda, eğitim veri seti etiketli olması gerekmektedir. İncelenen eserlerde çeşitli makine öğrenme algoritmaları kullanıldığı görülmüştür, örneğin, Aafer ve arkadaşları, Android uygulamalarının kötü niyetli davranışını tespit edebilen bir sınıflandırıcı oluşturmak için genel bir veri madenciliği yaklaşımı

önermişlerdir [63]. Karar Ağacı, K-En Yakın Komşular ve doğrusal SVM, uygulamalardaki API sıklığına bağlı olarak kötü amaçlı yazılım ve zararsız yazılım arasında ayrım yapacak şekilde kullanılmıştır. Arp ve arkadaşları, çok sayıda statik özelliği ayıklanıp iyi huylu ve kötü niyetli davranışları ayırt etmek için doğrusal destek vektör makine (SVM) sınıflandırıcısı kullanmışlardır [54]. Feldman ve arkadaşları, Manilyzer adlı bir statik analiz aracı önermişlerdir [84]. Manilyzer, manifest dosya bilgilerine ve makine öğrenme tekniklerine dayanmaktadır. Naive Bayes, Destek Vektör Makinesi (SVM), K-En Yakın Komşular (KNN) ve karar ağacı algoritmaları, kötü niyetli ve zararsız uygulamaları ayırt etmek için benimsenmiştir. Sheen ve arkadaşları, Android kötü amaçlı yazılımları, tespit etmek için toplu sınıflandırıcı tabanlı bir yöntem sunmuşlardır [91]. Karar ağacı ve Rastgele ağaçlar gibi bazı zayıf sınıflandırıcıları güçlendirmek için AdaBoost ve Bagging gibi son teknoloji topluluk yöntemleri kullanılmıştır. Fereidooni ve arkadaşları, ANASTASIA adlı makine öğrenme tekniklerine dayalı bir statik analiz çerçeveyi önermişlerdir [92]. Bu amaçla, Android uygulamalarından olabildiğince çok bilgilendirici özellikleri ayıklamak için uniPDroid adlı ve Androguard tabanlı bir araç uygulanmıştır. Bundan sonra, bir Android uygulamaları, kötü amaçlı veya iyi huylu olarak sınıflandırmak için AdaBoost, Random Forest, SVM, K-NN, Logistic Regression, Naive Bayes, Decision Tree gibi çeşitli makine öğrenme algoritmaları kullanılmıştır. Ma ve arkadaşları, kötü amaçlı yazılım tespiti optimize etmek için (2) seviyeli makine öğrenme tekniğini, statik analiz teknikleriyle birleştiren bir Android kötü amaçlı yazılım algılama yöntemi önermişlerdir [93]. Du ve arkadaşları, kontrol akış graf topluluk yapısına dayanan bir Android kötü amaçlı yazılım algılama yöntemi önermişlerdir [94]. Önerilen yöntem, karar ağacı, SVM, Naive Bayes ve BayesNet olmak üzere bazı makine öğrenme sınıflandırıcıları, kontrol akış graf topluluk yapısından ayıklanan özellikleri kullanılarak test edilmiştir. Ayrıca, Kirubavathi ve arkadaşı, Botnet sınıflandırmak için Naive Bayesian, destek vektör makinesi (SVM) ve azaltılmış hata budama ağacı (REPTree) kullanmışlardır [40]. Milosevic ve arkadaşlarındaki çalışmasında [66], Android'daki iyi huylu ve kötü amaçlı yazılım uygulamaları arasında ayrım yapmak için SVM, Naive Bayes, Karar ağaçları, AdaBoost ve Basit K-ortalama algoritması gibi birçok sınıflandırma ve kümeleme algoritmaları kullanılmıştır. Chen ve arkadaşlarındaki çalışmasında [38] ve Sen arkadaşlarındaki çalışmasında [65], SVM, Random Forest (RF) ve K-Nearest

Neighbor (KNN) dahil olmak üzere çeşitli makine öğrenme algoritmaları kullanılmıştır.

B. Kümeleme Algoritmaları

Bu tür algoritmalar, denetimsiz öğrenmeye dayanmaktadır. Kümeleme algoritmaları, veri örnekler arasındaki benzerlik miktarına bağlı olarak kümelere ayırmak için kullanılmaktadır. Bu nedenle, Öklid mesafesi veya kosinüs mesafesi gibi mesafe ölçüm yöntemleri, veri örnekler arasındaki benzerliği ölçmek için bu tür algoritmalarda kullanılabilmektedir. Ele alınan eserlerin çoğunda K-ortalamaları algoritmasının kullanıldığını gözlemlenmiştir. Bu algoritma, yinelemeli bir şekilde çalışarak ve kümelerin yeni merkezleri, veri konumlarına göre yeniden hesaplayarak her veri örneği K kümelerinden birine atamayı amaçlamaktadır. Bu algoritma bazı çalışmalarda kullanılmıştır, örneğin, Wu ve arkadaşları, birden çok statik özelliği ayıklayıp k-ortalama algoritması, uygulamaları birden çok kümeye bölmek için kullanmışlardır [61]. Daha sonra uygulamaları iyi huylu veya kötü niyetli olarak sınıflandırmak için KNN algoritması kullanılmıştır. Ayrıca, Aung ve arkadaşlarındaki çalışmasında [95], ayıklanan özellikleri birden çok kümeye kümelemek için k-ortalama algoritması, ilk aşama olarak kullanılmıştır. Bundan sonra, Android uygulamaları, birçok sınıfa sınıflandırmak için makine öğrenme sınıflandırma algoritmaları kullanılmıştır. Ayrıca, Verma ve arkadaşlarındaki çalışmasında [96], kümeleme aşaması olarak k-ortalama algoritmasına, ardından bir sınıflandırma aşaması olarak J48 ve ID 3 sınıflandırıcılarına dayanan melez bir sınıflandırma yöntemi önerilmiştir.

C. Derin Öğrenme Teknikleri

Derin öğrenme modelleri, giriş katmanı, çıktı katmanı ve çoklu gizli katmanları olmak üzere çok sayıda katmana dağılmış çok sayıda nörondan oluşmaktadır. Derin öğrenme teknikleri, geleneksel makine tekniklerde analizör tarafından özellikleri ayıklamak yerine özellikleri otomatik olarak ayıklama becerisiyle geleneksel makine öğrenme tekniklerinden daha iyi performans göstermektedir. İncelenen eserlerin küçük bir kısmında derin öğrenme teknikleri kullanılmıştır. Örneğin, Karbab ve

arkadaşları, Android uygulama veri kümesinden ham API yöntem çağrıları çıkartmışlardır [97]. Daha sonra, her uygulama için bir anlamsal vektörü oluşturulmuştur. Son olarak, inşa edilen vektörleri, uygulama sınıflandırmasında kullanılan çok katmanlı bir sinir ağı eğitmek için kullanılmıştır. Hou ve arkadaşları, uygulamalardaki kötü niyetli davranışları tespit etmek için dinamik bir analiz çerçeveyi geliştirmişlerdir [98]. Önerilen yöntem, Android uygulamaları, kötü niyetli veya iyi huylu olarak sınıflandırmak için Yığınlanmış Otomatik Kodlayıcılar (Stacked AutoEncoders SAEs) modeli içeren bir derin öğrenme mimarisine dayanmaktadır. Yuan ve arkadaşları, Android uygulamaları karakterize etmek için Deep Belief Networks (DBN) tabanlı bir derin öğrenme modeli önermişlerdir [52]. Huang ve arkadaşı tarafından gerçekleştirilen çalışmada [31], Android uygulamanın classes.dex dosyası RGB görüntüsüne dönüştürülmüş ve oluşturulan görüntüleri, uygulamaları kötü niyetli ve iyi huylu olarak sınıflandırmak için bir evrişimli sinir ağına girdi olarak uygulanmıştır. Yen ve arkadaşı tarafından gerçekleştirilen çalışmada [34], uygulamaların kaynak kodlarında bulunan önemli terimlerin toplam koddaki önemini hesaplamak ve sayısallaştırmak için uygulamalardaki kaynak kodları ayrıştırılmıştır. Sayısallaştırılmış değerleri görüntüye dönüştürülmüş ve oluşturulan görüntüleri, sınıflandırıcı olarak benimsenen evrişimli sinir ağına girdi olarak uygulanmıştır. Ayrıca, Junaid ve arkadaşları, Android kötü amaçlı yazılımları karakterize etmek için DroidDeepLearner adlı bir statik analiz çerçeveyi geliştirmişlerdir [77]. Birden çok statik özellikleri ayıklanmış ve kötü amaçlı yazılım ile zararlı yazılımı ayırt etmek için kullanılan derin öğrenme modeli eğitmek için kullanılmıştır. Ayrıca, Wang ve arkadaşlarındaki çalışmasında [39], kötü amaçlı yazılım algılama doğruluğu artırmak için yüksek boyutlu özellik vektörleri oluşturulmuş ve Android kötü amaçlı yazılımları algılamak için çoklu evrişimli sinir ağı (ESA) modelleri benimsenmiştir. Seyrekliği artırmak için doğrusal olmayan bir aktivasyon fonksiyonu ve aşırı uyumu önlemek için bırakma tekniği kullanan bir seri evrişimli sinir ağı mimarisi (ESA-S) kullanılmıştır. Son olarak, eğitim süresini kısaltmak için ESA'in eğitim öncesi yöntemi olarak derin AutoEncoder (DAE) kullanılmıştır. Ayrıca, Nix ve Zhang tarafından bir Evrişimli Sinir Ağı (ESA) ve Uzun Kısa Süreli Hafızalı (LSTM) Tekrarlayan Sinir Ağı (RNN) sınıflandırıcı olarak kullanılmıştır [99].

2.1.1.2. Kullanılan Veri Kümesine Dayalı Taksonomi

Önerilen kötü amaçlı yazılım tespit sistemleri eğitmek ve test etmek için kullanılan veri kümesi, tespit sistemlerin gücünü değerlendirmek için en önemli kriterlerinden biridir. Bu nedenle, incelenen eserleri, kullanılan değerlendirme veri kümesine göre sınıflandırmayı önerilmiştir.

I. İyi Huylu Veri Kümesi

Genel olarak, [37, 41, 44, 51, 56, 59, 73-77, 100] gibi incelenen eserlerin çoğunda iyi huylu veri seti resmi Android uygulama pazarından (Google play) toplanmıştır. Ayrıca, bazı diğer çalışmalarda, iyi huylu veri seti üçüncü taraf pazarlarından toplanmıştır [39, 101-103]. Onun dışında, bazı çalışmalarda hem resmi pazarlarından ve hem de üçüncü pazarlarından indirilen uygulamaları içeren iyi huylu bir veri kümesi kullanılmıştır [40, 104, 105].

II. Kötü Amaçlı Veri Kümesi

Malgenome [106] gibi iyi bilinen kötü amaçlı yazılım veri kümelerin, incelenen eserlerin çoğunda kullanıldığını gözlemlenmiştir. İyi bilinen kötü amaçlı veri setleri [36, 38, 42, 77] gibi birçok çalışmada kullanılmıştır. Ayrıca, Zhu ve arkadaşlarındaki çalışması gibi bazı çalışmalarda internetten toplanan özel veri setleri kullanılmıştır [37]. Ayrıca, iyi bilinen kötü amaçlı veri kümelerden ve internetten toplanan bazı kötü amaçlı yazılım örneklerden bir kombinasyonu, bazı diğer çalışmalarda kullanılmıştır [38, 51].

2.1.1.3. Zorlukların Karşı Önlemine Dayalı Taksonomi

Önceki çalışmalarda yaygın olarak kullanılan tüm analiz yöntemlerinin (yani statik analiz, dinamik analiz ve hibrit analiz) bazı zorlukları ve zayıflıkları vardır. Bu nedenle, önceki çalışmaları, bu zorluklara karşı alınan önlemlere dayanarak sınıflandırmayı önerilmiştir.

I. Statik Analiz Zorlukları

Statik analiz, Android kötü amaçlı yazılımlarını düşük hesaplama karmaşıklığı ve oldukça yüksek performansla hızlı bir şekilde tespit edebilen hafif bir algılama yöntemi olmasına rağmen, yine de gizleme teknikleri ve kod dinamik yüklenmesi gibi bazı sorunlarla ve zorluklarla karşı karşıya kalmaktadır. Bu sorunların en önemlileri aşağıdaki bölümlerde tartışılmıştır.

1) Gizleme Teknikleri

Bu teknikler, uygulama tersine mühendislikten korumak gibi birçok nedenden dolayı geliştiricilere önerilse de bu teknik, statik analiz yöntemlerinin üstesinden gelmek ve uygulamalardaki kötü niyetli davranışları gizlemek için kötü amaçlı yazılım geliştiriciler tarafından kullanılan en önemli yöntemlerden biridir. Kod gizleme teknikler, kötü amaçlı uygulama davranışını değiştirmeden uygulamadaki boyutu ve içeriği değiştirmektedir [131]. Bu amaca ulaşmak için kullanılan birçok teknik vardır, en önemlileri aşağıda tartışılacaktır:

İsim Gizleme: En basit gizleme yöntemlerinden biridir. Bu yöntemde, kötü amaçlı uygulamanın paket adı veya kötü amaçlı yazılımın kodundaki diğer bazı terimleri (sınıf adları veya yöntem adları gibi) bazı analiz yöntemleri atlayacak şekilde değiştirilmektedir.

Kontrol Akış Gizlenmesi: Daha önce belirtildiği gibi, kontrol akışı ve veri akışı, herhangi bir şüpheli davranışı tespit etmek için statik analiz yöntemlerinde yaygın olarak kullanılmıştır [72], [73, 107]. Uygulama kontrol akışına dayanan analiz algılama yöntemleri atlamak için kötü amaçlı uygulama çağrı grafı değiştirilebilmektedir. Goto-gizleme, en popüler kontrol akış gizleme yöntemlerinden biridir; burada kod içinde atlamalar yapmak ve orijinal talimat dizisi değiştirmek için Goto komutu kullanılmaktadır. Ayrıca, daha da derin bir gizleme elde etmek için gereksiz yöntemleri eklenip kod içinde çağrılabilmektedir [108, 109].

Dizeler şifreleme: Daha önce da belirtildiği gibi, kötü amaçlı yazılımın C&C sunucusuna bağlaması veya kurban cihazından Premium mesajları göndermesi için kullanılabilen ve kaynak kodda bulunan IP adresleri, alan adları veya Premium numaraları gibi terimleri birçok statik analize dayalı çalışmalarda bir özellik olarak kullanılmıştır. Bu nedenle, kötü amaçlı yazılım geliştiriciler, bu algılama yöntemleri atlamak için, uygulama yürütülmesi sırasında işlendiğinde çözülecek şekilde bu tip terimleri şifrelemektedir [108].

Sınıf Şifreleme: Tüm sınıfın kodlandığı, sıkıştırıldığı ve bir veri dizisinde depolandığı gelişmiş bir gizleme yöntemidir. Daha sonra, sınıf şifresi çözmek ve yürütme sırasında yüklemek için bir yöntem geliştirilmekte, böylece gizlenmiş sınıfın şifresi uygulama yürütme sırasında çözülüp belleğe yüklenmektedir [110]. Bu teknik, en karmaşık gizleme teknik olmasına rağmen statik analiz tekniklerini önleyen en etkili yollarından biridir.

Yansıma: Bu teknikte, sınıflar ve yöntemleri, derleme zamanında açık ismine gerek kalmaksızın çalışma zamanında çağrılabilmektedir. Örneğin, şekil 2.3'te gösterildiği gibi, sınıf çağrısı veya sınıfın yeni örnek oluşturma işlemi, uygulamanın analiz edilmesini zorlaştırmak için başka yöntemleri kullanılarak gerçekleştirilebilmektedir.

Önemsiz kod ekleme: Bu teknikte, sınıflar koduna ve yöntemler koduna önemsiz kod enjekte edilmektedir, böylece enjekte edilen önemsiz kod, uygulamanın yürütülmesini etkilemeden (uygulamadaki işlevi koruyarak) yürütülebilmektedir. Bu koda ölü kod veya işlemsiz kod olarak bilinmektedir [111]. Bu yöntem, kod akış tabanlı analiz yöntemleri atlatmak için kullanılan ünlü gizleme yöntemlerinden biridir.

```
Orijinal kod:
System.out.println("Hello World.");
Gizlenmiş kod:
Class c = Class.forName ("java.io.PrintStream");
Method m = c.getMethod("println", new Class[] { String.class});
m.invoke(null, new Object[] { "Hello World. " });
```

Şekil 2.3.Sınıf çağırma yansıma örneği.

Yukardaki gizleme teknikleri, daha önce yapılan eserlerin çok sınırlı sayıda ele alınmıştır. Örneğin, Yerima ve arkadaşlarındaki çalışmasında [112], kod gizleme ve diğer anti-analiz tekniklerine karşı sağlamlık ve esneklik sağlamak için rastgele orman sınıflandırıcısı, kapsamlı bir melez özellik seti kullanılarak eğitilmiştir. Ayrıca, Shen ve arkadaşları, çeşitli gizleme teknikleri kullanılarak üretilen kötü amaçlı yazılım varyantları tespit etmeyi amaçlayan bir imza veri seti oluşturmak için Android bileşenleri tabanlı bir topolojik grafı kullanmışlardır [86]. Ayrıca, Faruki ve arkadaşları, güçlü istatistiksel özellikleri tabanlı imzalara dayalı olarak Android uygulamalarında yeniden paketleme ve kod gizlemeyi tespit etmek için DroidOLytics adlı bir çerçeveyi önermişlerdir [87]. Ayrıca, Grace ve arkadaşları, gizleme ve dinamik yük yükleme tekniklerinin kullanımını tespit edebilen RiskRanker adlı bir çerçeveyi önermişlerdir [88].

2) Yerel Kod Yürütme (Native Code)

Android sistemi, JNI ara yüzü kullanılarak erişilen yerel kitaplıkları olarak uygulamadaki kaynak kodundan bir bölümü yazma olanağı sağlamaktadır. Bu özellik, kötü amaçlı yazılım geliştirici tarafından, kötü amaçlı uygulamaların analizini daha zor hale getirmek için yerel kitaplıkları kullanarak kaynak kodundaki kötü amaçlı kısmı yazmak için kullanılmıştır. DroidDream, bu yöntemi kullanan kötü amaçlı uygulamaların bir örnektir, bu kötü amaçlı yazılımda kötü amaçlı içerik yerel kodu kullanılarak yazılmış ve standart olmayan bir konuma yerleştirilmiştir. Ayrıca, DroidKungFu kötü amaçlı uygulamalar ailesinde, kötü amaçlı içerik yerel kod olarak yazılmış, şifrelenmiş ve uygulama koduna yerleştirilmiştir [113]. Genel olarak, yerel

kod analiz, bayt kodu analizinden çok daha zor olduğunu kabul edilmektedir, bu nedenle önceki çalışmaların az bir kısmında, yerel kod analizine dayanmıştır. Örneğin, Lantz ve Johansson, Android API'ler çağırmasında yerel kodun kullanımını analiz etmek için bir metodoloji önermişlerdir. Spreitzenbarth ve arkadaşlarındaki çalışmasında [115], geliştirilen Mobil-Sandbox melez analiz çerçevesinin geliştirilmiş bir kopyası [102], orijinal çerçeveye makine öğrenme teknikleri eklenerek uygulanmıştır. Statik analiz aşamasında, uygulama derlenmiş ve şüpheli görünen izinleri ve Intentleri toplamak için Manifest dosyası ayrıştırılmıştır. Dinamik analiz aşamasında, uygulama, Yerel API çağrıları da dahil olmak üzere gerçekleştirilen tüm eylemleri günlüğe kaydetmek için bir öykünücüde yürütülmüştür. Glodek ve Harang tarafından gerçekleştirilen çalışmada [116], uygulamadaki kötü niyetli davranışını tespit etmek için rastgele karar ormanlar sınıflandırıcısı, istenilen izinleri, yayın alıcıları ve yerel kodu varlığı kullanılarak eğitilmiştir.

3) Dinamik Yük

Android sistem, uygulamaların DEX, JAR, SO ve ELF dosyalarını yüklemesine ve bu dosyaları yürütme zamanında yürütmesine izin vermektedir. Bu özellik kötü amaçlı yazılım geliştiriciler tarafından kötü amaçlı içeriği şifrelenmiş bir Dex, Jar veya yerel kod dosyalarına yerleştirmek için kullanmıştır, böylece kötü amaçlı içeriği, uygulama yürütme sırasında çağrılıp şifresi çözülerek çalıştırılmaktadır. Ayrıca, bazı durumlarda, kötü amaçlı içeriği uygulamaya gömülmemekte, ancak yürütme sırasında uzak bir sunucudan indirilmektedir, bu da kötü amaçlı yazılımın statik analiz yöntemleri kullanılarak tespit edilmesini imkânsız haline getirmektedir. Dinamik yük, önceki bazı çalışmalarda ele alınmıştır, örneğin, Zhou ve arkadaşları, sistem çağrıları toplayarak dinamik olarak yüklenmiş yerel kodu tespit etmek için bir karma analiz çerçeveyi önermişlerdir [117]. Zheng ve arkadaşları, dinamik yük davranışı keşfetmeye odaklanan DroidTrace adlı dinamik bir analiz sistemi önermişlerdir [118]. Ptrace aracı, sistem çağrıları izlemek için kullanılmış ve ileri yürütme, hedef süreçte farklı dinamik yükleme davranışları tetiklemek için kullanılmıştır. Önerilen çerçevenin hem Java hem de yerel kod düzeyinde tüm dinamik yük davranışlarını izleyebileceği, tüm Android sürümleri için gerçek

cihazlarda yürütülebileceği ve dosya işlemi, ağ bağlantısı, süreç içindeki iletişimi ve ayrıcalık artırma olmak üzere dört tür davranışı izleyebileceği belirtilmiştir. Ayrıca, Spreitzenbarth ve arkadaşları, yeniden paketleme, kod gizleme veya dinamik yükleri kullanan kötü amaçlı yazılımlara karşı mücadele eden ve DroidAnalytics olarak adlandırılan bir çerçeveyi geliştirmişlerdir [115].

4) Uygulama Yeniden Paketleme

Bu yöntem, kötü amaçlı uygulamaları geliştirmek için kullanılan en yaygın yöntemlerden biridir. Basitçe, uygulama yeniden paketleme, popüler uygulamalardan birini yeniden derlenip kötü amaçlı bir içeriği kaynak koduna eklemeye dayanmaktadır. Bundan sonra, uygulama tekrar derlenip yeni bir imza ile imzalanıp ve resmi veya üçüncü taraf uygulama mağazalarında yeniden yayınlanmaktadır. Bu tür teknikler, incelenen bazı çalışmalarda ele alınmıştır, örneğin, Gurulian ve arkadaşları, uygulama yeniden paketlenmeyi tespit etmek için bir yöntem önermişlerdir [36]. Önerilen yöntem, orijinal uygulamanın popülerliğinden yararlanmak için saldırganın, uygulama adı ve simgesi gibi bazı orijinal uygulama verileri değiştirmemesine bağlıdır. Hu ve arkadaşları, yeniden paketleme tabanlı kötü amaçlı yazılımları tespit etmek için MIGDroid adlı bir statik analiz yaklaşımı sunmuşlardır [119]. İlk olarak, API çağrı dizileri çıkarılmıştır, ardından yöntem çağırma grafı oluşturulmuştur. Bundan sonra, oluşturulan graf alt graflara ayrılmış ve alt grafın tehdit derecesi, bu alt grafların her birinde kullanılan hassas API'lere göre hesaplanmıştır. Sonunda, önceden tanımlanmış eşik değeri aşan alt grafları zararlı olarak etiketlenmiştir. Ayrıca, Zhou ve arkadaşları, üçüncü taraf pazarlarda yeniden paketlenmiş uygulamaları tespit etmek için DroidMOSS adlı bir prototip sistemi geliştirmişlerdir [120]. Yeniden paketlenen uygulamadaki olası değişiklikleri tespit etmek için bulanık bir hashing tekniği kullanılmıştır. Önerilen sistem, altı farklı üçüncü taraf pazarında yeniden paketlenen uygulamaları tanımlamak için test edilmiş olup test edilen pazarlarda barındırılan uygulamaların %5 ile %13'ünün yeniden paketlenmiş uygulamalar olduğu görülmüştür.

II. Dinamik Analiz Zorlukları

Dinamik analiz, statik analizin zayıflıklarından kaçınmak için kullanılmakta, ancak bu tür analizler, bazı zorluklarla da karşı karşıya kalmaktadır.

1) Tüm Yürütme Yollarının Kapsanması

Dinamik analiz yönteminin karşılaştığı en önemli zorluklardan biridir, çünkü uygulamadaki davranışı tam olarak analiz etmek için dinamik analiz yöntem, uygulamanın tüm olası yürütme yollarını kapsanması gerektirmektedir. Genellikle, uygulamadaki herhangi bir etkinlik birden fazla öğe (düğmeler, metin kutusu, radyo düğmesi vb.) içermekte ve Monkey Runner gibi UI tetikleme araçların çoğu uygulamalarla etkileşim kurmak için rastgele olayları oluşturduğundan yürütmenin bazı yolları gözden kaçabilmektedir. Bu sorunun tam anlamıyla çözülebilmesi mümkün olmasa da incelenen bazı çalışmalarda bu zorluk ele alınmaya çalışılmıştır. Örneğin, Ali-Gombe ve arkadaşları, bir uygulamanın yürütme yolları tam olarak kapsamak için, uygulamadaki öğeleri tetikleyen bir Python aracı geliştirmişlerdir [47]. Ayrıca, Alzaylaee ve arkadaşları, Android MonkeyRunner'ı, DroidBot UI tetikleme aracıyla melezleştirerek otomatik kullanıcı ara yüzü tetikleyicisi geliştirmek için melez bir yöntem önermişlerdir [121].

2) Anti-Emülasyon Teknikleri

Kötü amaçlı yazılım geliştirici, uygulamanın analiz ortamda mı yoksa normal ortamda mı çalıştığını öğrenmek için birçok yol ve tekniği kullanmaktadır. Örneğin, kötü amaçlı bir uygulamanın, IMSI ve IMEI değeri gibi belirli donanım tanımlayıcı değeri inceleyerek öykünücüde çalıştırıldığını algılayabilmektedir. Kötü amaçlı uygulama, analiz ortamı ile normal ortam arasındaki kaynaklar kapasitesindeki (işlemci, bellek ...vb.) farkına dayanarak analiz ortamı da belirleyebilmektedir. Bu teknik, öykünücülerin kaynaklarının genellikle gerçek aygıtların kaynaklarından daha fazla olmasına dayanmaktadır [122]. Ayrıca, kullanıcının uygulama ile etkileşimini gözlemleyerek kötü amaçlı uygulamanın bir analiz ortamında çalıştırılıp çalıştırılmadığını keşfetmek de mümkündür. Bu yöntem, UI tetikleyiciler tarafından

oluşturulan olay sıklığının, normal kullanıcı tarafından oluşturulanlara kıyasla çok daha yüksek ve düzensiz olduğu gerçeğine dayanmaktadır. Kötü amaçlı yazılım, bir analiz ortamında yürütüldüğünü tespit edince, kötü niyetli davranışı gizleyip algılamadan kaçmak için zararsız görevleri gerçekleştirmektedir. Ayrıca, bazı kötü amaçlı uygulamaların, analiz süresi atlamak için kötü niyetli davranışları bir süre gizlemektedir.

Bu teknik etkileri hafifletmek için çok sınırlı sayıda çalışmada izlenen bazı basit prosedürlerin dışında, önceki çalışmalarda bu soruna net bir çözüm önerilmemiştir. Örneğin, Alzaylaee ve arkadaşlarındaki çalışmasında [123] ve Alzaylaee ve arkadaşlarındaki çalışmasında [45], kullanılan öykünücü geliştirmek için kişi iletişim bilgilerinin eklenmesi ve öykünücünün IMEI numarasının gerçek bir IMEI numarası ile değiştirilmesi gibi bazı prosedürler gerçekleştirilmiştir. Şekil 2.4, kullanılan analiz yöntemlerinin zorlukların taksonomisi göstermektedir.

Şekil 2.4. Kullanılan analiz yöntemlerin zorluklar taksonomisi.

2.1.2. Dinamik Analiz Ortamları

Bu kategori, uygulama davranışı analiz etmede kullanılabilecek dinamik analiz ortamları geliştirmeyi amaçlayan çalışmaları kapsamaktadır. Bu tür çalışmalar, uygulama davranışı izleyip analiz edebilecek yeni bir işletim sistemi oluşturmak için Android işletim sistemi değiştirmeyi amaçlayan çalışmaları içermektedir. Ayrıca, bir analiz altyapı (Test-Bed) tasarlama ve geliştirmeye yönelik çalışmaları da içermektedir. Örneğin, Rastogi ve arkadaşları, AppsPlayground adlı Android uygulamaları analiz etmek amacıyla kullanılabilecek bir test ortamı önermişlerdir

[48]. Önerilen çerçeve, bir dizi algılama, keşif ve kılık değiştirme teknikleri entegre ederek uygulamaların dinamik analizi için otomatik bir ortam sağlamayı amaçlamaktadır. Önerilen platform, Android SDK ile birlikte gelen standart Android öykünücüsü temel alınarak geliştirilmiştir. Li ve arkadaşları, Andlantis adlandırılan ve saatte 3000'den fazla Android uygulaması işleyebilen dinamik bir analiz platformu önermişlerdir [124]. Sistem, kötü amaçlı yazılım analist, uygulamaların anormal davranışı tanımlamasına ve anlamasına yardımcı olmaktadır. Ayrıca, Amos ve arkadaşları, Android kötü amaçlı yazılım sınıflandırıcıları, otomatik olarak eğitmek ve değerlendirmek için STREAM adlı bir sistem önermişlerdir [82]. STREAM, APK kurulumu, kullanıcı girişi oluşturma, özellik vektörü toplama ve kötü amaçlı yazılım sınıflandırması ele alan otomasyon yaklaşımları sunmuştur. Ayrıca, kötü amaçlı yazılımların profilini hızlı bir şekilde çıkarmak ve makine öğrenme sınıflandırıcıları eğitmek için bir yöntem sağlamaktadır. Ayrıca, Tam ve arkadaşları, Android kötü amaçlı yazılımlar için ayrıntılı davranış profilleri oluşturmak için CopperDroid adlı sanal makine iç gözlem (VMI) tabanlı dinamik analiz platformu geliştirmişlerdir [125]. CopperDroid'in hem Java hem de yerel kod yürütmesinden başlatılan eylemleri yakalayabileceği belirtilmiştir.

2.1.3. Politika Uygulama Çerçeveleri

Uygulamaların yüklenmesi veya çalıştırılması sırasında uygulanacak bir dizi kural oluşturmayı amaçlayan çalışmaları içermektedir. Örneğin, Bugiel ve arkadaşları, Android izin çerçevesini genişleten XManDroid adlı bir çerçeveyi sunmuşlardır [126]. XManDroid, tanımlanan politikaya göre olası kötü niyetli bağlantıları önlemek için uygulamalar arasında iletişim bağlantıları çalışma zaman izlenmesi ve analiz edilmesi gerçekleştirmektedir. Önerilen yöntem, çalışma zamanında, uygulama düzeyinde ayrıcalık yükseltme saldırıları algılamayı ve önlemeyi amaçlamaktadır. Ayrıca, Ongtang ve arkadaşları, kurulum-zaman izin verme politikaları ve uygulamalar arası iletişim (IPC) politikaları ekleyerek Android güvenlikteki sınırlamaları ele almak için Saint adlı genişletilmiş bir Android platformu geliştirmişlerdir [127].

2.1.4. Kod Paketleyici Araçları / Kod Paket Çözme Araçları

Paketleme teknikleri, uygulamaları kurcalama ve tersine mühendislikten korumak için uygulama geliştiriciler tarafından kullanılmaktadır, bu amaçla kod paketleyici adı verilen araçları kullanılmaktadır. Bu tekniklerde, kod yeniden kullanımı önlemek için uygulama kaynak kodu gizlemek için daha önce bahsedilen gizleme, yansıtma, yerel kod ve dinamik yük gibi yöntemlerin bir kombinasyonu kullanılmaktadır. Ne yazık ki, kötü amaçlı uygulama geliştiricileri, kötücül yazılım analizini engellemek veya daha zor hale getirmek için bu tip teknikleri kullanmaktadır. İncelenen çalışmaların bazıları, kötü amaçlı yazılımdan koruma sistemlerin sağlamlığını test etmek için bazı kamuflaj teknikleri önermeyi ve test etmeyi amaçlamaktadır. Bu tür çalışmaları, kod paketleyici olarak sınıflandırılmıştır. Örneğin, Zheng ve arkadaşları, kötü amaçlı yazılım mutasyonuna karşı anti-virüs sistemlerinin sağlamlığını değerlendirmek için, yeniden paketleme ve gizleme teknikleri kullanarak bir kötü amaçlı yazılımdan birden fazla kötü amaçlı yazılım örneği oluşturabilen ADAM adlı bir çerçeveyi geliştirmişlerdir [108]. Rastogi ve arkadaşları, gizleme tekniklerine karşı bazı ticari kötü amaçlı yazılımdan koruma sistemlerinin sağlamlığını test etmek için DroidChameleon adlandırılan sistematik bir çerçeveyi geliştirmişlerdir [128]. Ayrıca, Maiorca ve arkadaşları, bazı kötü amaçlı yazılımdan koruma sistemlerinin sağlamlığını test etmek için kullanılan kötü amaçlı yazılım davranışı gizlemek için bir dizi gizleme tekniği kullanmışlardır [110]. Huang ve arkadaşları, yeniden paketleme algılama sistemlerinin gizleme tekniklerine karşı dayanıklılığını test etmek için bir çerçeveyi geliştirmişlerdir [129]. Öte yandan, ele alınan bazı çalışmalar, kod paketleme teknikleri kullanan uygulamaları paketten çıkarmak ve uygulamanın orijinal kaynak kodunu elde etmek için araçları geliştirmeyi amaçlamaktadır, bu tip çalışmaları kod paket çözücü olarak sınıflandırılmıştır. Örneğin, Xue ve arkadaşları, paketlenmiş DEX dosyası çıkartmak için PackerGrind olarak adlandırılan bir aracı önermişlerdir [130]. Önerilen araç, bazı gerçek paketlenmiş uygulamaları kullanılarak test edilmiş olup paketlenmiş uygulamadan orijinal DEX dosyaları geri getirmede verimli olduğu belirtilmiştir. Li ve arkadaşlarındaki çalışmasında [131], Android kod paketleme teknikleri, Dex koruma teknikleri, yerel koruma teknikleri, bellek koruma teknikleri ve kod serbest bırakma koruma teknikleri olmak üzere dört türe göre ayrılmıştır. Ayrıca, AppSpear adlı

otomatik bir paket açma sistemi önerilmiştir. Önerilen çerçeve, ByteCode şifre çözme ve Dex yeniden birleştirmeye dayalı yeni bir yaklaşım kullanmaktadır, bu da geleneksel manuel analitik ve bellek dökümü tabanlı paket açma tekniklerinin yerini alabilmektedir.

2.1.5. Kullanıcı Ara Yüzü Tetikleme Araçları

Daha önce de belirtildiği gibi, uygulamayla etkileşim ve uygulamanın normal kullanımını simüle etme sorunu, dinamik analiz yöntemlerinin en önemli zorluklarından biridir. Kullanılan UI tetikleme araçlarının çoğu rastgele olay oluşturmaya bağlı olduğundan, uygulamadaki gerçek davranış hakkında önemli bilgileri verebileceği yürütme yollarının bazıları gözden kaçabilmektedir. Bu nedenle, kullanıcı ara yüzü tetikleme araçların geliştirilmesi, önceki çalışmaların en önemli araştırma trendlerinden biridir. Örneğin, Amalfitano ve arkadaşları, AndroidRipper, GUI ripleme tabanlı (GUI Ripping-based) bir otomatik kullanıcı ara yüzü olay tetikleme aracı sunmuşlardır [132]. Önerilen tekniğin, MonkeyRunner aracıyla uygulanan rastgele test tekniğine göre daha etkili olduğunu belirtilmiştir. Machiry ve arkadaşları, Android uygulamaları için kullanıcı arabirim girdileri üreten ve Dynodroid olarak adlandırılan bir sistem sunmuşlardır [133]. Üreten girdileri, verimli bir şekilde üretmek için yeni bir gözlemle-seç-çalıştır prensibi kullanılmıştır. Önerilen araç, değiştirilmemiş uygulama ikili dosyalar düzeyinde çalışmış olup kullanıcı ve makineden gelen girdileri birleştirme yeteneği ile kullanıcı ara yüzü girdileri ve sistem girdileri oluşturabilmiştir. Chang ve arkadaşları, mobil uygulamalarla anlamlı bir sırayla etkileşime girme ve uygulama davranışı izleme yeteneğine sahip bir UI otomatik tetikleme aracı gerçekleştirmişlerdir [81]. Önerilen araç, çeşitli makine öğrenme algoritmalarına dayalı bir karar modeli oluşturularak test edilmiş ve elde edilen sonuçları, diğer bazı araçların sonuçlarıyla karşılaştırılmıştır.

Çizelge 2.1, önerilen taksonomiye göre incelenen eserlerin bilgilerini göstermektedir.

Çizelge 2.1. Önerilen taksonomiye göre bu çalışmadaki ele alınan eserlerin sınıflandırması

#	Araç adı	Yıl	Amaç	Yöntem	Kullanılan özellikleri	Özellik Seçimi	Algılama yöntemi	Statik zorluklarla mücadele				Dinamik zorluklarla mücadele	
								R	N	D	O	X_P	Em
1	DL-Droid [134]	2020	B_analiz	Dinamik	Mani_F, Dyn_beh	-	Deep_L					✗	✓
2	Ünver et al [135]	2020	B_analiz	Vis tabanlı	Lokal ve global Özellikleri		ML	✗	✗	✗	✗		
3	VisDroid [136]	2020	B_analiz	Vis tabanlı	Lokal ve global Özellikleri	-	ML	✗	✗	✗	✗		
4	Taheri et al [137]	2020	B_analiz	Statik	Mani_F, Cod_F	Rastgele Orman	Ptrn_M	✗	✗	✗	✗		
5	Pektaş et al [138]	2020	B_analiz	Statik	Cod_F	-	Deep_L	✗	✗	✗	✗	✓	✗
6	NSDroid [139]	2020	B_analiz	Statik	Cod_F	-	Sign_sim	✗	✗	✗	✗		
7	Xiao et al [140]	2020	B_analiz	Statik	Cod_F	n-gram	Deep_L	✗	✗	✗	✗		
8	AMalNet [141]	2020	B_analiz	Statik	Mani_F, Cod_F	-	Deep_L	✗	✗	✗	✗		
9	Mateless et al [142]	2020	B_analiz	Statik	Cod_F	Chi2, TF–IDF	ML	✗	✗	✗	✗		
10	SaaS [143]	2019	B_analiz	Vis&static based	Mani_F, Cod_F, Görüntü özellikleri	TF-IDF	ML	✓	✗	✗	✗		
11	Saint et al [34]	2019	B_ analiz	Vis tabanlı	Cod_F, Görüntü özellikleri	Deep_L	Deep_L	✗	✗	✗	✗		
12	Zhang et al [144]	2019	B_ analiz	Statik	Mani_F, Cod_F	-	ML	✗	✗	✗	✗		
13	NDroid [144]	2019	B_ analiz	Dinamik	Sistem çağrılar, API çağrılar	-	-	✗	✓	✓	✗	✗	✗
14	CloneSpot [145]	2019	B_ analiz	Statik	Meta_F	-	Ptrn_M	✓	✗	✗	✗		
15	DroidDet [37]	2018	B_ analiz	Statik	Mani_F, cod_F	TF-IDF, Cos_Sim	ML	✗	✗	✗	✗		
16	Gurulian.et al [36]	2016	B_ analiz	Statik	Meta_F	-	Sign_sim	✓	✗	✗	✗		
17	KUAFUDET [38]	2018	B_ analiz	Statik	Cod_F, Semtc_F	Manuel budama	ML+ Sim	✓	✗	✗	✗		
18	Mad4a [51]	2018	B_ analiz	Hibrit	Mani_F, Net_beh	-	Ptrn_M					✗	✗

19	SAFEDroid [65]	2018	B_ analiz	Statik	Cod_F, Mani_F, Meta_F	-	ML	✗	✗	✓	✓		
20	Wang et al [74]	2018	B_ analiz	Statik	Mani_F, Semtc_F	-	Ptrn_M	✗	✗	✗	✗		
21	Wang et al [39]	2018	B_ analiz	Statik	Cod_F, Mani_F	-	Deep_L	✗	✗	✗	✗		
22	DroidFusion [67]	2018	B_ analiz	Statik	Cod_F, Mani_F	IG	ML	✗	✗	✓	✗		
23	Kirubavathi et al [40]	2017	B_ analiz	Statik	Mani_F	IG	ML	✗	✗	✗	✗		
24	MalPat [41]	2018	B_ analiz	Statik	Cod_F, Mani_F	-	ML	✗	✗	✗	✗		
25	FalDroid [146]	2018	B_ analiz	Statik	Semtc_F	TF-IDF	ML	✗	✗	✗	✗		
26	AppSpear [131]	2018	Packer/Unpacker	Dinamik	-	-	ML	✓	✓	✓	✓	✓	✗
27	Papadopoulos et al [147]	2018	B_ analiz	Dinamik	R_Con	-	ML					✗	✗
28	Jha et al [148]	2018	B_ analiz	Statik	Mani_F	-	-	✗	✗	✗	✗		
29	SIGPID [149]	2018	B_ analiz	Statik	Mani_F	-	ML	✗	✗	✗	✗		
30	Chunlei et al [150]	2018	B_ analiz	Statik	Cod_F	MI	ML	✗	✗	✗	✗		
31	Özkan et al]151[	2018	B_ analiz	Statik	Mani_F	-	ML	✗	✗	✗	✗		
32	Yangxu et al [152]	2018	B_ analiz	Statik	Semtc_F	-	ML	✗	✗	✗	✗		
33	Chihiro et al [153]	2018	B_ analiz	Statik	Cod_F, Mani_F	Deep_L	Deep_L	✗	✗	✗	✗		
34	RanDroid [154]	2018	B_ analiz	Statik	Cod_F, Mani_F	-	ML	✗	✓	✓	✓		
35	ONAMD [155]	2018	B_ analiz	Statik	Cod_F, Mani_F, Semtc_F		Sign, ML	✗	✗	✗	✗		
36	Jaemin et al [156]	2018	B_ analiz	Statik	Cod_F	-	ML	✗	✗	✗	✗		
37	Somarriba et al [42]	2017	B_ analiz	Dinamik	Net_Beh	-	Ptrn_M					✗	✗
38	AspectDroid [47]	2018	B_ analiz	Hibrit	Semtc+Dyn_Beh	-	Ptrn_M	✗	✓	✓	✗	✓	✗
39	SAMADroid [157]	2018	B_ analiz	Hibrit	Mani_F, cod_F, Sistem çağrıları		ML	✗	✗	✗	✗	✗	✗
40	NTPDroid [158]	2018	B_ analiz	Hibrit	Mani_F, Net_beh	-	Ptrn_M	✗	✗	✗	✗	✗	✗
41	Alzaylaee [121]	2017	UI tetikleme	Dinamik	API imzaları	-	-					✓	✗
42	MOCDroid [62]	2016	B_ analiz	Statik	Cod_F	Genetik Algoritma	ML+Sign	✗	✗	✗	✗		
43	AndroDialysis [58]	2017	B_ analiz	Statik	Mani_F	-	ML	✗	✗	✗	✗		
44	Wang et al [159]	2017	B_ analiz	Statik	Cod_F, Mani_F,	MI,Chi2,ANOVA	ML	✗	✓	✓	✓		

					Meta_F								
45	Pındroid [104]	2017	B_ analiz	Statik	Cod_F, Mani_F	-	ML	✗	✗	✗	✗		
46	DroidNative [73]	2017	B_ analiz	Statik	Semtc_F	-	Sign	✗	✓	✗	✗		
47	Sokolova et al [160]	2017	B_ analiz	Statik	Semtc_F	-	ML	✗	✗	✗	✗		
48	Du et al [94]	2017	B_ analiz	Statik	Semtc_F	-	ML	✗	✗	✗	✗		
49	Palumbo et al [71]	2017	B_ analiz	Statik	Cod_F , Meta_F	Eşik temelli	ML	✗	✗	✗	✗		
50	Yang et al [90]	2017	B_ analiz	Statik	Meta_F	IG	Sign	✗	✗	✗	✗		
51	Tong and Yan [161]	2017	B_ analiz	Hibrit	Semtc, Dyn_Beh	-	Ptrn_M	✗	✓	✓	✗	✗	✗
52	Milosevic et al [66]	2017	B_ analiz	Statik	Cod_F, Mani_F	-	ML	✗	✗	✗	✗		
53	MalDozer [97]	2017	B_ analiz	Statik	Cod_F	Deep_L	Deep_L	✗	✗	✗	✗		
54	Buchanan et al [162]	2017	B_ analiz	-	-	-	-						
55	Rehman et al [163]	2017	B_ analiz	Hibrit	Mani_F, Cod_F, Dyn_Beh	-	ML	✗	✗	✗	✗	✗	✗
56	Alzaylaee et al [45]	2017	B_ analiz	Dinamik	API çağrıları ve Intentler	IG	ML					✓	✓
57	Wang et al. [80]	2017	B_ analiz	Dinamik	Sistem çağrıları	-	ML					✗	✗
58	PackerGrind [130]	2017	Packer/Unpacker	Hibrit	-	-	-	✗	✓	✓	✓	✗	✗
59	Martinelli et al [164]	2017	B_ analiz	Dinamik	Sistem çağrıları	Deep_L	Deep_L					✗	✗
60	Liang et al [165]	2017	B_ analiz	Dinamik	Sistem çağrıları	Deep_L	Deep_L					✗	✗
61	Su et al [166]	2017		Hibrit	Cod_F, Mani_F, Dyn_Beh	-	ML	✗	✗	✗	✗	✗	✗
62	CASANDRA [167]	2017	B_ analiz	Statik	Semtc_F	-	ML	✗	✗	✗	✗		
63	FgDetector [168]	2017	B_ analiz	Statik	Cod_F, Mani_F	PCA algoritması	ML	✓	✗	✗	✓		
64	Mohsen et al [169]	2017	B_ analiz	Statik	Cod_F, Mani_F	-	ML	✗	✗	✗	✗		
65	DeepFlow [170]	2017	B_ analiz	Statik	Semtc_F	Deep_L	Deep_L	✗	✗	✗	✗		
66	Nix et al [99]	2017	B_ analiz	Statik	Cod_F	Deep_L	Deep_L	✗	✗	✗	✓		
67	DAPASA [171]	2017	B_ analiz	Statik	Semtc_F	-	ML	✗	✗	✗	✗		
68	R2-D2 [31]	2017	B_ analiz	Vis tabanlı	Cod_F, Görüntü Özellikleri	Deep_L	Deep_L	✗	✗	✗	✗		
69	Yang et al [32]	2017	B_ analiz	Vis tabanlı	Görüntü Özellikleri	-	ML	✗	✗	✗	✗		

70	Karimi et al [33]	2017	B_ analiz	Vis tabanlı	Semtc_F, Görüntü Özellikler	-	Sign	✗	✗	✗	✗		
71	Dexteroid [77]	2016	B_ analiz	Statik	Cod_F	-	Ptrn_M	✗	✗	✗	✗		
72	Brown et al. [76]	2016	B_ analiz	Statik	Semtc_F	-	ML	✗	✗	✗	✗		
73	Ma et al. [93]	2016	B_ analiz	Statik	Cod_F, Mani_F	-	ML	✗	✗	✗	✗		
74	ANASTASIA [92]	2016	B_ analiz	Statik	Cod_F, Mani_F	Ağaçlar Sınıflandırıcı	Deep_L	✗	✗	✗	✗		
75	Andro-Dumpsys [172]	2016	B_ analiz	Hibrit	Cod_F, Mani_F, Meta_F, Dyn_Beh	-	Ptrn_M	✗	✓	✓	✓	✗	✗
76	Alba et al. [173]	2016	B_ analiz	Statik	Cod_F, Mani_F	Chi2,Relief	ML	✗	✗	✗	✗		
77	SafeDroid [174]	2016	B_ analiz	Statik	Cod_F	-	ML	✗	✗	✗	✗		
78	ASE [175]	2016	B_analiz	Statik	Mani_F, Meta_F	-	Ptrn_M	✗	✗	✗	✗		
79	Kumar et al. [83]	2016	B_analiz	Vis tabanlı	GIST Görüntü Özellikleri	-	ML	✗	✗	✗	✗		
80	Mamadroid [105]	2016	B_analiz	Statik	Semtc_F	PCA	ML	✗	✗	✗	✗		
81	Chang et al. [81]	2016	UI trigger	Hibrit	Mani_F, Dyn_Beh	IG		✗	✗	✓	✓	✗	✗
82	Verma and Muttoo [96]	2016	B_analiz	Statik	Meta_F	IG	ML	✗	✗	✗	✗		
83	Wu et al. [101]	2016	B_analiz	Statik	Semtc_F	-	ML	✗	✗	✗	✗	✗	✗
84	Deep4MalDroid [98]	2016	UI trigger	Dinamik	Sistem çağrıları	-	Deep_L	✗	✗	✗	✗		
85	DroidOL [176]	2016	B_analiz	Statik	Semtc_F	-	ML	✗	✗	✗	✗		
86	DroidDetector [52]	2016	B_analiz	Hibrit	Mani_F, Code_F, Dyn_Beh	-	Deep_L	✗	✗	✓	✓	✗	✗
87	Dynalog [123]	2016	D_Env	Dinamik	API calls, Diğer Özellikleri	-	-					✗	✓
88	StormDroid [53]	2016	B_analiz	Hibrit	Mani_F, Code_F, Dyn_Beh	-	ML	✗	✗	✓	✗	✗	✗
89	Chen et al. [177]	2016	B_analiz	Statik	Code_F, Semtc_F	TF-IDF	ML	✗	✗	✓	✗		
90	Ju et al. [178]	2016	B_analiz	Statik	Mani_F	-	Ptrn_M	✗	✗	✗	✗		
91	Mmda [179]	2016	B_analiz	Statik	Mani_F	-	ML	✗	✗	✗	✗		
92	DroidDeepLearner [180]	2016	B_analiz	Statik	Cod_F, Mani_F	Deep_L	Deep_L	✗	✗	✗	✗		

93	Xiaotian et al. [181]	2016	B_analiz	Statik	Cod_F, Mani_F	Markov kapsamlı keşif algoritması	ML	✗	✗	✗	✗		
94	Manzhi et al. [182]	2016	B_analiz	Statik	Cod_F, Mani_F		ML	✗	✗	✗	✓		
95	Alejandro et al. [183]	2016	B_analiz	Statik	Mani_F, Semtc_F	-	ML	✗	✗	✗	✗		
96	DroidDeep [184]	2016	B_analiz	Statik	Cod_F, Mani_F	Derin Belief Ağı (DBN)	Deep_L	✗	✗	✗	✗		
97	DroidChain [185]	2016	B_analiz	Statik	Semtc_F	-	Sing	✗	✗	✗	✗		
98	ROAR [186]	2016	B_analiz	Statik	Mani_F, Semtc_F	-	Sing	✗	✓	✗	✗		
99	Xiaotian et al. [187]	2016	B_analiz	Statik	Mani_F, Semtc_F, Cod_F	-	ML	✗	✗	✗	✓		
100	Salvador et al. [188]	2016	B_analiz	Statik	Mani_F	IG, Relief, Chi2	ML	✗	✗	✗	✗		
101	Wei et al. [43]	2015	B_analiz	Dinamik	Net_Beh	-	ML					✗	✗
102	Elish et al. [75]	2015	B_analiz	Statik	Semtc_F	-	Ptrn_M	✗	✗	✗	✗		
103	APK Auditor [56]	2015	B_analiz	Statik	Mani_F	-	Sign	✗	✗	✗	✗		
104	Kurniawan et al. [44]	2015	B_analiz	Dinamik	R_Con, Net_Beh	-	ML					✗	✗
105	Yerima et al. [112]	2015	B_analiz	Statik	Cod_F, Mani_F	MI	ML	✗	✗	✗	✗		
106	Canfora et al. [189]	2015	B_analiz	Statik	Semtc_F	MI	ML	✗	✗	✗	✗		
107	Lantz et al. [114]	2015	B_analiz	Statik	Cod_F	-	Ptrn_M	✗	✓	✗	✗		
108	Mobile-Sandbox [102]	2015	B_analiz	Hibrit	Mani_F, Dyn_Beh	-	ML	✗	✓	✓	✓	✗	✗
109	Li et al. [190]	2015	B_analiz	Statik	Cod_F	-	Ptrn_M	✗	✗	✗	✗		
110	Westyarian et al. [191]	2015	B_analiz	Statik	Cod_F	-	ML	✗	✗	✗	✗		
111	Li et al. [192]	2015	B_analiz	Statik	Cod_F, Mani_F	-	ML	✗	✗	✗	✗		
112	DroidSafe [193]	2015	B_analiz	Statik	Cod_F	-	-	✗	✓	✗	✗		
113	Sheen et al. [91]	2015	B_analiz	Statik	Cod_F	IG, Relief, Chi2	ML	✗	✗	✗	✗		
114	Lee et al. [70]	2015	B_analiz	Statik	Semtc_F, Cod_F	-	Sign	✗	✗	✗	✗		
115	M0Droid [194]	2015	B_analiz	Dinamik	Sistem çağrıları	-	Sign					✗	✗
116	Bushra et al. [195]	2015	B_analiz	Dinamik	Dyn_Beh	-	ML					✗	✗
117	MARVIN [196]	2015	B_analiz	Hibrit	Cod_F, Mani_F, Net_Beh, File_Acc	Fisher Puanı	ML	✗	✓	✓	✓	✗	✗
118	Ashutosh et al. [197]	2015	Packer/Unpacker	Vis tabanlı	-	-	-	✗	✗	✗	✓		

119	Maiorca et al. [110]	2015	Packer/Unpacker	Statik	-	-	-	✓	✓	✓	✓			
120	Andlantis [198]	2014	D_Env	Dinamik	Net_Beh	-	-	✗	✗	✓	✗	✗	✗	
121	Adebayo et al. [199]	2014	B_analiz	Statik	Cod_F	Apriori-PSO	ML	✗	✗	✗	✗			
122	TaintDroid [49]	2014	B_analiz	Dinamik	Semtc_F	-	-					✗	✗	
123	DroidSentinel [200]	2014	B_analiz	Dinamik	-	-	Sign					✗	✗	
125	Li et al. [124]	2014	B_analiz	Dinamik	Net_Beh	-	ML					✗	✗	
126	Mdoctor[201]	2014	B_analiz	Statik	Mani_F, Meta_F		Sign	✗	✗	✗	✗			
127	Merlo et al. [202]	2014	B_analiz	Dinamik	R_Con	-	-					✗	✗	
128	DroidTrace [118]	2014	B_analiz	Hibrit	Mani_F, Dyn_beh	-	Ptrn_M	✗	✗	✓	✓	✗	✗	
129	Dendroıd [203]	2014	B_analiz	Statik	Semtc_F	-	ML	✗	✗	✗	✗			
130	Moonsamy et al. [103]	2014	B_analiz	Statik	Cod_F, Mani_F	-	ML	✗	✗	✗	✗			
131	PasDroid [204]	2014	B_analiz	Dinamik	Semtc_F	-	-					✗	✗	
132	DroidSIFT [59]	2014	B_analiz	Statik	Semtc_F, Cod_F	-	ML	✗	✗	✗	✗			
133	Drebin [54]	2014	B_analiz	Statik	Cod_F, Mani_F	-	ML	✗	✗	✗	✗			
134	Yerima et al. [205]	2014	B_analiz	Statik	Cod_F, Mani_F	-	ML	✗	✗	✗	✗			
135	Ng et al. [100]	2014	B_analiz	Dinamik	Sistem çağrıları	-	Ptrn_M					✗	✗	
136	Manilyzer [84]	2014	B_analiz	Statik	Mani_F	-	ML	✗	✗	✗	✗			
137	AMDetector [206]	2014	B_analiz	Hibrit	Mani_F, Cod_F, Dyn_Beh	-	Ptrn_M	✗	✗	✓	✗	✗	✗	
138	Apposcopy [207]	2014	B_analiz	Statik	Semtc_F	-	Sign	✗	✗	✗	✗			
139	Xiaoyan et al. [208]	2014	B_analiz	Statik	Mani_F	PCA algoritması	ML	✗	✗	✗	✗			
140	Xiangyu et al. [209]	2014	B_analiz	Statik	Mani_F	IG, Fisher Puanı, Chi2	ML	✗	✗	✗	✗			
141	Droid Detective [210]	2014	B_analiz	Statik	Mani_F	-	Ptrn_M	✗	✗	✗	✗			
142	MIGDroid [119]	2014	B_analiz	Statik	Cod_F	-	G_Sim	✓	✗	✗	✗			
143	Idrees et al. [211]	2014	B_analiz	Statik	Mani_F	-	ML	✗	✗	✗	✗			
144	Raphael et al. [212]	2014	B_analiz	Statik	Cod_F, Mani_F	X–ANOVA, X–Utest	G_Sim	✗	✗	✗	✗			
145	Park et al. [69]	2014	B_analiz	Statik	Cod_F, Mani_F	-	G_Sim	✗	✗	✗	✗			
146	DroidGraph [72]	2014	B_analiz	Statik	Cod_F	-	Sign	✗	✗	✗	✗			

147	Moghaddam et al. [60]	2014	B_analiz	Statik	Cod_F, Mani_F	-	ML	✗	✗	✗	✗		
148	Shen et al. [86]	2014	B_analiz	Statik	Cod_F, Mani_F	Sezgisel tabanlı	G_Sim	✗	✗	✗	✗		
149	Britton et al. [213]	2014	B_analiz	Statik	Mani_F, Semtc_F	-	ML	✗	✗	✗	✗		
150	DroidAnalyzer [214]	2014	B_analiz	Statik	Cod_F, Mani_F	-	Sign	✗	✗	✗	✗		
151	Deepa et al. [215]	2014	B_analiz	Statik	Cod_F	GK, IG &CFS	ML	✗	✗	✗	✗		
152	Shabtai et al. [216]	2014	B_analiz	Dinamik	Net_Beh	-	ML	✓	✗	✓	✗		
153	Yerima et al. [217]	2013	B_analiz	Statik	Cod_F, Mani_F	-	ML	✗	✗	✓	✓		
154	AppsPlayground [48]	2013	D_Env	Dinamik	-	-	-					✗	✗
155	Mobile-Sandbox [115]	2013	B_analiz	Hibrit	Mani_F, Dyn_Beh	-	-	✗	✓	✗	✓	✗	✗
156	DroidChameleon [113]	2013	Packer/Unpacker	Statik	-	-	-					✗	✗
157	Dynodroid [133]	2013	UI trigger	Dinamik	-	-	-					✗	✗
158	Chekina et al. [218]	2013	B_analiz	Dinamik	Net_Beh	-	-					✗	✗
159	Karami et al. [219]	2013	UI trigger	Dinamik	Net_Beh, File_Acc	-	-					✓	✗
160	Stream [82]	2013	D_Env	Dinamik	R_Con, Net_Beh		ML					✗	✗
161	Mas'ud et al. [55]	2013	B_analiz	Hibrit	Mani_F, Cod_F, Sys_call, Net_Beh	-	-	✗	✗	✗	✗	✗	✗
162	AEMF [220]	2013	B_analiz	Dinamik	Semtc_F	-	Ptrn_M					✗	✗
163	DroidAPIMiner [63]	2013	B_analiz	Statik	Cod_F, Mani_F	-	ML	✗	✓	✓	✗		
164	Samra et al. [57]	2013	B_analiz	Statik	Mani_F	-	ML	✗	✗	✗	✗		
165	AppGuard [221]	2013	Policy_Enf	Statik	-	-	Policy	✗	✗	✗	✗		
166	A3 [68]	2013	B_analiz	Statik	Semtc_F	-	Ptrn_M	✗	✗	✗	✗		
167	Aung et al. [95]	2013	B_analiz	Statik	Mani_F	IG	ML	✗	✗	✗	✗		
168	WHYPER[222]	2013	B_analiz	Statik	Mani_F	-	-	✗	✗	✗	✗		
169	DroidOLytics [87]	2015	B_analiz	Statik	Semtc_F	Benzerlik özet karması	Sign	✓	✗	✗	✗		
170	Peiravian et al. [223]	2013	B_analiz	Statik	Cod_F, Mani_F	-	ML	✗	✗	✗	✗		
171	Glodek et al. [116]	2013	B_analiz	Statik	Cod_F, Mani_F	-	ML	✗	✓	✗	✗		
172	Lu et al. [224]	2013	B_analiz	Statik	Mani_F	-	ML	✗	✗	✗	✗		
173	AnDarwin [89]	2013	B_analiz	Statik	Semtc_F	-	Ptrn_M	✓	✗	✗	✗		

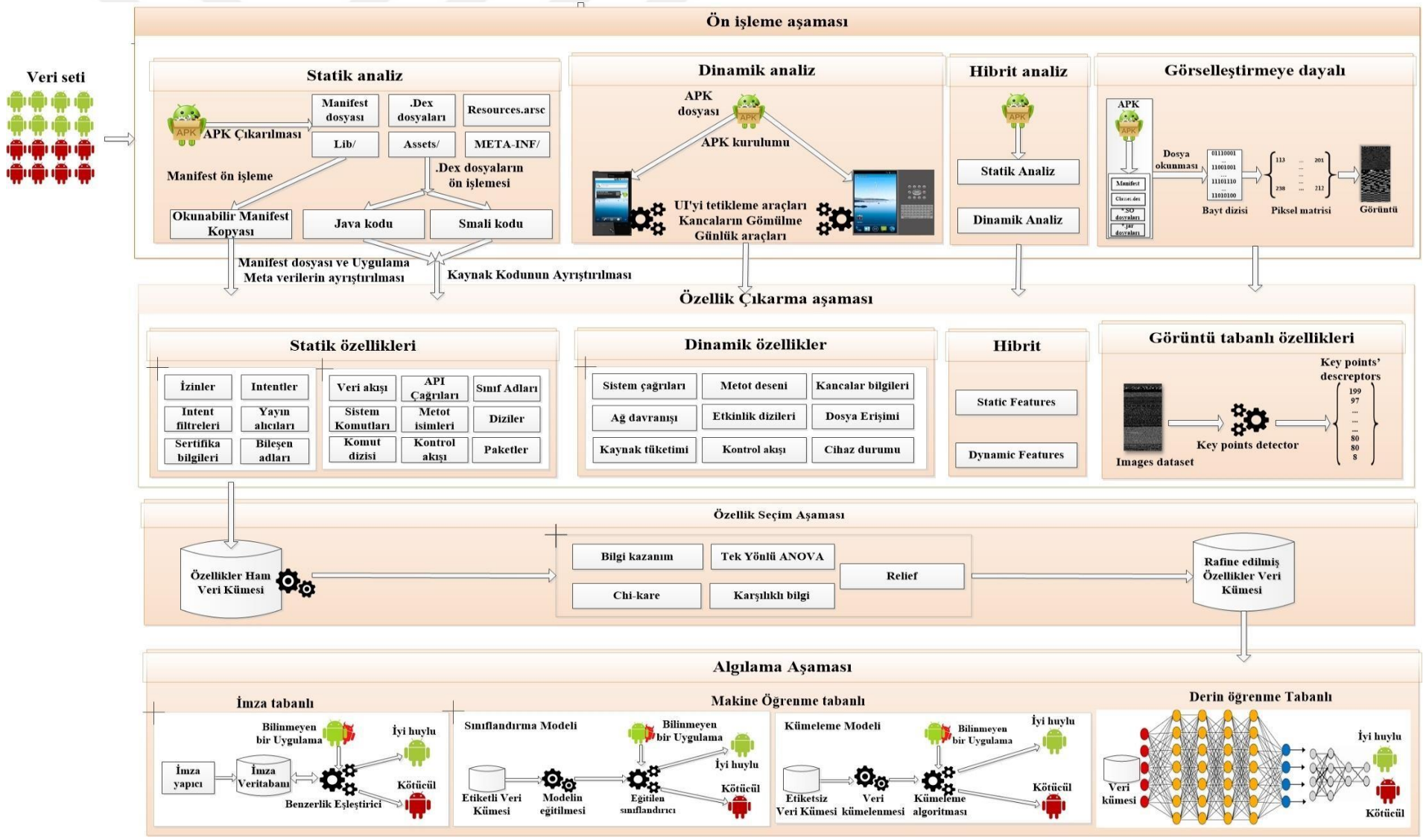
174	Yu et al. [225]	2013	B_analiz	Dinamik	Sistem çağrıları	-	ML					✗	✗
175	Alam et al. [226]	2013	B_analiz	Dinamik	Kombinasyon	-	ML					✗	✗
176	Ham et al. [227]	2013	B_analiz	Dinamik	R_Con	-	ML					✗	✗
177	VetDroid [228]	2013	B_analiz	Dinamik	-	-	-					✗	✗
178	CopperDroid [125]	2013	B_analiz	Dinamik	Sistem çağrıları	-	-	✗	✓	✗	✗	✓	✗
179	DroidAnalytics [229]	2013	B_analiz	Hibrit	Mani_F, API çağrıları	-	Sign	✓	✗	✓	✓	✗	✗
180	ANANAS [230]	2013	B_analiz	Dinamik	Net_Beh, Sistem çağrıları	-	-	✗	✓	✓	✗		
181	Puma [231]	2013	B_analiz	Statik	Mani_F	-	ML	✗	✗	✗	✗		
182	Huang et al. [129]	2013	Packer/Unpacker	Statik	-	-	-						
183	DroidLogger [46]	2012	B_analiz	Hibrit	Cod_F, Dyn_Beh	-	-	✗	✗	✗	✓	✗	✗
184	SmartDroid [232]	2012	UI trigger	Hibrit	Semtc_F, Dyn_Beh	-	-	✗	✗	✗	✗	✗	✗
185	DroidRanger [117]	2012	UI trigger	Hibrit	Mani_F, Dyn_B	-	Sign	✗	✓	✓	✗	✗	✗
186	ProfileDroid [233]	2012	D_Env	Hibrit	Mani_F, Dyn_B	-	-	✗	✗	✗	✗	✗	✗
187	DroidScope [50]	2012	D_Env	Dinamik	Semtc_F	-	-					✗	✗
188	AndroidRipper [132]	2012	UI trigger	Dinamik	-	-	-					✓	✗
189	AndroidLeaks [234]	2012	B_analiz	Statik	Cod_F, Mani_F	-	Sign	✗	✗	✗	✗		
190	Sanz et al. [78]	2012	B_analiz	Statik	Cod_F, Mani_F, Meta_F	IG	ML	✗	✗	✗	✗		
191	DroidMat [234]	2012	B_analiz	Statik	Cod_F, Mani_F	-	ML	✗	✗	✗	✗		
192	DroidMOSS [120]	2012	B_analiz	Statik	Cod_F, Meta_F	Bulanık hashing	Sign	✓	✗	✗	✗		
193	RiskRanker [88]	2012	B_analiz	Statik	Semtc_F	-	Sign	✗	✓	✗	✓		
194	Wei et al. [235]	2012	B_analiz	Statik	Mani_F	-	-	✗	✗	✗	✗		
195	MADAM [236]	2012	B_analiz	Dinamik	Sistem çağrıları & diğer Dyn_Beh	-	ML					✗	✗
196	Sahs et al. [237]	2012	B_analiz	Statik	Mani_F, Semtc_F	-	ML	✗	✗	✗	✗		
197	Sanz et al. [78]	2012	B_analiz	Statik	Cod_F, Mani_F, Meta_F	IG	ML	✗	✗	✗	✗		
198	Yang et al. [238]	2012	B_analiz	Statik	Semtc_F	-	-	✗	✗	✗	✗		

199	Gascon et al. [239]	2012	B_analiz	Statik	Semtc_F	-	ML	✗	✗	✗	✓		
200	ADAM [108]	2012	Packer/Unpacker	Statik	-	-	-						
201	YAASE [240]	2011	Policy_Enf	Dinamik	-	-	Politika					✗	✗
202	XmanDroid [126]	2011	Policy_Enf	Dinamik	-	-	Politika					✗	✗
203	Saint [127]	2012	Policy_Enf	Dinamik	-	-	Politika					✗	✗
204	Su et al. [241]	2011	B_analiz	Dinamik	Net_Beh, Sistem çağrıları	-	ML					✗	✗
205	QUIRE [242]	2011	D_Env	Dinamik	-	-	Politika					✗	✗
206	Stowaway [64]	2011	B_analiz	Statik	Cod_F, Mani_F	-	Ptrn_M	✗	✗	✗	✗		
207	Crowdroid [243]	2011	B_analiz	Dinamik	Sistem çağrıları	-	ML					✗	✗
208	ComDroid [244]	2011	B_analiz	Statik	Cod_F, Mani_F	-	-	✗	✗	✗	✗		
209	Isohara [245]	2011	B_analiz	Dinamik	System calles	-	Sign					✗	✗
210	Apex [246]	2010	Policy_Enf	Dinamik	-	-	Politika					✗	✗
211	Porscha [247]	2010	Policy_Enf	Dinamik	-	-	Politika					✗	✗
212	CrePE [248]	2010	Policy_Enf	Dinamik	-	-	Politika					✗	✗
213	Paranoid [249]	2010	B_analiz	Dinamik	Birden çok Dyn_Beh özelliği	-	-					✗	✗
214	Barrera et al. [250]	2010	B_analiz	Statik	Mani_F	-	ML	✗	✗	✗	✗		
215	AASandbox [251]	2010	D_Env	Hibrit	Cod_F, Mani_F, Sistem çağrıları	-	-	✗	✓	✗	✓	✗	✗
216	Shabtai et al. [252]	2010	B_analiz	Statik	Cod_F, Mani_F	-	ML	✗	✗	✗	✗		
217	Kirin [253]	2009	Policy_Enf	Statik	Mani_F	-	Politika	✗	✗	✗	✗		

Cod_F: Kod tabanlı özellikler | Semtc_F: Anlamsal özellikler | Mani_F: Manifest özellikleri | Meta_F: Uygulama meta veri özellikleri
R_Con: Kaynak tüketimi | Net_Beh: Ağ davranışı, Dyn_Beh: Dinamik Davranış || B_analiz: Davranış analiz
R: Yeniden paketleme | N: Yerel kod | D: Dinamik Yük | O: Gizleme teknikleri
X_P: Tüm yürütme yollarının kapsanması | Em: Anti-Emulator teknikleri
ML: Makine öğrenme | Sign: İmza tabanlı | Ptrn_M: Desen Eşleştirme | Polc&Rul: Politika ve Kurallar tabanlı | G_Sim: Graf Benzerliği
Vis-based: Görselleştirme Tabanlı | Kural: kural tabanlı | Politika: Politika tabanlı | Deep_L: Derin öğrenme | Policy_Enf: Politika uygulanması
MI: Karşılıklı bilgi | IG: Bilgi Kazançları | Cos_Sim: Kosinüs Benzerliği | CFS: Korelasyon Özellikler Seçimi | GK: Goodman Kruskals

2.2. Önerilen Şematik İnceleme Modeli

Bu çalışmada, ele alınan eserleri ve önerilen taksonomi ışığında şematik bir modeli önerilmiştir. Önerilen model, "Şematik İnceleme Modeli" (“Schematic Review Model”) olarak adlandırılmış olup araştırmacının fazla çaba harcamadan etki alanına kapsamlı bir bakış atmasını sağlayacak şekilde kötü amaçlı yazılım analiz sürecinin eksiksiz bir açıklaması sunmaktadır. Bu tür modellerin varlığının, tek bir model kullanarak alandaki en çok araştırma eğiliminin soyutlanmış bir tanımı vermek için tüm alanlardaki inceleme makalelerinde çok önemli olduğu kanaati hasıl olmuştur. Kötü amaçlı yazılım analizi süreci birden fazla aşama olarak açıklanmıştır, bu aşamaları, şekil 2-6'de gösterildiği gibi şematik bir şekilde ayrıntılı olarak açıklanmıştır. Gerçekleştirilen literatür taramasında görüldüğü kadarıyla, bu süreci, bu kadar ayrıntıyla ilk kez bu şekilde anlatılmaktadır. İncelenen eserlerde kullanılan çoğu teknik ve yöntem gözden geçirilmiş ve birden çok aşamada düzenlenmiştir. Bu aşamalar, “Kullanılan teknik aşamalara dayalı taksonomi” bölümünde ayrıntılı olarak tartışılmıştır.



Şekil 2.5.Önerilen Android kötü amaçlı yazılım tespit metodolojilerinin şematik incelemesi

3. MATERYAL VE YÖNTEM

Tez kapsamında, gerçekleştirilen sistematik incelemenin dışında iki ana çalışma gerçekleştirilmiştir. Birinci çalışma, bazı iyi bilinen Anti-virüs sistemlerin sağlamlığını değerlendirmeyi amaçlamaktadır. İkinci çalışma ise, Android uygulamaları sınıflandırmak için, görüntü işleme tekniklerine ve makine öğrenme algoritmalarına dayalı yeni bir yöntem önermeyi amaçlamaktadır.

3.1. Tez Çalışmasının Katkıları

Tez kapsamında üç ana çalışma gerçekleştirilmiştir. Birinci çalışmada, Android kötü amaçlı yazılım analiz alanında yürütülen 200'den fazla makaleyi analiz ederek önceki eserler için yeni ve kapsamlı bir taksonomi önerilmiştir. İkinci çalışma, bazı iyi bilinen Anti-virüs sistemlerin sağlamlığını değerlendirmeyi amaçlamaktadır. Bu amaçla, kötücül yazılım örneklerinden mutasyonlar oluşturmak için birkaç enjeksiyon saldırısı önerilmiş, ardından bazı ticari Anti virüs sistemleri, oluşturulan kötücül yazılım mutasyonlarının tespit edilebilirliği açısından değerlendirilmiştir. Üçüncü çalışma, Android uygulamaları sınıflandırmak için, görüntü işleme tekniklerine ve makine öğrenme algoritmalarına dayalı yeni bir yöntem önermeyi amaçlamaktadır. Üçüncü çalışma, iki tane alt çalışma içermektedir. İlk alt çalışmada, kötü amaçlı yazılım örnekleri birden çok kötücül yazılım ailesine sınıflandırmak için VisDroid olarak adlandırılan yeni bir model önerilmiştir. İkinci alt çalışmada, DeepVisDroid olarak adlandırılan yeni bir derin öğrenme modeli önerilmiş olup önerilen VisDroid modeli ve diğer bazı modellerle birlikte Android uygulamaları iyi huylu veya kötü amaçlı yazılım olarak sınıflandırmak için kullanılmıştır. Tez çalışmasında Sağlanan katkıları aşağıdaki gibi özetlenebilmektedir:

3.1.1. Gerçekleştirilen İlk Çalışma

- 2009 ve 2020 yılları arasında yayınlanan 200'den fazla makale kullanılarak sistematik bir inceleme yapılmıştır.
- İncelenen makalelere dayanarak önceki eserler için yeni ve kapsamlı bir taksonomi önerilmiştir.
- Şematik İnceleme Modeli (Schematic Review Model) olarak adlandırılan yeni ve ayrıntılı bir şematik inceleme yöntem önerilmiştir.

3.1.2. Gerçekleştirilen İkinci Çalışma

Bu çalışmada aşağıdaki katkıları sağlanmıştır:

- Bazı ticari Anti-virüs sistemlerinin kamuflaj tekniklerine karşı sağlamlığını değerlendirmek için birden çok enjeksiyon saldırısı önerilmiştir.
- Bu amaçla, izin enjeksiyon saldırısı ve izin-kodu enjeksiyon saldırısı olmak üzere iki enjeksiyon saldırısı önerip uygulanmıştır. Yapılan incelemelere göre bu tür saldırılar, Android kötü amaçlı yazılım algılama alanında ilk kez kullanılmıştır. Ayrıca, önerilen izin enjeksiyon saldırısı ve izin-kodu enjeksiyon saldırısı, uygulama yeniden imzalama adı verilen bir saldırı ile beraber, mevcut kötü amaçlı yazılım örneklerinden mutasyon oluşturmak için uygulanmıştır.
- Ayrıca, ticari Anti virüs sistemlerini daha da fazla şaşırtmak için ve yüksek derecede kamuflaj elde etmek için önerilen enjeksiyon saldırıları, dizi şifreleme, sınıf şifreleme ve kod çağırma yansıtma gibi bazı şaşırtma teknikleriyle melezleştirilmiştir. Yapılan incelemelere göre ilk kez bu çalışmada, şaşırtma teknikleri ve diğer saldırı türleri bir araya getirilerek melezleştirilmiştir.
- Elde edilen sonuçlar, test edilen kötü amaçlı yazılımdan koruma sistemlerinin çoğunun algılama doğruluğunun yaklaşık %10 veya daha azına düştüğünü göstermiştir. Ayrıca, kötü amaçlı yazılım örneklerini tespit edebilen Anti virüs sistem sayısı, 60'tan fazla sistemden yaklaşık 12 sisteme kadar düşmüştür.

3.1.3. Gerçekleştirilen Üçüncü Çalışma

Üçüncü çalışma kapsamında iki alt çalışma gerçekleştirilmiştir. Birinci alt çalışmada, kötü amaçlı yazılım örnekleri birden çok kötücül yazılım ailesine sınıflandırmak için VisDroid adlı yeni bir model önerilmiştir. İkinci alt çalışmada ise, DeepVisDroid adlı yeni bir derin öğrenme modeli önerilmiş ve birinci alt çalışmada önerilen VisDroid modeli ve diğer bazı modellerle beraber Android uygulamaları iyi huylu veya kötü amaçlı yazılım olarak sınıflandırmak için kullanılmıştır. Bu çalışmada aşağıdaki katkıları bulunmuştur:

3.1.3.1. Birinci Alt Çalışmada Sağlanan Katkılar

- Android kötücül yazılım örnekleri birden fazla kötücül yazılım aileye sınıflandırılmak amacıyla genel bir gri tonlamalı görüntü tabanlı modeli olan VisDroid önerilmiştir. Bu amaçla, her biri 4850 gri tonlamalı görüntü içeren beş kötü amaçlı yazılım görüntü veri kümesi oluşturulmuştur.
- Oluşturulan görüntü veri kümelerinden iki tür görüntü özelliği ayıklanmış ve önerilen modeli eğitmek için kullanılmıştır. İlk ayıklanan görüntü özellik türü, Renk Histogramı, Hu Momentleri ve Haralick Dokusu olmak üzere üç farklı görüntü global özelliği içermektedir. İkinci ayıklanan görüntü özellik türü, ÖBÖD (SIFT), HSÖ (SURF), YDB (ORB) ve KAZE olmak üzere dört farklı lokal görüntü özelliği içermektedir. Önerilen modeli eğitmek için ayıklanan görüntü tabanlı global özellikleri tek bir vektörde istiflenmiş ve önerilen modeli eğitmek için girdi olarak kullanılmıştır. Bunun karşılığı, ayıklanan görüntü tabanlı lokal özellik tanımlayıcılarından bir özellik vektörü oluşturmak için, görsel sözcük çantası gösterimi (GSÇ) kullanılmıştır. Literatür taramasında elde edilen bilgiler çerçevesinde, bu tür görüntü özellikler, Android kötü amaçlı yazılım sınıflandırma alanında ilk kez kullanılmıştır.
- Ayıklanan global ve lokal özellikleri, Random forest, K-Nearest-Neighbors, Decision Tree, Bagging, AdaBoost, Gradient Boost ve oylama sınıflandırıcıları olmak üzere birden çok makine öğrenme sınıflandırıcısı eğitmek için kullanılmıştır. Ayrıca, lokal ve global görüntü özelliklerinin bir karışımı

kullanılarak eğitilmiş bir dizi sınıflandırıcıya dayanarak sınıflandırma kararını alan yeni bir hibrit topluluk oylama sınıflandırıcısı önerilmiştir.

- Ayrıca, Rezidüel Sinir ağı (ResNet) ve Inception V3 olmak üzere iki iyi bilinen derin öğrenme modeli, oluşturulan gri tonlamalı görüntü veri setleri kullanılarak test edilmiştir.
- Elde edilen sonuçlar, bazı son teknoloji modellerin sonuçlarıyla karşılaştırılmıştır.
- Sonuçlar, önerilen yöntemin çok verimli olduğunu ve sınıflandırma doğruluğunun yaklaşık %98.2 'ye ulaştığını göstermiştir.

3.1.3.2. İkinci Alt Çalışmada Sağlanan Katkılar

- Android uygulamaları, kötü amaçlı veya iyi huylu olarak sınıflandırmak için, birinci alt çalışmada önerilen VisDroid modeli kullanılmıştır.
- Ayrıca, Android uygulamaları, kötü amaçlı veya iyi huylu olarak sınıflandırmak için, görüntü tabanlı özellikleri derin öğrenme teknikleriyle melezlenmesine dayanan DeepVisDroid adlı yeni bir model önerilmiş ve geliştirilmiştir. DeepVisDroid'de, yukarıda bahsedilen görüntü tabanlı özellikler, oluşturulmuş görüntü veri kümelerinden ayıklanmış ve 1-D evrişimli katman tabanlı bir evrişimli sinir ağ modeli eğitmek için kullanılmıştır. Literatür taramasında elde edilen bilgiler çerçevesinde, bu, ilk kez bir evrişimli sinir ağı model, görüntü tabanlı özellikleri kullanılarak eğitilmiş ve Android uygulamaları sınıflandırmak için kullanılmıştır.
- Ayrıca, önerilen DeepVisDroid modelinin sonuçları ile klasik 2-D katman tabanlı evrişimli sinir ağ modelleri kullanılarak elde edilebilecek sonuçlar arasında bir karşılaştırma yapmak için iki klasik sinir ağı modeli önerilmiştir. İlk modelde, VGG16 iyi bilinen modeldeki katman bloklarından ilham alan bir evrişimli sinir ağı modeli önerilmiş ve oluşturulan gri tonlamalı görüntü veri kümeleri kullanılarak eğitilmiştir, (her katman bloğu, birden fazla evrişimli katmanından ve bir MaxPooling katmanından oluşan bir katman grubundan oluşmaktadır). İkinci modelde ise, klasik bir 2-D evrişimli katman tabanlı evrişimli sinir modeli önerilmiş ve oluşturulan gri tonlamalı görüntü veri kümeleri kullanılarak eğitilmiştir.

- Ayrıca, Residual Sinir ağı (ResNet) ve Inception V3 olmak üzere iki iyi bilinen derin öğrenme modeli, oluşturulan gri tonlamalı görüntü veri setleri kullanılarak test edilmiştir.

3.2. Gerçekleştirilen İlk Çalışma

Derin kamuflaj: Melezleştirilmiş enjeksiyon saldırıları ve şaşırtma tekniklerine karşı, Android anti virüs sistemlerinin sağlamlığının değerlendirilmesi [254]

3.2.1. Kullanılan Materyal

Aşağıdaki paragraflarda, tezin bu kısımda kullanılan malzemelerden bahsedilecektir.

3.2.1.1. Kullanılan Kötücül Yazılım Veri Kümeleri

Bu çalışmada kullanılan kamufle edilmiş kötücül yazılım veri kümeleri oluşturmak için, önerilen enjeksiyon saldırıları, üç iyi bilinen Android kötücül yazılım veri kümesine uygulanmıştır. İlk kullanılan kötücül yazılım veri kümesi, 179 farklı aileye ait 5.560 Android kötü amaçlı uygulama içeren olan Drebin [54] Android kötücül yazılım veri kümesidir. İkinci kullanılan veri kümesi, 49 farklı kötü amaçlı yazılım ailesine ait 1260 örnek içeren Malgenom [255] Android kötücül yazılım veri kümesidir. Kullanılan üçüncü kötücül yazılım veri kümesi, Malgenom kötücül yazılım veri küme örneklerine aşağıdaki gibi bazı şaşırtma teknikleri uygulanarak oluşturulan PRAGuard [110] Android kötücül yazılım veri kümesidir:

I. Yansıma Veri Kümesi (Reflection Dataset)

Bu veri kümesi, orijinal Malgenom veri kümesine kod çağırma yansıması uygulanarak oluşturulmuştur. Kod çağırma yansıması, kaynak kodda bulunan metotların çağırma talimatlarını aynı görevi gerçekleştiren başka talimatları ile değiştirilerek elde edilmektedir.

II. Dize Şifreleme Veri Kümesi

Bu veri kümesi, Malgenom örneklerinin kaynak kodunda bulunan dize değerleri şifrelenerek oluşturulmuştur. Dize şifreleme, kötücül yazılımdaki kaynak kodu ayrıştırıp, kodda bulunan tüm dize değerleri, XOR işlem tabanlı bir algoritma ile şifrelenerek elde edilmiştir. Ayrıca, uygulamayı orijinal formatında çalıştırmak için, yürütme zamanında düz dizelerdeki değerlerini elde etmek için bir şifreleme çözme işlevi kullanılmıştır.

III. Sınıf Şifreleme Veri Kümesi

Bu veri kümesinde, Malgenom örneklerindeki her kötücül yazılım uygulamasının kaynak kodunda bulunan sınıfları, GZIP algoritması kullanılarak sıkıştırılmıştır. Ardından her sınıfın kodu, tamamen şifrelenmiştir. Yürütme sırasında sınıfların kaynak kodunun şifresini çözmek için bir işlev geliştirilmiştir.

IV. Tam Şaşırtma Veri Kümesi

Bu veri kümesinde, kod çağırma yansıması, dize şifreleme ve sınıf şifreleme ile birlikte Malgenom veri kümesine uygulanmıştır.

Bu çalışmada, önerilen enjeksiyon saldırıları, şaşırtma teknikleriyle melezlenmesi için, PRAGuard veri kümesi, bir şaşırtma veri kümesi olarak kullanılmıştır. Her biri 49 kötücül yazılım örneği içeren 24 kamufle edilmiş veri kümesi oluşturmak için, önerilen enjeksiyon saldırıları, Malgenom ve PRAGuard veri kümelerinin her birine uygulanmıştır.

DroidKungFuSapp gibi bazı zararlı yazılımlardaki kodunun çıkarılamaması nedeniyle, bazı kamufle edilmiş veri kümeleri, 49 kötücül yazılım değil, 47 veya 46 kötücül yazılım örneği içermektedir. Bu sebeple bu çalışmada, kodu çıkarılabilen toplam kamufle edilmiş kötücül yazılım sayısı 1154 ‘tür. Ayrıca, Drebin Android kötücül yazılım veri setinden rastgele seçilen ve 22 aileye ait 100 kötücül yazılım örneklerine, önerilen enjeksiyon saldırıları uygulanmıştır. Bu veri kümesine enjekte

edilen kod miktarı arttırılarak, Anti virüs sistemlerinin performanslarının nasıl etkileneceğini analiz etmek için kullanılmıştır.

3.2.1.2. Virüstotal

Virustotal, 60'tan fazla ticari Anti-virüs sistemdeki veri tabanına bağlanarak, gönderilen her türlü datanın, kötü amaçlı içeriğe sahip olup olmadığını analiz eden çevrimiçi bir hizmettir. Bu çalışmada, önerilen saldırılar, McAfee, Symantec, ESET NOD32, Kaspersky, Dr.Web ve Avira olmak üzere altı tane iyi bilinen ve popüler olarak kullanılan anti virüs sistemdeki algılama doğruluğunu değerlendirmek için kullanılmıştır.

3.2.2. Önerilen Enjeksiyon Saldırıları

3.2.2.1. Uygulama Yeniden İmzalama Saldırısı

Android uygulamalar, son kullanıcı tarafından kullanılmak üzere yayınlanmadan önce, geliştirici tarafından kendi anahtarı kullanılarak imzalanmalıdır. Bazı anti-virüs sistemleri, kötü amaçlı uygulamaları tespit etmek için, sadece uygulama üzerinde bulunan sertifika bilgilerine bakmaktadır.

Bu nedenle, McAfee, Symantec, ESET NOD32, Kaspersky, Dr.Web ve Avira sistemlerinde, bu ilkel yöntemin, kullanılıp kullanılmadığından emin olabilmek için, bu Antivirüs sistemlerin algılama doğruluklarını, kötücül yazılımdaki orijinal sertifikaları değiştirilerek test edilmiştir.

3.2.2.2. Önerilen İzin Enjeksiyon Saldırısı

İşletim sistemdaki kaynakları veya uygulama bileşenleri korumak için, Android işletim sisteminde, izin tabanlı güvenlik diye bir platform kullanılmaktadır. Herhangi bir program, doğru izinlere sahip olmadığı sürece, belirli bir korumalı sistem kaynağına veya diğer uygulamalara ait bileşenlere erişememektedir. İzinler, Manifest.xml dosyasına yazılmakta ve bu izinler, yükleme sırasında kullanıcı tarafından onaylanmalıdır. İzinler, daha önce gerçekleştirilen kötücül yazılım

algılama çalışmalarındaki modelleri eğitmek için, bir özellik olarak sıklıkla kullanılmıştır. Bu nedenle, Kötücül yazılımların, Antivirüs sistemleri tarafından tespit edilmelerini önlemek için, Manifest dosyasına zararsız izinlerinin enjekte edilmesini önerilmiştir. Başka bir deyişle, iyi huylu bir uygulamanın izinleri ayıklanıp, kötücül yazılımların Manifest dosyasında bulunan izinler bölümüne enjekte edilmiştir.

3.2.2.3. Önerilen İzinler-Kod Enjeksiyon Saldırısı

Genelde, Android uygulamalar geliştirirken, Java programlama dili kullanılmaktadır. Ardından, java kodu, DEX bayt koduna derlenmektedir. Dex bayt kodu, cep telefonu gibi cihazlarda, sınırlı kaynak ortamlarında çalıştırılmak üzere hazırlanmış ve optimize edilmiş bir bayt kodudur. Ayrıca, Dex bayt kod, Dalvik sanal makinesi kullanılarak çalıştırılabilmektedir. Dex bayt kodunun okunması ve analiz edilmesi kolay olmadığından, Android uygulamanın kaynak kodunu analiz etmek için, Java ve Dex bayt kodu arasında orta bir kod sözdizimi olan Smali kodu kullanılmaktadır. Herhangi bir uygulamanın gerçek davranışını analiz etmek için, uygulamadaki kaynak kodu çok önemli olduğundan dolayı, bu çalışmada, algılama teknikleri atlatmak için kötü niyetli uygulamanın kaynak koduna iyi huylu bir iyi huylu bir uygulamanın kodunun enjekte edilmesini önerilmiştir. Bu saldırıda, kötücül yazılım kodu, son kamufle edilmiş kötücül yazılımdaki kaynak kodunun küçük bir oranı olacak şekilde, kötücül yazılım koduna büyük miktarda iyi huylu kodu enjekte edip, derin bir kamuflaj elde etmeyi amaçlamaktadır. Bu amaçla, Smali kod dosyaları, iyi huylu bir uygulamadan çıkartılmış ve kötü amaçlı yazılımın Smali koduna enjekte edilmiştir. Ayrıca, iyi huylu bir uygulamadaki Manifest dosyasında bulunan izinleri ayıklanmış ve kötü amaçlı yazılımdaki Manifest dosyasına enjekte edilmiştir.

3.2.3. Uygulanan Enjeksiyon Saldırı Senaryoları

Önerilen enjeksiyon saldırıları üç farklı senaryoda uygulanmıştır. Şekil 3.1‘deki Seviye 1'de gösterildiği gibi ilk senaryoda, uygulama yeniden imzalama saldırısı kullanılarak mevcut bazı kötücül uygulamalara hafif bir şaşırtma saldırısı uygulanmıştır. Yani uygulamaların orijinal sertifikaları, yeni bir sertifika ile

değiştirilmiştir. Şekilde gösterildiği gibi, ilk olarak, kötücül APK arşivlerinin orijinal kaynak kodunu elde etmek için, ters derlenme yapılmıştır. Ardından, uygulamanın sertifika bilgilerini içeren META-INF klasörünün içeriği silinmiştir. Bundan sonra, uygulama yeniden derlenmiş, yeni bir sertifika oluşturulmuş ve oluşturulan sertifikayı kullanarak uygulama yeniden imzalanmıştır.

Şekil 3.1-Seviye 2 de gösterildiği gibi, ikinci uygulanan saldırı senaryosunda, uygulama yeniden imzalama saldırısı ile beraber izin enjeksiyon saldırısı uygulanmıştır. Bu amaçla, uygulama üzerinde, ters derleme yapılmış ve META-INF klasörünün içeriğini silinmiştir. Bundan sonra, iyi huylu izinler listesi oluşturmak için, iyi huylu bir uygulamanın Manifest dosyası ayrıştırılmıştır. Daha sonra, toplanan iyi huylu izinler, kötü amaçlı yazılımın Manifest dosyasının izinler bölümüne enjekte edilmiştir. Son olarak, kötü amaçlı yazılımın APK arşivi geri derlenmiş olup, yeni bir sertifika kullanılarak yeniden imzalanmıştır.

Şekil 3.1- Seviye 3 de gösterildiği gibi, uygulanan üçüncü saldırı senaryosunda, uygulama yeniden imzalama saldırısının yanı sıra, izin-kod enjeksiyon saldırısı da uygulanmıştır. Bu nedenle, öncelikle kötücül uygulamanın derlemesi çözülmüş ve META-INF klasörünün içeriğini silinmiştir. Ayrıca, iyi huylu uygulamadaki Manifest dosyası da ayrıştırılıp, iyi huylu izinler listesini oluşturulmuş ve kötücül uygulamasındaki Manifest dosyasına enjekte edilmiştir. Bundan sonra, iyi huylu APK arşivinin içeriği ayrıştırılarak, iyi huylu Smali kodu toplanmış ve toplanan iyi huylu Smali kod, kötücül uygulamanın Smali koduna enjekte edilmiştir. Son olarak, kötücül uygulama tekrar derlenmiş ve yeni bir sertifika kullanılarak yeniden imzalanmıştır.

Bu çalışmada, Telegram v4.8.9, iyi bilinen sohbet uygulaması, önerilen saldırıları uygulamak için, iyi huylu bir uygulama olarak kullanılmıştır. Burada, Telegram'ın, çok sayıda izinler ve büyük miktarda Smali kodu içermesinden başka herhangi bir husus için kullanılmadığını belirtmek gerekmektedir. Algoritma 3.1, önerilen saldırıları kullanılarak uygulanan senaryolarının sözde kodunu göstermektedir, ve algoritma 3.2, VirusTotal API kullanarak oluşturulan veri kümeleri test etmek için kullanılan algoritmayı göstermektedir.

Girdi: datasetPath, APKReBuildPath, apktoolPath, keystorePath, benignAppPath.
Çıktı: Kamufle edilmiş kötücül yazılım veri kümesi.

```
benignPermissions = benignPermissionCollector(benignAppPath)
benignCode = smaliCodeCollector(benignAppPath)
DatasetPath'teki her kötücül yazılım örneği için:
    ApkDecompile(samplePath, sampleExtractionPath, apktoolPath)
    # Mükerrer izinlerin silinmesi
    refinePermissions(maliciousAppPermissions, benignPermissions)
    permissionsInjector(sampleExtractionPath, benignPermissions, Ratio)
    # Ratio: enjekte edilen izinlerin sayısının, kötü amaçlı uygulamanın orjinal
      izinlerinin sayısına oranıdır.
    smaliCodeInjector(sampleExtractionPath, benignCode, Ratio)
    # Ratio: enjekte edilen kodun, kötü amaçlı uygulamanın orijinal koduna
       oranıdır.
    ApkReBuilding(sampleExtractionPath, APKReBuildPath, apktoolPath)
    APKSign(APKReBuildPath, keystorePath)
```

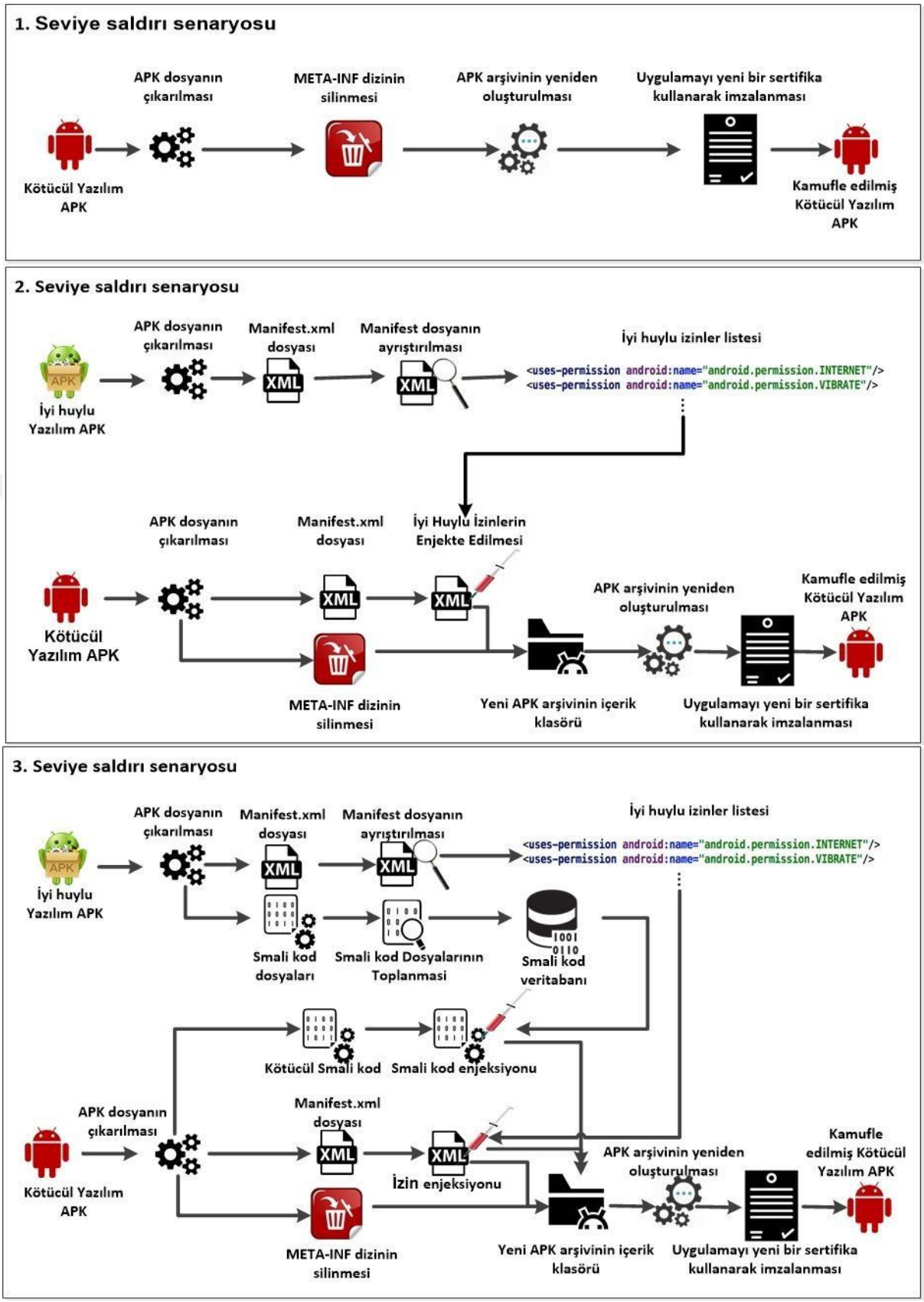
Algoritma 3.1. Önerilen saldırı senaryoları uygulamak için kullanılan algoritması.

Girdi: APKArchivesPath
Çıktı: scaningReports

```
# Kötücül yazılımları taramak üzere, VirusTotal API hizmetine tarama isteklerinin
    gönderilmesi
APKArchivesPath'deki her kötücül yazılım örneği için:
        myVirusTotalAPI.scan_file(samplePath)
        time.sleep(30)

# VirusTotal API hizmetinden tarama raporların istenilmesi
APKArchivesPath'deki her kötücül yazılım örneği için:
        hashValue = fileChecksum(samplePath, "sha256")
        response =myVirusTotalAPI.get_file_report(hashValue)
        rawJSONData = json.dumps(response, sort_keys=True, indent=4)
        JSONDictionary = json.loads(rawJSONData)
        reportAnalysis(JSONDictionary)
```

Algoritma 3.2. Oluşturulmuş kamufle edilmiş kötücül yazılım veri kümelerine karşı Anti virüs sistemlerinin performansını değerlendirmek için kullanılan algoritması.



Şekil 3.1. Önerilen enjeksiyon saldırıları kullanılarak uygulanan saldırı senaryoları.

3.3. Gerçekleştirilen İkinci Çalışma

İkinci çalışma kapsamında iki alt çalışma gerçekleştirilmiştir [135, 136]. Birinci alt çalışmada, kötü amaçlı yazılım örnekleri birden çok kötücül yazılım ailesine sınıflandırmak için VisDroid adlı bir model önerilmiştir. İkinci alt çalışmada ise, DeepVisDroid adlı derin öğrenme tabanlı bir model önerilmiş ve birinci alt çalışmada önerilen VisDroid modeli ve diğer bazı modellerle beraber Android uygulamaları iyi huylu veya kötü amaçlı yazılım olarak sınıflandırmak için kullanılmıştır.

3.3.1. Kullanılan Materyal

Bu bölümde, önerilen sınıflandırma modellerde kullanılan metodolojilerinden ve tekniklerinden bahsedilmiştir.

3.3.1.1. Kötücül Yazılımının Gri Tonlamalı Görüntüye Dönüştürülmesi

İlk olarak, kötücül yazılım APK arşivlerinin derlemesi çözülmüş ve içerikleri çıkartılmıştır. Sonra, önerilen görüntü veri kümeleri oluşturmak için, kullanılan APK arşivlerindeki dosyaları, doğalarına veya biçimine bakmaksızın ikili dosya olarak kabul edilmiştir. İstenilen APK içerikleri ikili bit akışı olarak okunmuş ve bayt matris olarak saklanmıştır. Bu çalışmada oluşturulan veri kümelerindeki kötücül yazılım kaynağı temsil etmek için gri tonlamalı gösterimi kullanılmıştır. Gri tonlamalı görüntüdeki piksel değerleri, 0 ile 255 değişmekte, böylece 0 değeri siyah rengi temsil eder iken 1 değeri beyaz rengi temsil etmektedir. Oluşturulan bayt matristeki her bir bayt 0 ile 255 arasında bir değer depolayabildiği için, matristeki her bayt, oluşturulan son gri tonlamalı görüntüdeki bir piksele dönüştürülmüştür. Bayt matrisi, 0 ile 255 arasında değerlere sahip bir matrisine dönüştürdükten sonra, son matrisi, gri tonlamalı bir görüntü olarak kaydedilmiştir. Önerilen veri kümelerindeki tüm yapılandırılmış görüntüleri, 256 piksel sabit genişliğe sahip iken, onların uzunlukları dosya boyutuna göre değişmektedir. Ancak, Manifest.xml ve Resources.arsc dosyalardaki boyutunun sınırlamalar nedeniyle, bu iki dosyaları kullanılarak oluşturulan görüntüler, 64 piksel genişliğine sahip olacak şekilde oluşturulmuştur.

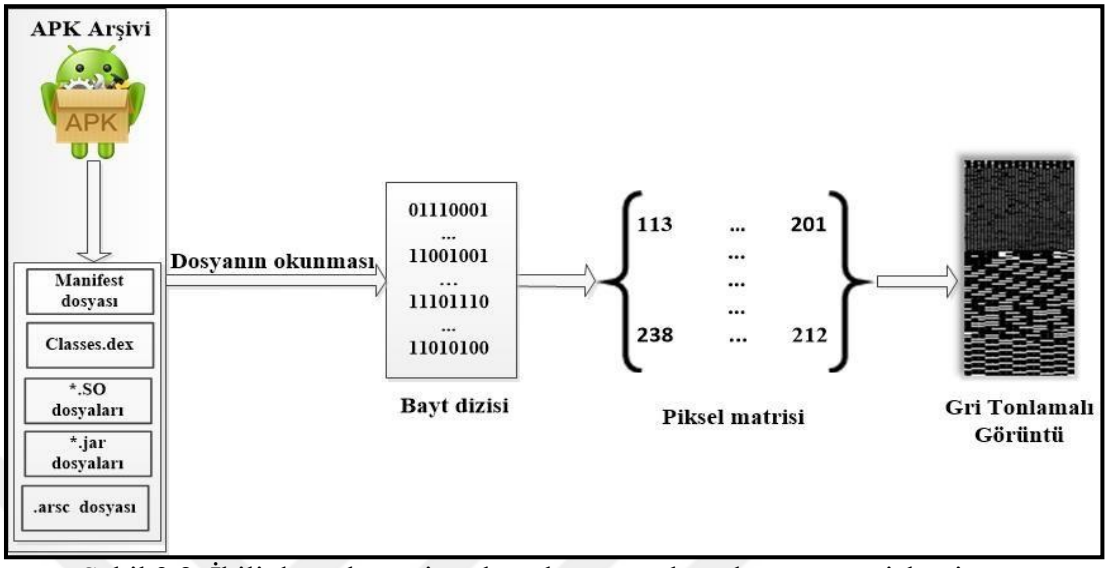
Şekil 3.2, kötü amaçlı yazılım örneğinin, gri tonlamalı bir görüntüye dönüştürme işlemini göstermektedir.

3.3.1.2. Önerilen ve Oluşturulan Görüntü Veri Kümeleri

Bu çalışmada, önerilen görüntü veri kümeleri oluşturmak için daha önce bahsedilen Drebin ve Malgenom kötücül yazılım veri kümeleri kullanılmıştır. Dolayısıyla, başlangıçta, kullanılan kötücül yazılım veri kümesinin toplam uygulama sayısı 6820 örnektir. Ancak, söz konusu olan kötücül yazılım veri kümeleri (Drebin ve Malgenom) bazı benzer kötü amaçlı uygulamaları içerdiğinden dolayı, kullanılan nihai veri kümesinde tekil kalacak şekilde, mükerrer olan kötücül yazılım örnekleri ihmal edilmiştir. Ayrıca, kullanılan kötü amaçlı yazılım aileler arasında bir miktar homojenlik sağlamak için 100'den az sayıda içeren kötücül yazılım aileleri de atlanmıştır. Sadece 100'den fazla sayıda içeren kötücül yazılım aileleri seçmemize rağmen, kullanılan nihai kötü amaçlı yazılım veri kümesi, uygulama sayısı açısından halen dengesizdir. Örneğin, ADRD ailesi, yalnızca 108 kötücül yazılım içermiş iken, DroidKungFu ailesi, 1103 kötücül yazılım içermiş ve FakeInstaller ailesi, 902 kötücül yazılım içermiştir. Ancak, 100 'den az sayıda uygulamaya sahip kötücül yazılım aileleri filtreledikten sonraki kalan kötücül yazılım ailelerin oldukları gibi görüntü veri kümeleri oluşturmak için girdi olarak kabul edilmiştir. Yukarıdaki işlem sonucunda, ilk sınıflandırma alt çalışmada kullanılan son kötücül yazılım veri kümesi, 12 farklı aileye ait 4850 örnek içermiştir.

Öte yandan, ikinci sınıflandırma alt çalışmasında, yukarıda belirtilen 4850 kötü amaçlı yazılım örneği, kötücül yazılım veri küme olarak kullanılmıştır. Ayrıca, 4850 iyi huylu uygulama, APKPure ücretsiz çevrimiçi hizmeti kullanılarak Google uygulama mağazasından indirilmiş ve ikinci sınıflandırma alt çalışmada iyi huylu veri küme olarak kullanılmıştır. İndirilen iyi huylu uygulamaların, iyi huylu olup olmadıklarından emin olmak için VirusTotal çevrimiçi API'yi kullanılmıştır, bu amaç ile Python tabanlı bir yazılım geliştirilmiştir. Bir uygulama, yalnızca VirusTotal hizmet altında barındırılan 60 'tan fazla Anti virüs sisteminden herhangi bir sistem tarafından algılanamadığı takdirde, iyi huylu olarak kabul edilmiştir. İyi huylu veri seti indirmek ve taramak için kullanılan algoritma, şekil 3.3'te

gösterilmektedir. Oluşturulan görüntü veri setleri aşağıdaki paragraflarda açıklanmıştır.



Şekil 3.2. İkili dosyaları gri tonlamalı görüntülere dönüştürme işlemi.

Manifest, Classes.dex, *.SO, *.Jar ve Resource.arsc dosyaları, herhangi bir Android APK arşivi ayıklanarak elde edilebilen içeriklerdir.

I. Manifest Dosyası Tabanlı Görüntü Veri Kümesi

Manifest.xml dosya, Android uygulamadaki en önemli dosyalarından biridir. Manifest dosyası, herhangi bir Android uygulama çalıştırırken sistem tarafından okunan ilk dosyadır. Ayrıca, bu dosya, uygulamanın nasıl yürütüleceğini ve nasıl davranacağını belirten bir yol haritası olarak kabul edilmektedir. Dolayısıyla, bu çalışmada, önerilen sınıflandırma modellerinin eğitiminde kullanılmak üzere, Manifest dosyalarının gri tonlamalı görüntülere dönüştürülmesini önerilmiştir. Manifest dosya boyutundaki sınırlama nedeniyle, bu dosyaları temsil etmek için, 64 piksel genişliğine ve dosya boyutuna göre değişen yüksekliğine sahip gri tonlamalı bir görüntüye dönüştürülmüştür. Manifest dosyayı temsil etmek için, 256, 128 ve 64 dahil olmak üzere birden fazla genişliği test ettikten sonra deneysel olarak 64 piksellik görüntü genişliği seçilmiştir. Aynı kötücül yazılım ailesine ait Android uygulamalarının Manifest dosyalarına dayalı görüntüler arasındaki büyük benzerliği, Şekil 3.4.a'de gösterilmiştir.

II. Manifest ve Resources.Arsc Dosyalar Tabanlı Veri Kümesi

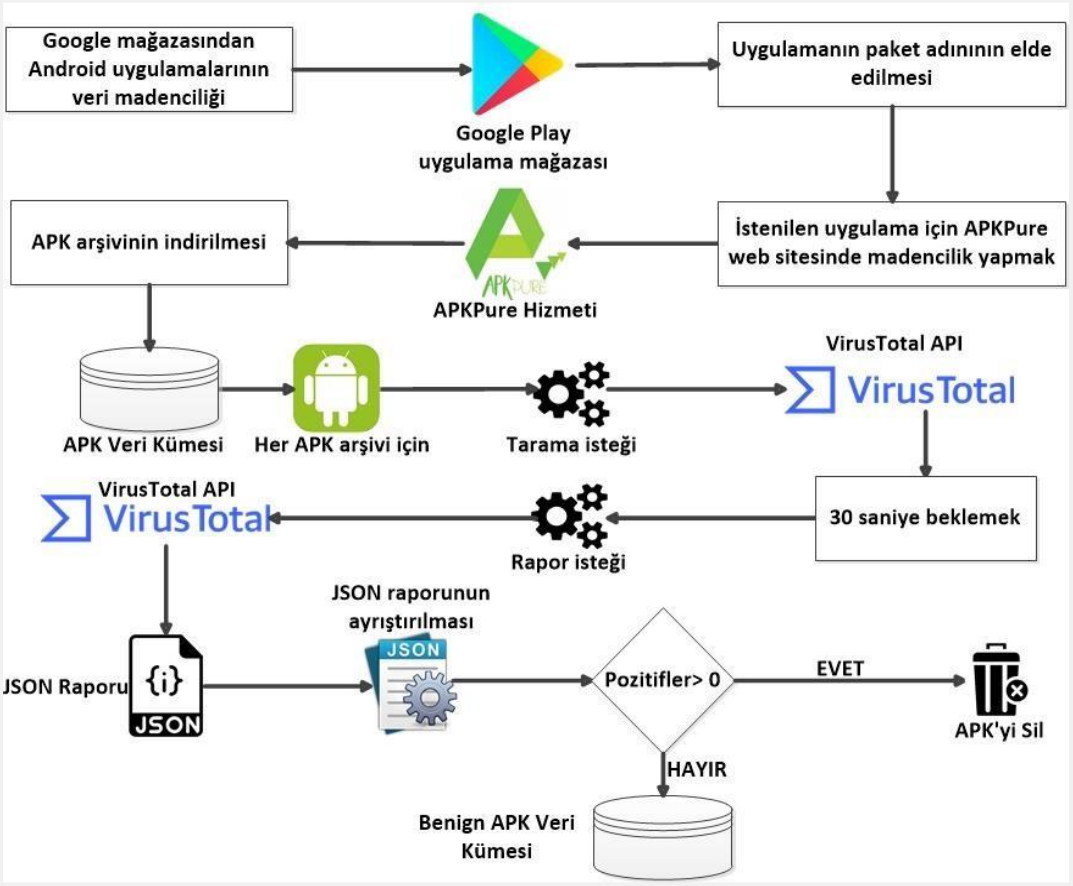
Bu veri kümede, her kötü amaçlı yazılım örneğinin Manifest.xml ve Resources.arsc dosyaları, bir gri tonlamalı görüntüsüne dönüştürülmüştür. Bu veri kümede oluşturulan görüntülerin genişliği 64 pikselde sabit tutulmuş iken, uzunlukları Manifest ve Resources.arsc dosyalarının boyutuna göre değişecek şekilde ayarlanmıştır. Oluşturulan son görüntüde, Manifest bölümü, Resources.arsc bölümünden 10 sıfır dolgu (10 satır siyah pikseli) kullanılarak ayrılmıştır. Şekil 3.4.b, bu yöntem kullanılarak oluşturulan aynı aileye ait iki görüntüyü göstermektedir.

III. DEX Dosya Tabanlı Veri Kümesi

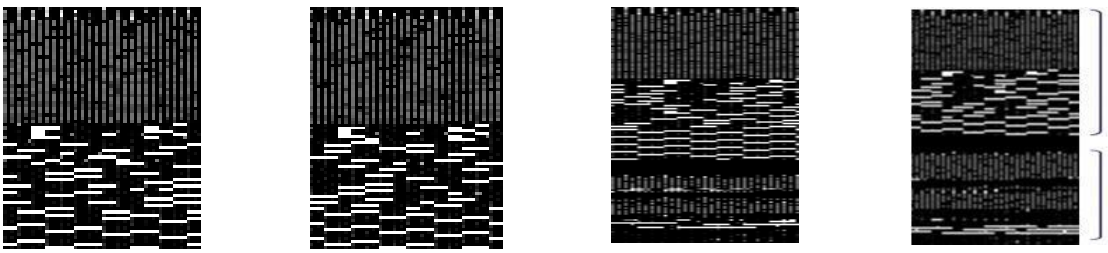
Önerilen üçüncü veri kümedeki gri tonlamalı görüntüler, kötücül yazılım örneklerinden çıkartılan DEX kod dosyaları kullanılarak oluşturulmuştur. Bu veri kümesi, Android uygulamaların sınıflandırılmasında, ham DEX bayt kodu kullanılarak oluşturulan gri tonlamalı görüntülerin etkinliğini test etmeyi amaçlamaktadır.

IV. Manifest, Resources.Arsc ve DEX Dosyalar Tabanlı Veri Kümesi

Adından da anlaşılacağı gibi, bu veri kümesindeki görüntüleri oluşturmak için, Manifest, Resouces.arsc ve DEX dosyaları kullanılmıştır. Bu veri kümede, oluşturulan görüntülerdeki genişliği 256 pikseldir. Manifest, Resouces.arsc ve DEX dosyalarının son görüntülerindeki bölümlerini, sıfır dolguları kullanılarak birbirinden ayrılmıştır.



Şekil 3.3. Kullanılan iyi huylu veri kümesi oluşturmak için kullanılan algoritma.



a. ADRD kötücül yazılım ailesinin Manifest dosyası tabanlı görüntüleri.

b. DroidKungFu ailenin Manifest ve .ARSC dosyalar tabanlı görüntüleri.

Şekil 3.4. Aynı kötü amaçlı yazılım ailelerine ait kötü amaçlı yazılım örneklerinden oluşturulan görüntüler arasındaki benzerliği.

3.3.1.3. Kullanılan Görüntü Tabanlı Özellikler

Görüntü tabanlı özellikler, nesne tanıma ve nesne algılama gibi çeşitli uygulamalarda kullanılmaktadır. Genel olarak, görüntü özellikleri, global özellikler ve lokal özellikler olmak üzere iki tip özelliklere ayrılmaktadır. Görüntü global özellikler, görüntüyü bir bütün olarak tanımlamaktadır. Başka bir deyişle, bu tür özellikler, görüntünün tümünü bir özellik vektörü kullanılarak tanımlamaktadır. Öte yandan, lokal özellikler, görüntü, bir nesne yığınları olarak tanımlamaktadır. Başka bir deyişle, bu tür özellikler, görüntüdeki çok sayıda farklı anahtar noktayı tespit etmeye ve bunların her birini uygun bir tanımlayıcı kullanarak açıklamaya dayanmaktadır. Dolayısıyla, lokal özellikleri kullanıldığında, her bir görüntü birden çok özellik vektörü (tanımlayıcılar) kullanılarak temsil edilmektedir. Neredeyse tüm makine öğrenme sınıflandırıcıları, n tane numune n tane özellik vektörü girdi olarak kabul ettiğinden dolayı, lokal özellik tanımlayıcıları, makine öğrenme sınıflandırıcılarına doğrudan girdi olarak kullanmak mümkün olmamaktadır (çünkü her bir görüntü örneği birden fazla tanımlayıcı vektörü kullanarak temsil edilmektedir). Bu nedenle, lokal özellikler, makine öğrenme sınıflandırıcılarına girdi olarak kullanılabilmesi için görsel sözcük çantası (GSÇ) gibi bazı teknikleri kullanılmaktadır. Global özelliklerle karşılaştırıldığında, lokal özelliklerin dağınıklığa ve tıkanmaya (Clutter ve Occlusion) karşı daha sağlam olduğunu bilinmektedir [256]. Bu çalışmada, önerilen modelleri eğitmek için, üç farklı global özellik ve dört farklı lokal özellik ayrı ayrı kullanılmıştır. Ayrıca, bu iki tür özelliğin birbirini tamamlayabileceği düşüncesiyle, bir topluluk oylama sınıflandırıcıyı eğitmek için, lokal özellikleri, global özellikleriyle melezleştirilmiştir. Başka bir deyişle, global özellikler, görüntüye genel bir bakış sağlayabilir iken, lokal özellikler, görüntüdeki her bir nesne hakkında ayrıntılı bilgi verebilmektedir. Bu nedenle, global özellikler, bir keşif (Exploration) rolü üstlenirken, lokal özellikleri, bir sömürü (Exploitation) rolü oynayabilmektedir. Sonraki iki paragrafta, kullanılan global ve lokal özellikler hakkında bazı ayrıntıları verilecektir.

I. Kullanılan Görüntü Tabanlı Global Özellikleri

Bu çalışmada üç farklı görüntü tabanlı global özellik kullanılmıştır; bu bölümde, her biri kısaca açıklanacaktır.

1) Renk Histogram

Genel olarak histogram, farklı yüksekliklere sahip çubukları kullanarak verilerin grafiksel gösterimidir. Renk histogram, görüntüdeki piksel yoğunluklarının dağılımını temsil eden global bir özelliktir. Bu çalışmada, gri tonlamalı görüntü temsili kullanıldığından, piksel yoğunluğu değerleri 0 ile 255 arasında değişmektedir. Başka bir deyişle, renk histogram, her çubuğun belirli bir yoğunluğa sahip piksel sayısı temsil ettiği şekilde bir dizi çubuk içermektedir. Bu özellik Denklem (1)'deki gibi tanımlanmıştır.

$$\boldsymbol{H} = \{\boldsymbol{H}[1], \boldsymbol{H}[2], \ldots, \boldsymbol{H}[N]\} \qquad (1).$$

H[i]: bin_i'deki piksel sayısıdır.
N: çubuk sayısıdır;
H: İstenilen görüntü için renk histogram vektörüdür.

2) Hu Anlar

Görüntü anlar, görüntüdeki piksellerin yoğunluklarının ağırlıklı ortalamasıdır [257], ve en basit formatı, Denklem (2)'de gösterildiği gibidir.

$$\boldsymbol{M} = \sum_x \sum_y \boldsymbol{x}^i \boldsymbol{y}^j I(\boldsymbol{x}, \boldsymbol{y}) \qquad (2)$$

Burada $I(x, y)$: görüntüdeki (x, y) konumundaki piksel yoğunluğudur (0 veya 1 bir değer alabilmektedir).

Hu anları; çevirme, ölçekleme, yansıtma ve döndürmeden etkilenmemektedir. Hu anları, merkezi anlar kullanılarak hesaplanan yedi sayıdan oluşmaktadır. Merkezi moment denklem (3) kullanılarak hesaplanmaktadır.

$$\mu_{ij} = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty}(x-\bar{x})^{i}(y-\bar{y})^{j} f(x,y)\,dx\,dy \qquad (3)$$

Burada: $i, j = 1,2,3\ldots$ $\bar{x} = \frac{m_{10}}{m_{00}}$, $\bar{y} = \frac{m_{01}}{m_{00}}$

Hu momentteki yedi sayısının hesaplama formülleri ve merkezi moment hesaplaması hakkındaki ayrıntılı bilgileri, [258]'de bulunabilmektedir.

3) Haralick Doku (Haralick Texture)

Doku, görüntülere göz atmak ve görüntüden bilgi almak için oldukça kullanışlı bir özelliktir. Ayrıca, doku özellik tanımlayıcısı, Smoothness, Coarseness, and Regularity gibi görüntüdeki özellikleri ölçmektedir [259]. Haralick dokuları, görüntü sınıflandırma alanında kullanılan en ünlü doku özelliklerinden biridir. Eş-oluşum matrisi (Co-occurrence matrix), Haralick doku özelliği hesaplamasında kullanılmaktadır. Haralick dokusu kullanılarak elde edilen öznitelik vektörleri, Fark Varyansı, Ters Fark Momenti, Entropi ve Enerji Dahil olmak üzere 13 öznitelikten oluşmaktadır [260]. Haralick doku özellik vektöründe kullanılan bileşenleri ile ilgili detaylı ayrıntıları, [98] 'te bulunabilmektedir.

II. Kullanılan Görüntü Tabanlı Lokal Özellikleri

Lokal özellikler, belirli desenleri veya karakteristik yapıları gibi, görüntüdeki ayrıntıları açıklayan her tür özelliği ifade etmektedir. Lokal özellikler, görüntüdeki küçük yamaları, kilit noktaları veya kenarları hakkında bilgi verebilmektedir. Bu tür özellikleri, görüntüdeki karakteristik alanlar tespit etmek için anahtar nokta detektörlerinin kullanılmasına dayanmaktadır. Sonra bu tür özellikte, tespit edilen karakteristik noktaları tanımlamak için özellik tanımlayıcıları kullanılmaktadır. Bu çalışmada, dört farklı yerel özellik kullanılmış; her biri kısaca anlatılacaktır.

1) Ölçekten Bağımsız Öznitelik Dönüşümü ÖBÖD (Scale-Invariant Feature Transform SIFT)

Bu algoritma 2004 yılında Lowe tarafından önerilmiştir [261]. ÖBÖD algoritması, ölçeklendirme ve döndürme ile etkilenmeyen özellikleri ayıklamayı amaçlamaktadır. ÖBÖD algoritmasının uygulanması, görüntüdeki karakteristik noktaları bulmak için ölçek alanı zirvelerinin seçilmesiyle başlamaktadır. Başka bir deyişle, bu algoritma, özelliklerin bulunabileceği potansiyel konumları tespit etmekle başlamaktadır. Bundan sonra, tespit edilen karakteristik noktaları, doğru bir şekilde lokalize edilmektedir ve her birine yönelim atanmaktadır. Bundan sonra, tespit edilen her bir anahtar noktası, 128 bit boyutuna sahip bir vektör kullanılarak tanımlanmaktadır. ÖBÖD algoritması, farklı ölçek değerlerinde (çoklu σ değerleri) Gauss'lu Laplacian (LoG) 'ının hesaplanmasına ve en iyi sonuçları veren ölçeğin seçilmesine dayanmaktadır. ÖBÖD algoritmasında, karakteristik noktaları, Gauss'lu Laplacian ölçek alanındaki lokal ekstrem bölgeleri (lokal minimum / maksimum) kullanarak belirlenmektedir. Böylece, farklı ölçekte birden fazla Gauss'lu Laplacian hesaplanır ve görüntüdeki her piksel için en iyi sonuçları veren σ değeri seçilmektedir. Gauss'lu Laplacian'ın hesaplanması biraz maliyetli olduğundan, ÖBÖD algoritmasında Gauss'lu Laplacian'ın yerine yaklaşık Gauss'lu Laplacian'ı kullanılmaktadır. Denklem (4), yaklaşık Gauss'lu Laplacian'ın nasıl hesaplanabildiğini göstermektedir.

$$\sigma\Delta^2 G = \frac{\partial G}{\partial \sigma} \approx \frac{G(x,y,k\sigma) - G(x,y,\sigma)}{k\sigma - \sigma} \qquad (4)$$

Burada: $G(x, y, \sigma)$, σ ölçeği için (x, y) konumundaki Gauss'lu Laplacian'dır.

$G(x, y, k\sigma)$, $k\sigma$ Ölçeği için (x, y) konumundaki Gauss'lu Laplacian'dır, $k\sigma$: σ'dan biraz daha büyük bir ölçektir.

$$G(x, y, k\sigma) - G(x, y, \sigma) \approx (k-1)\,\sigma^2.\Delta^2.G \qquad (5)$$

Burada: $G = \frac{1}{2\pi k\sigma^2} . e^{\frac{-(x^2+y^2)}{2k^2\sigma}}$

Özet olarak, ÖBÖD özelliğin arkasındaki fikir, Gauss'lu Laplacian'ın yerine Gauss'lu Laplacian'ın farkı kullanmaya dayanmaktadır, çünkü Gauss'lu Laplacian'ın fark hesaplanması, Gauss'lu Laplacian'ın hesaplanmasından daha basittir ve Gauss'lu Laplacian, yaklaşık olarak Gauss'lu Laplacian'ın farkına eşittir.

2) Hızlandırılmış Sağlam Öznitelikler HSÖ (Speeded Up Robust Features SURF)

Bu algoritma, ÖBÖD algoritmasına alternatif olarak tanıtılmıştır, çünkü ÖBÖD algoritması oldukça yavaştır [262]. HSÖ ve ÖBÖD arasındaki temel fark, Gauss'lu Laplacian'ı hesaplamak için kullanılan yöntemdir. HSÖ algoritmasında, Gauss'lu Laplacian'ı kutu filtresini kullanılarak hesaplanmaktadır. ÖBÖD algoritmasında olduğu gibi, HSÖ algoritması da Gauss ölçeği-uzay analizine dayanmaktadır, bu amaçla, görüntü integrali hesaplanarak görüntüdeki karakteristik noktaları tespit edilmektedir. Burada, görüntü integrali, girilen görüntüdeki tüm piksellerin yoğunluk değerlerinin toplamıdır (Denklem (6) kullanılarak).

$$I_{\sum}(x) = \sum_{i=0}^{i \le x} \sum_{j=0}^{j \le y} I(i,j) \qquad (6)$$

Ayrıca, kilit noktaları (karakteristik noktalar) algılama hızı artırmak için HSÖ algoritması hem ölçek hem de konum için Hessian matrisinin belirleyicisine dayanmaktadır. Belirli bir noktadaki Hessian Matrisi, Denklem (7) kullanılarak hesaplanmaktadır.

$$\boldsymbol{H}(f(x,y)) = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial x \partial y} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix} \qquad (7)$$

Başka bir deyişle, görüntü bir Gauss çekirdeği tarafından filtrelenmekte, böylece belirli bir X = (x, y) noktası için σ ölçeğindeki X'deki Hessian matrisi denklem (8) olarak tanımlanmaktadır.

$$\mathcal{H}(x, y) = \begin{pmatrix} L_{xx}(x, \sigma) & L_{xy}(x, \sigma) \\ L_{xy}(x, \sigma) & L_{yy}(x, \sigma) \end{pmatrix} \quad (8)$$

Burada $L_{xx}(x, \sigma)$, x noktasındaki Gauss'ın ikinci dereceden türevi temsil etmekte ve benzer şekilde $L_{xy}(x, \sigma)$ ve $L_{yy}(x, \sigma)$.

HSÖ algoritması, oryantasyonu belirlemek için yatay ve dikey yönde Gauss ağırlıkları ve Haar dalgacık yanıtları kullanmaktadır. Gauss ağırlıkları ve Haar dalgacık yanıtları, 6*s*'lik dairesel bir komşu bölgesi için hesaplanmaktadır; burada, s, karakteristik noktanın içinde bulunduğu ölçek değeridir. HSÖ algoritması, dairesel komşu bölgesi için dalgacık yanıtlarına göre tespit edilen anahtar noktaları (karakteristik noktaları) tanımlamak için 64-bit bir tanımlayıcı (Descriptor) kullanmaktadır.

3) KAZE Lokal Özellik

Bu algoritma, doğrusal olmayan ölçek uzaylarındaki özellikleri tespit etmek ve tanımlamak için bir algoritma olarak 2012 yılında Alcantarilla ve arkadaşları tarafından tanıtılmıştır [263]. Bu algoritma, her yeni oktavda herhangi bir alt örnekleme gerçekleştirmeksizin görüntünün orijinal çözünürlüğünde çalışmaktadır. KAZE detektörü, Hessian matrisin determinantının, farklı ölçek seviyelerde hesaplanmaya ve normalleştirilmeye dayanmaktadır. Lokal minima/maxima (Lokal Extremum), mevcut, üst ve alt ölçeklerde $\sigma i \times \sigma i$ boyutunda dikdörtgen bir pencereden seçilmektedir. HSÖ algoritmasında olduğu gibi, dönüşle değişmeyen tanımlayıcıları elde etmek için baskın oryantasyon (Dominant Orientation), tespit edilen her karakteristik noktanın etrafındaki $6\sigma i$ yarıçaplı dairesel bir alanda bulunmaktadır. Bu algoritmadaki tanımlayıcı uzunluğu 64 bittir.

4) Yönlendirilmiş FAST ve Döndürülmüş BRIEF YDB (Oriented FAST ve Rotated BRIEF ORB)

Adından da anlaşılacağı gibi, YDB [264], FAST iyi bilinen karakteristik nokta detektörüne [265] ve BRIEF tanımlayıcısına [266] dayanan hızlı bir görüntü ikili

lokal özelliktir. YDB özellik, dönüşle değişmez ve gürültüye dayanıklı bir görüntü lokal özelliği olarak bilinmektedir. YDB algoritması, karakteristik noktaları tespit etmek için FAST-9 karakteristik nokta detektörü kullanmaktadır ve seçilen karakteristik noktaları düzenlemek ve bunlardan en üst N noktayı seçmek için Harris köşe algoritması kullanmaktadır. YDB algoritması, görüntünün piramit ölçekteki her seviyede Harris köşe algoritması uygulayarak FAST özellikleri üretmeye dayanmaktadır. Ayrıca, YDB algoritması, tespit edilen karakteristik noktaları tanıtmak için, denklem (9) kullanarak gerçekleştirilen ikili yoğunluk test seti ile hesaplanan BRIEF tanımlayıcısı kullanmaktadır.

$$\tau(\boldsymbol{P}; \boldsymbol{x}, \boldsymbol{y}) = \begin{cases} \mathbf{1} : \boldsymbol{P}(\boldsymbol{x}) < \boldsymbol{p}(\boldsymbol{y}) \\ \mathbf{0} : \boldsymbol{P}(\boldsymbol{x}) \geq \boldsymbol{P}(\boldsymbol{y}) \end{cases} \quad (9)$$

Burada: P, düzleştirilmiş bir görüntü nesnesidir.
$P(x)$, bir x noktasında P 'nin yoğunluğudur.

3.3.1.4. Görsel Sözcük Çantası Gösterimi GSÇ (Bag Of Visual Words BOVW)

Bu yaklaşım başlangıçta doğal dil işleme (Natural Language Processing) alanında kullanılmış ve Sözcük Çantası (Bag Of Words) olarak adlandırılmıştır. Aynı kavram, yaygın olarak görüntü sınıflandırma alanında kullanılmaktadır ve Görsel Sözcük Çantası veya Görsel Özellikler Çantası (GSÇ) olarak adlandırılmaktadır. GSÇ gösteriminde, kullanılan görüntü özelliğine bağılı olarak ayıklanan tanımlayıcıları birden fazla kümeye bölünmekte ve her bir görüntü, tanımlayıcı frekans histogramı (görsel sözcükler) olarak temsil edilmektedir. GSÇ birden çok adımdan oluşmaktadır:

1) İlk adımda, görüntüdeki karakteristik noktaları tespit edilmekte ve tespit edilen karakteristik noktalarına dayanarak görüntü tanımlayıcıları oluşturulmaktadır (herhangi bir görüntü lokal özellik algoritması kullanılarak).
2) İkinci adımda, K-Means algoritması gibi herhangi bir kümeleme algoritması, oluşturulan görüntü lokal özellik tanımlayıcıları çoklu kümeler halinde kümelemek için kullanılmaktadır.

3) Son adımda, görüntüdeki görsel sözcüklerin sıklığına (her tanımlayıcı kümesine ait tanımlayıcıların sıklığına) bağlı olarak her görüntü için bir histogram oluşturulmaktadır.

Böylece, görüntü veri setindeki görüntü sayısına göre bir frekans histogramı veri kümesi oluşturulmaktadır ve istenilen makine öğrenme modelleri eğitmek ve test etmek için kullanılmaktadır. Bu histogramların her birine bir Görsel Sözcük Çantası denilmektedir.

3.3.1.5. Kullanılan Geleneksel Makine Öğrenme Sınıflandırıcıları

I. K-En Yakın Komşu (K-Nearest Neighbour KNN)

K-en yakın komşu algoritması (KNN), hem sınıflandırma hem de regresyon problemleri çözmek için kullanılabilen basit, uygulaması kolay, denetimli bir makine öğrenme algoritmasıdır. K en yakın komşu, mevcut tüm veri örnekleri depolayan ve yeni veri örnekleri bir benzerlik ölçüsüne (ör. Uzaklık fonksiyonları) göre sınıflandıran basit bir algoritmadır. Bu algoritmada, yeni bir veri örnek, ona en yakın komşularının çoğunluk oyuyla sınıflandırılmaktadır. Başka bir deyiş ile, herhangi veri örneğin depolanan veri örnekleriyle arasındaki mesafeyi bir mesafe fonksiyonu kullanılarak ölçülmekte ve söz konusu olan veri örneği, en yakın K komşularının arasında en yaygın olan sınıfa atanmaktadır.

II. Karar Ağacı (Decision Tree)

Karar Ağaç algoritması, denetimli öğrenme algoritma ailesine aittir. Karar ağaç algoritması, regresyon ve sınıflandırma problemleri çözmek için de kullanılabilmektedir. Karar Ağaç algoritması, eğitim verilerinden çıkartılan basit karar kuralları öğrenerek hedef veri örneğinin sınıfını tahmin etmek için kullanılabilecek bir eğitim modelin oluşturulmasını amaçlamaktadır. Karar Ağaçlarında, bir veri örneğinin sınıf etiketini tahmin etmek için ağacın kökünden başlanmaktadır. Kök özniteliğinin değerini veri örneğinin özniteliğiyle karşılaştırılmaktadır. Karşılaştırma temelinde, şu andaki değere karşılık gelen dalı izlenip bir sonraki düğüme atlanmaktadır.

III. Rastgele Ormanı (Random Forest)

Sınıflandırma, regresyon ve diğer görevler için kullanılan toplu bir öğrenme yöntemidir. Rastgele orman algoritmada, veri örnekleri üzerinde karar ağaçları oluşturulmakta ve daha sonra her karar ağacından bir tahmini alınmakta ve son olarak oylama yoluyla en iyi çözümü seçilmektedir. Test edilen veri örnek sınıf hakkında son kararı vermek için farklı karar ağaçlarından alınan oyları toplamaktadır. Test edilen veri örneğinin sınıfı, toplanan oyların çoğunluğuna göre belirlenmektedir. Rastgele orman algoritması, tek bir karar ağacından daha iyi toplu bir yöntem olarak bilinmektedir, çünkü rastgele ormanda, sonucun ortalaması alarak fazla uyumu (Over-fitting) azaltmaktadır.

IV. Torbalama Sınıflandırıcısı (Bagging Classifier)

Torbalama sınıflandırıcısı, birden çok temel sınıflandırıcıya uyan toplu bir meta-tahminleyicidir. Temel sınıflandırıcıların her biri, orijinal veri kümesinden rastgele seçilen alt kümeleri kullanılarak eğitilmektedir. Ardından, son tahmini oluşturmak için temel sınıflandırıcıların bireysel tahminleri toplanmaktadır. Bu metot, karar ağacı gibi bazı öğrenme metotların karmaşıklığını basitleştirmek için kullanılmaktadır. Bu da karar ağacının yapım prosedürüne rastgeleleştirmeyi ekleyerek ve ardından bundan bir topluluk oluşturarak başarılmıştır. Bu çalışmada, K-en yakın komşu, torbalama sınıflandırıcısında, temel sınıflandırıcı olarak kullanılmıştır.

V. AdaBoost

AdaBoost veya Adaptive Boosting, 1996 yılında Yoav Freund ve Robert Schapire tarafından önerilen topluluk arttırma sınıflandırıcılarından biridir. AdaBoost sınıflandırıcısı, sınıflandırma doğruluğu arttırmak için çoklu sınıflandırıcıları birleştirmektedir. AdaBoost, yinelemeli bir sınıflandırıcı birleştirme yöntemidir. AdaBoost sınıflandırıcı kullanılarak, çok düşük performans gösteren sınıflandırıcıları birleştirerek güçlü bir sınıflandırıcıyı oluşturulabilmektedir. Buradaki fikir, sınıflandırıcıları, doğru bir şekilde sınıflandırılması zor olan veri örneklerine

konsantre olmaya zorlamak için hem sınıflandırıcılara hem de veri örneklerine ağırlık ayarlanmaktadır. Herhangi bir makine öğrenme algoritması, AdaBoost algoritması için temel sınıflandırıcı olarak kullanılabilmektedir. Bu çalışmada, rastgele orman sınıflandırıcısı, AdaBoost sınıflandırıcı için temel tahmin edici olarak kullanılmıştır.

VI. Gradyan artırma (Gradient Boost)

Gradyan artırma sınıflandırıcısı, güçlü bir sınıflandırma modeli oluşturmak için birçok zayıf öğrenme modeli bir araya getirmektedir. Karar ağaçları genellikle gradyan artırma yapılırken kullanılmaktadır. Gradyan artırma modeller, karmaşık veri kümeleri sınıflandırmadaki etkinlikleri nedeniyle, bu modellerin kullanılması popüler haline gelmiştir. AdaBoost'ta olduğu gibi, bu teknik, sonraki sınıflandırıcıları, önceki sınıflandırıcıların hatalarından öğrenme mantığı ile çalışmaktadır.

3.3.1.6. Kullanılan Derin Öğrenme Teknikleri

I. Evrişimli Sinir Ağ (ESA)

Evrişimli sinir ağ (ESA), genellikle görüntü işleme ve nesne tanıma alanında kullanılan bir derin öğrenme algoritmasıdır. Evrişimli sinir ağ modellerinde, evrişimli (Convolutional) ve havuzlama (Pooling) olmak üzere iki ana işlemi kullanılmaktadır. Bu iki işlem, veri boyutları küçültmek için, hesaplama karmaşıklığı basitleştirmek için ve sınıflandırma ve tanımada kullanılabilecek şekilde görüntüyü temel özelliklerine indirgemek için kullanılmaktadır. Genel olarak, herhangi bir evrişimli sinir ağı aşağıdaki temel bloklardan oluşmaktadır:

1) Evrişimli Katmanlar

Evrişimli sinir ağında kullanılan ana yapı taşlarıdır. Görüntüdeki karakteristik noktaları algılamak ve görüntü boyutu küçültmek için, görüntü üzerinden çekirdek adı verilen birçok filtre geçirilmektedir. Başka bir deyişilye, 3 X 3 gibi belirli boyutlu

bir filtre görüntü üzerinden geçirilerek filtrede tanımlanan ağırlıkları, orijinal piksel değerleriyle nokta çarpım toplamı hesaplanmaktadır. Görüntüdeki her piksel blok, iç çarpım toplam sonucu ile bir sonraki etkinleştirme katmanında temsil edilecektir.

2) Aktivasyon Katmanları

Bu katmanları, genellikle ReLu işlevi kullanarak bir miktar doğrusalsızlığı oluşturmak için kullanılmaktadır. Başka bir deyişiyle, ReLu, evrişimli katmanının çıktı matris üzerinde bir miktar doğrusalsızlık üretmek için bir etkinleştirme işlevi olarak kullanılmaktadır.

3) Pooling Katmanları

Genel olarak, havuz katmanı, matrisin boyutları azaltmak amacıyla daha fazla alt örnekleme yapmak için kullanılmaktadır. Tipik olarak, bu katman, matris üzerinden bir filtre geçirilmektedir ve her bir piksel grubu, gruptaki maksimum/minimum değere sahip olan pikseli ile temsil edilmektedir.

4) Tamamen Bağlı Katman

Bu katman, her bir sınıf etiketinin olasılığını elde etmek için kullanılmaktadır. Dolayısıyla, bu katman, geleneksel çok katmanlı bir algılayıcı yapıya sahiptir. Tamamen bağlı katman, önceki katmanlardan gelen vektörü girdi olarak alıp her sınıfın olasılığını çıktı olarak vermektedir.

3.3.1.7. Kullanılan Son Teknoloji Derin Öğrenme Modelleri

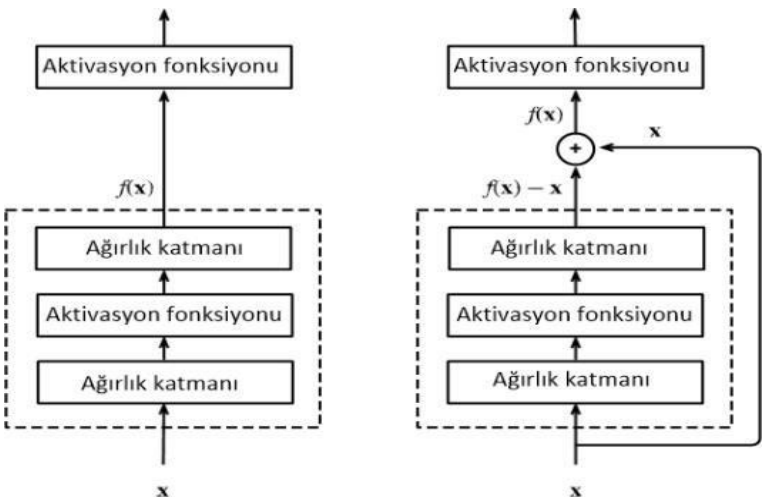
Önerilen sınıflandırma modelleri ile mevcut derin öğrenme modelleri arasında bir karşılaştırma yapmak için, iyi bilinen iki derin öğrenme modeli, oluşturulan kötü amaçlı yazılım görüntü veri kümeleri kullanılarak test edilmiştir. Kullanılan iki model, aşağıdaki iki bölümde kısaca açıklanmıştır.

I. Kalıntı Ağları (ResNet) Model

ResNet, 2015 yılında He ve arkadaşları tarafından önerilen bir evrişimli sinir ağ modelidir [267]. ResNet modeldeki temel amacı, yüksek hesaplama performansıyla yüzlerce hatta binlerce katmanı eğitmek için kullanılabilirliği sağlamaktadır. Başka bir deyişiyle, önceki ESA mimarilerin hesaplama performansı katman sayısının artmasıyla düşerken, ResNet model, güçlü performansı koruyarak çok sayıda katman eklemeye imkân sağlamaktadır. ResNet model, daha hızlı eğitim sağlamak için önemsiz katmanları veya başlangıçta hiçbir şey yapmayan katmanları atlamaya dayanmaktadır. Böylece ResNet model, hesaplama süresi kabul edilebilir bir seviyede tutarak çok sayıda hiperparametre içeren daha derin modelleri eğitebilmektedir. Bu nedenle ResNet, bilgisayarla görme alanında kullanılan en derin öğrenme modellerinden biri haline gelmiştir. ResNet model, denklem (10) gösterdiği gibi birkaç yığılmış katman için bir kalıntı öğrenme bloğu (Residual Block) benimsemeye dayanmaktadır.

$$y = \mathcal{F}(x, \{W_i\}) + x \qquad (10)$$

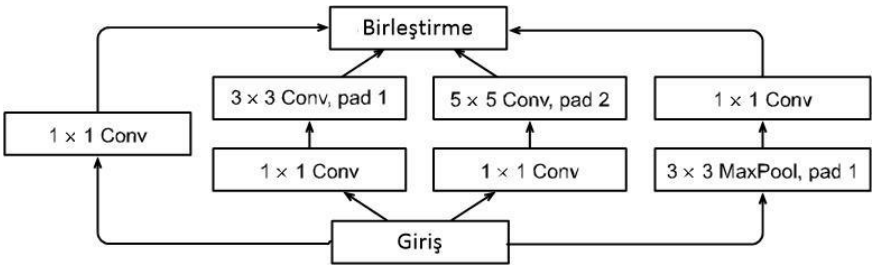
Burada: x ve y, dikkate alınan katmanların giriş ve çıkış vektörleri ve $\mathcal{F}(x, \{W_i\})$ öğrenilecek Residual blok haritalamayı temsil etmektedir. ResNet modeldeki Residual öğrenme bloğu şekil 3.5'te gösterilmektedir.



Şekil 3.5. Normal bir blok (solda) ve ResNet'in bir residual öğrenme bloğu (sağda).

II. Inception V3 Model

Inception V3 model [268], performansı hem hız hem de doğruluk açısından ilerletmek için önerilen daha karmaşık ve yoğun bir şekilde tasarlanmış bir evrişimli sinir ağdır. Bu tür modeller, bir zorluk haline gelen evrişimli katmanları için doğru çekirdeği seçebilme sorununu çözmek için önerilmiştir. Burada, büyük çekirdekleri, global olarak dağıtılan bilgileri için daha yararlı iken, küçük çekirdekleri, lokal olarak dağıtılan bilgileri için daha yararlı olduğunu bilinmektedir. Bu nedenle, Inception modellerde aynı seviyede birden çok filtre boyutu kullanılması önerilmiştir, böylece toplam model çok büyük bir derinlik yerine biraz daha geniş olmuştur. Bu modellerin en basit türü, birden çok Naive Inception modülü içeren Inception V1'dir [269]. Her Naive modül, $1x1$ boyutlu filtre, $3x3$ boyutlu filtre ve $5x5$ boyutlu filtre olmak üzere üç farklı boyutlu filtrelerden ve bir Max Pooling katmanından oluşmaktadır. Şekil 3.6 de gösterildiği gibi, önceki modülün filtrelerinin çıktıları birleştirilerek bir sonraki modüle gönderilmektedir. Bu filtreler, Inception v2 ve Inception v3 sürüm modellerinde optimize edilmiştir, böylece sınıflandırma doğruluk ve hız açısından en iyi performans elde edilebilmektedir. Örneğin, nxn boyutundaki filtreler, %33 daha hesaplama maliyeti düşük olan $1xn$ ve $nx1$ filtrelerinden bir kombinasyon ile değiştirilmiştir.



Şekil 3.6. Inception bloğunun yapısı.

3.3.2. Önerilen Modeller

Android kötü amaçlı yazılım örnekleri ailelerine göre sınıflandırmanın yanı sıra Android uygulamaları, kötü amaçlı veya zararsız olarak sınıflandırmak için birden fazla model önerilmiş ve geliştirilmiştir. VisDroid adlı ve geleneksel makine

öğrenme sınıflandırıcılar tabanlı bir model önerilmiş ve android uygulamaların ikili sınıflandırmasının yanı sıra Android kötü amaçlı yazılımların çok sınıflı sınıflandırması için kullanılmıştır. Ayrıca, Android uygulamaları, zararsız veya kötü amaçlı yazılım olarak sınıflandırmak için üç derin öğrenme modeli önerilmiş ve geliştirilmiştir.

3.3.2.1. Geleneksel Makine Öğrenme Sınıflandırıcı Tabanlı Modeller

I. Önerilen VisDroid Model

Bu bölümde önerilen VisDroid sınıflandırma modeli detaylı olarak anlatılacaktır. İlk olarak, daha önce bahsedilen kötü amaçlı yazılım görüntü veri kümeleri oluşturulmuştur; sonra, bir takım görüntü tabanlı global ve lokal özellikleri ayıklanmıştır. Detaylı olarak anlatılacak olursa, her kötü amaçlı yazılım görüntüsünden Renk Histogramı, Hu Moments ve Haralick Dokusu olmak üzere üç tane global özellikleri ayıklanmıştır. Ayıklanan global özellikleri, tek bir özellik vektörde yığılmış, normaliz edilmiş ve kullanılan makine öğrenme sınıflandırıcıları eğitmek için girdi olarak kullanılmıştır. Ayrıca, oluşturulan görüntü veri setlerinden ÖBÖD, HSÖ, YDB ve KAZE olmak üzere dört tane görüntü tabanlı lokal özellikleri ayıklanmıştır. Lokal özellikleri, her bir görüntünün birden çok vektörü (birden çok tanımlayıcı) kullanılarak temsil edilmesi açısından global özelliklerinden farklıdır. Neredeyse tüm makine öğrenme algoritmaları n tane numune n tane özellik vektörü girdi olarak kabul ettiğinden dolayı, lokal özelliklerin orijinal formatıyla, makine öğrenme sınıflandırıcılarına girdi olarak kullanılamamaktadır. Bu nedenle, her bir görüntüden ayıklanan lokal özelliklerin tanımlayıcılarından bir özellik vektörü oluşturmak için görsel sözcük çanta algoritması (GSÇ) kullanılmıştır. Görsel sözcük çanta algoritmasının ilk adımda, ayıklanan lokal tanımlayıcı vektörler, K-ortalama algoritması kullanılarak çoklu kümeler halinde gruplandırılmıştır. Ardından, görüntüdeki her tanımlayıcı kümesi, hazırlanan K-ortalama algoritması kullanılarak tahmin edip her görüntü için bir histogram oluşturulmuştur. Başka bir deyişiyle, K-ortalama algoritmasındaki her bir kümeye ait tanımlayıcı sayısı hesaplayarak her görüntü için bir özellik vektörü oluşturulmuştur. Sonuç olarak, her bir görüntüyü temsil etmek için K uzunluğunda bir vektör elde edilmiştir. Elde edilen vektörleri,

makine öğrenme sınırlandırıcılarına girdi olarak kullanılmıştır. Algoritma 3.3, GSÇ algoritması kullanarak lokal öznitelik vektörleri oluşturmak için kullanılan yöntemi göstermektedir. Her küme için histogram değeri, denklem (11) kullanılarak hesaplanması sırasında normaliz edilmiştir.

$$\boldsymbol{H}(\boldsymbol{C}) = \boldsymbol{H}(\boldsymbol{C}) + \frac{1}{\textit{Karakteristik nokta sayısı}} \qquad (11)$$

Burada $H(C)$: kelime dağarcığındaki belirli bir kümeye ait tanımlayıcıların sıklığını temsil eden bir değerdir; Karakteristik nokta sayısı: Resimdeki kilit nokta sayısı temsil etmektedir.

Şekil 3.7, önerilen VisDroid görüntü tabanlı sınıflandırma modeli göstermektedir. Birden çok makine öğrenme sınıflandırıcısı eğitmek için ayıklanan lokal ve global özellikleri ayrı ayrı kullanılmıştır. Ayıklanan özellikleri, Rastgele Orman, K-en yakın Komşu, Karar Ağacı, Torbalama, AdaBoost, Gradient Boost ve zor oylama sınıflandırıcıları eğitmek için kullanılmıştır. Ayıklanan üç global özellikleri, bir vektörde istiflenmiş, normalleştirilmiştir. Ardından, oluşturulan global özellik vektörleri, kötücül yazılım örnekleri on iki farklı aileye sınıflandırmak için ve Android uygulamaları kötücül veya iyi huylu olarak sınıflandırmak için kullanılan sınıflandırıcıları eğitmek için kullanılmıştır.

Öte yandan, lokal özellikleri, global özelliklerde olduğu gibi istiflenmemiş, ancak yukarıda bahsedilen makine öğrenme sınıflandırıcıları eğitmek için ayıklanan lokal özellikleri ayrı ayrı kullanılmıştır. Başka bir deyişiyle, ayıklanan dört lokal özelliğin her birinin etkinliğini ayrı ayrı test edilmiştir. Örneğin, ÖBÖD özelliği ayıklanmış, ardından ÖBÖD histogram vektörleri oluşturmak için GSÇ algoritması kullanılmıştır. Bundan sonra, inşa edilen ÖBÖD histogram vektörleri, kötü amaçlı yazılım örnekleri, on iki farklı aileye sınıflandırmak için ve Android uygulamaları, kötücül veya iyi huylu olarak sınıflandırmak için yukarıdaki bahsedilen sınıflandırıcıları eğitmek için kullanılmıştır.

Algoritma 3.4, Android uygulamaları, iyi huylu yazılım veya kötü amaçlı yazılım olarak sınıflandırmak (İkili sınıf sınıflandırılması) için kullanılan VisDroid modeli göstermektedir.

II. Önerilen Melez Oylama Sınıflandırıcı

Global ve lokal görüntü özelliklerinin etkinliğini ayrı ayrı test ettikten sonra, bu iki tür özelliğin bir karışımı, topluluk bir sınıflandırıcı eğitiminde kullanmayı önerilmiştir. Global özellikleri, görüntü hakkında genel bilgi verebildiğinden ve lokal özellikleri, görüntüdeki her bir nesne hakkında bilgi verebildiğinden, bu iki tür özelliğin birleştirilmesinin daha doğru bir performans verebileceğini düşünmekteyiz. Böylece, global özellikleri, görüntü üzerinde keşif (Exploration) yapabilir iken, lokal özellikleri, görüntü üzerinde istismar (Exploitation) yapabilmektedir. Bu amaçla, lokal ve global özellikleri kullanılarak eğitilen bir dizi sınıflandırıcılarına dayanan melez bir toplu oylama sınıflandırıcısı önerilmiştir. Başka bir deyişiyle, daha önce bahsedilen altı makine öğrenme sınıflandırıcıları iki sefer eğitilmiştir. İlk seferde, ayıklanan global özellikleri kullanılarak eğitilmiş olup altı karar alınmıştır (her sınıflandırıcıdan bir karardır). Bundan sonra, GSÇ algoritması, ayıklanan lokal özelliklerin tanımlayıcılarından (yani ÖBÖD, HSÖ, YDB veya KAZE) bir histogram vektörü oluşturmak için kullanılmıştır. Ardından, oluşturulan lokal özellik vektörleri kullanarak bahsedilen altı sınıflandırıcıları eğiterek altı tane karar daha elde edilmiştir. Böylece, on iki karar alınmıştır (global özellikleri kullanılarak eğitilen sınıflandırıcıları kullanarak altı tane karar alınmış ve lokal özelliklerin birini kullanarak eğitilen sınıflandırıcıları kullanarak altı tane karar alınmıştır). Zor oylama sınıflandırıcısına benzer bir şekilde çalışan bir topluluk oylama sınıflandırıcısı önerilmiştir. Elde edilen on iki kararı, önerilen bu topluluk oylama sınıflandırıcısına girdi olarak kullanılmıştır. Dolayısıyla, önerilen topluluk oylama sınıflandırıcısının nihai karar, elde edilen on iki kararların çoğunluğuna dayanmaktadır. Başka bir deyişiyle, android yazılımın numunesi, en fazla oyu alan sınıfına ait olduğunu karar alınmıştır. Şekil 3.8, önerilen melez oylama sınıflandırıcısının şematik diyagramı göstermektedir.

Giriş: Lokal özellik türü: F = ÖBÖD, HSÖ, YDB veya KAZE
Görüntü Veri Kümesi: $Images = \{img_1, img_2, \ldots img_i\}$
Çıkış: Histogram vektör listesi: $\mathcal{H} = \{h_1, h_2, \ldots h_j\}$

#1. Lokal özellikleri ayıklanmak (ÖBÖD, HSÖ, YDB veya KAZE).

For each $class_n$ in $Images$ **do** // $class_n$: belirli bir Android uygulama sınıfı
  For each img_i in $class_n$ **do**
    If F == 'ÖBÖD':
      $kps_i, dsc_i \leftarrow$ *SIFT* özelliğin img_i'den ayıklanması
    Else if F == 'HSÖ':
      $kps_i, dsc_i \leftarrow$ *SURF* özelliğin img_i'den ayıklanması
    Else if F == 'YDB':
      $kps_i, dsc_i \leftarrow$ *SURF* özelliğin img_i'den ayıklanması
    Else if F == 'KAZE':
      $kps_i, dsc_i \leftarrow$ *KAZE* özelliğin img_i'den ayıklanması
      $dscDataset \leftarrow dsc_i$ tanımlayıcı veri kümesine eklenmesi
    $keyPointsList \leftarrow$ kps_i karakteristik nokta veri kümesine eklenmesi
    $labelList \leftarrow class_n$ etiket listesine eklenmesi
  End for

End for

  Return $dscDataset, keyPointsList, labelList$

#2. Görsel Kelime Çanta yaklaşımı kullanarak kelime dağarcığı oluşturmak

$i \leftarrow 0$
For all $dsc_i \in dscDataset$ **do**
  $h_i \leftarrow$ görüntü veri kümesindeki küme sayısına kadar bir uzunluğa sahip sıfır değerleri içeren bir vektörün oluşturulması
  $KP_i \leftarrow$ img_i'de tespit edilen anahtar noktaların sayısı

  For all $d \in dsc_i$ **do** // d, dsc_i'de bir tanımlayıcı vektördür
    $clusterId \leftarrow K-ortalama$ algoritması kullanarak d'in kümeyi belirlenmesi
    $h[clusterId] \leftarrow h[clusterId] + \frac{1}{KP_i}$
  End for

  $i \leftarrow i + 1$
  $\mathcal{H} \leftarrow$ Lokal özellik histogram veri kümesine h_i eklemek
End for

Return $\mathcal{H}$ // Lokal özellik histogram veri kümesi (lokal özellik vektörleri)

Algoritma 3.3. Lokal öznitelik vektörleri oluşturmak için kullanılan GSÇ algoritması.

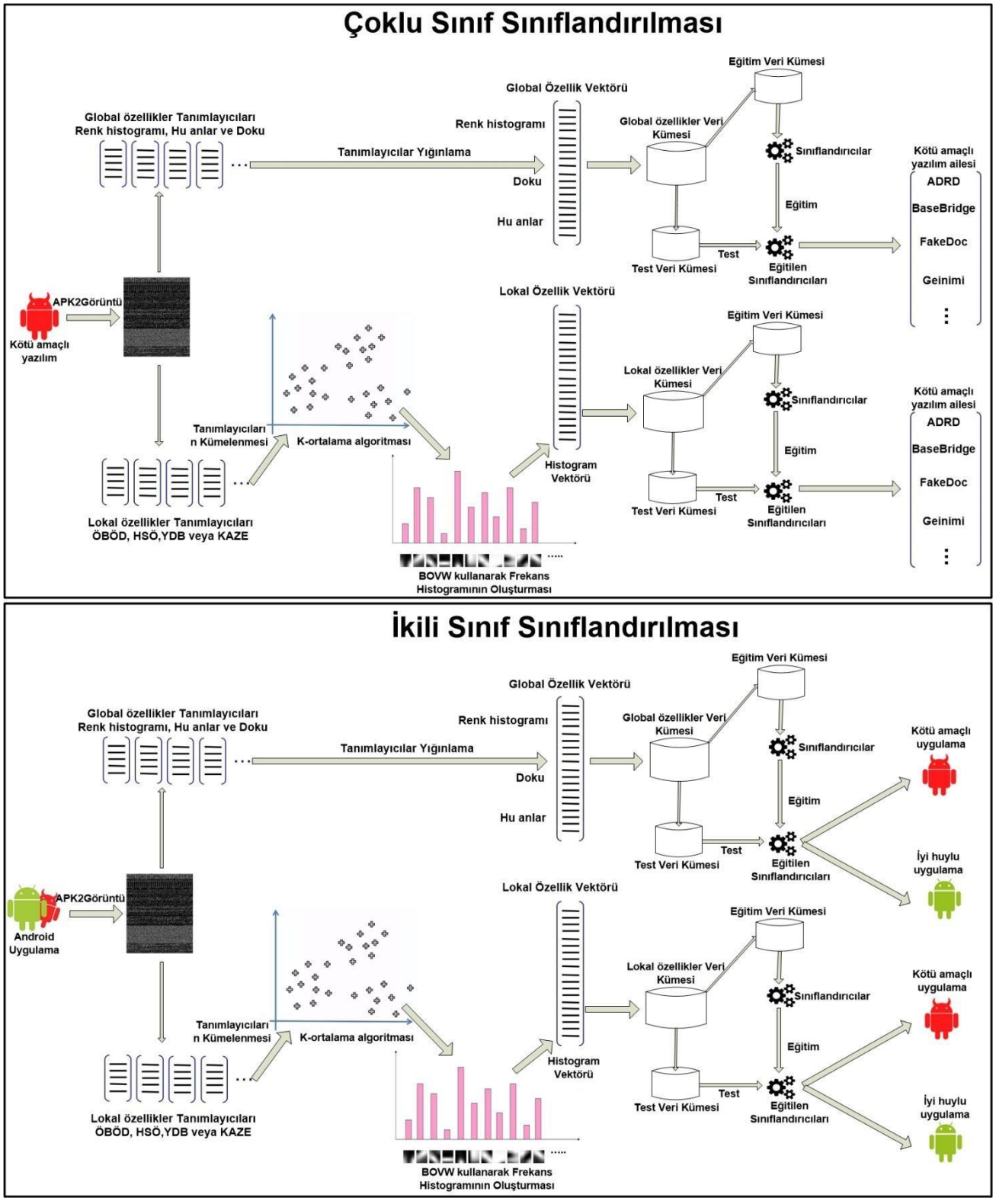
Giriş: İyi huylu uygulama veri kümesi: $APK_\beta = \{apk_1, apk_2, \dots apk_i\}$
Kötücül yazılım veri kümesi: $APK_\mu = \{apk_1, apk_2, \dots apk_j\}$
Çıkış: Uygulama etiketi
Görüntü veri kümenin oluşturma aşaması:
1. **For all** $apk_i \in APK_\beta$ **do**
apk_i 'nin içeriğini çıkarılması
İstenilen gri tonlamalı görüntülerinin oluşturulması
End for
2. **For all** $apk_j \in APK_\mu$ **do**
apk_j 'nin içeriğini çıkarılması
İstenilen gri tonlamalı görüntülerinin oluşturulması
End for

Global özellik tabanlı sınıflandırma modeli
3. **For all** $img_i \in images$ **do** **{***images*: oluşturulan görüntü veri kümesi**}**
$\mathcal{H} \leftarrow$ *renk histogram özelliğinin ayıklanması*
$\mu \leftarrow$ *Hu moment özelliğinin ayıklanması*
Texture $\leftarrow$ *Haralik doku özelliğinin ayıklanması*
$GFV_i \leftarrow \mathcal{H}, \mu$ *and Texture bir vektörde istiflenmesi* {GFV_i global özellik vektörü}
$GFV_i \leftarrow$ *Global özellik vektörünün normalleştirilmesi*
$X \leftarrow GFV_i$ *Global özelliklerin veri kümesine eklenmesi*
$Label_i \leftarrow 0$ *or* 1 **{** $Lable_i$: uygulama etiketi**}**
End for

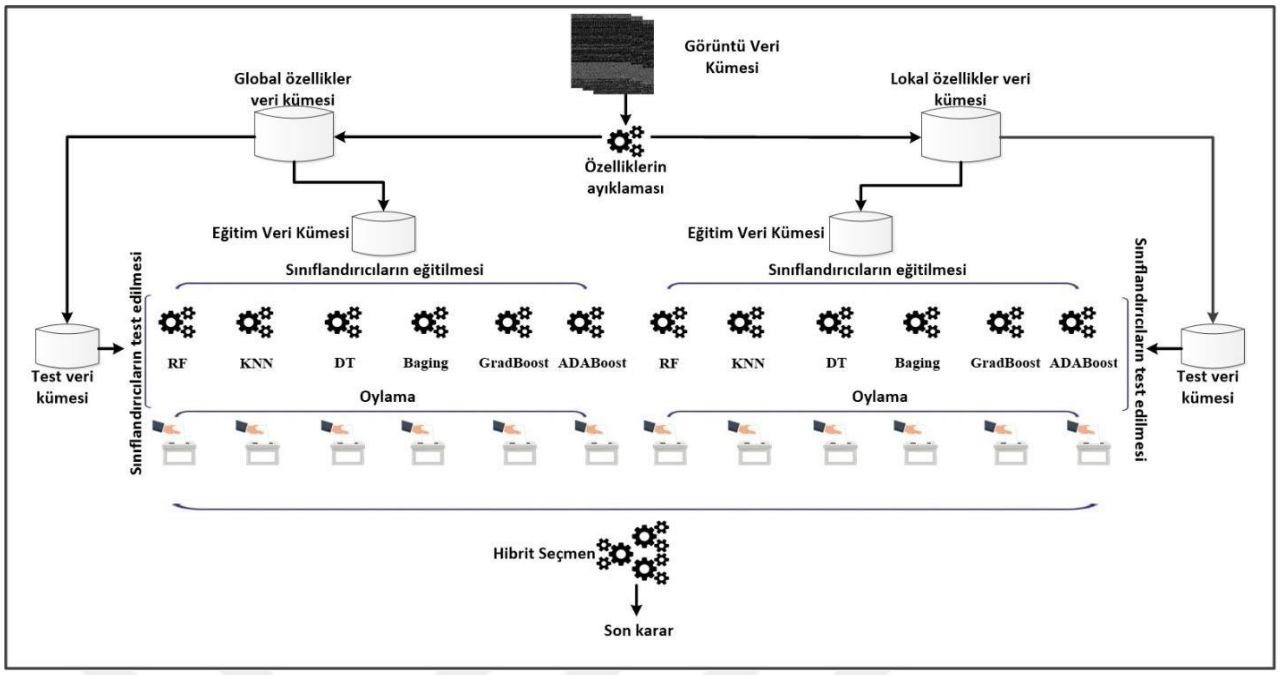
Lokal özellik tabanlı sınıflandırma modeli
1. For all $img_i \in images$ **do** **{***images*: oluşturulan görüntü veri kümesi**}**
$Desc_i, KP_i \leftarrow$ *SIFT, SURF, KAZE veya ORB lokal özelliğin ayıklanması*
#Burada: $Desc_i, KP_i$: *ayıklanan özelliğin tanımlayıcıları ve karakteristik nokta listesi*
$Label_i \leftarrow 0$ *or* 1 **{** $Lable_i$: uygulama etiketi}
End for
2. Görsel Kelime Çantası kullanarak kelime dağarcığı oluşturma
$i \leftarrow 0$
For all $desc_i \in extracted\_image\_descriptors\_dataset$ **do**
Histogram $\leftarrow$ 2 *uzunluğuna sahip sıfırlı bir vektörün oluşturulması*
$KP_{No} \leftarrow$ *her görüntüdeki tespit edilen karakteristik nokta sayısı*
For all $vector \in desc_i$ **do**
clusterId
$\leftarrow K -$ *ortalama algoritması kullanarak d 'in kümesini belirlemek*
Histogram [*clusterId*] $\leftarrow$ *Histogram* [*clusterId*] + $\frac{1}{KP_{No}}$
End for
$i \leftarrow i + 1$
$X \leftarrow$ *Hesaplanan histogram vektörünün, özellikler veri kümesine eklenmesi*
End for

Veri kümesi, test ve eğitim veri kümesine bölmek
1. *Ayıklanan özelliklerin, 10 kat Çapraz Doğrulamaya dayalı olarak bölünmesi*
2. *Makine öğrenme sınıflandırıcılarının eğitilmesi ve test edilmesi*
Return Uygulama etiketi

Algoritma 3.4. Android uygulamaları ikili sınıflandırmak için kullanılan VisDroid modelinin sözde kodu.



Şekil 3.7. Önerilen VisDroid görüntü tabanlı android uygulamaları sınıflandırma model



Şekil 3.8. Önerilen melez oylama sınıflandırıcısı

3.2.4.2. Önerilen Derin Öğrenme Modelleri

Üç farklı evrişimli sinir ağ modeli önerilmiş olup Android uygulamaları, iyi huylu veya kötü amaçlı olarak sınıflandırmak için kullanılmıştır. İlk olarak, bazı görüntü tabanlı özelliklerine ve 1B-evrişimli katman tabanlı bir ESA modeline dayanan ve DeepVisDroid olarak adlandırılan melez bir modeli önerilmiştir. Ayrıca, önerilen DeepVisDroid modelin sonuçları ile geleneksel ESA modellerinin sonuçları ile bir karşılaştırma yapmak için iki tane klasik 2B-evrişimli katman tabanlı ESA modeli önerilmiştir. Önerilen üç ESA modelleri, en iyi sonuçlar veren mimarileri seçilecek şekilde deneysel olarak tasarlanmıştır. Bu amaçla, bir evrişimli katman ve bir havuzlama katman ile başlanmış, ardından ağın hibrit parametreleri kademeli olarak değiştirilmiş ve en iyi performansı veren mimari seçilmiştir. Önerilen modelleri ilerleyen paragraflarda detaylı olarak anlatılacaktır.

I. Önerilen Görüntü Özelliklerine Dayalı DeepVisDroid Model

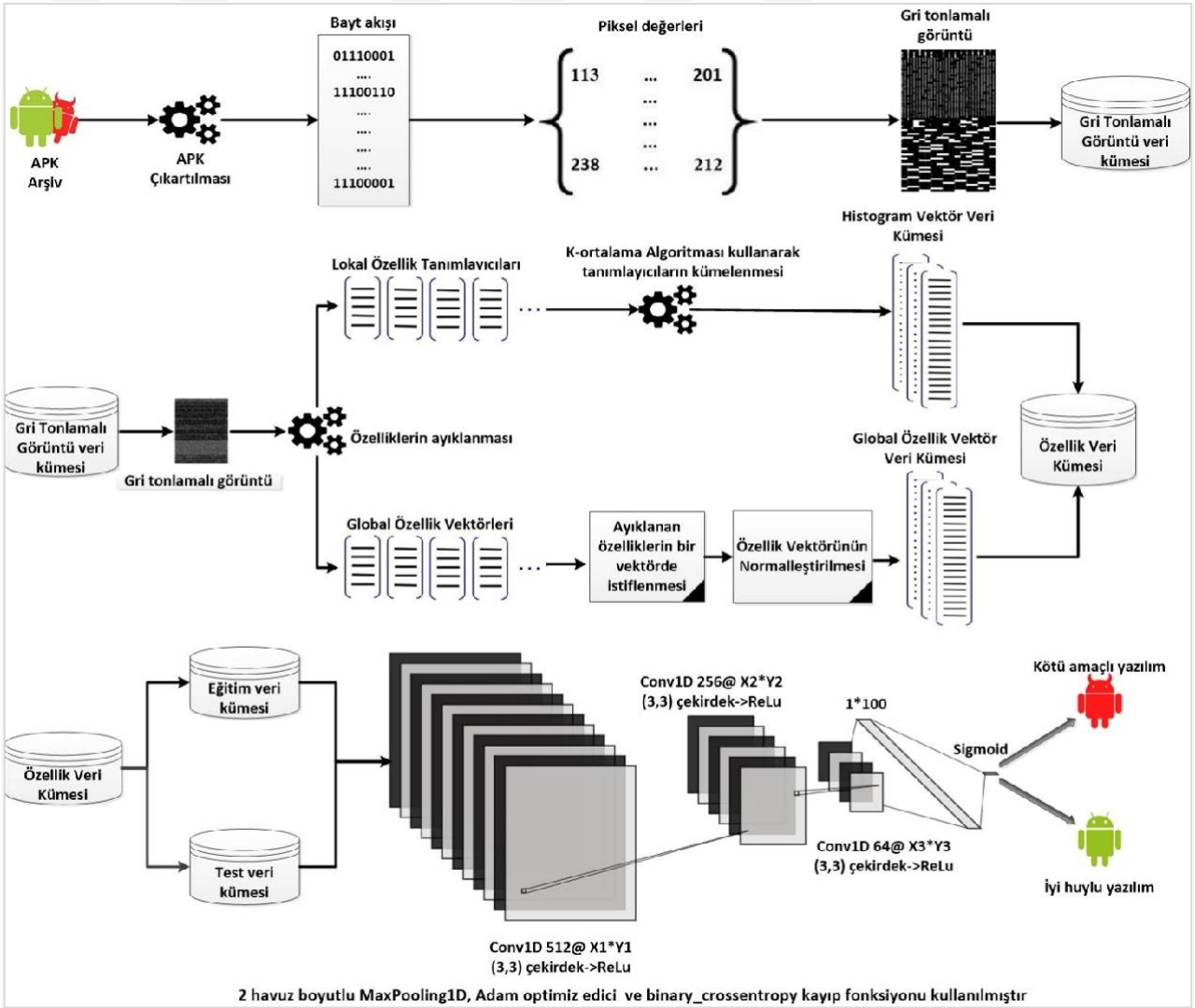
DeepVisDroid adlı 1-B evrişimli katman tabanlı bir evrişimli sinir ağ model önerilmiş ve bazı görüntü tabanlı özellikleri kullanılarak eğitilmiştir. Bu amaçla,

daha önce bahsedilen görüntü tabanlı global özellikleri ve lokal özellikleri oluşturulan dört gri tonlamalı görüntü veri setlerinden ayıklanmış ve önerilen DeepVisDroid modeli eğitmek için kullanılmıştır. Görüntü tabanlı lokal özellikleri ayıkladıktan sonra, ayıklanan çoklu lokal özellik tanımlayıcılarından bir özellik vektörü oluşturmak için görsel sözcük çanta algoritması (GSÇ) kullanılmıştır. Oluşturulan GSÇ tabanlı özellik vektörleri, bir evrişimli sinir ağ modeli eğitmek için kullanılmıştır. Öte yandan, bahsedilen üç global özellik ayıklanmış, bir vektörde istiflenmiş, normalleştirilmiş ve önerilen DeepVisDroid evrişimli sinir ağ modeli eğitmek için kullanılmıştır. Önerilen DeepVisDroid deneysel olarak tasarlanmıştır, böylece farklı evrişimli katmanlar ve Max Pooling katmanlar koleksiyonları test edilmiş ve en iyi sonuçlar veren modeli seçilmiştir. Son seçilen model, her biri (3,3) sabit çekirdek boyutuna sahip farklı sayıda filtre içeren üç 1-B evrişimli katmanı içermektedir. ReLu işlevi, kullanılan tüm evrişimli katmanlarında bir etkinleştirme işlevi olarak kullanılmıştır. Ayrıca, modelde, (2,2) boyutlu filtrelere sahip üç MaxPooling katmanı kullanılmıştır. Ayrıca sigmoid, çıkış katmanında bir etkinleştirme işlevi olarak, ikili çapraz Entropi, bir kayıp fonksiyonu olarak ve Adam, bir optimize edici olarak kullanılmıştır. Önerilen DeepVisDroid modeli şekil 3.9'de gösterilmektedir.

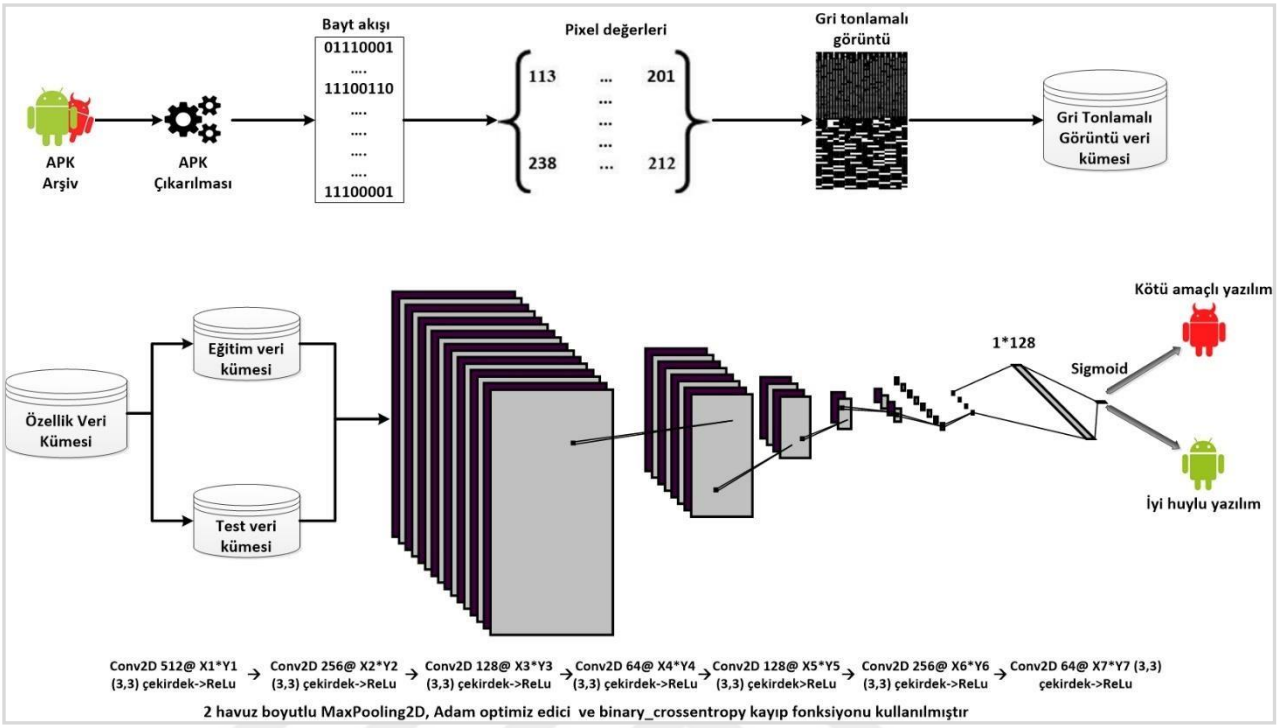
II. Önerilen Klasik 2B Evrişimli Katman Tabanlı ESA Model

Önerilen DeepVisDroid modelin sonuçları klasik ESA modellerin sonuçlarıyla karşılaştırmak için klasik bir 2B evrişimli katmanlar tabanlı ESA modeli deneysel olarak inşa edilmiş ve oluşturulan gri tonlamalı görüntü veri kümeleri kullanılarak eğitilmiştir. Önerilen model, deneysel olarak oluşturulmuştur, böylece hiperparametreler için farklı değerleri test edilmiş ve en iyi sonuçları veren değerler seçilmiştir. Örneğin, evrişimli katmanlarda kullanılan en iyi filtre sayısı seçmek için 64 ile 512 arasında değişen farklı değerleri test edilmiştir. Ayrıca, filtre boyutu için farklı değerler test edilmiş ve (3,3) boyutlu filtrenin en iyi sonuçlar verdiğini ortaya çıkmıştır. Ayrıca, (2,2) filtre boyutuna sahip MaxPooling, önerilen modelde havuz katmanı olarak kullanılmıştır. Başka bir deyişiyle, evrişimli katmanların sayısı, Maxpooling katmanları ve filtre sayısı kademeli olarak değiştirilmiş ve en iyi sonuçlar veren katman sayısı nihai modelde kullanılmak üzere seçilmiştir. Önerilen

modelin mimarisi, şekil 3.10'te gösterildiği gibi olduğunda en iyi sonuçlari elde edilmiştir. Son model, ReLu etkinleştirme işlevlerine sahip yedi evrişimli katmanından ve yedi MaxPooling katmanından oluşmaktadır. Kullanılan evrişimli katmanlardaki filtre sayısı 64 ile 512 filtre arasındadır ve her katmanda (3,3)'lük bir çekirdeği kullanılmıştır. MaxPooling katmanlarında hem x hem de y yönlerinde 1 Stride değerine ve (2,2) boyutuna sahip bir filtre kullanılmıştır. Ayrıca, eğitimin aşırı uyması önlemek için 0.3 bir Dropout değeri kullanılmıştır. Bu çalışmada ele alınacak problem ikili sınıflandırma problemi olduğundan dolayı, önerilen modelin çıktı katmanında, sigmoid fonksiyonu aktivasyon fonksiyon olarak, ikili çapraz Entropi kayıp fonksiyon olarak ve Adam optimize edici olarak kullanılmıştır.



Şekil 3.9. Önerilen DeepVisDroid görüntü özelliklerine ve 1B-evrişimli katmanlarına dayanan evrişimli sinir ağı model.

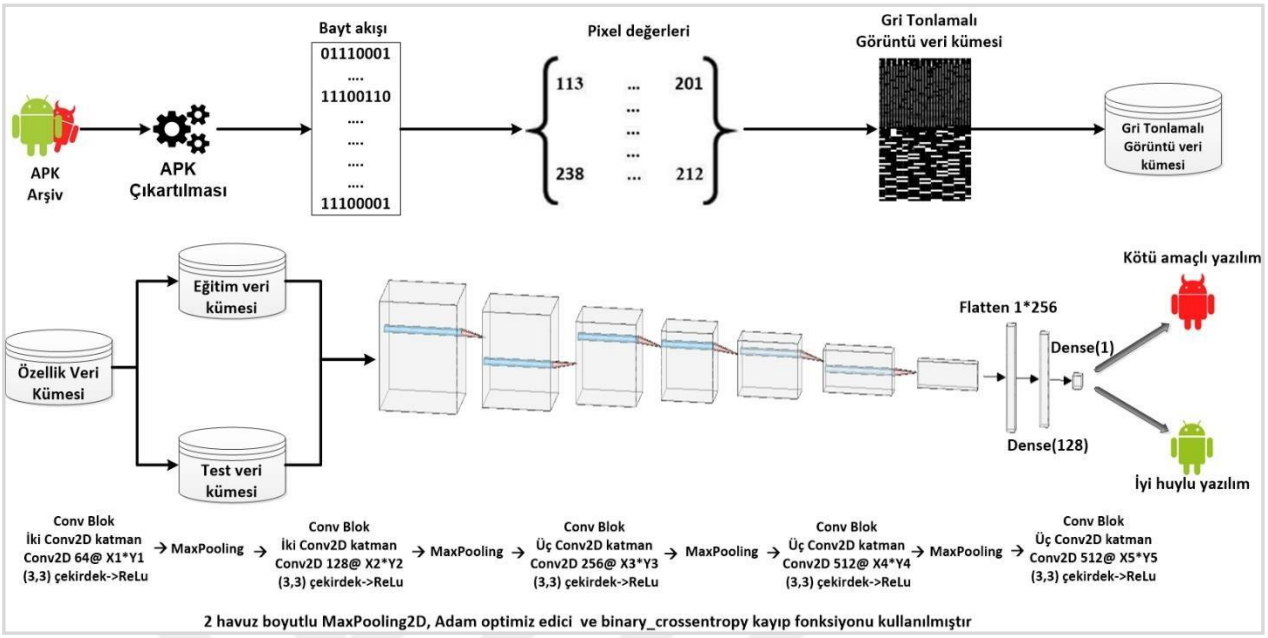


Şekil 3.10. Önerilen 2B-evrişimli katman tabanlı klasik evrişimli sinir ağı model.

III. Önerilen VGG16 Katman Blok Tabanlı ESA Model

Çalışmanın bu bölümde, VGG16, iyi bilinen evrişimli sinir ağ modelinden esinlenen bir model tasarlanmıştır, ancak önerilen model çok daha az sayıda katmanları içermektedir. Önerilen modeldeki katman blok sayısı deneysel olarak seçilmiştir, böylece en iyi sonuçları verebilecek katman sayısı ve hiperparametre değerleri seçilmiştir. Şekil 3.11, önerilen son model mimarisi göstermektedir. Önerilen model, her birine bir MaxPooling katmanı izleyen çoklu evrişimli katman bloklarından oluşmaktadır. Önerilen modelin girdi bölümü, her biri iki evrişimli katmanı ve ardından bir MaxPolling katmanı içeren iki evrişimli katman bloğundan oluşturulmuştur. Modelin ikinci bölümü, her biri üç evrişimli katman ve ardından bir MaxPooling katmanı içeren üç evrişimli katman bloğu içermektedir. Tüm katman blokların katmanlarında, her biri (3,3) boyutuna sahip 64, 128, 256 veya 512 bir filtre kullanılmıştır. Ayrıca, önerilen modelde, "Same" dolgu, ReLu, evrişimli

katmanlarında etkinleştirme işlevi olarak, Sigmoid, sınıflandırma işlevi olarak ve (2,2) boyutlu filtrelerine sahip MaxPooling katmanları kullanılmıştır.



Şekil 3.11. Önerilen VGG16 model katman blok formatından esinlenilen katman blokları içeren evrişimli sinir ağı modeli.

4. ARAŞTIRMA BULGULARI

4.1.Önerilmiş Enjeksiyon Saldırıları Kullanılarak Bazı İyi Bilinen Anti-Virüs Sistemlerinin Değerlendirilmesi

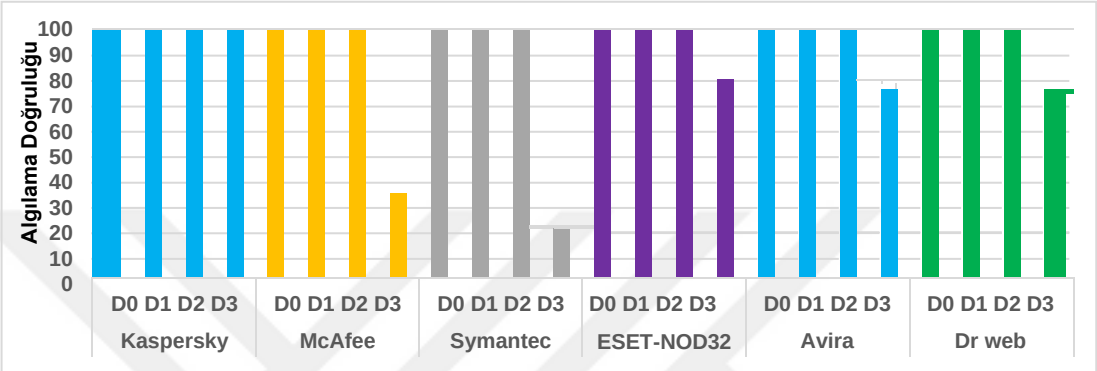
Önerilen saldırıları ve bu tür saldırılara karşı iyi bilinen bazı anti-virüs sistemlerinin sağlamlığını test etmek için birçok vaka çalışması gerçekleştirilmiştir. Önerilen enjeksiyon saldırıları yaklaşık 450 satır Python kod kullanılarak uygulanmıştır. Tüm deneyler Intel Core i7-6700HQ CPU ve 16 GB RAM özelliğine sahip bir laptop kullanılarak uygulanmıştır. Gerçekleştirilen tüm vaka çalışmalarındaki sonuçları, aşağıdaki paragraflarda detaylı bir şekilde anlatılmıştır. Saldırıların etkinliğini değerlendirmek amacıyla iki metrik kullanılmıştır; birincisi, test edilen anti-virüs sistemlerinin algılama doğruluğu ve ikincisi ise, test edilen örnekleri algılayabilen ortalama anti-virüs sistem sayısıdır.

4.1.1. Vaka Çalışma 1

Çizelge 4.1'de gösterildiği gibi dört farklı veri kümesi oluşturmak için, önerilen saldırıları, orijinal Malgenom android kötü amaçlı yazılım veri kümesine uygulanmıştır. Şekil 4.1’de gösterildiği gibi elde edilen sonuçlar; önerilen izin-kodu enjeksiyon saldırısı uygulandığında, bazı sistemlerin doğruluğunda önemli bir azalma olduğunu göstermiştir. Örneğin, McAfee ve Symantec'in algılama doğruluğu sırasıyla yaklaşık %33 ve %20'ye kadar düşmüştür. Şekil 4.2.a, uygulanan saldırılara göre kötü amaçlı yazılım örneklerini algılayabilen ortalama anti virüs sistem sayısındaki değişikliği göstermektedir. Şekilde gösterildiği gibi, uygulamalar da yeniden imzalama saldırısı uygulandığında, ortalama Anti virüs sistem sayısı yaklaşık 45'ten, 30'a kadar düşmüştür. Bu da bazı anti virüs sistemlerinin uygulamadaki sertifika bilgisine dayalı, ilkel algılama yöntemi kullandıkları anlamına gelmektedir.

Çizelge 4.1. İlk vaka çalışmasında oluşturulan veri kümeleri.

D0: Malgenom veri kümesinden 49 kötü amaçlı yazılım örneği rasgele olarak seçilmiştir (her kötü amaçlı yazılım ailesinden bir örnek).
D1: Önerilen yeniden imzalama saldırısı, D0 veri kümesindeki örneklere uygulanmıştır.
D2: Önerilen İzin enjeksiyon saldırısı, D0 veri kümesindeki örneklere uygulanmıştır.
D3: Önerilen izin-kodu enjeksiyon saldırısı, D0 veri kümesindeki örneklere uygulanmıştır.



Şekil 4.1. Değerlendirilen Anti virüs sistemlerinin algılama doğruluğu (Vaka çalışma1).

4.1.2. Vaka Çalışma 2

Bu vaka çalışmasında, Çizelge 4.2'de gösterildiği gibi beş veri seti oluşturmak için önerilen saldırıları, kod çağırma yansıtma gizleme (Invocation Reflection) tekniği ile melezleştirilmiştir.

Şekil 4.3 gösterdiği gibi önerilen saldırılar, kod çağırma yansıtma gizleme tekniği ile melezleştirildiğinde; çoğu anti virüs sisteminin tespit doğruluğunda önemli bir azalma olduğunu göstermiştir. Örneğin, McAfee ve Symantec'in algılama doğruluğu, sırasıyla %22 ve %18'e kadar düşmüştür. Ayrıca, izin enjeksiyon saldırısının etkinliği, ilk vaka çalışmasına kıyasla artmıştır. Şekil 4.2.b, kötü amaçlı yazılım örneklerini algılayabilen ortalama anti virüs sistem sayısındaki azalmayı göstermektedir. Bu vaka çalışmasında elde edilen sonuçlara göre, izin-kodu enjeksiyon saldırısı uygulandığında oluşmuş olan kötücül yazılımları, algılayabilen Anti virüs sistem sayısı 26 ya kadar düşmüştür.

4.1.3. Vaka Çalışma 3

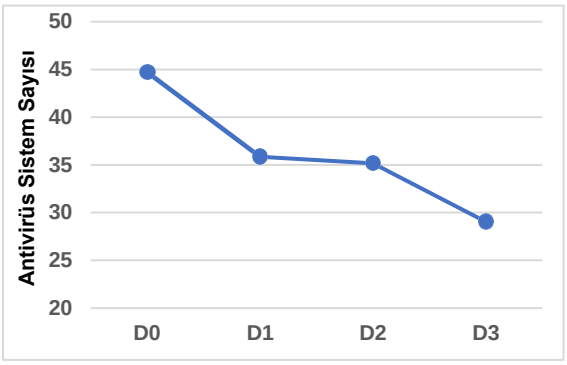
Bu vaka çalışmasında, Çizelge 4.3'te gösterildiği gibi, beş veri seti oluşturmak için önerilen saldırılar, string şifreleme gizleme tekniği ile melezleştirilmiştir. Şekil 4.4 gösterdiği gibi elde edilen sonuçlar, çoğu anti virüs sistemdeki algılama doğruluğunun önceki vaka çalışmalarına kıyasla azalmaya devam ettiğini göstermektedir. Örneğin, Kaspersky, McAfee, Symantec, ESET-NOD32 ve Avira'nın algılama doğruluğu sırasıyla %72, %18.75, %10.42, %41.67 ve %31.25'e kadar düşmüştür. Ayrıca, Şekil 4.2.c, kötü amaçlı yazılım örneklerini tespit edebilen ortalama sistem sayısının önceki vaka çalışmalarına kıyasla azaldığını ve 23 sisteme ulaştığını göstermektedir.

4.1.4. Vaka Çalışma 4

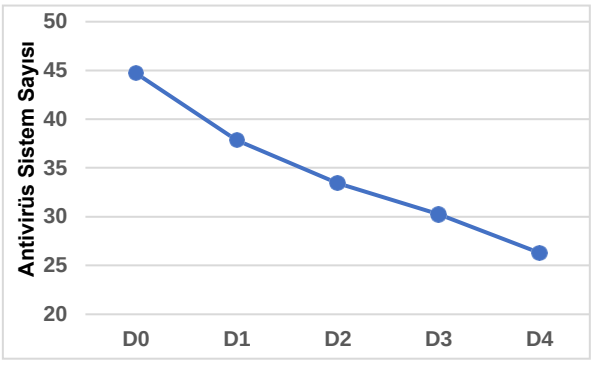
Çizelge 4.4'te gösterildiği gibi, önerilen saldırıları, PRAGuard ‘daki sınıfı şifreleme veri kümesiyle melezleştirilerek beş veri kümesi oluşturulmuştur. Şekil 4.5 gösterdiği gibi elde edilen sonuçlar, bazı anti virüs sistemdeki algılama doğruluğunun önceki vaka çalışmasına kıyasla azaldığını göstermiştir. Örneğin, Kaspersky, Eset-nod32, Avira ve Dr.Web'in algılama doğrulukları, sırasıyla %53.19, %38.3, %23.40 ve %27.66'ya kadar düşmüştür. Ayrıca, Şekil 4.2.d 'de gösterildiği gibi kötü amaçlı yazılım örneklerini tespit edebilen ortalama sistem sayısı, önceki vaka çalışmasına kıyasla, 23 ‘ten 20 ‘ye kadar düştüğü gözlemlenmiştir.

Çizelge 4.2. İkinci vaka çalışmasında oluşturulan veri kümeleri.

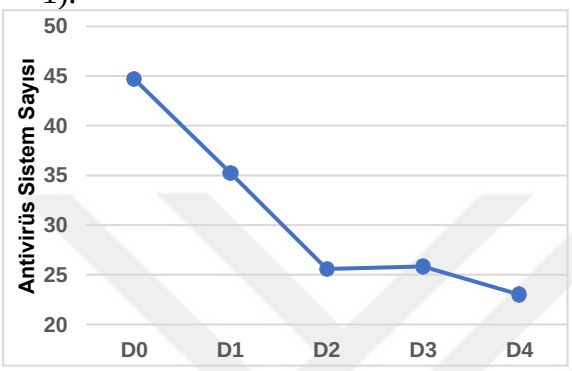
D0: Malgenom veri kümesinden 49 kötü amaçlı yazılım örneği rasgele olarak seçilmiştir (her kötü amaçlı yazılım ailesinden bir örnek alınmıştır).
D1: Kod çağırma yansıtma gizleme tekniği, D0 veri kümesindeki örneklere uygulanmıştır. (bu veri kümesinin örnekleri PRAGuard ‘daki yansıma veri kümesinden seçilmiştir).
D2: Önerilen yeniden imzalama saldırısı, D1 veri kümesinin örneklerine uygulanmıştır.
D3: Önerilen İzin enjeksiyon saldırısı, D1 veri kümesinin örneklerine uygulanmıştır.
D4: Önerilen izin-kodu enjeksiyon saldırısı, D1 veri kümesinin örneklerine uygulanmıştır.



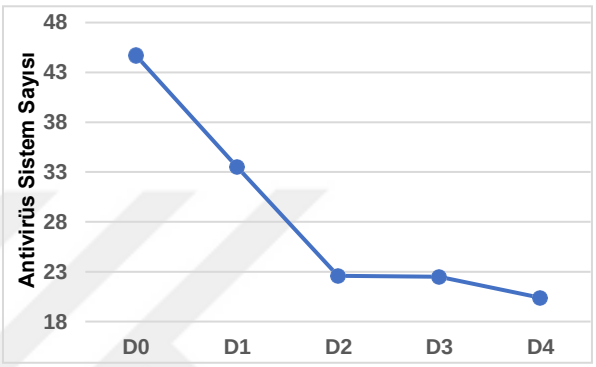
a. Ortalama anti virüs sayısı (Vaka çalışma 1).



b. Ortalama anti virüs sayısı (Vaka çalışma 2).

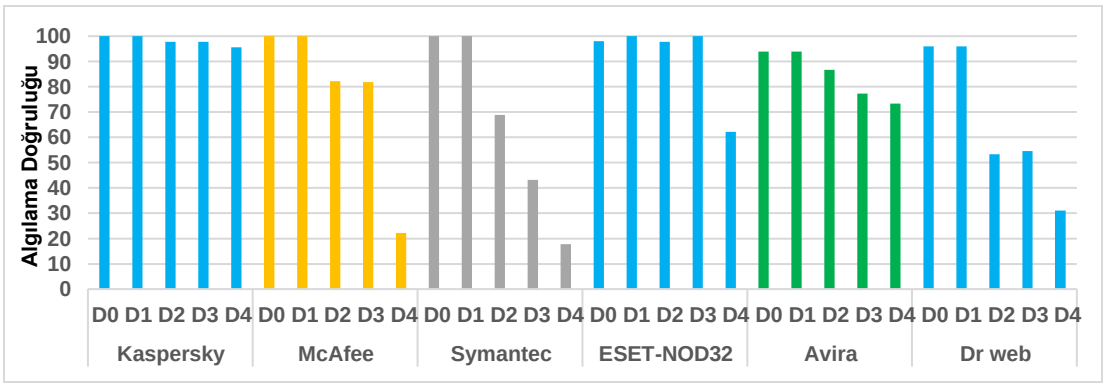


c. Ortalama anti virüs sayısı (Vaka çalışma 3).



d. Ortalama anti virüs sayısı (Vaka çalışma 4).

Şekil 4.2. Kötücül yazılım numuneleri tespit edebilen ortalama anti virüs sistem sayısı.



Şekil 4.3.Değerlendirilen Anti virüs sistemlerinin algılama doğruluğu (Vaka çalışma 2).

4.1.5. Vaka Çalışma 5

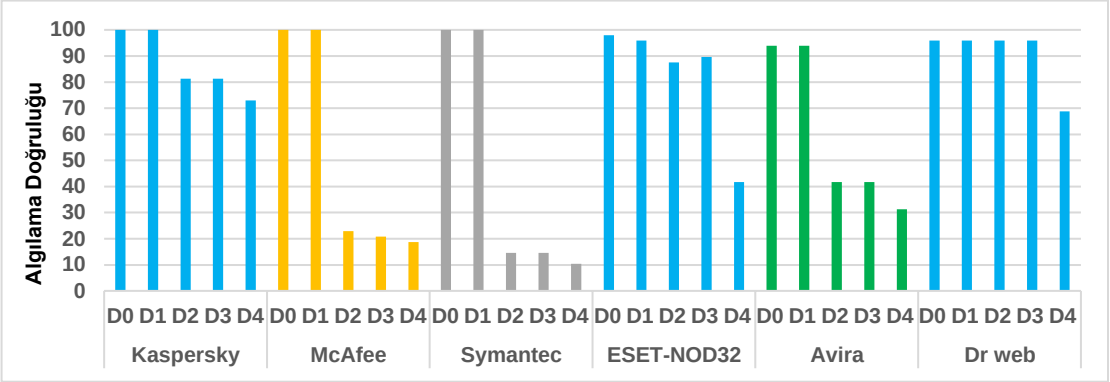
Bu vaka çalışmasında, önerilen saldırıları, PRAGuard ‘daki tam şaşırtma (Obfuscation) veri seti ile Çizelge 4.5'te gösterildiği gibi melezleştirilmiştir. Şekil 4.6'teki sonuçlar, önceki sonuçlarla karşılaştırıldığında çoğu anti virüs sisteminin performansında önemli bir düşüş olduğunu göstermektedir. Örneğin, izin-kod enjeksiyon saldırısı, PRAGuard ‘daki tam obfuskasyon veri setinin kötücül yazılımlarına uygulandığında; Kaspersky, Symantec, ESET-NOD32, Avira ve DrWeb Anti virüslerin tespit doğrulukları yaklaşık %10 ‘a kadar düşmüştür. Ayrıca, test edilen kötücül yazılım örneklerini algılayabilen ortalama anti virüs sistem sayısında, önemli ölçüde bir azalma görülmüştür. Örneğin, Şekil 4.2'da gösterildiği gibi, izin-kod enjeksiyon saldırısı uygulandığında, bu saldırıyı algılayabilen ortalama anti virüs sistem sayısı, yaklaşık 12 sisteme kadar düşmüştür.

Bazı kötücül yazılım örneklerine, izin-kodu enjeksiyon saldırısı uygulandığında, bu saldırıyı algılayabilen anti virüs sistem sayısının, sıfıra düştüğü görülmüştür. Başka bir deyişle, ticari olarak faaliyet gösteren, 60’tan fazla anti virüs sisteminin hiçbiri, bazı kamufle edilmiş kötücül yazılım örnekleri tespit edemediği gözlemlenmiştir. Örneğin, şekil 4.8, izin-kod saldırısı, PRAGuard ‘daki tam obfuskasyon veri seti ile hibritlendikten sonra kamufle edilmiş, CruseWin kötücül uygulamasının test sonucunu göstermektedir.

Buraya kadar anlatılan beş vaka çalışma testleri, 24 Temmuz 2018 ile 3 Ağustos 2018 arasında VirusTotal API ‘ı kullanılarak gerçekleştirilmiştir.

Çizelge 4.3. Üçüncü vaka çalışmasında oluşturulan veri kümeleri.

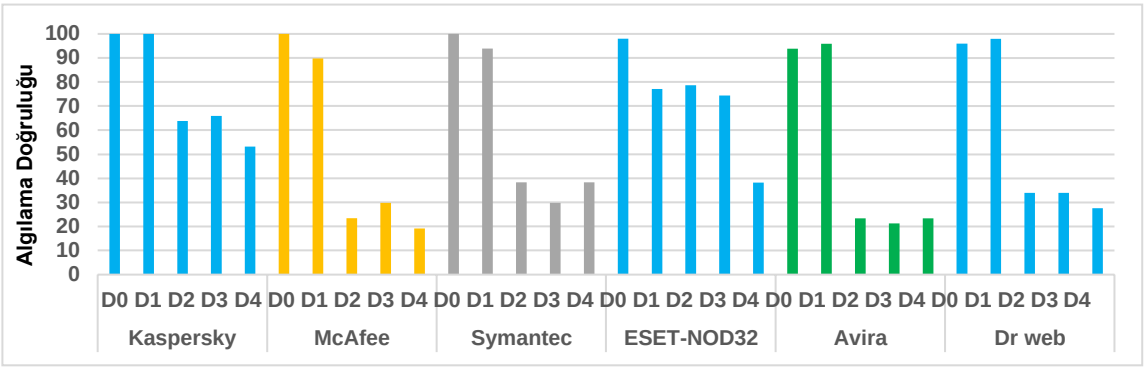
D0: Malgenom veri kümesinden 49 kötü amaçlı yazılım örneği rasgele olarak seçilmiştir (her kötü amaçlı yazılım ailesinden bir örnek alınmıştır.)
D1: String şifreleme gizleme tekniği, D0 veri kümesindeki örneklere uygulanmıştır. (bu veri kümesinin örnekleri PRAGuard ‘daki string şifreleme obfuskasyon veri kümesinden seçilmiştir.)
D2: Önerilen yeniden imzalama saldırısı, D1 veri kümesinin örneklerine uygulanmıştır.
D3: Önerilen izin enjeksiyon saldırısı, D1 veri kümesinin örneklerine uygulanmıştır.
D4: Önerilen izin-kodu enjeksiyon saldırısı, D1 veri kümesinin örneklerine uygulanmıştır.



Şekil 4.4.Değerlendirilen Anti virüs sistemlerinin algılama doğruluğu (Vaka çalışma3).

Çizelge 4.4. Dördüncü vaka çalışmasında oluşturulan veri kümeleri.

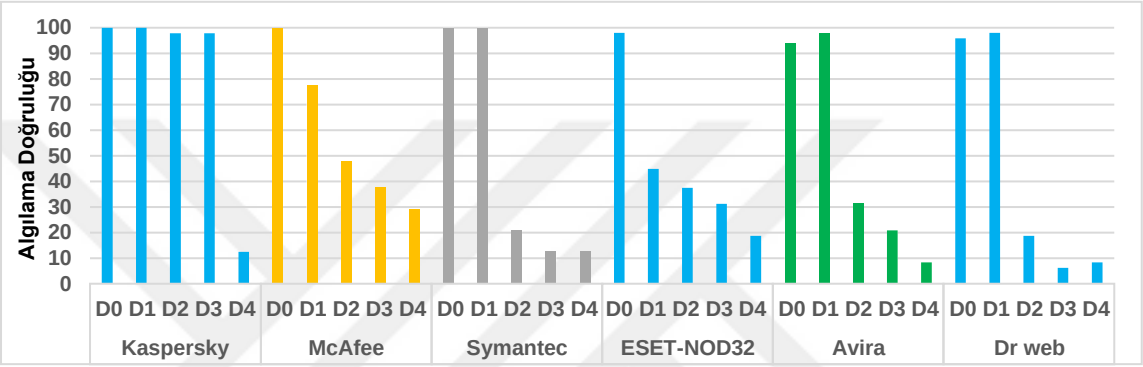
D0: Malgenom veri kümesinden 49 kötü amaçlı yazılım örneği rasgele olarak seçilmiştir (her kötü amaçlı yazılım ailesinden bir örnek alınmıştır).
D1: Sınıf şifreleme gizleme tekniği, D0 veri kümesindeki örneklere uygulanmıştır. (bu veri kümesinin örnekleri PRAGuard 'daki sınıf şifreleme obfuskasyon veri kümesinden seçilmiştir).
D2: Önerilen yeniden imzalama saldırısı, D1 veri kümesinin örneklerine uygulanmıştır.
D3: Önerilen izin enjeksiyon saldırısı, D1 veri kümesinin örneklerine uygulanmıştır.
D4: Önerilen izin-kodu enjeksiyon saldırısı, D1 veri kümesinin örneklerine uygulanmıştır.



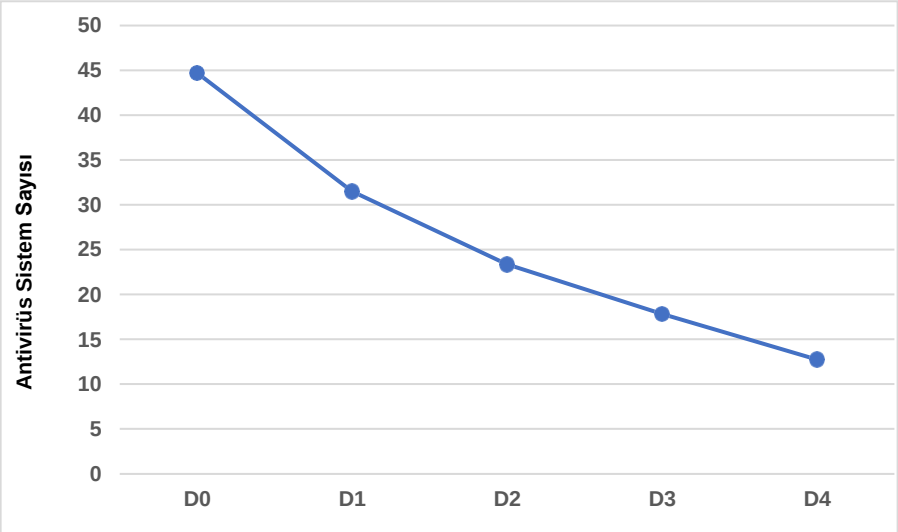
Şekil 4.5. Değerlendirilen Anti virüs sistemlerinin algılama doğruluğu (Vaka çalışma 4).

Çizelge 4.5. Beşinci vaka çalışmasında oluşturulan veri kümeleri.

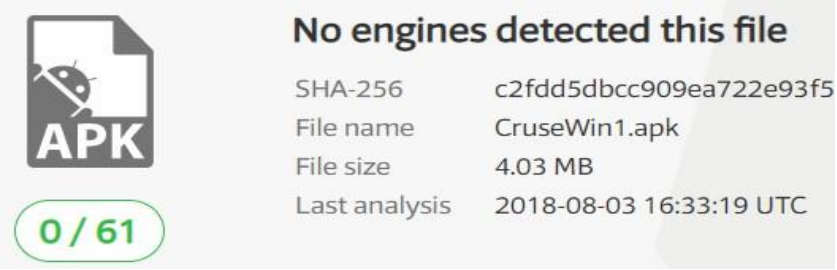
D0: Malgenom veri kümesinden 49 kötü amaçlı yazılım örneği rasgele olarak seçilmiştir (her kötü amaçlı yazılım ailesinden bir örnek alınmıştır).
D1: Önceki vaka çalışmalarında kullanılan tüm obfuskasyon teknikleri, D0 veri kümesindeki örneklere uygulanmıştır. (bu veri kümesinin örnekleri PRAGuard 'daki tam obfuskasyon veri kümesinden seçilmiştir).
D2: Önerilen yeniden imzalama saldırısı, D1 veri kümesinin örneklerine uygulanmıştır.
D3: Önerilen İzin enjeksiyon saldırısı, D1 veri kümesinin örneklerine uygulanmıştır.
D4: Önerilen izin-kodu enjeksiyon saldırısı, D1 veri kümesinin örneklerine uygulanmıştır.



Şekil 4.6. Değerlendirilen Anti virüs sistemlerinin algılama doğruluğu (Vaka çalışma 5).



Şekil 4.7. Beşinci vaka çalışmasına göre, kötücül yazılım örnekleri tespit edebilen ortalama anti virüs sistem sayısı.



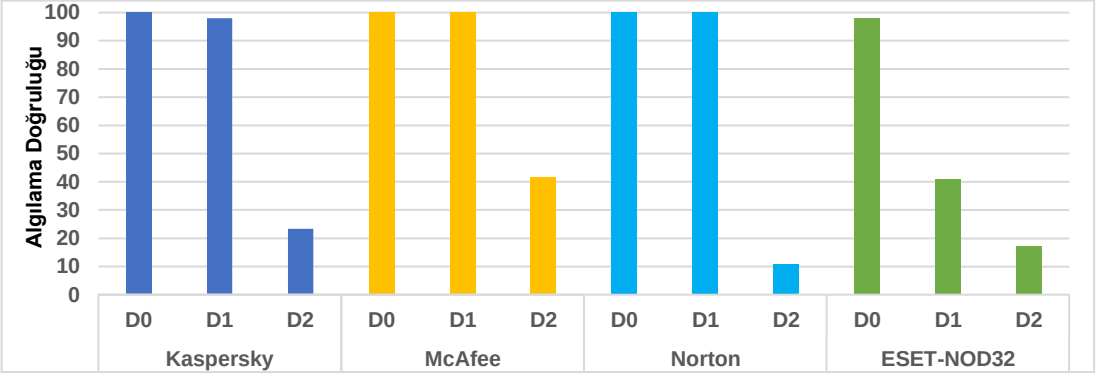
Şekil 4.8 . Kamufle edilmiş CrusWin kötü amaçlı yazılımın test sonuçları.

4.1.6. Vaka Çalışma 6

Bu vaka çalışması, önerilen saldırıların etkinliğinin, gerçek bir Android ortamda değerlendirmesini amaçlamaktadır. Bu amaçla, MEmu Android emülatörü kullanarak Norton Security ve Anti Virüs, McAfee Mobile Security, Kasper Mobile Anti Virüs ve Eset Mobile Security ve Anti Virüs olmak üzere dört farklı mobil anti virüs sisteminin algılama doğruluğunu test edilmiştir. Dolayısıyla, dört farklı MEmu emülatör kurulup, bunların içeriğine de yukarıda bahsi geçen dört mobil anti virüs sistemleri kurulmuştur. Ayrıca, söz konusu anti virüs sistemlerinin performansını test etmek amacıyla üç veri kümesi aşağıdaki gibi oluşturulmuştur:

- D0: Bu veri kümesi Malgenom'dan rastgele seçilen 49 örnek içermektedir (her kötü amaçlı yazılım ailesinden bir örnek seçilmiştir).
- D1: Bu veri kümesinde, tüm obfuskasyon teknikleri D0 veri kümedeki örneklerine uygulanmıştır (bu veri kümesinin örnekleri PRAGuard tam obfuskasyon veri kümesinden seçilmiştir).
- D3: Önerilen izin-kodu enjeksiyon saldırısı D1 veri kümedeki örneklerine uygulanmıştır.

İzin-kodu enjeksiyon saldırısı uygulandıktan sonra, incelenen tüm anti virüs sistemlerinin tespit doğruluğundaki önemli azalmayı şekil 4.9'te gösterilmektedir.
Bu vaka çalışmada yapılan testleri, 6 Mayıs 2019 ve 9 Mayıs 2019 tarihler arasında gerçekleştirilmiştir.



Şekil 4.9 . Altıncı vaka çalışmada değerlendirilen anti virüs sistemlerin algılama doğruluğu.

4.1.7. Vaka Çalışma 7

Bu vaka çalışması, kademeli olarak artan miktarda kod enjekte edilmesinin anti virüs sistemlerinin performansının üzerindeki etkisini test etmeyi amaçlamaktadır. Başka bir deyişiyle, enjekte edilen kod miktarı kötü amaçlı uygulama boyutuyla orantılı olarak arttığında, anti virüs sistemlerinin algılama performansını analiz etmeyi amaçlanmaktadır. Böylece, bu vaka çalışmada, zararlı olmayan izinleri ve Smali kodunu artan parçalar şeklinde enjekte edilmiştir. Bu amaçla, Drebin veri kümesinden 24 aileye ait 100 Android kötücül yazılım örneği seçilip dört farklı veri kümesi aşağıdaki gibi oluşturulmuştur:

- 1: 1 enjeksiyon oranı: Bu veri kümede, enjekte edilen iyi huylu izin sayısı, kötü amaçlı uygulamanın Manifest dosyasında bulunan orijinal izinler sayısına eşittir. Benzer şekilde, enjekte edilen iyi huylu Smali kod boyutu, kötü amaçlı uygulamanın orijinal Smali kodunun boyutuna eşittir.

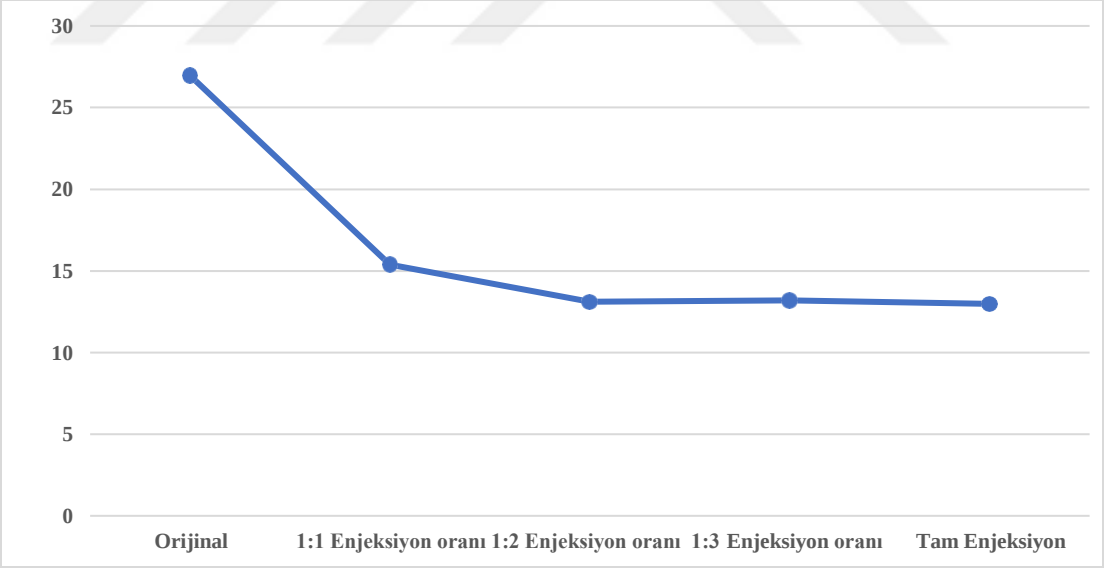
- 1: 2 enjeksiyon oranı: Bu veri kümede, enjekte edilen iyi huylu izin sayısı, kötü amaçlı uygulamanın Manifest dosyasındaki orijinal izin sayısının iki katıdır. Ayrıca, enjekte edilen iyi huylu Smali kod boyutu, orijinal kötü amaçlı uygulamanın Smali kodunun iki katıdır.

- 1: 3 enjeksiyon oranı: Bu veri kümede, enjekte edilen iyi huylu izin sayısı, kötü amaçlı uygulamanın Manifest dosyasındaki orijinal izin sayısının üç katıdır. Ayrıca, enjekte edilen iyi huylu Smali kod boyutu, orijinal kötü amaçlı uygulamanın Smali kodunun üç katıdır.

- Tam Enjeksiyon: Bu veri kümede, kötü amaçlı yazılım örneklerin kaynaklarına tüm Telegram uygulamasındaki Smali kodu ve izinleri (89 MB Smali kodu ve 56 izin) enjekte edilmiştir.

Şekil 4.10, oluşturulmuş kötü amaçlı yazılım örneklerini algılayabilen ortalama anti virüs sistem sayısı göstermektedir. Enjekte edilen kod oranı orijinal kodun iki katını aştığında ortalama sistem sayısının neredeyse sabit hale geldiğini gözlemlenmiştir.

Bu vaka çalışmasında testleri, 01-04-2019 ve 05-04-2019 arasında VirusTotal kullanılarak gerçekleştirilmiştir.



Şekil 4.10. Enjekte edilen kod miktarının etkisini analiz edilmesi.

4.1.8. Önerilen Saldırıların Zaman Karmaşıklığının Analiz Edilmesi

Bu bölümde, önerilen saldırıların zaman karmaşıklığını incelenmiştir. Daha önce de belirtildiği gibi, tüm saldırı senaryoları otomatik olarak yürütebilecek bir Python modeli geliştirilmiştir. Geliştirilen modelin zaman karmaşıklığı n kötücül yazılım numuneye saldırmak için $O(n^2)$ ve bir kötücül yazılım numuneye saldırmak için $O(n)$ 'dir. Ayrıca, önerilen saldırıların her aşamasını yürütmek için harcanan zaman saniye cinsinden Çizelge 4.6'da hesaplanmıştır.

4.1.9. Önerilen Enjeksiyon Saldırılarının Sonuçlarının Tartışılması

Deney sonuçlar, önerilen saldırıların, kötü amaçlı yazılımdan koruma sistemlerinin performansını azaltmadaki etkinliğini göstermiştir. Bazı kötü amaçlı yazılımdan koruma sistemlerinin ilkel bir algılama yöntemine bağlı olduğunu fark edilmiştir. Bu ilkel yöntem, önerilen yeniden imzalama saldırısı uygulandıktan sonra kötü amaçlı yazılım örneklerini algılayabilen sistem sayısındaki azalmasından gözlemlenmiştir. Ayrıca, elde edilen sonuçları, önerilen saldırıların geleneksel şaşırtma ve gizleme tekniklerinden daha etkin olduğunu göstermiştir. Ek olarak, önerilen saldırıların bazı gizleme teknikleri ile melezleştirilmesiyle derin kamuflaj edilmiş kötü amaçlı yazılım örnekleri elde edilmiştir. Elde edilen kamuflaj edilmiş kötü amaçlı yazılım örneklerinden bazıları, 60'tan fazla anti virüs sistemlerinden hiçbir sistem tarafından algılanamadığını ortaya çıkmıştır. McAfee, Symantec gibi bazı kötü amaçlı yazılımdan koruma sistemlerinin kararının, onların imza veri tabanları diğer kötü amaçlı yazılımdan koruma sistemlerinin kararlarına göre güncellenmiş gibi kısa bir süre içinde değişebileceğini görülmüştür. Ancak bunun sebebi tam olarak bilmek mümkün olmamaktadır, çünkü ticari şirketler, kullandığı tespit yöntemleri açıkça ortaya koymamaktadır. Ayrıca, kamuflaj edilmiş kötü amaçlı yazılımları gerçek bir Android ortamda test edildiğinde, VirusTotal tarafından elde edilen sonuçlara kıyasıyla McAfee gibi bazı kötü amaçlı yazılım önleme sistemlerinin performansında bazı değişiklikleri fark edilmiştir. Bu değişikliği, iki vaka çalışmasının deneyleri arasındaki zaman farkından kaynaklanmış olabilmektedir. Ayrıca, VirusTotal tarafından kullanılan anti virüs sistem sürümlerinin cep telefonlarında kullanılanlardan farklı olmasından kaynaklanmış olabilmektedir.

Çizelge 4.6. Önerilen saldırılardaki hesaplama süresinin analiz edilmesi

Enjeksiyon Oranı	APK geri derleme	İzin enjeksiyonu	Kod enjeksiyonu	APK yeniden derleme	APK yeniden imzalaması	Toplam süre/Saniye
1:1	2.72	0.011	4.42	8.77	1	17.021
1:2	2.72	0.037	7.70	9.14	1.05	20,65
1:3	2.72	0.04	12.14	52.61	1.18	68.69
Tam Enjeksiyon	2.72	0.05	18.06	61.83	2.37	85.03

4.2. Önerilen Android Kötücül Yazılım Sınıflandırma Modellerinin Sonuçları

Gerçekleştirilen tüm çok sınıflı ve iki sınıflı sınıflandırma deneyleri 8 çekirdekli Intel Xeon E5-2680 bir işlemci ve 64 GB RAM özelliğine sahip bir sunucu kullanılarak uygulanmıştır. VisDroid, DeepVisDroid ve diğer önerilen sınıflandırma modelleri, 1500'den fazla Python kod satırı kullanılarak geliştirilmiştir. Önerilen sınıflandırma modellerin geliştirilmesinde, OpenCV, Sklearn, Keras, Tensorflow gibi birden çok iyi bilinen Python kütüphaneleri kullanılmıştır. GSÇ parametreleri ampirik olarak ayarlanmış olup 120 büyüklüğünde bir kelime dağarcığı (sınıfı sayısı × 10) seçilmiştir. Başka bir deyişle, oluşturulmuş GSÇ tabanlı lokal özellik vektörleri 120 uzunluğa sahiptir. Ayrıca, Sklearn Python kütüphanesine ait olan MiniBatchKMeans algoritması, GSÇ algoritmasında bir kümeleme yöntemi olarak kullanılmıştır. Ayrıca, veri kümesinin %90'ı önerilen modelleri eğitmek için ve %10'u önerilen modelleri test etmek için kullanılacak şekilde, 10-katlı çapraz doğrulaması (k = 10 k-fold cross-validation yaklaşımında) kullanılmıştır. Önerilen modellerin performansını değerlendirmek için genel sınıflandırma doğruluğu ve hesaplama maliyeti metrik olarak kullanılmıştır.

Birden çok sınıflandırma modeli önerilip Android yazılımları sınıflandırmak için iki farklı amaçlı deney gruplar altında eğitilmiştir. İlk deney grubunda, Android kötü amaçlı yazılım örnekleri birçok aileye sınıflandırmak için (çok sınıflı sınıflandırma) VisDroid adlı bir model önerilmiştir. Önerilen VisDroid 'te, birçok geleneksel makine öğrenme sınıflandırıcısı bağımsız olarak veya topluluk oylama sınıflandırıcı şeklinde eğitilip Android kötücül uygulamaları birden fazla aileye sınıflandırmak için

kullanılmayı önerilmiştir. Ayrıca, kötü amaçlı yazılım örnekleri birçok kötücül yazılım aileye sınıflandırmak için ResNet ve Inception V3 olmak üzere iki tane iyi bilinen derin öğrenme modeli kullanılmıştır.

İkinci denme grubu, Android uygulamaları kötücül yazılım veya iyi huylu yazılım (iki sınıflı sınıflandırma) olarak sınıflandırmayı amaçlamaktadır. Bu amaçla, önerilen VisDroid geleneksel makine öğrenme sınıflandırıcılar tabanlı modeli, önerilen DeepVisDroid hibrit derin öğrenme tabanlı modeli, önerilen iki geleneksel ESA derin öğrenme modeli ve iki tane iyi bilinen derin öğrenme modeli kullanılmıştır.
Şekil 4.11'da yukarıda bahsedildiği iki deney grup kapsamında önerilen modelleri ve gerçekleştirilen deneyleri özetlenmiştir.

4.2.1. Önerilen Çok Sınıflı Sınıflandırma Modelleri

Çalışmanın bu bölümde, Android kötü amaçlı yazılım örnekleri, ailelerine göre sınıflandırmak için geleneksel makine öğrenme sınıflandırıcılarına dayanan ve VisDroid olarak adlandırılan bir model önerilmiştir. Ayrıca, Android kötü amaçlı yazılım örnekleri, ailelerine göre sınıflandırmak için Residual Network (ResNet) ve Inception v3 olmak üzere iki iyi bilinen derin öğrenme modeli kullanılmıştır.

4.2.1.1. Önerilen VisDroid Çok Sınıflı Sınıflandırma Model

Şekil 4.11'de gösterildiği gibi önerilen VisDroid modeli kullanılarak oluşturulmuş kötücül yazılım görüntü veri kümelerinin her birine dayanarak dört deney grubu gerçekleştirilmiştir. Yürütülen her deneyinde ayaklanan özellikleri, Rastgele orman, K-en yakın komşu, Karar ağacı, Torbalama (Bagging), AdaBoost (Adaptive Boosting) ve Gradyan Artırma (Gradient Boosting) olmak üzere altı tane geleneksel makine öğrenme sınıflandırıcıları eğitmek ve test etmek için kullanılmıştır. İlk deney grubunda, söz konusu makine öğrenme sınıflandırıcıları eğitmek için lokal özellikleri oluşturulmuş kötücül yazılım görüntü veri kümelerinden ayıklanıp kullanılmıştır. Ayrıca, lokal özellikleri kullanılarak eğitilen sınıflandırıcıları, önerilen lokal özellik tabanlı topluluk oylama sınıflandırıcısında karar vermek için seçmen olarak kullanılmıştır. İkinci deney grubunda, oluşturulmuş kötücül yazılım görüntü veri

kümelerinin her birinden global özellikleri ayıklanıp yukarıda bahsedilen altı makine öğrenme sınıflandırıcıları bağımsız bir şekilde eğitmek için kullanılmıştır. Ayrıca, global özelliklerine dayanarak eğitimini gören sınıflandırıcıları, global özellik tabanlı bir topluluk oylama sınıflandırıcısı oluşturmak için seçmen olarak kullanılmıştır. Birinci ve ikinci deney grubu, Android kötücül yazılım algılama alanında görüntü tabanlı lokal ve global özelliklerin etkinliğini ayrı ayrı test etmeyi amaçlamaktadır. Üçüncü deney grubunda, melez bir topluluk oylama sınıflandırıcısı oluşturmak için görüntü tabanlı lokal ve global özelliklerinin melezleştirilmesini önerilmiştir. Dördüncü deney grubunda, derin öğrenme teknikleri, önerilen geleneksel makine öğrenme sınıflandırıcılar tabanlı VisDroid modeli ile karşılaştırmak için oluşturulmuş görüntü veri kümelerinin bazılarını kullanılarak iki iyi bilinen derin öğrenme modeli test edilmiştir. Aşağıdaki paragraflarda gerçekleştirilen dört çok sınıflı sınıflandırma deney grupların her birinde elde edilen sonuçları anlatılmıştır.

I. Görüntü Tabanlı Lokal Özelliklerine Dayanan VisDroid Modeli Kullanılarak Elde Edilen Çok Sınıflı Sınıflandırma Sonuçları

Oluşturulan görüntü veri kümelerinin her birinden lokal özellikleri ayıklanıp daha önce belirtilen altı makine öğrenme sınıflandırıcıları içeren dört makine öğrenme sınıflandırıcı grubu eğitilmiştir. Her sınıflandırıcı grubu, bu tür özelliklerin verimliliğini test etmek için ayıklanan lokal özelliklerden biri kullanılarak bağımsız olarak eğitilmiştir. Bundan sonra, her grupta eğitilen sınıflandırıcıları, çoğunluk oylama (hard-voting) sınıflandırıcıda seçmen olarak kullanılmıştır. Dolayısıyla, her biri farklı bir lokal özellik (yani, ÖBÖD, HSÖ, YDB veya KAZE) ile eğitilmiş bir sınıflandırıcı grubuna dayanan dört oylama sınıflandırıcısı elde edilmiştir. Lokal özellikleri kullanılarak elde edilen sonuçları Çizelge 4.7'de gösterilmiş olup her birini aşağıdaki paragraflarda tartışılacaktır:

1) Manifest Dosya Görüntü Veri Kümesinden Ayıklanan Lokal Özellikleri Kullanılarak Elde Edilen Sonuçları

Manifest tabanlı görüntü veri kümesinden ayıklanan lokal özellikleri kullanarak dört sınıflandırıcı grubu eğitilmiştir, böylece her bir sınıflandırıcı grup, farklı bir

lokal özelliği kullanılarak eğitimini görmüştür. Bu veri kümesinden ayıklanan ÖBÖD özellikleri kullanıldığında sınıflandırıcıların sınıflandırma doğruluğunun çoğu %90'a aştığını ve en iyi lokal özellik tabanlı sınıflandırma doğruluğu elde edildiğini fark edilmiştir. Öte yandan, YDB özelliği kullanıldığında %45 ile %66,5 arasında bir sınıflandırma doğruluğuna ulaşıp en kötü sonuçları elde edildiğini gözlemlenmiştir.

2) Manifest ve Resources.Arsc Dosyalar Tabanlı Görüntü Veri Kümesinden Ayıklanan Lokal Özellikleri Kullanılarak Elde Edilen Sonuçları

Bu deneyde, lokal özellikleri Manifest ve Resources.arsc dosyalar tabanlı görüntü veri kümesinden ayıklanmıştır. Ayıklanan özellikleri dört sınıflandırıcı grubu eğitmek için kullanılmıştır, böylece her özellik farklı bir sınıflandırıcı grubunu eğitmek için kullanılmıştır. Bundan sonra, her bir grupta eğitilen sınıflandırıcıları, dört oylama sınıflandırıcısında seçmen olarak kullanılmıştır. Bu deneydeki en iyi sınıflandırma doğruluğu, %90.30'a ulaşmış olup KAZE özelliği kullanılarak elde edilmiştir.

3) DEX Kod Dosya Tabanlı Görüntü Veri Kümesinden Ayıklanan Lokal Özellikleri Kullanılarak Elde Edilen Sonuçları

Bu deneyde, dört sınıflandırıcı grubunu eğitmek için DEX dosya tabanlı görüntü veri kümesinden ayıklanan lokal özellikleri kullanılmıştır. Ayrıca, her grupta eğitilen sınıflandırıcıları, bir oylama sınıflandırıcısında karar almak için seçmen olarak kullanılmıştır. Genel olarak, bu deneydeki tüm eğitilen sınıflandırıcıların sınıflandırma doğruluğunda, önceki deneylere kıyasla bir iyileşme gözlemlenmiştir. Bu deneydeki en iyi sınıflandırma doğruluğu, %91.74'e ulaşmış olup ayıklanan KAZE özelliği kullanılarak eğitilen oylama sınıflandırıcısı kullanılarak elde edilmiştir.

4) Manifest, Resources.Arsc ve DEX Kod Dosyalar Tabanlı Görüntü Veri Kümesinden Ayıklanan Lokal Özellikleri Kullanılarak Elde Edilen Sonuçları

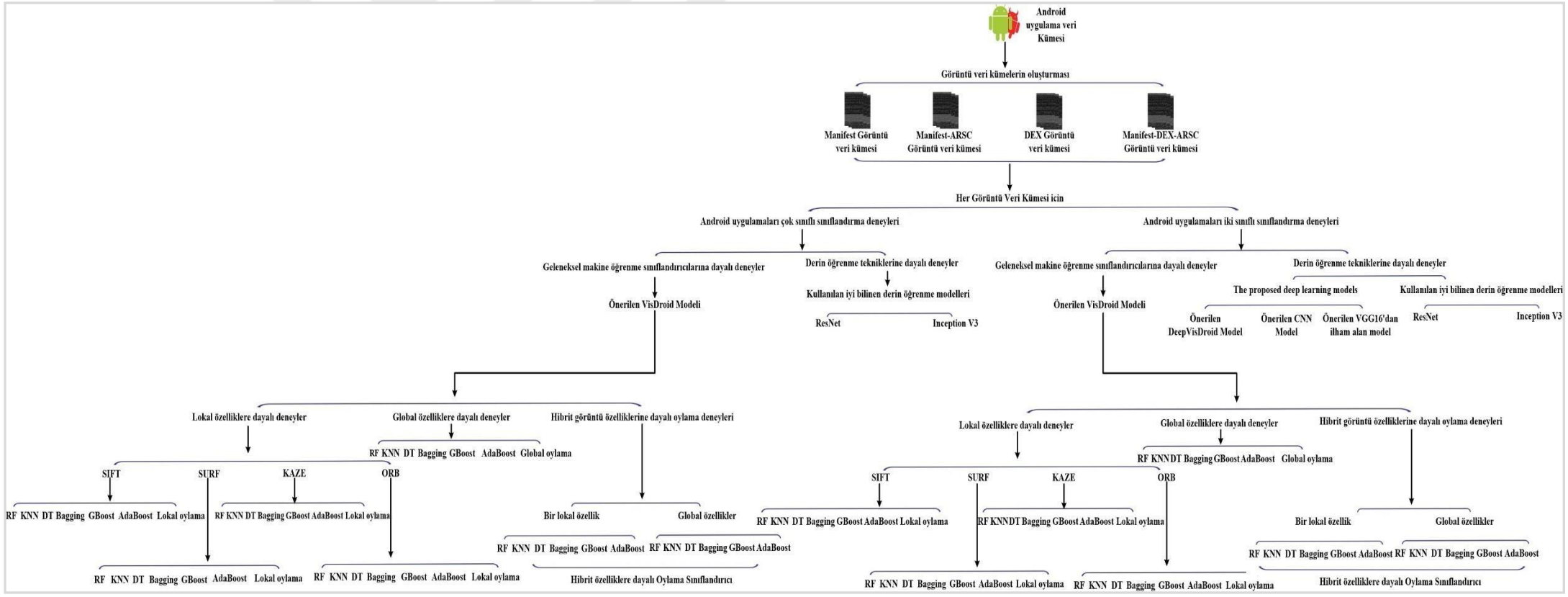
Bu deneyde, lokal özellikleri Manifest, ARSC ve DEX dosyaları kullanılarak oluşturulan görüntü veri kümesinden ayıklanmıştır. Ayıklanan lokal özelliklerin her birini, farklı bir sınıflandırıcı grubunu eğitmek ve test etmek için kullanılmıştır. Her grupta eğitilen sınıflandırıcıları bir oylama sınıflandırıcısı oluşturmak için üyeler (seçmen) olarak kullanılmıştır. Bu deneydeki en iyi sınıflandırma doğruluğu, %89.48 'i yakalamış olup bu veri kümesinden ayıklanan KAZE lokal özellikleri kullanılarak eğitilen Rastgele Orman sınıflandırıcısı kullanılarak elde edilmiştir.

II. Görüntü Tabanlı Global Özelliklere Dayanan VisDroid Modeli Kullanılarak Elde Edilen Çok Sınıflı Sınıflandırma Sonuçları

Daha önce bahsedilen üç global özellik ayıklanmış ve bir özellik vektörü oluşturmak üzere yan yana istif edip normaliz edilmiştir. İnşa edilen global özellik vektörleri, kötü amaçlı yazılım örnekleri, ailelerine göre sınıflandırmak üzere önerilen VisDroid modelde kullanılan sınıflandırıcı grupları eğitmek için kullanılmıştır. Global özellikleri kullanılarak elde edilen sonuçları, çizelge 4.8'de gösterilmiş olup her birini aşağıdaki paragraflarda açıklanmıştır:

1) Manifest Dosya Görüntü Veri Kümesinden Ayıklanan Global Özellikleri Kullanılarak Elde Edilen Sonuçları

Manifest tabanlı görüntü veri kümesinden ayıklanan global özellikleri kullanılarak elde edilen en iyi sınıflandırma doğruluğu, %92.16'ya ulaşmış olup Rastgele Orman sınıflandırıcısı kullanılarak elde edilmiştir.



Şekil 4.11. Gerçekleştirilen deneylerin özeti.

ResNet: Artık Ağ; RF: Rastgele Orman sınıflandırıcısı; KNN: K-En Yakın Komşular sınıflandırıcısı; DT: Karar Ağaçlar sınıflandırıcısı; Bagging: Torbalama sınıflandırıcısı; AdaBoost: AdaBoost sınıflandırıcı; GBoost: Gradyan Boost sınıflandırıcısı; Lokal Oylama: Lokal özelliklere dayalı oylama sınıflandırıcısı; Global Oylama: global özelliklere dayalı oylama sınıflandırıcısı; Hibrit Oylama: Lokal ve global özelliklere dayalı oylama sınıflandırıcısı;

2) Manifest ve Resources.Arsc Dosyalar Tabanlı Görüntü Veri Kümesinden Ayıklanan Global Özellikleri Kullanılarak Elde Edilen Sonuçları

Bu deneyde, global özellikleri Manifest ve Resources.arsc dosyaları kullanılarak oluşturulan veri kümesinden ayıklanıp önerilen VisDroid modeli eğitmek için kullanılmıştır. Bu deneydeki en iyi sınıflandırma doğruluğu, %92.16'ya ulaşmış olup AdaBoost sınıflandırıcısı kullanılarak elde edilmiştir.

3) DEX Kod Dosya Tabanlı Görüntü Veri Kümesinden Ayıklanan Global Özellikleri Kullanılarak Elde Edilen Sonuçları

Bu deneyde, DEX kod dosyaları kullanılarak oluşturulan görüntü veri kümesinden ayıklanan global özellikleri, önerilen VisDroid modeli eğitmek için kullanılmıştır. Bu deneydeki en iyi sınıflandırma doğruluğu, 92.77'ye ulaşmış olup AdaBoost ve Rastgele Orman sınıflandırıcıları kullanılarak elde edilmiştir.

4) Manifest, Resources.Arsc ve DEX Kod Dosyalar Tabanlı Görüntü Veri Kümesinden Ayıklanan Global Özellikleri Kullanılarak Elde Edilen Sonuçları

Bu deneyde, önerilen VisDroid modeli eğitmek için Manifest, Dex ve Resources.arsc dosyaları kullanılarak oluşturulan veri kümesinden ayıklanan global özellikleri kullanılmıştır. Bu deneyin en iyi sınıflandırma doğruluğu, %90.52 'ye ulaşmış olup AdaBoost sınıflandırıcısı kullanılarak elde edilmiştir.

III. Önerilen Oylama Sınıflandırıcılarının Sonuçları

Çizelge 4.9 'da lokal özellik tabanlı oylama sınıflandırıcısı, global özellik tabanlı oylama sınıflandırıcısı ve melez özelliklere dayalı oylama sınıflandırıcısının her birinden elde edilen sonuçları gösterilmektedir. Önerilen melez oylama sınıflandırıcısının, Manifest veri kümesinden ayıklanan özellikleri kullanılarak eğitildiğinde, %93.40 bir sınıflandırma doğruluğu elde etiğini ve en iyi sınıflandırma doğruluğu verdiğini gözlemlenmiştir.

I. VisDroid Model Tabanlı Çok Sınıflı Sınıflandırma Sonuçlarının İyileştirilmesi

Önerilen modelde kullanılan makine öğrenme sınıflandırıcılarının sınıflandırma raporlarını ve hata matrislerini incelendiğinde, AnserverBot ve BaseBridge kötücül yazılım ailelerin arasında yanlış tahminlerin en büyük oranının olduğunu fark edilmiştir. Bu nedenle, bu iki kötücül yazılım ailelerideki kaynaklar arasında büyük bir benzerlik olduğu sonucuna varılmıştır. Örneğin, Çizelge 4.10, Manifest-Aarsc-Dex görüntü veri kümesinden ayıklanan ÖBÖD özellikleri kullanılarak eğitilmiş Rastgele orman sınıflandırıcısının hata matrisini göstermektedir. Ayrıca Çizelge 4.11, DEX görüntü veri kümesinden ayıklanan KAZE özellikleri kullanılarak eğitilen Karar Ağacı sınıflandırıcısının hata matrisini göstermektedir. Dolayısıyla, önerilen modelin performansının nasıl etkilenebileceğini test etmek için AnserverBot ve BaseBridge kötücül yazılım ailelerdeki görüntüleri birleştirmeye karar verilmiştir. Başka bir deyişle, kötücül yazılım örneklerinin sınıflandırmasını on iki sınıf yerine on bir sınıfa göre test edilmiştir. Yukarıda bahsedilen iki ailenin örneklerinin birleştirilmesinden sonra hem lokal hem de global özellikleri kullanılarak elde edilen sınıflandırma sonuçlarında önemli bir iyileşme gözlemlenmiştir. Bahsedilen iki kötücül yazılım ailesinin birleştirilmesinden sonra ayıklanan lokal özellikleri kullanılarak elde edilen sonuçları, çizelge 4.12'da gösterilmektedir. Lokal özellikleri Manifest görüntü veri kümesinden ayıklandığında, en iyi sınıflandırma doğruluğu %95.46'ya ulaşmış olup ÖBÖD lokal özelliği kullanılarak eğitilen rastgele orman sınıflandırıcısı kullanılarak elde edilmiştir. Öte yandan, lokal özellikleri Manifest ve Resources.arsc dosyalar tabanlı görüntü veri setinden ayıklandığında, en iyi sınıflandırma doğruluğu %93,96'ya ulaşmış olup rastgele orman sınıflandırıcısı, HSÖ özelliği kullanılarak eğitildiğinde elde edilmiştir. Ayrıca, lokal özellikleri DEX veri kümesinden ayıklandığında, en iyi lokal özellik tabanlı sınıflandırma doğruluğu %96'a aşmış olup HSÖ özelliği kullanılarak eğitilen AdaBoost sınıflandırıcısından elde edilmiştir. Son olarak, lokal özellikleri, Manifest, Resources.arsc ve DEX kod dosyalar tabanlı veri kümesinden ayıklandığında, en iyi sınıflandırma doğruluğu yaklaşık %94.85'e ulaşmış olup Rastgele Orman ve AdaBoost sınıflandırıcıları eğitmek için, ÖBÖD özelliği kullanıldığında elde edilmiştir.

Çizelge 4.7. Ayıklanan lokal özellikleri kullanılarak elde edilen çok sınıflı sınıflandırma doğrulukları.

Özellik	Veri Kümesi	RF	K-NN	DT	Bagging	AdaBoost	Gboost	Oylama
ÖBÖD	Manifest	91.75	87.42	87.00	87.83	90.10	91.96	91.96
	Manifest-Resources.arsc	85.97	82.68	80.00	82.26	84.50	85.10	85.36
	DEX	89.00	85.95	86.98	84.92	90.50	88.84	88.43
	Manifest-DEX-ARSC	87.22	85.57	82.68	85.15	87.84	87.42	87.22
HSÖ	Manifest	89.48	84.90	83.90	84.54	88.20	87.40	88.86
	Manifest-Resources.arsc	87.20	84.12	80.80	84.30	86.59	85.36	86.19
	DEX	91.73	86.57	86.16	85.95	90.50	89.00	89.67
	Manifest-DEX-ARSC	87.63	85.36	83.30	83.71	87.01	87.42	87.84
KAZE	Manifest	85.50	76.90	82.80	78.30	86.80	84.53	88.00
	Manifest-Resources.arsc	87.42	83.50	83.30	82.00	87.40	84.95	90.30
	DEX	91.32	90.08	89.67	87.19	91.53	91.32	91.74
	Manifest-DEX-ARSC	89.48	87.22	86.39	88.04	87.84	88.25	88.04
YDB	Manifest	66.50	45.00	64.10	55.00	64.50	61.00	61.80
	Manifest-Resources.arsc	77.94	55.46	71.55	65.98	78.97	73.61	74.43
	DEX	80.78	72.10	76.86	76.45	81.82	82.44	82.23
	Manifest-DEX-ARSC	80.82	75.46	75.46	79.38	81.65	80.21	80.62

RF: Rastgele Orman; KNN: K-En Yakın Komşu; DT: Karar Ağacı; Bagging: Torbalama sınıflandırıcısı; AdaBoost: AdaBoost sınıflandırıcısı; GBoost: Gradient Boost sınıflandırıcısı; Voting: lokal özelliklere dayalı oylama sınıflandırıcısı;

Çizelge 4.8. Ayıklanan global özellikleri kullanarak önerilen VisDroid modeli eğitildiğinde elde edilen çok sınıflı sınıflandırma doğrulukları.

Görüntü Veri Kümesi	RF	KNN	DT	Bagging	AdaBoost	GBoost	Oylama
Manifest	92.16	84.10	89.89	85.77	90.72	91.75	90.52
Manifest-Resources.arsc	91.90	86.80	88.24	85.98	92.16	90.10	89.80
DEX	92.77	90.29	90.29	88.43	92.77	91.94	91.32
Manifest-ARSC-DEX	90.10	88.45	88.00	88.66	90.52	89.48	88.87

Çizelge 4.9. Önerilen oylama sınıflandırıcıları kullanılarak elde edilen çok sınıflı sınıflandırma doğrulukları.

Veri kümesi	Lokal	Global	Hibrit
Manifest	91.96	92.16	93.40
Manifest-Resources.arsc	90.3	92.16	91.34
DEX	91.73	91.32	92.56
Manifest-ARSC-DEX	88.04	88.87	90.10

Öte yandan, daha önce bahsedilen iki kötücül yazılım ailesinin birleştirilmesinden sonra ayıklanan global özellikleri kullanılarak elde edilen sonuçları, çizelge 4.13 'te gösterilmiştir. Manifest dosya tabanlı görüntü veri setinden ayıklanan global özellikleri, Rastgele Orman ve AdaBoost sınıflandırıcıları eğitmek için kullanıldığında, yaklaşık %98.2 bir sınıflandırma doğruluğu elde edilmiştir. Ayrıca, önerilen VisDroid modeli eğitmek için Manifest ve Resources.arsc dosyalar tabanlı görüntü veri kümesinden ayıklanan global özellikleri kullanıldığında, en iyi sınıflandırma doğrulukları %96.9 'ya ve %96.7 'ye ulaşmış olup AdaBoost ve Rastgele Orman sınıflandırıcıları kullanılarak elde edilmiştir. DEX kod tabanlı görüntü veri kümesinden global özellikleri ayıklandığında, en iyi sınıflandırma doğrulukları %98.14 ve %97.3'e ulaşmış olup Rastgele Orman ve AdaBoost sınıflandırıcıları kullanılarak elde edilmiştir. Son olarak, Manifest-Aarsc-Dex görüntü veri setinden ayıklanan global özellikleri, önerilen VisDroid modeli eğitmek için kullanıldığında, Rastgele Orman sınıflandırıcısı kullanılarak elde edilen sınıflandırma doğruluğu %97.73'e ulaşmıştır.

Çizelge 4.14, önerilen lokal özellik tabanlı oylama sınıflandırıcısı, global özellik tabanlı oylama sınıflandırıcısı ve melez özellik tabanlı topluluk oylama sınıflandırıcısı (Ensemble Voting) kullanılarak elde edilen sonuçları göstermektedir. Önerilen melez sınıflandırıcısı, %97.73'e ulaşan bir sınıflandırma doğruluğu yakalamış olup oylama sınıflandırıcıların arasında en iyi sınıflandırma doğruluğu verdiğini gözlemlenmiştir.

Çizelge 4.10. Manifest-Aarsc-Dex kötücül yazılım görüntü veri kümesinden ayıklanan özellikleri kullanılarak eğitilen rastgele orman sınıflandırıcısının hata matrisi.

	ADRD	AnserverBot	BaseBridge	DroidKungFu	FakeDoc	FakeInstaller	Geinimi	GinMaster	Iconosys	Kmin	Opfake	Plankton
ADRD	6	0	0	3	0	0	0	2	0	0	0	0
AnserverBot	0	3	18	0	0	0	0	0	0	0	0	0
BaseBridge	0	23	27	0	0	0	0	0	0	0	0	0
DroidKungFu	0	0	0	99	0	0	0	0	0	0	0	0
FakeDoc	0	0	0	1	10	0	0	0	0	0	0	0
FakeInstaller	0	0	1	0	0	90	0	0	0	0	0	0
Geinimi	0	0	0	0	0	0	8	0	0	0	0	0
GinMaster	0	0	0	12	0	0	1	23	0	0	0	1
Iconosys	0	0	0	1	0	0	0	0	13	0	0	0
Kmin	0	0	0	0	0	0	0	0	0	22	0	0
Opfake	0	0	0	0	0	4	0	0	0	0	63	0
Plankton	0	0	0	0	0	0	0	0	0	0	0	54

Çizelge 4.11. Dex kod tabanlı kötücül yazılım görüntü veri kümesinden ayıklanan özellikleri kullanılarak eğitilen karar ağacı sınıflandırıcısının hata matrisi.

	ADRD	AnserverBot	BaseBridge	DroidKungFu	FakeDoc	FakeInstaller	Geinimi	GinMaster	Iconosys	Kmin	Opfake	Plankton
ADRD	4	0	0	3	0	0	0	1	0	0	0	1
AnserverBot	0	10	2	0	1	0	0	0	0	0	0	0
BaseBridge	0	20	28	0	1	0	0	2	0	0	0	1
DroidKungFu	2	0	0	88	0	0	0	1	0	0	0	1
FakeDoc	0	0	0	0	8	0	0	0	0	0	0	0
FakeInstaller	0	0	0	0	0	89	0	0	0	0	2	0
Geinimi	0	0	0	0	0	0	12	2	0	0	0	0
GinMaster	1	0	2	6	1	1	0	22	0	0	0	2
Iconosys	0	0	0	1	0	0	0	0	10	0	0	1
Kmin	1	0	0	0	0	0	0	0	0	14	0	0
Opfake	0	0	0	0	0	3	0	0	0	0	63	1
Plankton	3	1	0	0	0	0	0	0	0	1	2	69

Çizelge 4.12. AnserverBot ve BaseBridge kötü amaçlı yazılım aileleri birleştirdikten sonra, ayıklanan lokal özellikleri kullanılarak elde edilen çok sınıflı sınıflandırma doğrulukları.

Özellik	Veri kümesi	RF	K-NN	DT	Bagging	AdaBoost	Gboost	Oylama
ÖBÖD	**Manifest**	95.46	92.99	91.13	90.93	95.46	95.05	95.05
	Manifest-Resources.arsc	92.58	87.63	87.63	87.42	92.16	89.69	91.75
	DEX	93.18	90.08	89.87	87.81	95.25	93.39	91.74
	Manifest-DEX-ARSC	94.85	90.10	90.31	89.28	94.85	94.02	92.58
HSÖ	**Manifest**	91.55	85.77	86.80	84.33	91.13	89.28	89.48
	Manifest-Resources.arsc	93.96	88.87	87.63	87.63	93.20	91.75	92.78
	DEX	95.87	92.15	92.77	90.50	96.07	94.01	93.80
	Manifest-DEX-ARSC	94.23	92.58	91.75	91.96	94.43	94.64	94.02
KAZE	**Manifest**	90.10	81.85	87.42	84.54	88.45	88.86	88.25
	Manifest-Resources.arsc	91.96	84.12	88.04	82.47	92.16	90.52	89.48
	DEX	94.83	91.73	90.70	90.29	94.42	93.80	93.60
	Manifest-DEX-ARSC	94.02	92.16	90.72	88.87	94.23	94.43	92.99
YDB	**Manifest**	70.72	46.80	67.83	62.06	70.72	64.74	67.63
	Manifest-Resources.arsc	82.68	64.33	78.97	66.80	83.50	78.35	79.18
	DEX	87.81	81.20	82.02	81.82	89.26	90.50	88.22
	Manifest-DEX-ARSC	89.28	83.71	84.12	84.74	91.75	89.48	89.69

Çizelge 4.13. AnserverBot ve BaseBridge kötü amaçlı yazılım aileleri birleştirdikten sonra ayıklanan global özellikleri kullanılarak elde edilen çok sınıflı sınıflandırma doğrulukları.

Görüntü Veri Kümesi	RF	KNN	DT	Bagging	AdaBoost	GBoost	Oylama
Manifest	98.14	88.66	95.26	88.87	98.14	97.73	95.67
Manifest-Resources.arsc	96.70	91.55	94.43	89.90	97.53	96.91	94.85
DEX	98.14	92.36	93.80	91.12	97.31	95.87	95.87
Manifest-ARSC-DEX	97.73	92.78	94.02	92.78	96.90	96.49	95.26

Çizelge 4.14. AnserverBot ve BaseBridge kötücül yazılım aileleri birleştirdikten sonra önerilen oylama sınıflandırıcıları kullanılarak elde edilen çok sınıflı sınıflandırma doğrulukları.

Veri kümesi	Lokal	Global	Hibrit
Manifest	95.05	95.67	97.74
Manifest-Resources.arsc	92.78	94.85	96.5
DEX	93.8	95.87	96.91
Manifest-ARSC-DEX	94.02	95.26	96.91

4.2.1.2. Bazı Son Teknoloji Derin Öğrenme Modelleri Kullanarak, Android Kötücül Yazılımların Çok Sınıflı Sınıflandırılması

Bu deneyde, Manifest, Manifest-Arsc, Dex ve Manifest-Dex-Arsc görüntü veri kümeleri olmak üzere üç tane veri kümesi kullanarak Residual Neural Network (ResNet) ve Inception V3 olmak üzere iki tane iyi bilinen derin öğrenme modeli test edilmiştir. Test edilen ResNet ve Inception V3 modellerinde 30 adet Epochs, Adam optimizer, SoftMax aktivasyon fonksiyonu ve 0.3 'lük bir Dropout oranı kullanılmıştır. Bu deney, önerilen VisDroid modeldeki sonuçları ile derin öğrenme modellerdeki sonuçlar arasında bir karşılaştırma yapılması hedeflemektedir. Kullanılan iki derin öğrenme modeli kullanılarak elde edilen sınıflandırma doğruluğu, çizelge 4.15 'da gösterilmektedir.

4.2.1.3. Kötücül Çok Sınıflı Sınıflandırma Modellerdeki Hesaplama Maliyeti

Özellik ayıklama aşamasından sınıflandırıcı eğitim aşamasına kadar önerilen modellerin toplam yürütme süresi hesaplanmıştır. Maliyet analiz çalışması, önerilen tüm bağımsız sınıflandırıcıları, önerilen topluluk sınıflandırıcıları ve test edilen derin öğrenme modelleri Manifest tabanlı görüntü veri kümesi kullanılarak gerçekleştirilmiştir. Çizelge 4.16 'daki hesaplama maliyet çalışmasının sonuçları, önerilen melez sınıflandırıcıdeki hesaplama maliyetinin, 401.97 saniyeye (en kötü durum) ulaştığını göstermektedir; bu da her numune için ortalama 0.08 saniyeyi anlamına gelmektedir. Ayrıca, bağımsız sınıflandırıcıların maliyeti, melez sınıflandırıcı tarafından ihtiyaç duyulan maliyeti ile karşılaştırıldığında, bağımsız sınıflandırıcılardaki maliyetinin daha düşük olduğunu not edilmiştir. Örneğin, rastgele orman sınıflandırıcısı, 14.04 ile 199.36 saniye arasında değişen bir süre ihtiyaç duymuştur. Ayrıca, tek türünden bir görüntü özelliği (lokal özelliği veya global özelliği) kullanılarak eğitilen oylama (yani yerel özellik tabanlı oylama sınıflandırıcısı veya global özelliklere dayalı oylama sınıflandırıcısı) sınıflandırıcılardaki hesaplama süresinin, 60.51 ile 289.55 saniye arasında değiştiğini gözlemlenmiştir. Öte yandan, test edilen ResNet model, ortalama olarak her Epoch için 1388.65 saniye bir hesaplama maliyeti ihtiyaç duymuştur; bu da ResNet modeldeki toplam ortalama maliyetinin, yaklaşık 1388.65 × 30 saniyeyi olduğunu anlamına gelmektedir. Ayrıca, test edilen Inception V3 model, ortalama olarak her Epoch için 813.25 bir hesaplama maliyeti ihtiyaç duymuştur; bu da Inception V3 modeldeki toplam ortalama maliyetinin, yaklaşık 813.25 × 30 saniyeyi olduğunu anlamına gelmektedir.

Çizelge 4.15. Test edilen derin öğrenme modelleri kullanılarak elde edilen sınıflandırma doğrulukları.

Veri Kümesi	RestNet	Inception V3
Manifest	96.34	95.12
Manifest-Arsc	93.75	92.68
DEX	94.92	94.11
Mnifest-DEX-Arsc	95.93	93.70

Çizelge 4.16. Kullanılan çok sınıflı sınıflandırma modelleri için saniye cinsinden hesaplama süresi.

Özellik	RF	KNN	DT	Bagging	AdaBoost	G-boost	Oylama Sınıflandırıcı	Hibrit Oylama
ÖBÖD	140.78	138.70	138.88	140.69	171.65	168.96	268.93	395.15
HSÖ	199.36	108.16	196.76	198.17	225.93	222.34	289.55	401.97
KAZE	164.22	163.22	162.68	164.51	178.65	174.43	210.08	380.58
YDB	14.04	12.78	12.26	14.03	26.65	24.87	60.51	342.09
Global	113.82	110.30	110.23	110.79	148.00	132.03	205.16	379.95

4.2.1.4. Kullanılan Çok Sınıflı Sınıflandırma Modellerinin Sonuçlarının Tartışılması

Android kötücül yazılım örnekleri, birden çok kötücül yazılım ailesine sınıflandırmak için VisDroid adlı yeni bir model önerilmiştir. Oluşturulan kötü amaçlı yazılım görüntü veri kümelerinden iki tür görüntü tabanlı özellikleri ayıklanıp birden çok senaryoda altı makine öğrenme sınıflandırıcısı eğitmek için kullanılmıştır. Hemen hemen tüm yapılan deneylerde global özelliklerin, lokal özelliklerinden daha iyi bir sınıflandırma doğruluğu verdiğini gözlemlenmiştir. Örneğin, rastgele orman, AdaBoost ve önerilen oylama sınıflandırıcıları eğitmek için global özellikleri kullanıldığında, sınıflandırma doğruluğunun, yaklaşık %98 veya daha fazlasına ulaştığını gözlemlenmiştir. Genel olarak, en iyi sınıflandırma doğrulukları, rastgele orman sınıflandırıcısı kullanılarak elde edilmiştir. Ayrıca, ayıklanan lokal ve global özellikleri, hibrit edilmiş bir şekilde, bir oylama sınıflandırıcısı eğitmek için kullanılmıştır. Önerilen melez topluluk oylama sınıflandırıcısının, birden çok deneyde sonuçları iyileştirdiğini not edilmiştir. Küçük boyutlu görüntü veri kümeleri (yani Manifest veya Manifest-Arsc veri kümeleri) kullanıldığında, ÖBÖD ve HSÖ özelliklerinin, YDB ve KAZE özellikleri kullanılarak elde edilen algılama doğruluklarından daha iyi bir algılama doğruluğu

verdiğini kaydedilmiştir. Öte yandan, YDB ve KAZE özellikleri kullanılarak elde edilen performans, görüntülere daha fazla ayrıntı eklenerek artmıştır. Örneğin, Manifest-ARSC-DEX veri kümesi kullanıldığında, KAZE ve YDB lokal özelliği kullanılarak elde edilen algılama doğruluklarının, sırayla %96 ve %91 'e aştıklarını gözlemlenmiştir. Elde edilen sonuçların, rastgele orman, AdaBoost ve oylama sınıflandırıcıları kullanılarak en iyi sınıflandırma doğruluğu verdiğini görülmüştür.

Gerçekleştirilen hesaplama maliyet analiz çalışmasının sonuçlarına göre, YDB özelliklerinin, özelliklerin ayıklanması, sınıflandırıcıların eğitilmesi ve kötücül yazılım örneklerin algılanması için ortalama olarak 63.40 saniyeye ihtiyaç duyarak en küçük ortalama toplam süre aldığını kaydedilmiştir; bu da her kötü amaçlı yazılım örneği için 0.01 saniyeye ihtiyaç duyduğunu anlamına gelmektedir. Ancak diğer yandan, YDB özelliklerinin, yapılan tüm deneylerde en kötü sınıflandırma doğruluğu verdiğini gözlemlenmiştir. Buna karşılık, diğer özellikler üzerinden gerçekleştirilen deneylerin hesaplama süresinin, 108.16 ile 225.93 saniye arasında değişmiş olup birbirine yakın olduğunu tespit edilmiştir, bu da, her örnek için 0.02-0.05 saniyeye ihtiyaç duyduğunu anlamına gelmektedir. Bu çalışmadaki en iyi sınıflandırma doğruluğu, Manifest görüntü veri setinden ayıklanan global özellikleri, rastgele ormanı ve Adaboost sınıflandırıcıları eğitmek için kullanıldığında elde edilmiş olup %98.14'e ulaşmıştır. Dolayısıyla, özellikleri ayıklamak, sınıflandırıcıları eğitmek ve en iyi doğruluğu elde etmek için gereken yürütme sürelerinin rastgele ormanı ve AdaBoost sınıflandırıcıları için sırasıyla 140,78 ve 171,65 saniye olduklarını kaydedilmiştir; bu da her numune için sırasıyla 0.03 ve 0.035 saniye ihtiyaç duyulduğunu anlamına gelmektedir. Lokal ve global özelliklere dayalı oylama sınıflandırıcılardaki hesaplama süresinin, bağımsız sınıflandırıcıların hesaplama süresinden biraz daha fazla olduğunu fark edilmiştir. Önerilen oylama sınıflandırıcılarının hesaplama süresi 60.51 ile 289.08 saniye arasında değiştiğini gözlemlenmiştir; bu da, her numune için 0.01 ile 0.06 saniye arasında ihtiyaç duyulduğunu anlamına gelmektedir. Önerilen melez sınıflandırıcının hesaplama süresinin, bağımsız sınıflandırıcılardaki maliyetinin yaklaşık iki katı olmasına rağmen, onun en büyük hesaplama süresi 0.08 saniyeyi olup hala makbul bir düzeyde olduğunu görülmüştür. Öte yandan, test edilen iki derin öğrenme modellerdeki hesaplama süresinin, sırasıyla 1388.65 × 30 ve 813.25

× 30 saniyeye ulaşmış olup önerilen modelin hesaplama süresinden çok daha büyük olduğunu görülmüştür. Önerilen VisDroid modelin, sınıflandırma doğruluğu ve hesaplama maliyeti açısından test edilen derin öğrenme modellerinden daha iyi performans gösterdiğini gözlemlenmiştir.

4.2.2. Kullanılan İki Sınıflı Sınıflandırma Modelleri

Çalışmanın bu bölümde, önceki çalışmada önerilen ve Android yazılımların çok sınıflı sınıflandırılması amacıyla kullanılan VisDroid modeli, Android uygulamaları, iyi huylu veya kötü amaçlı yazılım olarak sınıflandırmak için kullanılmıştır. Ayrıca, DeepVisDroid adlandırılan bir görüntü özellikler tabanlı derin öğrenme modeli önerip Android uygulamaları iyi huylu veya kötücül olarak sınıflandırmak için kullanılmıştır. Ayrıca, önerilen DeepVisDroid modeldeki sonuçları, geleneksel iki boyutlu evrişimli katmanlar tabanlı evrişimli sinir ağı (ESA) modelleri kullanılarak elde edilebilecek sonuçları ile karşılaştırmak için iki evrişimli sinir ağı (ESA) modeli önerilmiştir. İlave olarak, önerilen modellerin etkinlikleri değerlendirmek için iki iyi bilinen derin öğrenme model, oluşturulmuş görüntü veri kümeleri kullanılarak test edip önerilen modelleri ile karşılaştırılmıştır.

4.2.2.1. Önerilen VisDroid İki Sınıflı Sınıflandırma Model

I. Lokal Özellik Tabanlı VisDroid Modeli Kullanılarak Elde Edilen İki Sınıflı Sınıflandırma Sonuçları

ÖBÖD, HSÖ, YDB ve KAZE lokal özellikleri, oluşturulan görüntü veri kümelerinden ayıklanıp Rastgele orman, K-en yakın komşu, Karar ağacı, Torbalama (Bagging), AdaBoost (Adaptive Boosting) ve Gradyan Artırma (Gradient Boosting) olmak üzere altı farklı geleneksel makine öğrenme sınıflandırıcıları eğitmek ve test etmek için kullanılmıştır. Aşağıdaki paragraflarda, oluşturulan gri tonlamalı görüntü veri kümelerinin her birinden ayıklanan lokal özellikleri kullanılarak elde edilen iki sınıflı sınıflandırma sonuçları anlatılacaktır. Oluşturulan görüntü veri kümelerinin her birine dört farklı deney uygulanmıştır. Her deneyde, kullanılan lokal özelliklerden biri ayıklanmış, GSÇ algoritması

kullanılarak bir özellik vektörü oluşturulmuş ve makine öğrenme sınıflandırıcıları eğitmek için kullanılmıştır. Çizelge 4.17, görüntü lokal özellikleri kullanılarak elde edilen android uygulama iki sınıflı sınıflandırma sonuçları göstermektedir.

1) Manifest Görüntü Veri Küme Tabanlı Lokal Özellikleri Kullanılarak, Elde Edilen İki Sınıflı Sınıflandırma Sonuçları

Bu görüntü veri seti üzerinde dört farklı deney gerçekleştirilmiştir, böylece her deneyde farklı lokal özellik ayıklanıp modeli eğitmek için kullanılmıştır. İlk deneyde, ÖBÖD lokal özelliğinin tanımlayıcıları, Manifest dosya tabanlı veri kümesindeki gri tonlamalı görüntülerinden ayıklanmıştır. Ayıklanan ÖBÖD tanımlayıcılarına dayanarak oluşturulan GSÇ vektörleri makine öğrenme sınıflandırıcıları eğitmek için kullanıldığında, rastgele orman, K-en yakın komşu, Karar ağacı, Torbalama, AdaBoost ve Gradient Boost sınıflandırıcılardaki sınıflandırma doğrulukları, sırasıyla %93.59, 90.69, 90.69, 89.89, 93.89 ve 89.19 'a ulaşmıştır.

İkinci deneyde, önerilen VisDroid modeli eğitmek üzere kullanılan GSÇ vektörleri oluşturmak için HSÖ lokal özellikteki tanımlayıcıları ayıklanmıştır. Elde edilen sonuçlar, yukarıda belirtilen sınıflandırıcılardaki doğruluklarının, sırasıyla %91.87, 87.45, 84.94, 88.25, 92.57 ve 84.34 'e ulaştığını göstermiştir.

Üçüncü deneyde, YDB lokal özellikleri Manifest dosya tabanlı veri kümesinden ayıklanıp önerilen VisDroid modeli eğitmek için kullanılmıştır. Kullanılan sınıflandırıcıların sınıflandırma doğrulukları sırasıyla %72.08, 65.16, 69.58, 67.97, 72.49 ve 68.17 'ye ulaşmıştır.

Son deneyde, önerilen VisDroid modeli eğitmek için KAZE özellikleri kullanılmıştır. Elde edilen sonuçlar, rastgele orman, K-en yakın komşu, Karar ağacı, Torbalama, AdaBoost ve Gradient Boost sınıflandırma doğruluğunun, sırasıyla %91.87, 86.55, 86.75, 86.14, 92.37 ve 83.83 ’e ulaştığını göstermiştir.

2) Manifest ve Resources.Arsc Görüntü Veri Küme Tabanlı Lokal Özellikleri Kullanılarak Elde Edilen İki Sınıflı Sınıflandırma Sonuçları

Android uygulamaların iki sınıflı sınıflandırmasında, Manifest ve Resources.arsc dosyalar tabanlı görüntü veri kümesinden ayıklanan lokal özelliklerin etkinliğini test etmek için dört deney gerçekleştirilmiştir. İlk deneyde, Manifest ve Resources.arsc dosyalar tabanlı görüntü veri kümesinin görüntülerinden ÖBÖD özellikleri ayıklanmıştır. Ayıklanan ÖBÖD tanımlayıcılarından bir özellik vektörü oluşturmak için GSÇ algoritması kullanılmıştır. Elde edilen sonuçların, rastgele orman, K-en yakın komşular, Karar ağacı, Torbalama, Ada-Boost ve Gradient Boost sınıflandırıcıların doğrulukları, sırasıyla %96.23, 96.61, 96.43, 96.05, 96.79 ve 96.45'e ulaştığını göstermiştir.

İkinci deneyde, kullanılan makine öğrenme sınıflandırıcıları eğitmek için HSÖ tanımlayıcı tabanlı GSÇ vektörleri kullanılmıştır. Elde edilen sonuçlar, rastgele orman, K-en yakın komşular, Decision tree, Bagging, AdaBoost ve Gradient Boost doğrulukların, sırasıyla %97.78, 96.42, 95.45, 96.23, 97.67 ve 95.19 olduğunu göstermiştir.

Üçüncü deneyde, makine öğrenme sınıflandırıcıları eğitmek üzere Manifest ve Resources.arsc dosyalar tabanlı görüntülerden YDB lokal özellikleri ayıklanıp GSÇ vektörleri oluşturulmuştur. Elde edilen sonuçlar, sınıflandırma doğrulukların, sırasıyla %89.23, 80.95, 84.27, 82.26, 86.34 ve 84.96 ’e ulaştığını göstermiştir.

Son deneyde, KAZE lokal özelliği Manifest ve Resources.arsc dosyalar tabanlı gri tonlamalı görüntülerinden ayıklanmıştır. Oluşturulan KAZE özellik tabanlı GSÇ vektörleri makine öğrenme sınıflandırıcıları eğitmek için kullanıldığında, elde edilen sınıflandırma doğrulukların, sırasıyla %91.61, 82.72, 89.45, 86.69, 90.94 ve 90.14 'e ulaştıklarını gözlemlenmiştir.

3) DEX Koduna Ait Görüntü Veri Küme Tabanlı Lokal Özellikleri Kullanarak, Elde Edilen İki Sınıflı Sınıflandırma Sonuçları

DEX kod tabanlı görüntü veri kümesinden ayıklanan lokal özelliklerin android uygulamaları sınıflandırma konusunda etkinliğini test etmek için, bu veri kümesi üzerinde dört deney yapılmıştır. İlk deneyde, DEX dosya tabanlı görüntü veri kümesindeki her görüntüden ÖBÖD özellikleri ayıklanmıştır. Veri kümesindeki her bir görüntüden ayıklanan ÖBÖD tanımlayıcılarından bir özellik vektörü oluşturmak için GSÇ algoritması kullanılmıştır. Elde edilen sonuçlar, rastgele orman, K-en yakın komşu, Karar ağacı, Torbalama, Ada-Boost ve Gradient Boost sınıflandırıcılardaki doğrulukların, sırasıyla %95.56, 94.12, 92.04, 94.33, 95.88 ve 93.61 'e ulaştığını göstermiştir.

İkinci deneyde, önerilen VisDroid modeldeki makine öğrenme sınıflandırıcıları eğitmek için HSÖ tanımlayıcı tabanlı GSÇ vektörleri kullanılmıştır. Sonuçlar, rastgele orman, K-en yakın komşu, Decision tree, Bagging, AdaBoost ve Gradient Boost doğrulukların, sırasıyla %96.90, 96.29, 95.88, 95.15, 97.22 ve 95.88 olduğunu göstermiştir.

Üçüncü deneyde, makine öğrenme sınıflandırıcıları eğitmek üzere DEX kod tabanlı görüntülerden YDB lokal özellikleri ayıklanıp GSÇ vektörleri oluşturulmuştur. Elde edilen sonuçlar, sınıflandırma doğrulukların, sırasıyla %90.21, 87.42, 86.49, 87.53, 90.72 ve 87.11 ’e ulaştığını göstermiştir.

DEX görüntü veri kümesi üzerinden gerçekleştirilen son deneyde, KAZE lokal özelliği DEX kod tabanlı gri tonlamalı görüntülerinden ayıklanmıştır. Oluşturulan KAZE özellik tabanlı GSÇ vektörleri, makine öğrenme sınıflandırıcıları eğitmek için kullanıldığında, elde edilen sınıflandırma doğrulukları, sırasıyla %97.42, 96.80, 94.64, 96.49, 97.22 ve 96.18 'e ulaşılmıştır.

4) Manifest, Resources.Arsc ve DEX Koduna Ait, Görüntü Veri Küme Tabanlı Lokal Özellikler Kullanılarak Elde Edilen İki Sınıflı Sınıflandırma Sonuçları

Bu veri seti kullanarak dört deney de gerçekleştirilmiştir. Her deneyde, makine öğrenme sınıflandırıcıları eğitmek için kullanılan GSÇ vektörleri oluşturmak için farklı bir lokal özellik tanımlayıcıları ayıklanmıştır. İlk deneyde, ÖBÖD özelliği Manifest-DEX-ARSC görüntü veri kümesinden ayıklanmıştır. Önerilen modeli eğitmek üzere ayıklanan ÖBÖD özelliğin tanımlayıcılarına dayanarak ve GSÇ algoritması kullanarak bir özellik vektörü oluşturulmuştur. Elde edilen sonuçlar, rastgele orman, K-en yakın komşu, Karar ağacı, Torbalama, Ada-Boost ve Gradient Boost sınıflandırıcılardaki sınıflandırma doğrulukların, sırasıyla %96,5, 94,8, 93,8, 95,4, 97 ve 95,7 'ye ulaştığını göstermiştir.

İkinci deneyde, HSÖ özelliği Manifest-DEX-ARSC veri kümesinden ayıklanıp makine öğrenme sınıflandırıcıları eğitmek için kullanılmıştır. Sonuçlar, rastgele orman, K-en yakın komşu, Karar ağacı, Torbalama, Ada-Boost ve Gradient Boost makine öğrenme sınıflandırıcılardaki sınıflandırma doğrulukların, sırasıyla % 97,8, 96,7, 94,7, 96,8, 97,8 ve% 97,4'e ulaştığını göstermiştir.

Üçüncü deneyde, YDB özellikleri, Manifest-DEX-ARSC görüntü veri setinden ayıklanıp her YDB özellik tanımlayıcılarından bir özellik vektörü oluşturmak için GSÇ algoritması kullanılmıştır. Sonuçlar, kullanılan sınıflandırıcılardaki sınıflandırma doğrulukların, sırasıyla %93.23, 91.71, 87.65, 90.92, 93.51 ve 88.73 olduğunu göstermiştir.

Son deneyde, KAZE özellikleri, Manifest-DEX-ARSC görüntü veri kümesindeki her görüntüden ayıklanmıştır. Ardından, ayıklanan her bir KAZE özellikteki tanımlayıcılarından bir özellik vektörü oluşturmak için GSÇ algoritması kullanılmıştır. Oluşturulan GSÇ vektörleri, önerilen VisDroid modeldeki makine öğrenme sınıflandırıcıları eğitmek için kullanılmıştır. Elde edilen sonuçlar, kullanılan makine öğrenme sınıflandırıcılardaki sınıflandırma doğrulukların, sırasıyla %98.32, 97.29, 95.41, 96.89, 98.12 ve 97.55 ‘e ulaştığını göstermiştir.

II. Global Özelliklerine Dayanan VisDroid Modeli Kullanılarak Elde Edilen İki Sınıflı Sınıflandırma Sonuçları

Daha önce bahsedilen global özellikleri ayıklanıp önerilen VisDroid modeli eğitmek için kullanıldığında elde edilen sonuçları, çizelge 4.18'de gösterilmiş olup detaylı olarak aşağıdaki paragraflarda anlatılacaktır.

1) Manifest Görüntü Veri Kümesi Tabanlı Global Özellikler Kullanarak Elde Edilen İki Sınıflı Sınıflandırma Sonuçları

İlk deneyde, global özellikleri, Manifest dosya tabanlı görüntü veri kümesinden ayıklanıp önerilen VisDroid modeli eğitmek için kullanılmıştır. Elde edilen sonuçlar, Rastgele orman, K-en yakın komşu, Karar ağacı, Torbalama, Ada-Boost ve Gradient Boost sınıflandırıcılardaki sınıflandırma doğruluğunun sırasıyla %98.19, 95.78, 95.78, 95.78, 98.59, 97.69 ve 98.09 olduğunu göstermiştir.

2) Manifest ve Resources.Arsc Görüntü Veri Küme Tabanlı Global Özellikleri Kullanarak Elde Edilen İki Sınıflı Sınıflandırma Sonuçları

İkinci deneyde, global özellikleri Manifest ve Resources.arsc dosyalar tabanlı görüntü veri kümesinden ayıklanıp önerilen VisDroid modeli eğitmek için kullanılmıştır. Elde edilen sonuçlar, Rastgele orman, K-en yakın komşu, Karar ağacı, Torbalama, Ada-Boost ve Gradient Boost sınıflandırıcılardaki sınıflandırma doğruluklarının, sırasıyla %98,61, 98,15, 96,92, 97,41, 98,97 ve 98,23 'e ulaştığını göstermiştir.

3) DEX Kod Tabanlı Görüntü Veri Küme Tabanlı Global Özellikleri Kullanarak Elde Edilen İki Sınıflı Sınıflandırma Sonuçları

Üçüncü deneyde, global özellikleri, DEX kod dosyalar tabanlı görüntü veri kümesinden ayıklanmış ve önerilen VisDroid modeli eğitmek için kullanılmıştır. Rastgele orman, K-en yakın komşu, Karar ağacı, Torbalama, Ada-Boost ve

Gradient Boost sınıflandırıcılardaki sınıflandırma doğrulukları, sırasıyla %98,45, 98,25, 97,22, 97,53, 98,55 ve% 98,25 'e ulaşmıştır.

Çizelge 4.17. Lokal özelliklerine dayanan VisDroid modeli kullanılarak elde edilen kötücül yazılım iki sınıflı sınıflandırma doğrulukları.

Özellik	Veri Kümesi	RF	K-NN	DT	Bagging	AdaBoost	Gboost	Oylama
ÖBÖD	Manifest	93.59	90.69	90.69	89.88	93.89	89.18	92,69
	Manifest-ARSC	96.23	96.61	96.43	96.05	96.79	96.45	97.08
	DEX	95.56	94.12	92.04	94.32	95.87	93.61	94.84
	Manifest-DEX-ARSC	96.5	94.81	93.83	95.46	97.01	95.79	96.68
HSÖ	Manifest	91.87	87.45	84.94	88.25	92.57	84.34	88.45
	Manifest-ARSC	97.78	96.42	95.45	96.23	97.67	95.19	97.18
	DEX	96.90	96.29	95.88	95.15	97.22	95.88	96.70
	Manifest-DEX-ARSC	97.81	96.73	94.78	96.81	97.87	97.43	97.81
KAZE	Manifest	93.59	90.69	90.69	89.89	93.89	89.19	88.75
	Manifest-ARSC	91.61	82.72	89.45	86.69	90.94	90.14	89.97
	DEX	97.42	96.80	94.64	96.49	97.22	96.19	97.22
	Manifest-DEX-ARSC	98.31	97.37	95.49	96.93	98.16	97.50	97.68
YDB	Manifest	72.09	65.16	69.58	67.97	72.49	68.17	71.59
	Manifest-ARSC	89.23	80.95	84.27	82.26	86.34	84.96	84.56
	DEX	90.21	87.42	86.49	87.52	90.72	87.11	89.07
	Manifest-DEX-ARSC	93.21	91.72	87.67	90.90	93.56	88.77	91.12

4) Manifest, Resources.Arsc ve DEX Kod Tabanlı Görüntü Veri Küme Tabanlı Global Özellikleri Kullanarak Elde Edilen İki Sınıflı Sınıflandırma Sonuçları

Dördüncü deneyde, global özellikleri Manifest, DEX ve ARSC dosyaları kullanılarak oluşturulan görüntü veri setinden ayıklanmıştır. Ayıklanan global özellikleri makine öğrenme sınıflandırıcıları eğitmek için kullanıldığında, Random forest, K-en yakın komşu, Karar ağacı, Torbalama, AdaBoost ve Gradient Boost

sınıflandırma doğrulukları, sırasıyla %98,75, 98,3, 98,1, 98,2, 98,7 ve 98'e ulaşmıştır.

III. Kullanılan Oylama Sınıflandırıcıları Kullanarak Elde Edilen Android Uygulamalar İki Sınıflı Sınıflandırma Sonuçları

Çizelge 4.19, android uygulamaları iyi huylu veya kötücül yazılım olarak sınıflandırmak için önerilen lokal özelliklere dayalı oylama sınıflandırıcısı, global özelliklere dayalı oylama sınıflandırıcısı ve hibrit özelliklere dayalı oylama sınıflandırıcısı kullanıldığında, elde edilen sonuçları göstermektedir. Önerilen melez sınıflandırıcısının, Manifest-ARSC-DEX dosya tabanlı görüntü veri setinden ayıklanan global ve lokal özellikleri kullanılarak eğitildiğinde, %93,40'a ulaşan ikinci en iyi sınıflandırma doğruluğunu elde etiğini gözlemlenmiştir.

Çizelge 4.18. VisDroid modelin, global özellikleri kullanarak eğitildiğinde, elde edilen iki sınıflı sınıflandırma doğrulukları.

Veri Kümesi	**RF**	**KNN**	**DT**	**Bagging**	**AdaBoost**	**GBoost**	**Oylama**
Manifest	98.29	94.78	96.18	94.98	98.49	97.09	98.09
Manifest-Resources.arsc	98.61	98.15	96.92	97.41	98.97	98.23	98.23
DEX	98.14	98.25	97.22	97.63	98.35	97.83	98.45
Manifest-ARSC-DEX	98.49	98.31	97.89	98.1	98.75	98.19	98.79

Çizelge 4.19. Önerilen oylama sınıflandırıcıları kullanarak, elde edilen Android yazılım iki sınıflı sınıflandırma doğrulukları.

Veri kümesi	**Lokal**	**Global**	**Melez**
Manifest	92,69	98.09	96.99
Manifest-Resources.arsc	97.18	98.23	98.51
DEX	97.22	98.45	98.14
Manifest-ARSC-DEX	97.81	98.79	98.61

IV. Test Edilen Diğer, Android Kötü Amaçlı Yazılım Veri Kümeleri Kullanarak Elde Edilen Sonuçlari

Önerilen metodolojinin herhangi bir android kötücül yazılım veri kümesi tespit edebilirliğini kanıtlamak için, önerilen VisDroid modeldeki algılama etkinliği, AMD (Android Malware Dataset) veri kümesi kullanılarak test edilmiştir. AMD android veri kümesi, 71 aileye ait 24000 'den fazla kötücül yazılım örneği içerir ve en büyük android kötücül yazılım veri kümelerinden biri olarak bilinmektedir. 4850 kötücül yazılım örneği, AMD veri kümesinden rastgele olarak seçip Manifest görüntü veri seti, DEX görüntü veri seti ve Manifest-DEX-ARSC görüntü veri seti olmak üzere üç kötücül yazılım görüntü veri seti oluşturulmuştur. Bundan sonra, önerilen model, oluşturulan AMD kötücül yazılım veri seti tabanlı görüntü veri kümelerine dayanarak test edilmiştir. Görüntü tabanlı global özelliklerin, tüm gerçekleştirilen deneylerde lokal özelliklerinden daha iyi sonuçları verdiği için, önerilen model, yalnızca bu veri setinden ayıklanan global özellikleri kullanılarak test edilmiştir. Ayrıca, çok sınıflı sınıflandırma, ikili sınıflandırma probleminden daha zor olduğundan dolayı, önerilen VisDroid modeli, bu veri küme kullanılarak yalnızca çok sınıflı sınıflandırma yapmak için test edilmiştir. Çizelge 4.20'da gösterildiği gibi, AMD Android kötücül yazılım veri küme tabanlı Manifest-ARSC-DEX görüntü veri setinden ayıklanan global özellikleri kullanılarak önerilen modeli eğitildiğinde, sınıflandırma doğruluğunun, %98'den fazlaya ulaştığını gözlemlenmiştir.

Çizelge 4.20. AMD kötücül yazılım veri kümesi kullanarak önerilen VisDroid modeli eğitildiğinde, elde edilen sonuçlar

Veri kümesi	**RF**	**KNN**	**DT**	**Bagging**	**AdaBoost**	**GBoost**
Manifest dataset	97.61	93.99	95.19	94.29	97.40	96.61
DEX dataset	97.11	96.71	97.02	95.99	97.12	96.40
Manifest-DEX-ARSC	**98.36**	96.81	97.1	96.71	**98.36**	96.81

Kalın değer, bu çalışmada elde edilen en iyi sınıflandırma doğruluğu göstermektedir.

4.2.2.2. Kullanılan Derin Öğrenme Tabanlı İki Sınıflı Sınıflandırma Modellerinin Sonuçları

Çalışmanın bu bölümde, derin öğrenme teknikleri, görüntü tabanlı özellikleriyle melezlemeye dayalı Android kötücül yazılım örneklerini tespit etmek için DeepVisDroid adlı yeni melez bir derin öğrenme modeli önerilmiştir. Ayrıca, önerilen melez modeldeki sonuçları, geleneksel ESA modellerinin sonuçlarıyla karşılaştırmak için iki tane klasik 2D-evrişimli katman tabanlı sinir ağ modeli önerip oluşturulmuş gri tonlamalı görüntü veri kümeleri kullanılarak eğitilmiştir. İlave olarak, önerilen üç tane derin öğrenme modelleri kullanılarak elde edilen sonuçları, daha da fazla karşılaştırmak için Artık Sinir ağı (ResNet) ve Inception V3 olmak üzere iki tane iyi bilinen derin öğrenme modeli, oluşturulan görüntü veri kümeleri kullanılarak test edilmiştir. Önerilen tüm derin öğrenme modelleri, en iyi sonuçları elde etmek için kullanılan hiperparametrelerin kademeli olarak değiştirilmesiyle ampirik olarak oluşturulmuştur. Başka bir deyişle, bir evrişimli katman ve bir havuz katmanından oluşan bir model ile başlayıp, modeldeki katman sayısı ve hiperparametreleri değiştirerek performansın nasıl etkilenebileceğini test edilmiştir. Ayrıca, önerilen modellerin performansı optimize etmek için filtre sayısı, havuz boyutu (Pool size), bırakma değeri (Dropout value) ve çekirdek boyutu için birçok değerleri test edilmiştir. Bu çalışmada ele alınmaya çalışılan problem ikili bir sınıflandırma problemi olduğundan, Sigmoid fonksiyon, çıktı katmanında bir aktivasyon fonksiyonu olarak kullanılmış, ReLu evrişimli katmanlarda aktivasyon fonksiyonu olarak kullanılmış ve binary_crossentropy kayıp fonksiyonu olarak kullanılmıştır [270]. Yerdeki sınırlamadan dolayı, sadece DEX kod tabanlı görüntü veri kümesi üzerinden gerçekleştirilen DeepVisDroid modelin oluşturma sürecinin (Ablation study) bir kısmını gösterilmiştir. Çizelge 4.21'de, DEX kod tabanlı görüntü veri kümesi üzerinden gerçekleştirilen Ablasyon çalışması gösterilmektedir.

Çizelge 4.22, önerilen DeepVisDroid modelinde ve önerilen iki klasik ESA modellerinde kullanılan hiperparametre sayısı göstermektedir. Ayrıca, önerilen derin öğrenme modellerinde kullanılan son fonksiyonlar ve parametreler Çizelge 4.23'te gösterilmektedir.

I. Önerilen Derin Öğrenme Modelleri Kullanılarak Elde Edilen İki Sınıflı Sınıflandırma Doğruluğu

Önerilen derin öğrenme modellerinin ve test edilen iki iyi bilinen derin öğrenme modelinin sınıflandırma doğrulukları, çizelge 4.24'te gösterilmektedir. Bu çalışmadaki en iyi sınıflandırma doğruluğunun, DEX görüntü veri setinden ayıklanan HSÖ görüntü lokal özellikleri kullanılarak eğitilen DeepVisDroid modeli kullanılarak elde edildiğini gözlemlenmiştir. Detaylı olarak, önerilen modelleri eğitmek için DEX dosyalarına dayalı görüntü veri seti kullanıldığında, bu çalışmada elde edilen en iyi sınıflandırma doğruluğu (% 98.96'ya ulaşmıştır) elde edilmiştir. Bu sınıflandırma doğruluğu, önerilen DeepVisDroid modelin, DEX dosya tabanlı görüntü veri setinden ayıklanan HSÖ özelliklerine dayanarak eğitilmesiyle elde edilmiştir. Ayrıca, DEX dosya tabanlı görüntü veri seti kullanılarak ResNet modeli eğitildiğinde, %98.04'e ulaşan iyi bir sınıflandırma doğruluğu elde edilmiştir. Manifest dosya tabanlı görüntü veri kümesi, önerilen modelleri eğitmek için kullanıldığında, sınıflandırma doğruluğu %97.38'e ulaşmıştır, bu da HSÖ yerel özellikleri kullanarak eğitilen DeepVisDroid modeli vasıtasıyla elde edilmiştir. Ayrıca, Manifest tabanlı görüntü veri kümesinden ayıklanan KAZE özellikleri kullanılarak DeepVisDroid modeli eğitildiğinde, %97,28'lik bir sınıflandırma doğruluğu elde edilmiştir. Ayrıca, Manifest tabanlı görüntü veri seti, önerilen klasik ESA modeli eğitmek için kullanıldığında, bu modeli kullanılarak elde edilen sınıflandırma doğruluğu %97.28'e ulaşmıştır. Ayrıca, Manifest dosyası ve Rasources.arsc dosyası kullanılarak oluşturulan görüntü veri setinden ayıklanan özellikleri, önerilen DeepVisDroid modeli eğitmek için kullanıldığında, ÖBÖD, HSÖ ve KAZE lokal özellikleri kullanılarak elde edilen sınıflandırma doğruluğu, 98.62, 98.71 ve 98.61'e sırasıyla ulaşmıştır. Ayrıca önerilen klasik ESA modeli ve test edilen ResNet modeli kullanılarak elde edilen sınıflandırma doğruluğu sırasıyla %98.50 ve %98'e ulaşmıştır. Manifest, Rasources.arsc ve DEX kod dosyaları kullanılarak oluşturulan görüntü veri setinden ayıklanan özellikleri, önerilen DeepVisDroid modeli eğitmek için kullanıldığında, onun sınıflandırma doğruluğu %98,05 ve %98,03'e ulaşmıştır; bu da sırasıyla, ayıklanan global özellikleri ve HSÖ özellikleri kullanılarak elde edilmiştir. Ayrıca,

ResNet modeli eğitmek için bu veri seti (Manifest-Arsc-DEX görüntü veri seti) kullanıldığında, %98,23'e ulaşan bir sınıflandırma doğruluğu elde edilmiştir.

Tüm deneylerde model eğitiminin ilk 15 dönemler (Epoch) boyunca eğitim kaybı, doğrulama kaybı, eğitim doğruluğu ve doğrulama doğruluğundaki değişiklikleri şekil 4.12 ve şekil 4.13'da çizilmiştir.

4.2.2.3. Kullanılan Kötü Amaçlı İki Sınıflı Sınıflandırma Modellerdeki Hesaplama Maliyeti

I. Kullanılan VisDroid Modelinin Hesaplama Maliyeti

Özellik ayıklama, modelin eğitilmesi ve test edilmesi dahil olmak üzere önerilen modeldeki her aşamanın hesaplama maliyetini analiz edilmiştir. Çizelge 4.25'te gösterildiği gibi, VisDroid 'in hesaplama maliyet analiz çalışması, Manifest dosya tabanlı görüntü veri seti deneyleri kullanılarak gerçekleştirilmiştir. Elde edilen sonuçlar, ÖBÖD özellikler tabanlı hesaplama süresinin, 866.43 saniyeye ulaşmış olup en yüksek hesaplama maliyeti olduğunu gözlemlenmiştir. Buna karşılık, YDB ve KAZE özelliklerdeki hesaplama zamanlarının, sırasıyla 74.31 ve 84.32 saniyeye ulaşmış olup en düşük ortalama toplam hesaplama maliyeti olduğunu görülmüştür.

II. Kullanılan Derin Öğrenme Modellerinin Hesaplama Maliyeti

DeepVisDroid model dahil olmak üzere önerilen tüm derin öğrenme modellerin hesaplama maliyet analiz çalışması, çizelge 4.26'da gösterildiği gibi gerçekleştirilmiştir. Genel olarak, önerilen DeepVisDroid modeldeki hesaplama maliyetinin, hem önerilen iki klasik ESA modellerdeki hesaplama maliyetinden hem de kullanılan iyi bilinen derin öğrenme modellerdeki hesaplama maliyetinden çok daha düşük olduğunu fark edilmiştir. Görüntü global özellik tabanlı DeepVisDroid modelinin, neredeyse tüm gerçekleştirilen deneylerde en düşük hesaplama süresine ihtiyaç duymuş olup, onun ortalama hesaplama süresi, 775.68 ile 1593.65 saniye arasında değiştiği gözlemlenmiştir.

Çizelge 4.21. Kullanılan DeepVisDroid model için gerçekleştirilen ablasyon çalışması.

Kullanılan Model	**Dropout değeri**	**Sınıflandırma doğruluğu**				
		ÖBÖD	**HSÖ**	**KAZE**	**YDB**	**Global**
Conv1D 512 (kernel (3,3)->ReLu) MaxPooling1D(pool_size=(2))	50	95.07	96.64	96.96	93.08	90.68
	30	93.13	95.91	96.65	92.67	92.35
Conv1D 512(kernel (3,3)->ReLu) MaxPooling1D(pool_size=(2)) Conv1D256 (kernel (3,3)->ReLu) MaxPooling1D(pool_size=(2))	50	94.03	96.54	96.54	93.71	93.61
	30	94.97	96.75	96.54	92.14	91.51
Conv1D 512(kernel (3,3)->ReLu) MaxPooling1D(pool_size=(2)) Conv1D 256(kernel (3,3)->ReLu) MaxPooling1D(pool_size=(2)) Conv1D 128(kernel (3,3)->ReLu) MaxPooling1D(pool_size=(2))	50	94.24	97.17	96.96	92.56	93.61
	30	94.63	97.21	94.76	91.2	88.72
Conv1D 512(kernel (3,3)->ReLu) MaxPooling1D(pool_size=(2)) Conv1D 256(kernel (3,3)->ReLu) MaxPooling1D(pool_size=(2)) Conv1D 128(kernel (3,3)->ReLu) MaxPooling1D(pool_size=(2)) Conv1D 64(kernel (3,3)->ReLu) MaxPooling1D(pool_size=(2))	50	**97.62**	**98.96**	97.25	**93.53**	**96.32**
	30	95.07	96.54	97.58	92.35	93.68
Conv1D 512(kernel (3,3)->ReLu) MaxPooling1D(pool_size=(2)) Conv1D 256(kernel (3,3)->ReLu) MaxPooling1D(pool_size=(2)) Conv1D 128(kernel (3,3)->ReLu) MaxPooling1D(pool_size=(2)) Conv1D 64(kernel (1,1)->ReLu) MaxPooling1D(pool_size=(1))	50	94.66	96.13	96.23	91.93	93.51
	30	95.91	96.54	95.28	90.68	93.72
Conv1D 512(kernel (3,3)->ReLu) MaxPooling1D(pool_size=(2)) Conv1D 256(kernel (3,3)->ReLu) MaxPooling1D(pool_size=(2)) Conv1D 128(kernel (3,3)->ReLu) MaxPooling1D(pool_size=(2)) Conv1D 64(kernel (1,1)->ReLu) MaxPooling1D(pool_size=(1)) Conv1D 32(kernel (1,1)->ReLu) MaxPooling1D(pool_size=(1))	50	94.86	94.76	95.92	89.42	91.72
	30	94.97	94.93	94.45	88.69	92.15

Kalın değerler, ablasyon çalışmasında elde edilen en iyi sınıflandırma doğruluğunu göstermektedir.

Ayrıca, YDB ve KAZE görüntü lokal özellikler tabanlı DeepVisDroid modellerinin, ÖBÖD ve HSÖ görüntü lokal özellikler tabanlı DeepVisDroid modellerine göre çok daha düşük hesaplama maliyetine ihtiyaç duyduğunu kaydedilmiştir. Önerilen iki klasik ESA modellerdeki hesaplama maliyetinin birbirine yakın oldukları ve genellikle önerilen DeepVisDroid modelinden çok daha fazla hesaplama maliyetine ihtiyaç duyduklarını ve hesaplama sürelerinin 28560 ile 297680 saniye arasında değiştiğini görülmüştür. Ayrıca, test edilen iki iyi bilinen derin öğrenme modellerin, 17509,5 ile 70392 saniye arasında değişen büyük bir hesaplama süresi tükettiklerini gözlemlenmiştir.

Çizelge 4.22. Önerilen modellerin her birinde kullanılan hiperparametre sayısı.

Model	**Hiperparametre sayısı**
DeepVisDroid	546089
Conv2d layer-based ESA model	2078849
VGG16 inspired model	7438017

Çizelge 4.23. Önerilen derin öğrenme modellerinde kullanılan parametreler.

Parametre	**Ayarlanmış değer**
Aktivasyon fonksiyonu	ReLu, Sigmoid
Padding	Same
Çekirdek boyutu	(3, 3)
Havuz katmanı (Pooling layer)	Max pooling
pool_size	(2, 2)
Filtre boyutu	512,256,128,64
Dropout	0.3-0.5
Kayıp fonksiyonu	binary_crossentropy
Optimize Edici	Adam
Epochs sayısı	30

Çizelge 4.24. Android uygulamaları iki sınıflı sınıflandırmak için önerilen derin öğrenme modellerinin algılama doğrulukları.

Veri Kümesi	DeepVisDroid Modeli					VGG16	ESA	ResNet	Inception V3
	Global	Lokal özellikleri							
		ÖBÖD	HSÖ	YDB	KAZE				
Manifest	94.35	96.01	97.38	93.82	97.28	96.86	97.28	96.43	95.28
Manifest-ARSC	97.31	98.62	98.71	96.8	98.61	96.40	98.50	98	91.70
DEX	96.32	97.62	**98.96**	93.53	97.25	97.50	97.63	98.04	97.94
Manifest-DEX-ARSC	98.05	97.65	98.03	92.78	97.32	97.65	97.90	98.23	97.80

Kalın değer, bu çalışmada elde edilen en iyi sınıflandırma doğruluğu göstermektedir.

Çizelge 4.25. ViDroid model tabanlı iki sınıflı sınıflandırmadaki hesaplama maliyeti (saniye cinsinden).

Özellik	RF	KNN	DT	Bagging	AdaBoost	G-boost	Oylama Sınıflandırıcı	Hibrit Oylama
ÖBÖD	868.84	864.94	864.39	865.61	867.04	867.75	893.72	1049.75
HSÖ	762.99	759.65	759.39	759.99	761.4	762.18	791.81	978,94
KAZE	49.36	48.31	47.77	48.95	262.87	48.66	328,648	361.58
YDB	41.22	39.83	39.36	40.51	244.73	40.19	299,54	342.09
Global	172.49	166.14	165.17	168.07	168.78	176.37	247,24	683,09

Çizelge 4.26. Önerilen derin öğrenme modeller tabanlı iki sınıflı sınıflandırmanın hesaplama maliyeti (saniye cinsinden).

Veri Kümesi	DeepVisDroid					VGG	ESA	ResNet	Inception V3
	Global	Lokal özellikleri							
		ÖBÖD	HSÖ	YDB	KAZE				
Manifest	775.68	1244.62	1162.96	**490.75**	535.71	29270	148556	43020	17509.5
Manifest-ARSC	852.38	14367.76	9955.58	1138.66	2203.12	54415	53065	33690	37903.35
DEX	1593.65	31015.09	17275.2	4085.12	12315.71	297680	287430	40165.5	46844.5
Manifest-DEX-ARSC	1223.76	31780.28	33852.5	4965.68	15433.22	28560	52878	69591	70392

Kalın değer, modeli eğitmek için harcanan en düşük hesaplama maliyeti göstermektedir.

4.2.2.4. Kullanılan Kötücül Yazılım İki Sınıflı Sınıflandırma Modellerdeki Sonuçlarının Tartışılması

I. Önerilen VisDroid Modeldeki Sonuçlarının Tartışılması

Çalışmanın bu bölümde, önerilen geleneksel makine öğrenme sınıflandırıcılarına dayanan olan VisDroid model, android uygulamaları iyi huylu veya kötücül olarak sınıflandırmak için kullanılmıştır. Global özellikleri kullanılarak elde edilen sınıflandırma doğruluğunun, hemen hemen tüm deneylerde lokal özellikleri kullanılarak elde edilen sınıflandırma doğruluğundan daha iyi olduğunu görülmüştür. Örneğin, AdaBoost sınıflandırıcısı eğitmek için, Manifest, DEX ve Manifest-DEX-ARSC görüntü veri kümelerinin her birinden ayıklanan global özellikleri kullanıldığında, onun sınıflandırma doğruluğu %98'in üzerine çıkmıştır. Ayrıca, Manifest-DEX-ARSC görüntü veri kümesinden ayıklanan lokal özellikleri, AdaBoost sınıflandırıcısı eğitmek için kullanıldığında, onun sınıflandırma doğruluğunun, %98'in üzerine çıktığını görülmüştür. Başka bir deyişle, bu çalışmadaki en iyi sınıflandırma doğrulukları, AdaBoost sınıflandırıcısı kullanılarak elde edilmiştir. Genel olarak, Manifest görüntü veri kümesinden ayıklanan lokal özellikleri kullanılarak elde edilen sınıflandırma doğruluğunun, DEX veya Manifest-DEX-ARSC görüntü veri kümelerinden ayıklanan lokal özellikleri kullanılarak elde edilen sınıflandırma doğruluğundan daha düşük olduğunu gözlemlenmiştir. Ayrıca, YDB lokal özellikleri kullanılarak elde edilen sınıflandırma doğruluğunun, kullanılan görüntü veri setine ve eğitilen sınıflandırıcıya göre %65.16 ile %93.56 arasında değişmiş olup tüm deneylerde en kötü sınıflandırma doğruluğu olduğunu görülmüştür.

Gerçekleştirilen zaman maliyet analiz çalışmasının sonuçlarına göre, YDB özelliklerinin, en küçük toplam hesaplama maliyetine ihtiyaç duyduğunu ortaya çıkmıştır. YDB özellikleri, önerilen modeli eğitmek için kullanıldığında, gereken ortalama toplam hesaplama maliyeti 74.31 saniyeye ulaşmıştır, bu da her kötücül yazılım örneği için ortalama 0,008 saniye ihtiyaç duyduğunu anlamına gelmektedir. Ancak aksine, YDB özelliklerinin, yürütülen tüm deneylerde en kötü sınıflandırma doğruluğu verdiğini gözlemlenmiştir. Ayrıca, ÖBÖD özelliklerinin, en yüksek

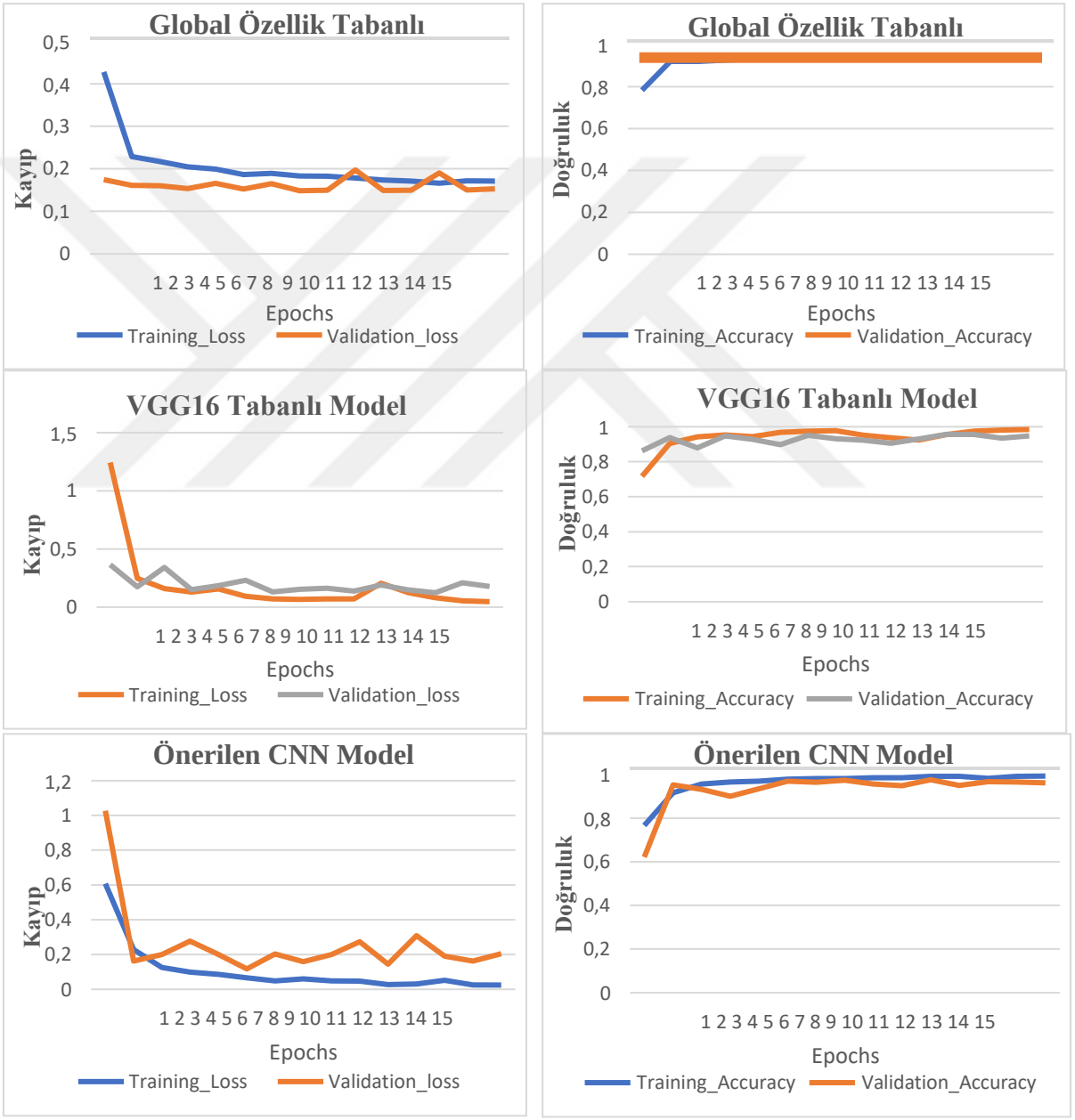
hesaplama maliyetine ihtiyaç duyduğunu sonucuna varılmıştır. ÖBÖD özellikleri tabanlı modeldeki ortalama hesaplama süresi 866.43 saniyeye ulaşmıştır, bu da her kötücül yazılım örneği için 0.091 saniyeye ihtiyaç duyduğunu anlamına gelmektedir. Ayrıca, sonuçların, KAZE özellikleri kullanılarak elde dilen sınıflandırma doğruluğu %98'e kadar ulaşmış olup lokal özellikleri kullanılarak elde edilen en iyi sınıflandırma doğruluk olduğunu göstermiştir. KAZE özelliğin ortalama hesaplama maliyetinin, YDB özelliğin ortamla hesaplama maliyetine yakın olup 84.32 saniyeye ulaştığını kaydedilmiştir. Öte yandan, global özellikleri, önerilen model eğitmek için kullanıldığında, önerilen modeldeki ortalama hesaplama maliyeti 169,50 saniyeye ulaşmıştır; bu da her kötücül yazılım numunesi için 0,018 saniyeye ihtiyaç duyulduğu anlamına gelmektedir. Başka bir deyişle, önerilen modelin, %98,75'e ulaşan bir sınıflandırma doğruluğu vermesi için, her kötücül yazılım örneği için 0,018 saniyeye ihtiyaç duyduğunu gözlemlenmiştir.

II. Önerilen Derin Öğrenme Modellerdeki Sonuçlarının Tartışılması

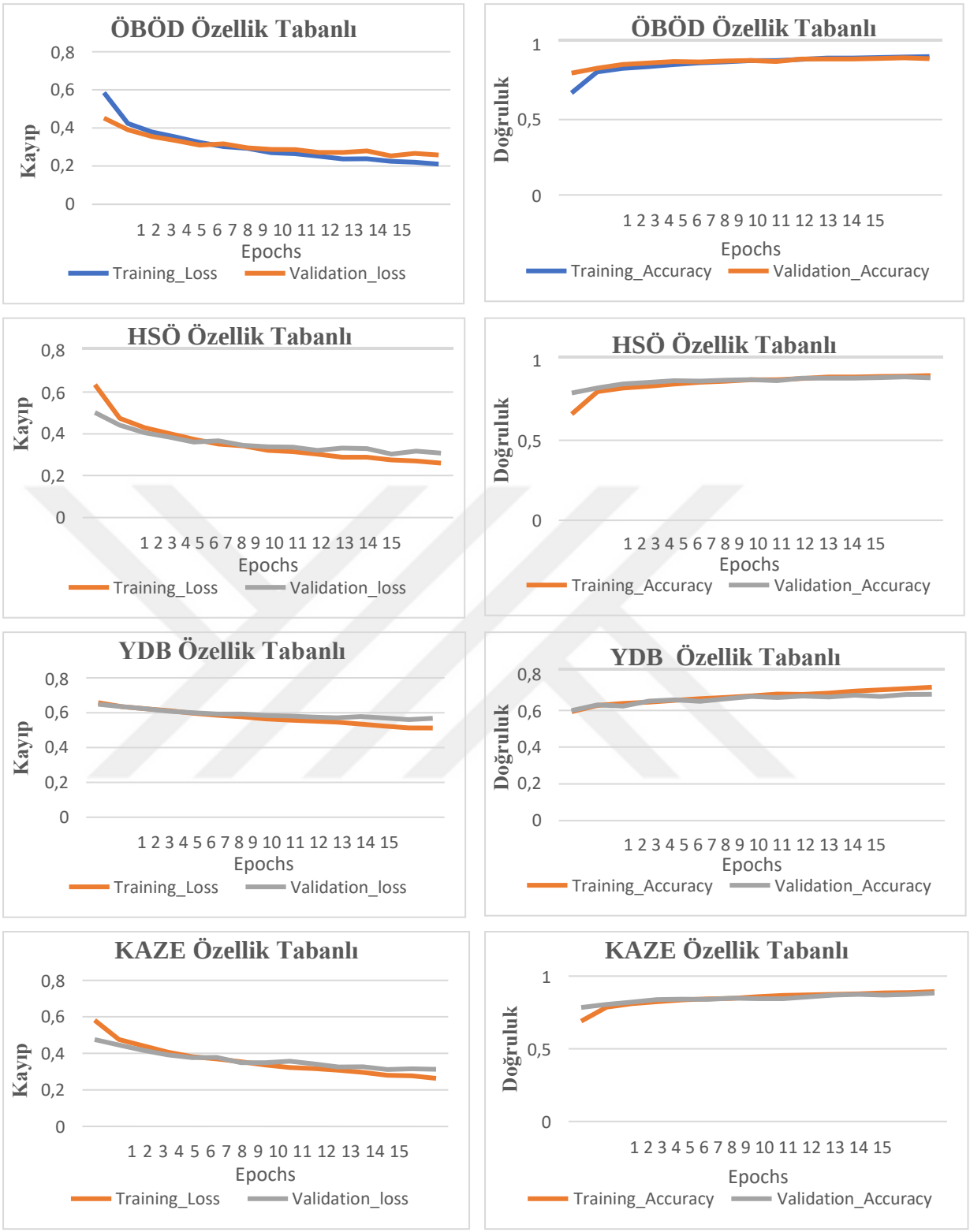
Elde edilen sonuçlar, önerilen tüm ESA modellerinin umut verici sonuçlar verdiğini ve önerilen DeepVisDroid modelin doğru bir Android kötü amaçlı yazılım detektörü olarak kullanılabileceği göstermiştir. Önerilen klasik ESA modellerinin ve kullanılan iyi bilinen derin öğrenme modellerinin her biri kullanılarak elde edilen sınıflandırma doğruluğunun %98'e ulaşmasına rağmen, onların hesaplama maliyetlerinin, önerilen DeerpVisDroid modeldeki hesaplama maliyetine kıyasla çok yüksek olduğunu gözlemlenmiştir. Ayrıntılı olarak anlatılacak olursa, önerilen klasik 2D-konvolüsyon katman tabanlı ESA modelinin, 135482.3 saniye bir hesaplama zamanına ihtiyaç duyduğunu görülmüştür, bu da her kötücül yazılım örneği için 13.97 saniyeye ihtiyaç duyulduğunu anlamına gelmektedir. Karşılığı olarak, önerilen klasik 2D-konvolüsyon katman tabanlı ESA modelinin, en fazla %98.50'lık bir sınıflandırma doğruluğu ulaşabildiğini gözlemlenmiştir. Ayrıca, önerilen VGG16 dan ilham eden ESA modelinin ortalama hesaplama maliyeti, 102481.3 saniye olduğunu ortaya çıkmıştır, bu da her örnek için 10.56 saniyeye ihtiyaç duyduğunu anlamına gelmektedir. Öte yandan, önerilen VGG16 modelinden ilham eden ESA modelin en fazla yakalayabildiği sınıflandırma doğruluğu %97.65'e ulaşmıştır. Ayrıca, test edilen iki son teknoloji derin öğrenme modeller,

bazı deneylerde %98'in üzerinde iyi bir sınıflandırma doğruluğu elde etmişlerdir. Ancak, bu iki model yüksek bir hesaplama süresi ihtiyaç duydukları görülmüş olup, sırasıyla, ResNet ve Inception V3 'deki ortalama hesaplama süresi 46616.63, 43162.35 saniyeye ulaşmıştır. Buna karşılık, bu çalışmada elde edilen sınıflandırma doğruluk, %98.96'ya ulaşmış olup önerilen DeepVisDroid görüntü özelliklerine dayalı derin öğrenme modeli kullanılarak elde edilmiştir. Ayrıca, önerilen DeepVisDroid modelin sınıflandırma doğruluğun, birden fazla deneyde %98'e aştığını gözlemlenmiştir. Ayrıca, önerilen DeepVisDroid model tarafından harcanan hesaplama süresinin, önerilen klasik ESA modeller ve test edilen iyi bilinen derin öğrenme modeller tarafından harcanan hesaplama sürelerinden daha az olduğunu görülmüştür. Bu da önerilen DeepVisDroid modelinde kullanılan 1D-evrişimli katmanların, önerilen klasik ESA modellerinde ve test edilen ResNet ve Inception V3 modellerinde kullanılan 2D-evrişimli katmanlarından çok daha basit olduğundan kaynamaktadır. Bu nedenle, önerilen DeepVisDroid'in hiperparametre sayısı, önerilen klasik ESA modellerdeki hiperparametre sayısından ve test edilen ResNet ve Inception V3 modellerdeki hiperparametre sayısından daha azdır. Detaylı olarak anlatılacak olursa, önerilen DeepVisDroid model tarafından eğitim esnasında kullanılan özelliğe göre harcanan ortalama hesaplama sürelerinin, global özellik, YDB özellik, KAZE özellik, HSÖ özellik ve ÖBÖD özellik için, sırasıyla 1111.36, 2670.05, 7621.94, 15561.56 ve 19601.93 saniye olduğunu not edilmiştir, bu da her numune için ortalama 0.11, 0.27, 0.78, 1.60 ve 2.02 saniyeye ihtiyaç duyulduğu anlamına gelmektedir. Genel olarak, HSÖ, ÖBÖD ve KAZE görüntü tabanlı lokal özelliklerine dayanarak eğitilen DeepVisDroid kullanılarak elde edilen sınıflandırma doğruluğunun, görüntü tabanlı global özelliklerine dayanarak eğitilen DeepVisDroid kullanılarak elde edilen sınıflandırma doğruluğundan daha yüksek olduğu belirtilmiştir. Ancak, bu global özelliklerin etkin olmadığını anlamına gelmemektedir, mesela, DeepVisDroid modeli eğitmek için Manifest-Arsc-DEX görüntü veri kümesinden ayıklanan global özellikleri kullanıldığında, her bir APK örnek için 0.11 saniyelik bir hesaplama süresi ile % 98,05'lik bir sınıflandırma doğruluğu elde edilmiştir. Ayrıca, görüntü tabanlı lokal özellik tabanlı DeepVisDroid modeldeki hesaplama maliyetinin, görüntü tabanlı global özellikler tabanlı DeepVisDroid modelinin hesaplama maliyetinden daha yüksek olduğunu görülmüştür. Aynı zamanda, görüntü tabanlı lokal özelliklere dayalı DeepVisDroid

modelin sınıflandırma doğruluğunun, görüntü tabanlı global özelliklere dayalı DeepVisDroid modelin sınıflandırma doğruluğundan daha yüksek olduğunu kaydedilmiştir. Önerilen DeepVisDroid model tarafından harcanan en yüksek ortalama hesaplama süresinin, 19601.94 saniyeye ulaşmış olup ÖBÖD özelliği tabanlı DeepVisDroid model tarafından harcandığını gözlemlenmiştir, bu da her APK örneği için 2.02 saniyeyi anlamına gelmektedir.



Şekil 4.12. Önerilen DeepVisDroid modeli ve klasik ESA modeller için kayıp ve doğruluğun analiz edilmesi.



Şekil 4.13. Lokal özellikler tabanlı DeepVisDroid model için kayıp ve doğruluğun analiz edilmesi.

4.3. Sonuçların Karşılaştırılması

Önerilen VisDroid ve DeepVisDroid modeli kullanılarak elde edilen sonuçları, çizelge 4.27'da gösterildiği gibi, sınıflandırma doğruluğu ve hesaplama süresi açısından bazı daha önceki yapılan çalışmaların sonuçlarıyla karşılaştırılmıştır. Önerilen modelin, sınıflandırma doğruluğunun ikili sınıflandırma için %98.96'ya ve çok sınıflı sınıflandırma için 98.14'e ulaştığını ve tüm karşılaştırılan çalışmaların sınıflandırma doğruluklarına aştığını gözlemlenmiştir. Buna karşılık, önerilen modellerin hesaplama süresinin, karşılaştırılan modellerin hesaplama sürelerine yakın olduğunu ve yüksek sınıflandırma doğruluğu ile karşılaştırıldığında hala çok kabul edilebilir bir seviyede olduğunu görülmüştür.

Çizelge 4.27. Önerilen modellerin sonuçlarının önceki bazı çalışmaların sonuçlarıyla karşılaştırılması.

Model		**Maliyeti/Saniye**	**Doğruluk**
Mateless.et al [142]		-	97.5%
Xiao.et al [140]		0.193	93.67%
Pektaş. et al [271]		0.017	91.42%
Wang. et al [272]		-	94%
DL-Droid [134]		0.08	98.5%
Yen.et al [34]		-	92.67%
R2-D2 [273]		0.5	93%
AspectDroid [47]		-	94.68%
SAFEDroid [274]		-	98.4%
DroidDet [37]		-	88.26%
FalDroid [146]		4.6	94.2%
DREBIN [54]		0.75	94%
DroidSIFT [275]		0.06	93%
Yang.et.al [276]		-	95.42%
Karimi.et.al [277]		-	97%
Dendroid [203]		-	94.06%
Lee.et.al [70]		2-10	97.86%
DroidMat [278]		-	97.87%
DroidLegacy [279]		-	98%
VisDroid	İki sınıflı sınıflandırma	0.018	**98.75%**
	Çok sınıflı sınıflandırma	0.08	**98.14%**
DeepVisDroid	İki sınıflı sınıflandırma	0.72-10.16	98.05 **-98.96%**

4.4. Önerilen Sınıflandırma Yöntemin Sınırlamaları

Önerilen modelin, bu çalışmada önerilen enjeksiyon saldırıları gibi bazı kod kamuflaj ve gizleme teknikleri tespit edilememesi ihtimal vardır. Bu bölümde, bu tür şaşırtma tekniklere karşı, önerilen DeepVisDroid modelinin sağlamlığını test etmek için bir vaka çalışmayı gerçekleştirilmiştir. Bu amaçla, kullanılan Android kötü amaçlı yazılım veri kümesinden rastgele olarak 50 kötü amaçlı yazılım örneği seçilmiştir. Daha sonra her APK arşivi ters derlenmiş, kod-izin enjeksiyon saldırısı ona uygulanmış ve yeniden paketlenerek yeni bir sertifikayı kullanılarak yeniden imzalanmıştır. Bundan sonra, APK tekrardan ters derlenmiş ve yukarıda belirtilen dört gri tonlamalı görüntü veri kümesi oluşturulmuştur. Oluşturulan görüntü veri kümelerinden ayıklanan global özellikleri kullanarak önerilen DeepVisDroid modeli eğitildiğinde, çizelge 4.27'deki sonuçları elde edilmiştir. Önerilen model, Manifest dosyaları kullanılarak oluşturulan görüntü veri kümesinden ayıklanan global özellikleri kullanılarak eğitildiğinde, %0'lik bir algılama doğruluğu elde edilmiştir; bu da model tarafından gizlenmiş 50 kötü amaçlı yazılım örneklerinden hiçbirinin tespit edilemediği anlamına gelmektedir. DeepVisDroid, Manifest ve Resouces.arsc dosyaları kullanılarak oluşturulan görüntü veri setinden ayıklanan global özellikleri kullanılarak eğitildiğinde, %4'lük bir algılama doğruluğu elde edilmiştir, bu da 50 gizlenmiş kötü amaçlı yazılım örneğinden yalnızca iki örneğinin tespit edildiğini anlamına gelmektedir. Ayrıca, DeepVisDroid modeli, DEX kod tabanlı görüntü veri kümesinden ayıklanan global özellikleri kullanılarak eğitildiğinde, algılama doğruluğu %58'e ulaşmıştır, bu da 50 gizlenmiş kötü amaçlı yazılım örneğinden 29 kötü amaçlı yazılım örneğinin tespit edildiğini anlamına gelmektedir. Son olarak, DeepVisDroid, Manifest, Resources.arsc ve DEX kod dosyaları kullanılarak oluşturulan görüntü veri kümesinden ayıklanan global özelliklere dayalı olarak eğitildiğinde, onun algılama doğruluğu %64'e ulaşmıştır, bu da 50 gizlenmiş kötücül yazılım örneğinden 32 örneğinin tespit edildiği anlamına gelmektedir. Planlanan ileri dönem çalışmalarında görüntü tabanlı nesne tespit teknikleri kullanarak bu tür saldırılardan korunma tekniklerin geliştirilmesi hedeflenmektedir.

Çizelge 4.28. DeepVisDroid modelinin, önerilen enjeksiyon saldırılarına karşı test edilmesi.

Veri Kümesi	**Per**	**Per-Arsc**	**DEX**	**Per-Arsc-DEX**
Algılama doğruluğu %	0	4	58	54

5. SONUÇ VE ÖNERİLER

Akıllı telefonları hedef alan kötü amaçlı uygulamalar, en tehlikeli siber uzay tehditlerinden biri haline gelmiştir. Dolayısıyla hem ticari hem de akademik alanda, çoklu algılama sistemleri geliştirilmiştir. Bu tez çalışması günümüzün vazgeçilmez cihazları olan akıllı telefon ve tabletleri hedef alan çeşitli kötücül amaçlı yazılımların tespit yöntemlerini içeren detaylı inceleme ve uygulamalar gerçekleştirilmiştir.

Öncelikle, Android kötü amaçlı yazılım tespit sistemlerine ve bu alanda gelecekteki çalışmalarda odaklanılması gereken araştırma trendlerine ışık tutmak için 2009 ile 2020 arasında yayınlanan 200'den fazla makale kullanılarak kapsamlı bir sistematik inceleme yapılmıştır.
Gerçekleştirilen ilk çalışmada incelenen çalışmalar ışığında, bu alanda yapılan çalışmaların çoğunun altında sınıflandırılabileceği şekilde sağlam ve kapsamlı bir taksonomi önerilmiştir. Ek olarak, Android kötücül yazılım algılama sürecini gösteren "Schematic Review Model" adlı ayrıntılı bir şematik model önerilmiştir. Önerilen taksonomi ve şematik inceleme literatüre ilk kez bu çalışmada ile kazandırılmıştır.

İkinci çalışma ise, yaygın olarak kullanılan bazı ticari anti virüs sistemlerinin sağlamlığını, kamuflaj tekniklerine karşı test etmeyi amaçlamaktadır. Bu amaçla, izin ve izin-kodu olmak üzere iki adet enjeksiyon saldırısı önerilmiştir. Ayrıca, daha fazla kamuflaj seviyesi elde etmek için, önerilen enjeksiyon saldırıları, yaygın olarak kullanılan bazı gizleme teknikleriyle melezleştirilmiştir. İncelenen literatürde görüldüğü kadarıyla bu tür saldırılar da Android platform alanında ilk kez bu çalışmada kullanılmıştır. Bazı anti virüs sistemleri değerlendirmek için önerilen saldırıları kullanarak birden fazla deney gerçekleştirilmiştir. Sonuçlara göre, test edilen anti virüs sistemlerinin çoğunun algılama doğruluğunda yaklaşık %10 düşüş gözlemlenmiştir. Ayrıca, izin-kod enjeksiyon saldırısı gizleme teknikleriyle melezleştirildiğinde kötü amaçlı yazılım örneklerini tespit edebilen ortalama anti virüs sistem sayısı yaklaşık 12'ye düşmüştür. Bu da Android tabanlı cihazların gerçek

tehlike altında olduğunu ve bu tür saldırılarla başa çıkmak için yeni yöntemlerin geliştirilmesi gerektiğini yansıtmaktadır.

Üçüncü çalışmada, android uygulamaları, iyi huylu veya kötücül olarak sınıflandırmak için ve ardından kötücül olarak algılanan uygulamaları türlerine göre sınıflandırmak için görüntü işleme teknikleri ve makine öğrenme algoritmalarına dayanan yeni bir yöntem önerilmiştir., Önerilen yöntem Android uygulamalarının kaynaklarından bazı dosyaları, gri tonlamalı görüntülere dönüştürmeye ve makine öğrenme sınıflandırıcılarını eğitmek için bazı görüntü tabanlı özellikleri çıkarmaya dayanmaktadır. ÖBÖD, HSÖ, KAZE ve YDB olmak üzere dört tane görüntü tabanlı lokal; renk histogramı, Hu anları ve Haralick doku olmak üzere üç tane görüntü tabanlı global özellik çıkarılmış ve önerilen modelleri eğitmek için kullanılmıştır. Ayıklanan lokal özellik tanımlayıcılarından bir özellik vektörü oluşturmak için Görsel Sözcük Çantası (GSÇ) kullanılmıştır. İki farklı makine öğrenme tabanlı model önerilmiş ve ayıklanan görüntü tabanlı lokal ve global özellikler kullanılarak eğitilmiştir. VisDroid olarak adlandırılan birinci model, birden çok geleneksel makine öğrenme ve bir melez oylama sınıflandırıcısını içermektedir. Önerilen VisDroid, Android uygulamalarını iyi huylu veya kötü niyetli olarak sınıflandırmak ve kötü amaçlı uygulamaları kendi ailelerine sınıflandırmak için kullanılmıştır. DeepVisDroid olarak adlandırılan ikinci model, 1 boyutlu evrişimli sinir ağ modelini eğitmek için ayıklanan özellikleri kullanmaya dayanmaktadır. Yine yapılan literatür incelemeler neticesinde önerilen VisDroid ve DeepVisDroid modellerinde kullanılan metodoloji ilk kez bu çalışmada kullanılmıştır. Ayrıca, DeepVisDroid modelinin sonuçlarını geleneksel derin öğrenme modellerinin sonuçlarıyla karşılaştırmak için iki klasik evrişimli sinir ağ modeli önerilmiştir. Ayrıca, iki iyi bilinen derin öğrenme modeli olan ResNet ve Inception V3 ile daha derin bir karşılaştırma yapmak için test edilmiştir. Önerilen yöntem, çok verimli hesaplama süresi ile %98'i aşan yüksek bir sınıflandırma doğruluğu elde etmiştir. Ayrıca önerilen model, sınıflandırma doğruluğu açısından karşılaştırılan tüm son teknoloji modellerden daha iyi performans göstermiştir.

Gelecekteki çalışmalarda, önerilen modelin verimliliği, masaüstü yazılım örnekleri gibi diğer kötü amaçlı uygulama türlerini sınıflandırma konusunda test edilecektir.

Ayrıca, görüntü tabanlı nesne algılama teknikleri, bu çalışmada önerilen enjeksiyon saldırılarını atlatmak için kullanılacaktır. Ayrıca önerilen yöntem geliştirilerek web tabanlı çevrimiçi hizmet olarak sunulacaktır.

KAYNAKLAR

1. Gartner_Q2 (2017). "Gartner Says Demand for 4G Smartphones in Emerging Markets Spurred Growth in Second Quarter of 2017." https://www.gartner.com/newsroom/id/3788963 Accessed 14-July-2018.
2. Costello, K. (2018). "Gartner Says Worldwide Sales of Smartphones Returned to Growth in First Quarter of 2018." https://www.gartner.com/newsroom/id/3876865 Accessed 29-07-2018.
3. Gartner_Q4 (2017). "Gartner Says Worldwide Sales of Smartphones Recorded First Ever Decline During the Fourth Quarter of 2017." https://www.gartner.com/newsroom/id/3859963 Accessed 11-July-2018.
4. Statista_a (2018). "Number of available applications in the Google Play Store from December 2009 to June 2018." https://www.statista.com/statistics/266210/number-of-available-applications-in-the-google-play-store/ Accessed 13-July-2018.
5. Statista_b (2018). "Growth of available mobile apps at Google Play worldwide from 2nd quarter 2015 to 1st quarter 2018." https://www.statista.com/statistics/185729/google-play-quarterly-growth-of-available-apps/ Accessed 13-July-2018.
6. Statista_c (2018). "Cumulative number of apps downloaded from the Google Play as of May 2016 (in billions)." https://www.statista.com/statistics/281106/number-of-android-app-downloads-from-google-play/ Accessed 14-July-2018.
7. Pulse_Secure (2015) 2015 MOBILE THREAT REPORT. Pulse Secure Mobile Threat Center (MTC)
8. Symantec (2016) Internet Security Threat ReportInternet Report. vol 21.
9. G-Data (2017). "8,400 new Android malware samples every day." https://www.gdatasoftware.com/blog/2017/04/29712-8-400-new-android-malware-samples-every-day Accessed 14-July-2018.
10. Lueg, C. (2018). "Malware figures for Android rise rapidly." https://www.gdatasoftware.com/blog/2018/07/30937-malware-figures-for-android-rise-rapidly Accessed 13-09-2018.
11. G-Data (2018). "The total count of mobile malware rises about 40 percent in 2018." https://www.gdatasoftware.com/blog/2018/11/31255-cyber-attacks-on-android-devices-on-the-rise Accessed 01-04-2019.
12. McAfee (2017). "New Android Malware Found in 144 GooglePlay Apps." https://securingtomorrow.mcafee.com/mcafee-labs/android-malware-grabos-exposed-millions-to-pay-per-install-scam-on-google-play/ Accessed 14-July-2018.
13. McAfee (2018) McAfee Mobile Threat Report Q1, 2018.
14. McAfee (2018) AsiaHitGroup Gang Again Sneaks Billing-Fraud Apps Onto Google Play. McAfee Labs Threats Report.
15. Brenner, B. (2017). "Malware rains on Google's Android Oreo parade." https://nakedsecurity.sophos.com/2017/08/24/malware-rains-on-googles-android-oreo-parade/ Accessed 30-07-2018.

16. Micro, T. (2019). Adware Disguised as Game, TV, Remote Control Apps Infect 9 Million Google Play Users https://blog.trendmicro.com/trendlabs-security-intelligence/adware-disguised-as-game-tv-remote-control-apps-infect-9-million-google-play-users/ Accessed 04-04-2019.
17. Martin_Zhang (2016). "Android ransomware variant uses clickjacking to become device administrator." https://www.symantec.com/connect/blogs/android-ransomware-variant-uses-clickjacking-become-device-administrator Accessed 13-09-2018.
18. Research, E. (2017). "DoubleLocker: Innovative Android Ransomware." https://www.welivesecurity.com/2017/10/13/doublelocker-innovative-android-malware/ Accessed 13-09-2018.
19. List, S. (2017). "Ztorg: money for infecting your smartphone." https://securelist.com/ztorg-money-for-infecting-your-smartphone/78325/ Accessed 13-03-2018.
20. Unuchek, R. (2017). "Dvmap: the first Android malware with code injection." https://securelist.com/dvmap-the-first-android-malware-with-code-injection/78648/ Accessed 13-09-2017.
21. Semantic (2010). "AndroidOS.FakePlayer." https://www.symantec.com/security-center/writeup/2010-081100-1646-99 Accessed 13-09-2018.
22. Roman Unuchek, F.S., Denis Parinov, Alexander Liskin (2017). "IT threat evolution Q3 2017. Statistics." https://securelist.com/it-threat-evolution-q3-2017-statistics/83131/.
23. Micro, T. (2018). "Mobile Adware RottenSys Can Infect Android Devices to Become Part of a Botnet." https://www.trendmicro.com/vinfo/dk/security/news/cybercrime-and-digital-threats/mobile-adware-rottensys-can-infect-android-devices-to-become-part-of-a-botnet Accessed 13-09-2018.
24. Kaspersky (2015). "Mobile banking Trojans, explained." https://www.kaspersky.com/blog/mobile-banking-trojans-faq/13243/Accessed 13-09-2018.
25. Snow, J. (2016). "The evolution of Asacub trojan: from small fish to ultimate weapon." https://www.kaspersky.com/blog/asacub-trojan/11108/.
26. securelist (2017). "A new era in mobile banking Trojans." https://securelist.com/a-new-era-in-mobile-banking-trojans/79198/ Accessed 13-9-2018.
27. f-secure "Trojan:Android/Anserverbot.A." https://www.f-secure.com/v-descs/trojan_android_anserverbot.shtml Accessed 13-09-2018.
28. F-secure "Trojan-Spy:Android/Tramp.A." https://www.f-secure.com/v-descs/trojan_android_tramp.shtml#summary Accessed 13-09-2018.
29. Android_PlayProtect "Play Protect." https://www.android.com/play-protect/ Accessed 14-July-2018.
30. Bakour, K., H.M. Ünver, and R. Ghanem (2019) The Android malware detection systems between hope and reality. SN Applied Sciences 1 (9):1120
31. Huang, T.H.-D. and H.-Y. Kao (2017) R2-D2: color-inspired convolutional neural network (cnn)-based android malware detections. arXiv preprint arXiv:1705.04448

32. Yang, M. and Q. Wen. Detecting android malware by applying classification techniques on images patterns. in Cloud Computing and Big Data Analysis (ICCCBDA), 2017 IEEE 2nd International Conference on. 2017. IEEE
33. Karimi, A. and M.H. Moattar. Android ransomware detection using reduced opcode sequence and image similarity. in Computer and Knowledge Engineering (ICCKE), 2017 7th International Conference on. 2017. IEEE
34. Yen, Y.-S. and H.-M. Sun (2019) An Android mutation malware detection based on deep learning using visualization of importance from codes. Microelectronics Reliability 93:109-114
35. APKTool (2018). "A tool for reverse engineering Android apk files." https://ibotpeaches.github.io/Apktool/ Accessed 14-July-2018.
36. Gurulian, I., K. Markantonakis, L. Cavallaro, and K. Mayes (2016) You can't touch this: Consumer-centric android application repackaging detection. Future Generation Computer Systems 65:1-9. doi:10.1016/j.future.2016.05.021
37. Zhu, H.-J., Z.-H. You, Z.-X. Zhu, W.-L. Shi, X. Chen, and L. Cheng (2018) DroidDet: Effective and robust detection of android malware using static analysis along with rotation forest model. Neurocomputing 272:638-646. doi:10.1016/j.neucom.2017.07.030
38. Chen, S., M. Xue, L. Fan, S. Hao, L. Xu, H. Zhu, and B. Li (2018) Automated poisoning attacks and defenses in malware detection systems: An adversarial machine learning approach. Computers & Security 73:326-344. doi:10.1016/j.cose.2017.11.007
39. Wang, W., M. Zhao, and J. Wang (2018) Effective android malware detection with a hybrid model based on deep autoencoder and convolutional neural network. Journal of Ambient Intelligence and Humanized Computing. doi:10.1007/s12652-018-0803-6
40. Kirubavathi, G. and R. Anitha (2017) Structural analysis and detection of android botnets using machine learning techniques. International Journal of Information Security 17 (2):153-167. doi:10.1007/s10207-017-0363-3
41. Tao, G., Z. Zheng, Z. Guo, and M.R. Lyu (2018) MalPat: Mining Patterns of Malicious and Benign Android Apps via Permission-Related APIs. IEEE Transactions on Reliability 67 (1):355-369. doi:10.1109/tr.2017.2778147
42. Somarriba, O. and U. Zurutuza. A collaborative framework for android malware detection using DNS & dynamic analysis. in 2017 IEEE 37th Central America and Panama Convention (CONCAPAN XXXVII). 2017. doi:10.1109/CONCAPAN.2017.8278529
43. Wei, S., G. Wu, Z. Zhou, and L. Yang. Mining network traffic for application category recognition on Android platform. in Progress in Informatics and Computing (PIC), 2015 IEEE International Conference on. 2015. IEEE. doi:https://doi.org/10.1109/PIC.2015.7489879
44. Kurniawan, H., Y. Rosmansyah, and B. Dabarsyah. Android anomaly detection system using machine learning classification. in Electrical Engineering and Informatics (ICEEI), 2015 International Conference on. 2015. IEEE. doi:https://doi.org/10.1109/ICEEI.2015.7352512
45. Alzaylaee, M.K., S.Y. Yerima, and S. Sezer. Emulator vs real phone: Android malware detection using machine learning. in Proceedings of the 3rd ACM on International Workshop on Security And Privacy Analytics. 2017. ACM. doi:https://doi.org/10.1145/3041008.3041010

46. Shuaifu, D., W. Tao, and Z. Wei. DroidLogger: Reveal suspicious behavior of Android applications via instrumentation. in 2012 7th International Conference on Computing and Convergence Technology (ICCCT). 2012.
47. Ali-Gombe, A.I., B. Saltaformaggio, J.R. Ramanujam, D. Xu, and G.G. Richard (2018) Toward a more dependable hybrid analysis of android malware using aspect-oriented programming. Computers & Security 73:235-248.
48. Rastogi, V., Y. Chen, and W. Enck. AppsPlayground: automatic security analysis of smartphone applications. in Proceedings of the third ACM conference on Data and application security and privacy. 2013. ACM. doi:https://doi.org/10.1145/2435349.2435379
49. Enck, W., P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L.P. Cox, J. Jung, P. McDaniel, and A.N. Sheth (2014) TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. ACM Transactions on Computer Systems (TOCS) 32 (2):5. doi:https://doi.org/10.1145/2619091
50. Yan, L.-K. and H. Yin. DroidScope: Seamlessly Reconstructing the OS and Dalvik Semantic Views for Dynamic Android Malware Analysis. in USENIX security symposium. 2012.
51. Kabakus, A.T. and I.A. Dogru (2018) An in-depth analysis of Android malware using hybrid techniques. Digital Investigation 24:25-33. doi:10.1016/j.diin.2018.01.001
52. Yuan, Z., Y. Lu, and Y. Xue (2016) Droiddetector: android malware characterization and detection using deep learning. Tsinghua Science and Technology 21 (1):114-123. doi:10.1109/TST.2016.7399288
53. Chen, S., M. Xue, Z. Tang, L. Xu, and H. Zhu (2016) StormDroid: A Streaminglized Machine Learning-Based System for Detecting Android Malware.377-388. doi:10.1145/2897845.2897860
54. Arp, D., M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens. Drebin: Effective and explainable detection of android malware in your pocket. in Ndss. 2014.
55. Mas'ud, M.Z., S. Shahrin, M.F. Abdollah, S.R. Selamat, R. Yusof, and R. Ahmad (2013) Profiling mobile malware behaviour through hybrid malware analysis approach. 2013 9th International Conference on Information Assurance and Security (IAS)
doi:https://doi.org/10.1109/ISIAS.2013.6947737
56. Talha, K.A., D.I. Alper, and C. Aydin (2015) APK Auditor: Permission-based Android malware detection system. Digital Investigation 13:1-14. doi:10.1016/j.diin.2015.01.001
57. Samra, A.A.A. and O.A. Ghanem (2013) Analysis of Clustering Technique in Android Malware Detection. 2013 Seventh International Conference on. IEEE:729-733. doi:10.1109/imis.2013.111
58. Feizollah, A., N.B. Anuar, R. Salleh, G. Suarez-Tangil, and S. Furnell (2017) AndroDialysis: Analysis of Android Intent Effectiveness in Malware Detection. Computers & Security 65:121-134. doi:10.1016/j.cose.2016.11.007
59. Zhang, M., Y. Duan, H. Yin, and Z. Zhao (2014) Semantics-Aware Android Malware Classification Using Weighted Contextual API Dependency Graphs. 2014 ACM SIGSAC Conference on Computer and Communications Security:1105-1116. doi:10.1145/2660267.2660359

60. Moghaddam, S.H. and M. Abbaspour. Sensitivity analysis of static features for Android malware detection. in Electrical Engineering (ICEE), 2014 22nd Iranian Conference on. 2014. IEEE. doi:10.1109/IranianCEE.2014.6999667
61. Wu, D.-J., C.-H. Mao, T.-E. Wei, H.-M. Lee, and K.-P. Wu. Droidmat: Android malware detection through manifest and api calls tracing. in Information Security (Asia JCIS), 2012 Seventh Asia Joint Conference on. 2012. IEEE.
62. Martín, A., H.D. Menéndez, and D. Camacho (2016) MOCDroid: multi-objective evolutionary classifier for Android malware detection. Soft Computing 21 (24):7405-7415. doi:10.1007/s00500-016-2283-y
63. Aafer, Y., W. Du, and H. Yin. DroidAPIMiner: Mining API-Level Features for Robust Malware Detection in Android. in Security and Privacy in Communication Networks. 2013. Cham: Springer International Publishing.
64. Felt, A.P., E. Chin, S. Hanna, D. Song, and D. Wagner. Android permissions demystified. in Proceedings of the 18th ACM conference on Computer and communications security. 2011 of Conference. Chicago, Illinois, USA: ACM. doi:10.1145/2046707.2046779
65. Sen, S., A.I. Aysan, and J.A. Clark. SAFEDroid: Using Structural Features for Detecting Android Malwares. in Security and Privacy in Communication Networks. 2018. Cham: Springer International Publishing.
66. Milosevic, N., A. Dehghantanha, and K.-K.R. Choo (2017) Machine learning aided Android malware classification. Computers & Electrical Engineering 61:266-274. doi:10.1016/j.compeleceng.2017.02.013
67. Yerima, S.Y. and S. Sezer (2018) DroidFusion: A Novel Multilevel Classifier Fusion Approach for Android Malware Detection. IEEE Transactions on Cybernetics:1-14. doi:10.1109/tcyb.2017.2777960
68. Zhang, L., Yan Niu, Xiao Wu, Zhaoguo Wang, and Yibo A3: Automatic Analysis of Android Malware. in International Workshop on Cloud Computing and Information Security. 2013 of Conference.
69. Park, W., K.-H. Lee, K.-S. Cho, and W. Ryu. Analyzing and detecting method of android malware via disassembling and visualization. in Information and Communication Technology Convergence (ICTC), 2014 International Conference on. 2014. IEEE. doi:10.1109/ICTC.2014.6983300
70. Lee, J., S. Lee, and H. Lee (2015) Screening smartphone applications using malware family signatures. Computers & Security 52:234-249. doi:10.1016/j.cose.2015.02.003
71. Palumbo, P., L. Sayfullina, D. Komashinskiy, E. Eirola, and J. Karhunen (2017) A pragmatic android malware detection procedure. Computers & Security 70:689-701. doi:10.1016/j.cose.2017.07.013
72. Kwon, J., J. Jeong, J. Lee, and H. Lee. Droidgraph: discovering android malware by analyzing semantic behavior. in Communications and Network Security (CNS), 2014 IEEE Conference on. 2014. IEEE. doi:10.1109/CNS.2014.6997523
73. Alam, S., Z. Qu, R. Riley, Y. Chen, and V. Rastogi (2017) DroidNative: Automating and optimizing detection of Android native code malware variants. Computers & Security 65:230-246. doi:10.1016/j.cose.2016.11.011
74. Wang, C., Q. Xu, X. Lin, and S. Liu (2018) Research on data mining of permissions mode for Android malware detection. Cluster Computing. doi:10.1007/s10586-018-1904-x

75. Elish, K.O., X. Shu, D. Yao, B.G. Ryder, and X. Jiang (2015) Profiling user-trigger dependence for Android malware detection. Computers & Security 49:255-273. doi:10.1016/j.cose.2014.11.001
76. Brown, J., M. Anwar, and G. Dozier (2016) Detection of Mobile Malware: An Artificial Immunity Approach.74-80. doi:10.1109/spw.2016.32
77. Junaid, M., D. Liu, and D. Kung (2016) Dexteroid: Detecting malicious behaviors in Android apps using reverse-engineered life cycle models. Computers & Security 59:92-117. doi:10.1016/j.cose.2016.01.008
78. Sanz, B., I. Santos, C. Laorden, X. Ugarte-Pedrero, and P.G. Bringas. On the automatic categorisation of android applications. in Consumer Communications and Networking Conference (CCNC), 2012 IEEE. 2012. IEEE.
79. Tan, M., M. Yu, Y. Wang, S. Li, and C. Liu. Android malware detection combining feature correlation and Bayes classification model. in Communication Software and Networks (ICCSN), 2017 IEEE 9th International Conference on. 2017. IEEE. doi:https://doi.org/10.1109/ICCSN.2017.8230195
80. Wang, C., Z. Li, X. Mo, H. Yang, and Y. Zhao (2017) An android malware dynamic detection method based on service call co-occurrence matrices. Annals of Telecommunications 72 (9-10):607-615. doi:10.1007/s12243-017-0580-9
81. Chang, W.-L., H.-M. Sun, and W. Wu. An Android Behavior-Based Malware Detection Method using Machine Learning. in Signal Processing, Communications and Computing (ICSPCC), 2016 IEEE International Conference on. 2016. IEEE.
82. Amos, B., H. Turner, and J. White. Applying machine learning classifiers to dynamic android malware detection at scale. in Wireless communications and mobile computing conference (iwcmc), 2013 9th international. 2013. IEEE.
83. Kumar, A., K.P. Sagar, K.S. Kuppusamy, and G. Aghila. Machine learning based malware classification for Android applications using multimodal image representations. in 2016 10th International Conference on Intelligent Systems and Control (ISCO). 2016. doi:10.1109/ISCO.2016.7726949
84. Feldman, S., D. Stadther, and B. Wang (2014) Manilyzer: Automated Android Malware Detection through Manifest Analysis.767-772. doi:10.1109/mass.2014.65
85. Bakour, K., G.S. Daş, and H.M. Ünver. An intrusion detection system based on a hybrid Tabu-genetic algorithm. in Computer Science and Engineering (UBMK), 2017 International Conference on. 2017. IEEE
86. Shen, T., Y. Zhongyang, Z. Xin, B. Mao, and H. Huang (2014) Detect Android Malware Variants Using Component Based Topology Graph. 2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom):406-413. doi:10.1109/TrustCom.2014.52
87. Faruki, P., V. Laxmi, A. Bharmal, M.S. Gaur, and V. Ganmoor (2015) AndroSimilar: Robust signature for detecting variants of Android malware. Journal of Information Security and Applications 22:66-80. doi:10.1016/j.jisa.2014.10.011
88. Grace, M., Y. Zhou, Q. Zhang, S. Zou, and X. Jiang. RiskRanker: Scalable and Accurate Zero-day Android. in Proceedings of the 10th international conference on Mobile systems, applications, and services. 2012 of Conference.: ACM. doi:10.1145/2307636.2307663

89. Crussell, J., C. Gibler, and H. Chen (2015) AnDarwin: Scalable Detection of Android Application Clones Based on Semantics. IEEE Transactions on Mobile Computing 14 (10):2007-2019. doi:10.1109/TMC.2014.2381212
90. Yang, X., D. Lo, L. Li, X. Xia, T.F. Bissyandé, and J. Klein (2017) Characterizing malicious Android apps by mining topic-specific data flow signatures. Information and Software Technology 90:27-39.
91. Sheen, S., R. Anitha, and V. Natarajan (2015) Android based malware detection using a multifeature collaborative decision fusion approach. Neurocomputing 151:905-912. doi:10.1016/j.neucom.2014.10.004
92. Fereidooni, H., M. Conti, D. Yao, and A. Sperduti. ANASTASIA: ANdroid mAlware detection using STatic analySIs of Applications. in New Technologies, Mobility and Security (NTMS), 2016 8th IFIP International Conference on. 2016. IEEE. doi:https://doi.org/10.1109/NTMS.2016.7792435
93. Ma, L., Y. Yang, X. Wang, and J. He (2016) Ultra-Lightweight Malware Detection of Android Using 2-Level Machine Learning.729-733. doi:https://doi.org/10.1109/ICISCE.2016.161
94. Du, Y., J. Wang, and Q. Li (2017) An Android Malware Detection Approach Using Community Structures of Weighted Function Call Graphs. IEEE Access 5:17478-17486. doi:10.1109/access.2017.2720160
95. Aung, Z. and W. Zaw (2013) Permission-based android malware detection. International Journal of Scientific & Technology Research 2 (3):228-234
96. Verma, S. and S.K. Muttoo (2016) An Android Malware Detection Framework-based on Permissions and Intents. Defence Science Journal 66 (6):618. doi:10.14429/dsj.66.10803
97. Karbab, E.B., M. Debbabi, A. Derhab, and D. Mouheb (2017) Android Malware Detection using Deep Learning on API Method Sequences. arXiv preprint arXiv:1712.08996. doi:https://arxiv.org/abs/1712.08996v1
98. Hou, S., A. Saas, L. Chen, and Y. Ye. Deep4maldroid: A deep learning framework for android malware detection based on linux kernel system call graphs. in Web Intelligence Workshops (WIW), IEEE/WIC/ACM International Conference on. 2016. IEEE.
99. Nix, R. and J. Zhang. Classification of Android apps and malware using deep neural networks. in Neural Networks (IJCNN), 2017 International Joint Conference on. 2017. IEEE
100. Ng, D.V. and J.-I.G. Hwang. Android malware detection using the dendritic cell algorithm. in Machine Learning and Cybernetics (ICMLC), 2014 International Conference on. 2014. IEEE. doi:https://doi.org/10.1109/ICMLC.2014.7009126
101. Wu, S., P. Wang, X. Li, and Y. Zhang (2016) Effective detection of android malware based on the usage of data flow APIs and machine learning. Information and Software Technology 75:17-25.
102. Spreitzenbarth, M., T. Schreck, F. Echtler, D. Arp, and J. Hoffmann (2014) Mobile-Sandbox: combining static and dynamic analysis with machine-learning techniques. International Journal of Information Security 14. doi:10.1007/s10207-014-0250-0
103. Moonsamy, V., J. Rong, and S. Liu (2014) Mining permission patterns for contrasting clean and malicious android applications. Future Generation Computer Systems 36:122-132. doi:10.1016/j.future.2013.09.014

104. Idrees, F., M. Rajarajan, M. Conti, T.M. Chen, and Y. Rahulamathavan (2017) PIndroid: A novel Android malware detection system using ensemble learning methods. Computers & Security 68:36-46. doi:10.1016/j.cose.2017.03.011
105. Mariconti, E., L. Onwuzurike, P. Andriotis, E. De Cristofaro, G. Ross, and G. Stringhini (2016) Mamadroid: Detecting android malware by building markov chains of behavioral models. arXiv preprint arXiv:1612.04433. doi:https://arxiv.org/abs/1612.04433v3
106. Zhou, Y. and X. Jiang (2012) Dissecting Android Malware: Characterization and Evolution.95-109. doi:10.1109/sp.2012.16
107. Yerima, S.Y., G. McWilliams, and S. Sezer (2014) Analysis of Bayesian classification-based approaches for Android malware detection. IET Information Security 8 (1):25-36. doi:10.1049/iet-ifs.2013.0095
108. Zheng, M., P.P. Lee, and J.C. Lui. ADAM: an automatic and extensible platform to stress test android anti-virus systems. in International conference on detection of intrusions and malware, and vulnerability assessment. 2012. Springer.
109. Faruki, P., H. Fereidooni, V. Laxmi, M. Conti, and M. Gaur (2016) Android Code Protection via Obfuscation Techniques: Past, Present and Future Directions. arXiv preprint arXiv:1611.10231.
110. Maiorca, D., D. Ariu, I. Corona, M. Aresu, and G. Giacinto (2015) Stealth attacks: An extended insight into the obfuscation effects on Android malware. Computers & Security 51:16-31. doi:https://doi.org/10.1016/j.cose.2015.02.007
111. Mavrogiannopoulos, N., N. Kisserli, and B. Preneel (2011) A taxonomy of self-modifying code for obfuscation. Computers & Security 30 (8):679-691.
112. Yerima, S.Y., I. Muttik, and S. Sezer (2015) High accuracy android malware detection using ensemble learning. IET Information Security 9 (6):313-320. doi:10.1049/iet-ifs.2014.0099
113. Rastogi, V., Y. Chen, and X. Jiang. DroidChameleon: evaluating Android anti-malware against transformation attacks. in Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security. 2013 of Conference. Hangzhou, China: ACM. doi:10.1145/2484313.2484355
114. Lantz, P. and B. Johansson. Towards bridging the gap between dalvik bytecode and native code during static analysis of android applications. in Wireless Communications and Mobile Computing Conference (IWCMC), 2015 International. 2015. IEEE. doi:10.1109/IWCMC.2015.7289149
115. Spreitzenbarth, M., F. Freiling, F. Echtler, T. Schreck, and J. Hoffmann. Mobile-sandbox: having a deeper look into android applications. in Proceedings of the 28th Annual ACM Symposium on Applied Computing. 2013 of Conference. Coimbra, Portugal: ACM. doi:10.1145/2480362.2480701
116. Glodek, W. and R. Harang (2013) Rapid Permissions-Based Detection and Analysis of Mobile Malware Using Random Decision Forests.980-985. doi:10.1109/milcom.2013.170
117. Zhou, Y., Z. Wang, W. Zhou, and X. Jiang (2012) Hey, You, Get Off of My Market: Detecting Malicious Apps in Official and Alternative Android Markets. NDSS 25
118. Zheng, M., M. Sun, and J.C.S. Lui. DroidTrace: A ptrace based Android dynamic analysis system with forward execution capability. in 2014 International Wireless Communications and Mobile Computing Conference (IWCMC). 2014. doi:10.1109/IWCMC.2014.6906344

119. Hu, W., J. Tao, X. Ma, W. Zhou, S. Zhao, and T. Han. Migdroid: Detecting app-repackaging android malware via method invocation graph. in Computer Communication and Networks (ICCCN), 2014 23rd International Conference on. 2014. IEEE. doi:10.1109/ICCCN.2014.6911805
120. Zhou, W., Y. Zhou, X. Jiang, and P. Ning. Detecting repackaged smartphone applications in third-party android marketplaces. in Proceedings of the second ACM conference on Data and Application Security and Privacy. 2012. ACM. doi:https://doi.org/10.1145/2133601.2133640
121. Alzaylaee, M.K., S.Y. Yerima, and S. Sezer. Improving dynamic analysis of android apps using hybrid test input generation. in Cyber Security And Protection Of Digital Services (Cyber Security), 2017 International Conference on. 2017. IEEE. doi:https://doi.org/10.1109/CyberSecPODS.2017.8074845
122. Vidas, T. and N. Christin. Evading android runtime analysis via sandbox detection. in Proceedings of the 9th ACM symposium on Information, computer and communications security - ASIA CCS '14. 2014 of Conference. doi:10.1145/2590296.2590325
123. Alzaylaee, M.K., S.Y. Yerima, and S. Sezer. DynaLog: An automated dynamic analysis framework for characterizing android applications. in Cyber Security And Protection Of Digital Services (Cyber Security), 2016 International Conference On. 2016. IEEE. doi:https://doi.org/10.1109/CyberSecPODS.2016.7502337
124. Li, J., L. Zhai, X. Zhang, and D. Quan. Research of android malware detection based on network traffic monitoring. in Industrial Electronics and Applications (ICIEA), 2014 IEEE 9th Conference on. 2014. IEEE. doi:https://doi.org/10.1109/ICIEA.2014.6931449
125. Tam, K., S.J. Khan, A. Fattori, and L. Cavallaro. CopperDroid: Automatic Reconstruction of Android Malware Behaviors. in NDSS. 2015.
126. Bugiel, S., L. Davi, A. Dmitrienko, T. Fischer, and A. Sadeghi (2011) XManDroid: A new Android evolution to mitigate privilege escalation attacks. Technische Universit. at Darmstadt, Technical Report TR-2011-04,
127. Ongtang, M., S. McLaughlin, W. Enck, and P. McDaniel (2012) Semantically rich application-centric security in Android. Security and Communication Networks 5 (6):658-673. doi:10.1002/sec.360
128. Rastogi, V., Y. Chen, and X. Jiang. Droidchameleon: evaluating android anti-malware against transformation attacks. in Proceedings of the 8th ACM SIGSAC symposium on Information, computer and communications security. 2013. ACM. doi:https://doi.org/10.1145/2484313.2484355
129. Huang, H., S. Zhu, P. Liu, and D. Wu. A framework for evaluating mobile app repackaging detection algorithms. in International Conference on Trust and Trustworthy Computing. 2013. Springer
130. Xue, L., X. Luo, L. Yu, S. Wang, and D. Wu (2017) Adaptive Unpacking of Android Apps. IEEE/ACM 39th International Conference 358-369.
131. Li, B., Y. Zhang, J. Li, W. Yang, and D. Gu (2018) AppSpear : Automating the hidden-code extraction and reassembling of packed android malware. Journal of Systems and Software 140:3-16. doi:10.1016/j.jss.2018.02.040
132. Amalfitano, D., A.R. Fasolino, P. Tramontana, S. De Carmine, and A.M. Memon. Using GUI ripping for automated testing of Android applications. in Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering. 2012. ACM.

133. Machiry, A., R. Tahiliani, and M. Naik. Dynodroid: An input generation system for android apps. in Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering. 2013. ACM.
134. Alzaylaee, M.K., S.Y. Yerima, and S. Sezer (2020) DL-Droid: Deep learning based android malware detection using real devices. Computers & Security 89:101663
135. Ünver, H.M. and K. Bakour (2020) Android malware detection based on image-based features and machine learning techniques. SN Applied Sciences 2 (7):1-15
136. Bakour, K. and H.M. Ünver (2020) VisDroid: Android malware classification based on local and global image features, bag of visual words and machine learning techniques. Neural Computing and Applications:1-21
137. Taheri, R., M. Ghahramani, R. Javidan, M. Shojafar, Z. Pooranian, and M. Conti (2020) Similarity-based Android malware detection using Hamming distance of static binary features. Future Generation Computer Systems 105:230-247
138. Pektaş, A. and T. Acarman (2020) Deep learning for effective Android malware detection using API call graph embeddings. Soft Computing 24 (2):1027-1043
139. Liu, P., W. Wang, X. Luo, H. Wang, and C. Liu (2020) NSDroid: efficient multi-classification of android malware using neighborhood signature in local function call graphs. International Journal of Information Security:1-13
140. Xiao, X., S. Zhang, F. Mercaldo, G. Hu, and A.K. Sangaiah (2019) Android malware detection based on system call sequences and LSTM. Multimedia Tools and Applications 78 (4):3979-3999
141. Pei, X., L. Yu, and S. Tian (2020) AMalNet: A deep learning framework based on graph convolutional networks for malware detection. Computers & Security:101792
142. Mateless, R., D. Rejabek, O. Margalit, and R. Moskovitch (2020) Decompiled APK based malicious code classification. Future Generation Computer Systems
143. Zhang, Y., W. Ren, T. Zhu, and Y. Ren (2019) SaaS: A situational awareness and analysis system for massive android malware detection. Future Generation Computer Systems
144. Zhang, L., V.L. Thing, and Y. Cheng (2019) A scalable and extensible framework for android malware detection and family attribution. Computers & Security 80:120-133
145. Martín, I. and J.A. Hernández (2019) CloneSpot: Fast detection of Android repackages. Future Generation Computer Systems 94:740-748
146. Fan, M., J. Liu, X. Luo, K. Chen, Z. Tian, Q. Zheng, and T. Liu (2018) Android Malware Familial Classification and Representative Sample Selection via Frequent Subgraph Analysis. IEEE Transactions on Information Forensics and Security 13 (8):1890-1905. doi:10.1109/tifs.2018.2806891
147. Papadopoulos, H., N. Georgiou, C. Eliades, and A. Konstantinidis (2018) Android malware detection with unbiased confidence guarantees. Neurocomputing 280:3-12. doi:10.1016/j.neucom.2017.08.072
148. Jha, A.K. and W.J. Lee (2018) An empirical study of collaborative model and its security risk in Android. Journal of Systems and Software 137:550-562.
149. Li, J., L. Sun, Q. Yan, Z. Li, W. Srisa-an, and H. Ye (2018) Significant Permission Identification for Machine Learning Based Android Malware Detection. IEEE Transactions on Industrial Informatics

150. Zhao, C., W. Zheng, L. Gong, M. Zhang, and C. Wang. Quick and Accurate Android Malware Detection Based on Sensitive APIs. in 2018 IEEE International Conference on Smart Internet of Things (SmartIoT). 2018. IEEE
151. Şahın, D.Ö., O.E. Kural, S. Akleylek, and E. Kiliç. New results on permission based static analysis for Android malware. in Digital Forensic and Security (ISDFS), 2018 6th International Symposium on. 2018. IEEE
152. Jin, Y., T. Liu, A. He, Y. Qu, and J. Chi. Android Malware Detector Exploiting Convolutional Neural Network and Adaptive Classifier Selection. in 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC). 2018. IEEE
153. Hasegawa, C. and H. Iyatomi. One-dimensional convolutional neural networks for Android malware detection. in Signal Processing & Its Applications (CSPA), 2018 IEEE 14th International Colloquium on. 2018. IEEE
154. Koli, J. RanDroid: Android malware detection using random machine learning classifiers. in Technologies for Smart-City Energy Security and Power (ICSESP), 2018. 2018. IEEE
155. Riasat, R., M. Sakeena, A.H. Sadiq, and Y.-J. Wang. Onamd: An Online Android Malware Detection Approach. in 2018 International Conference on Machine Learning and Cybernetics (ICMLC). 2018. IEEE
156. Jung, J., H. Kim, D. Shin, M. Lee, H. Lee, S.-j. Cho, and K. Suh. Android Malware Detection Based on Useful API Calls and Machine Learning. in 2018 IEEE First International Conference on Artificial Intelligence and Knowledge Engineering (AIKE). 2018. IEEE
157. Arshad, S., M.A. Shah, A. Wahid, A. Mehmood, H. Song, and H. Yu (2018) SAMADroid: A Novel 3-Level Hybrid Malware Detection Model for Android Operating System. IEEE Access 6:4321-4339
158. Arora, A. and S.K. Peddoju. NTPDroid: A Hybrid Android Malware Detector Using Network Traffic and System Permissions. in 2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science And Engineering (TrustCom/BigDataSE). 2018. IEEE
159. Wang, X., W. Wang, Y. He, J. Liu, Z. Han, and X. Zhang (2017) Characterizing Android apps' behavior for effective detection of malapps at large scale. Future Generation Computer Systems 75:30-45. doi:10.1016/j.future.2017.04.041
160. Sokolova, K., C. Perez, and M. Lemercier (2017) Android application classification and anomaly detection with graph-based permission patterns. Decision Support Systems 93:62-76. doi:10.1016/j.dss.2016.09.006
161. Tong, F. and Z. Yan (2017) A hybrid approach of mobile malware detection in Android. Journal of Parallel and Distributed Computing 103:22-31. doi:10.1016/j.jpdc.2016.10.012
162. Buchanan, W.J., S. Chiale, and R. Macfarlane (2017) A methodology for the security evaluation within third-party Android Marketplaces. Digital Investigation 23:88-98. doi:10.1016/j.diin.2017.10.002
163. Rehman, Z.-U., S.N. Khan, K. Muhammad, J.W. Lee, Z. Lv, S.W. Baik, P.A. Shah, K. Awan, and I. Mehmood (2017) Machine learning-assisted signature and heuristic-based detection of malwares in Android devices. Computers & Electrical Engineering. doi:10.1016/j.compeleceng.2017.11.028

164. Martinelli, F., F. Marulli, and F. Mercaldo (2017) Evaluating convolutional neural network for effective mobile malware detection. Procedia Computer Science 112:2372-2381
165. Liang, H., Y. Song, and D. Xiao. An end-to-end model for Android malware detection. in Intelligence and Security Informatics (ISI), 2017 IEEE International Conference on. 2017. IEEE
166. Su, M.-Y., J.-Y. Chang, and K.-T. Fung. Machine learning on merging static and dynamic features to identify malicious mobile apps. in Ubiquitous and Future Networks (ICUFN), 2017 Ninth International Conference on. 2017. IEEE
167. Narayanan, A., M. Chandramohan, L. Chen, and Y. Liu (2017) Context-aware, adaptive, and scalable Android malware detection through online learning. IEEE Transactions on Emerging Topics in Computational Intelligence 1 (3):157-175
168. Li, D., Z. Wang, L. Li, Z. Wang, Y. Wang, and Y. Xue. FgDetector: Fine-Grained Android Malware Detection. in Data Science in Cyberspace (DSC), 2017 IEEE Second International Conference on. 2017. IEEE
169. Mohsen, F., H. Bisgin, Z. Scott, and K. Strait. Detecting Android Malwares by Mining Statically Registered Broadcast Receivers. in Collaboration and Internet Computing (CIC), 2017 IEEE 3rd International Conference on. 2017. IEEE
170. Zhu, D., H. Jin, Y. Yang, D. Wu, and W. Chen. DeepFlow: Deep learning-based malware detection by mining Android application for abnormal usage of sensitive data. in Computers and Communications (ISCC), 2017 IEEE Symposium on. 2017. IEEE
171. Fan, M., J. Liu, W. Wang, H. Li, Z. Tian, and T. Liu (2017) Dapasa: detecting android piggybacked apps through sensitive subgraph analysis. IEEE Transactions on Information Forensics and Security 12 (8):1772-1785
172. Jang, J.-w., H. Kang, J. Woo, A. Mohaisen, and H.K. Kim (2016) Andro-Dumpsys: Anti-malware system based on the similarity of malware creator and malware centric information. Computers & Security 58:125-138. doi:10.1016/j.cose.2015.12.005
173. Coronado-De-Alba, L.D., A. Rodríguez-Mota, and P.J. Escamilla-Ambrosio. Feature selection and ensemble of classifiers for Android malware detection. in Communications (LATINCOM), 2016 8th IEEE Latin-American Conference on. 2016. IEEE. doi:https://doi.org/10.1109/LATINCOM.2016.7811605
174. Goyal, R., A. Spognardi, N. Dragoni, and M. Argyriou (2016) SafeDroid: A Distributed Malware Detection Service for Android.59-66. doi:https://doi.org/10.1109/SOCA.2016.14
175. Song, J., C. Han, K. Wang, J. Zhao, R. Ranjan, and L. Wang (2016) An integrated static detection and analysis framework for android. Pervasive and Mobile Computing 32:15-25. doi:https://doi.org/10.1016/j.pmcj.2016.03.003
176. Narayanan, A., L. Yang, L. Chen, and L. Jinliang. Adaptive and scalable android malware detection through online learning. in Neural Networks (IJCNN), 2016 International Joint Conference on. 2016. IEEE. doi:https://doi.org/10.1109/IJCNN.2016.7727508
177. Chen, W., D. Aspinall, A.D. Gordon, C. Sutton, and I. Muttik (2016) More Semantics More Robust.147-158. doi:10.1145/2939918.2939931
178. Ju, S.-h., H.-s. Seo, and J. Kwak (2016) Research on android malware permission pattern using permission monitoring system. Multimedia Tools and Applications 75 (22):14807-14817. doi:10.1007/s11042-016-3273-x

179. Wang, K., T. Song, and A. Liang. Mmda: Metadata Based Malware Detection on Android. in Computational Intelligence and Security (CIS), 2016 12th International Conference on. 2016. IEEE
180. Wang, Z., J. Cai, S. Cheng, and W. Li. DroidDeepLearner: Identifying Android malware using deep learning. in Sarnoff Symposium, 2016 IEEE 37th. 2016. IEEE
181. Zhang, X., D. Hu, Y. Fan, and K. Yu. A novel android malware detection method based on markov blanket. in Data Science in Cyberspace (DSC), IEEE International Conference on. 2016. IEEE
182. Yang, M. and Q. Wen. Detecting Android malware with intensive feature engineering. in Software Engineering and Service Science (ICSESS), 2016 7th IEEE International Conference on. 2016. IEEE
183. Martín, A., H.D. Menéndez, and D. Camacho. String-based malware detection for android environments. in International Symposium on Intelligent and Distributed Computing. 2016. Springer
184. Su, X., D. Zhang, W. Li, and K. Zhao. A deep learning approach to android malware feature learning and detection. in Trustcom/BigDataSE/I SPA, 2016 IEEE. 2016. IEEE
185. Wang, Z., C. Li, Z. Yuan, Y. Guan, and Y. Xue (2016) DroidChain: A novel Android malware detection method based on behavior chains. Pervasive and Mobile Computing 32:3-14
186. Karbab, E.B., M. Debbabi, and D. Mouheb (2016) Fingerprinting Android packaging: Generating DNAs for malware detection. Digital Investigation 18:S33-S45
187. Zhang, X. and Z. Jin. A new semantics-based android malware detection. in Computer and Communications (ICCC), 2016 2nd IEEE International Conference on. 2016. IEEE
188. Morales-Ortega, S., P.J. Escamilla-Ambrosio, A. Rodriguez-Mota, and L.D. Coronado-De-Alba. Native malware detection in smartphones with android os using static analysis, feature selection and ensemble classifiers. in Malicious and Unwanted Software (MALWARE), 2016 11th International Conference on. 2016. IEEE
189. Canfora, G., A.D. Lorenzo, E. Medvet, F. Mercaldo, and C.A. Visaggio (2015) Effectiveness of Opcode ngrams for Detection of Multi Family Android Malware.333-340. doi:10.1109/ares.2015.57
190. Li, Q. and X. Li (2015) Android Malware Detection Based on Static Analysis of Characteristic Tree.84-91. doi:10.1109/CyberC.2015.88
191. Westyarian, Y. Rosmansyah, and B. Dabarsyah. Malware detection on Android smartphones using API class and machine learning. in 2015 International Conference on Electrical Engineering and Informatics (ICEEI). 2015. doi:10.1109/ICEEI.2015.7352513
192. Li, W., J. Ge, and G. Dai. Detecting malware for android platform: An svm-based approach. in Cyber Security and Cloud Computing (CSCloud), 2015 IEEE 2nd International Conference on. 2015. IEEE. doi:https://doi.org/10.1109/CSCloud.2015.50
193. Gordon, M.I., D. Kim, J. Perkins, L. Gilham, N. Nguyen, and M. Rinard (2015) Information-Flow Analysis of Android Applications in DroidSafe. doi:10.14722/ndss.2015.23089

194. Damshenas, M., A. Dehghantanha, K.-K.R. Choo, and R. Mahmud (2015) M0Droid: An Android Behavioral-Based Malware Detection Model. Journal of Information Privacy and Security 11 (3):141-157. doi:10.1080/15536548.2015.1073510
195. Almin, S.B. and M. Chatterjee (2015) A Novel Approach to Detect Android Malware. Procedia Computer Science 45:407-417.
196. Lindorfer, M., M. Neugschwandtner, and C. Platzer. Marvin: Efficient and comprehensive mobile app classification through static and dynamic analysis. in Computer Software and Applications Conference (COMPSAC), 2015 IEEE 39th Annual. 2015. IEEE
197. Jain, A., H. Gonzalez, and N. Stakhanova. Enriching reverse engineering through visual exploration of Android binaries. in Proceedings of the 5th Program Protection and Reverse Engineering Workshop. 2015. ACM
198. Bierma, M., E. Gustafson, J. Erickson, D. Fritz, and Y.R. Choe (2014) Andlantis: Large-scale Android dynamic analysis. arXiv preprint arXiv:1410.7751. doi:https://arxiv.org/abs/1410.7751v1
199. Adebayo, O.S. and N. AbdulAziz. Android malware classification using static code analysis and apriori algorithm improved with particle swarm optimization. in Information and Communication Technologies (WICT), 2014 Fourth World Congress on. 2014. IEEE. doi:10.1109/WICT.2014.7077314
200. Liang, S., X. Du, C.C. Tan, and W. Yu. An effective online scheme for detecting Android malware. in Computer Communication and Networks (ICCCN), 2014 23rd International Conference on. 2014. IEEE.
201. Lagerspetz, E., H.T.T. Truong, S. Tarkoma, and N. Asokan (2014) MDoctor: A Mobile Malware Prognosis Application.201-206. doi:10.1109/icdcsw.2014.36
202. Merlo, A., M. Migliardi, and P. Fontanelli. On energy-based profiling of malware in Android. in 2014 International Conference on High Performance Computing & Simulation (HPCS). 2014. doi:10.1109/HPCSim.2014.6903732
203. Suarez-Tangil, G., J.E. Tapiador, P. Peris-Lopez, and J. Blasco (2014) Dendroid: A text mining approach to analyzing and classifying code structures in Android malware families. Expert Systems with Applications 41 (4):1104-1117. doi:10.1016/j.eswa.2013.07.106
204. Hsiao, S.W., S.H. Hung, R. Chien, and C.W. Yeh (2014) PasDroid: Real-Time Security Enhancement for Android.229-235. doi:10.1109/imis.2014.28
205. Yerima, S.Y., S. Sezer, and I. Muttik. Android malware detection using parallel machine learning classifiers. in Next Generation Mobile Apps, Services and Technologies (NGMAST), 2014 Eighth International Conference on. 2014. IEEE. doi:10.1109/NGMAST.2014.23
206. Zhao, S., X. Li, G. Xu, L. Zhang, and Z. Feng (2014) Attack Tree Based Android Malware Detection with Hybrid Analysis. Trust, Security and Privacy in Computing and Communications (TrustCom):380-387. doi:10.1109/TrustCom.2014.49
207. Feng, Y., S. Anand, I. Dillig, and A. Aiken. Apposcopy: Semantics-based detection of android malware through static analysis. in Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering. 2014. ACM. doi:10.1145/2635868.2635869
208. Xiaoyan, Z., F. Juan, and W. Xiujuan (2014) Android malware detection based on permissions. doi:10.1049/cp.2014.0605

209. Xiangyu, J. Android malware detection through permission and package. in 2014 International Conference on Wavelet Analysis and Pattern Recognition. 2014. doi:10.1109/ICWAPR.2014.6961291
210. Liang, S. and X. Du. Permission-combination-based scheme for android mobile malware detection. in Communications (ICC), 2014 IEEE International Conference on. 2014. IEEE. doi:10.1109/ICC.2014.6883666
211. Idrees, F. and M. Rajarajan. Investigating the android intents and permissions for malware detection. in Wireless and Mobile Computing, Networking and Communications (WiMob), 2014 IEEE 10th International Conference on. 2014. IEEE. doi:10.1109/WiMOB.2014.6962194
212. Raphael, R., P. Vinod, and B. Omman. X-ANOVA and X-Utest features for Android malware analysis. in Advances in Computing, Communications and Informatics (ICACCI, 2014 International Conference on. 2014. IEEE. doi:10.1109/ICACCI.2014.6968608
213. Wolfe, B., K.O. Elish, and D.D. Yao. Comprehensive behavior profiling for proactive Android malware detection. in International Conference on Information Security. 2014. Springer
214. Seo, S.-H., A. Gupta, A.M. Sallam, E. Bertino, and K. Yim (2014) Detecting mobile malware threats to homeland security through static analysis. Journal of Network and Computer Applications 38:43-53
215. Deepa, K., G. Radhamani, and P. Vinod (2015) Investigation of Feature Selection Methods for Android Malware Analysis. Procedia Computer Science 46:841-848
216. Shabtai, A., L. Tenenboim-Chekina, D. Mimran, L. Rokach, B. Shapira, and Y. Elovici (2014) Mobile malware detection through analysis of deviations in application network behavior. Computers & Security 43:1-18. doi:10.1016/j.cose.2014.02.009
217. Yerima, S.Y., S. Sezer, G. McWilliams, and I. Muttik. A New Android Malware Detection Approach Using Bayesian Classification. in 2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA). 2013. doi:10.1109/AINA.2013.88
218. Tenenboim-Chekina, L., O. Barad, A. Shabtai, D. Mimran, L. Rokach, B. Shapira, and Y. Elovici. Detecting application update attack on mobile devices through network featur. in Computer Communications Workshops (INFOCOM WKSHPS), 2013 IEEE Conference on. 2013. IEEE. doi:https://doi.org/10.1109/INFCOMW.2013.6970755
219. Karami, M., M. Elsabagh, P. Najafiborazjani, and A. Stavrou (2013) Behavioral Analysis of Android Applications Using Automated Instrumentation.182-187. doi:10.1109/sere-c.2013.35
220. Vasquez, S. and J. Simmonds (2013) Mobile Application Monitoring. 2013 32nd International Conference of the Chilean Computer Science Society:30-32. doi:10.1109/sccc.2013.16
221. Backes, M., S. Gerling, C. Hammer, M. Maffei, and P. Styp-Rekowsky. AppGuard–Fine-grained policy enforcement for untrusted Android applications. in Revised Selected Papers of the 8th International Workshop on Data Privacy Management and Autonomous Spontaneous Security - Volume 8247. 2014 of Conference.: Springer-Verlag. doi:10.1007/978-3-642-54568-9_14
222. Pandita, R., Xusheng Xiao, Wei Yang, William Enck, and Tao Xie. WHYPER: Towards Automating Risk Assessment of Mobile Applications. in USENIX

Security Symposium. 2013 of Conference. doi:http://citeseerx.ist.psu.edu/viewdoc/similar?doi=10.1.1.674.5247&type=cc
223. Peiravian, N. and X. Zhu (2013) Machine Learning for Android Malware Detection Using Permission and API Calls.300-305. doi:10.1109/ictai.2013.53
224. Lu, Y., P. Zulie, L. Jingju, and S. Yi (2013) Android Malware Detection Technology Based on Improved Bayesian Classification.1338-1341. doi:10.1109/imccc.2013.297
225. Wei, Y., H. Zhang, L. Ge, and R. Hardy. On behavior-based detection of malware on android platform. in Global Communications Conference (GLOBECOM), 2013 IEEE. 2013. IEEE
226. Alam, M.S. and S.T. Vuong. Random forest classification for detecting android malware. in Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing. 2013. IEEE
227. Ham, H.-S. and M.-J. Choi. Analysis of android malware detection performance using machine learning classifiers. in ICT Convergence (ICTC), 2013 International Conference on. 2013. IEEE
228. Zhang, Y., M. Yang, B. Xu, Z. Yang, G. Gu, P. Ning, X.S. Wang, and B. Zang. Vetting undesirable behaviors in android apps with permission use analysis. in Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security. 2013. ACM
229. Zheng, M., M. Sun, and J. Lui (2013) Droidanalytics: a signature based analytic system to collect, extract, analyze and associate android malware. arXiv preprint arXiv:1302.7212
230. Eder, T., M. Rodler, D. Vymazal, and M. Zeilinger. ANANAS - A Framework for Analyzing Android Applications. in 2013 International Conference on Availability, Reliability and Security. 2013 of Conference. doi:10.1109/ares.2013.93
231. Sanz, B., I. Santos, C. Laorden, X. Ugarte-Pedrero, P.G. Bringas, and G. Álvarez. Puma: Permission usage to detect malware in android. in International Joint Conference CISIS'12-ICEUTE 12-SOCO 12 Special Sessions. 2013. Springer
232. Zheng, C., S. Zhu, S. Dai, G. Gu, X. Gong, X. Han, and W. Zou. SmartDroid: an automatic system for revealing UI-based trigger conditions in android applications. in Proceedings of the second ACM workshop on Security and privacy in smartphones and mobile devices. 2012 of Conference. Raleigh, North Carolina, USA: ACM. doi:10.1145/2381934.2381950
233. Wei, X., L. Gomez, I. Neamtiu, and M. Faloutsos. ProfileDroid: multi-layer profiling of android applications. 2012 of Conference.: 18th annual international conference on Mobile computing and networking. ACM. doi:10.1145/2348543.2348563
234. Gibler, C., J. Crussell, J. Erickson, and H. Chen. AndroidLeaks: automatically detecting potential privacy leaks in android applications on a large scale. in International Conference on Trust and Trustworthy Computing. 2012. Springer. doi:https://doi.org/10.1007/978-3-642-30921-2_17
235. Wei, X., L. Gomez, I. Neamtiu, and M. Faloutsos. Permission evolution in the android ecosystem. in Proceedings of the 28th Annual Computer Security Applications Conference. 2012. ACM

236. Dini, G., F. Martinelli, A. Saracino, and D. Sgandurra. MADAM: a multi-level anomaly detector for android malware. in International Conference on Mathematical Methods, Models, and Architectures for Computer Network Security. 2012. Springer. doi:https://doi.org/10.1007/978-3-642-33704-8_21
237. Sahs, J. and L. Khan. A machine learning approach to android malware detection. in Intelligence and security informatics conference (eisic), 2012 european. 2012. IEEE
238. Yang, Z. and M. Yang. Leakminer: Detect information leakage on android with static taint analysis. in Software Engineering (WCSE), 2012 Third World Congress on. 2012. IEEE
239. Gascon, H., F. Yamaguchi, D. Arp, and K. Rieck. Structural detection of android malware using embedded call graphs. in Proceedings of the 2013 ACM workshop on Artificial intelligence and security. 2013. ACM
240. Russello, G., B. Crispo, E. Fernandes, and Y. Zhauniarovich. Yaase: Yet another android security extension. in Privacy, Security, Risk and Trust (PASSAT) and 2011 IEEE Third Inernational Conference on Social Computing (SocialCom), 2011 IEEE Third International Conference on. 2011. IEEE
241. Su, X., M. Chuah, and G. Tan. Smartphone Dual Defense Protection Framework: Detecting Malicious Applications in Android Markets. in 2012 8th International Conference on Mobile Ad-hoc and Sensor Networks (MSN). 2012 of Conference. doi:10.1109/msn.2012.43
242. Dietz, M., S. Shekhar, Y. Pisetsky, A. Shu, and D.S. Wallach. Quire: Lightweight provenance for smart phone operating systems. in USENIX security symposium. 2011. San Francisco, CA;
243. Burguera, I., U. Zurutuza, and S. Nadjm-Tehrani. Crowdroid: behavior-based malware detection system for android. in Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices. 2011. ACM. doi:https://doi.org/10.1145/2046614.2046619
244. Chin, E., A.P. Felt, K. Greenwood, and D. Wagner. Analyzing inter-application communication in Android. in Proceedings of the 9th international conference on Mobile systems, applications, and services. 2011 of Conference. Bethesda, Maryland, USA: ACM. doi:10.1145/1999995.2000018
245. Isohara, T., K. Takemori, and A. Kubota. Kernel-based Behavior Analysis for Android Malware Detection. in 2011 Seventh International Conference on Computational Intelligence and Security. 2011 of Conference. doi:10.1109/cis.2011.226
246. Nauman, M., S. Khan, and X. Zhang. Apex: extending android permission model and enforcement with user-defined runtime constraints. in Proceedings of the 5th ACM symposium on information, computer and communications security. 2010. ACM. doi:https://doi.org/10.1145/1755688.1755732
247. Ongtang, M., K. Butler, and P. McDaniel. Porscha: Policy oriented secure content handling in Android. in Proceedings of the 26th Annual Computer Security Applications Conference. 2010. ACM. doi:https://doi.org/10.1145/1920261.1920295
248. Conti, M., V.T.N. Nguyen, and B. Crispo (2011) CRePE: Context-Related Policy Enforcement for Android. 6531:331-345. doi:10.1007/978-3-642-18178-8_29
249. Portokalidis, G., P. Homburg, K. Anagnostakis, and H. Bos. Paranoid Android: versatile protection for smartphones. in Proceedings of the 26th Annual

Computer Security Applications Conference. 2010. ACM. doi:https://doi.org/10.1145/1920261.1920313
250. Barrera, D., G. Kayacık, P. C. van Oorschot, and A. Somayaji (2010) A Methodology for Empirical Analysis of Permission-Based Security Models and its Application to Android. doi:10.1145/1866307.1866317
251. Blasing, T., L. Batyuk, A.-D. Schmidt, S.A. Camtepe, and S. Albayrak. An android application sandbox system for suspicious software detection. in 2010 5th International Conference on Malicious and Unwanted Software (MALWARE 2010). 2010. IEEE
252. Shabtai, A., Y. Fledel, and Y. Elovici. Automated Static Code Analysis for Classifying Android Applications Using Machine Learning. in 2010 International Conference on Computational Intelligence and Security. 2010 of Conference. doi:10.1109/cis.2010.77
253. Enck, W., M. Ongtang, and P. McDaniel. On lightweight mobile phone application certification. in Proceedings of the 16th ACM conference on Computer and communications security. 2009 of Conference. Chicago, Illinois, USA: ACM. doi:10.1145/1653662.1653691
254. Bakour, K., H.M. Ünver, and R. Ghanem (2019) A Deep Camouflage: Evaluating Android's Anti-malware Systems Robustness Against Hybridization of Obfuscation Techniques with Injection Attacks. Arabian Journal for Science and Engineering 44 (11):9333-9347
255. Zhou, Y. and X. Jiang. Dissecting android malware: Characterization and evolution. in 2012 IEEE symposium on security and privacy. 2012. IEEE
256. Lisin, D.A., M.A. Mattar, M.B. Blaschko, E.G. Learned-Miller, and M.C. Benfield. Combining local and global image features for object class recognition. in 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)-Workshops. 2005. IEEE
257. MALLICK, S. (2018). "Shape Matching using Hu Moments " Shape Matching using Hu Moments https://www.learnopencv.com/shape-matching-using-hu-moments-c-python/ Accessed 19-04-2019.
258. Huang, Z. and J. Leng. Analysis of Hu's moment invariants on image scaling and rotation. in 2010 2nd International Conference on Computer Engineering and Technology. 2010. IEEE
259. Kumar, R.M. and K. Sreekumar (2014) A survey on image feature descriptors. Int J Comput Sci Inf Technol 5:7668-7673
260. Haralick, R.M., K. Shanmugam, and I.H. Dinstein (1973) Textural features for image classification. IEEE Transactions on systems, man, and cybernetics (6):610-621
261. Lowe, D.G. (2004) Distinctive image features from scale-invariant keypoints. International journal of computer vision 60 (2):91-110
262. Bay, H., T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. in European conference on computer vision. 2006. Springer
263. Alcantarilla, P.F., A. Bartoli, and A.J. Davison. KAZE features. in European Conference on Computer Vision. 2012. Springer
264. Rublee, E., V. Rabaud, K. Konolige, and G.R. Bradski. ORB: An efficient alternative to SIFT or SURF. in ICCV. 2011. Citeseer
265. Rosten, E. and T. Drummond. Machine learning for high-speed corner detection. in European conference on computer vision. 2006. Springer

266. Calonder, M., V. Lepetit, C. Strecha, and P. Fua. Brief: Binary robust independent elementary features. in European conference on computer vision. 2010. Springer
267. He, K., X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. in Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.
268. Szegedy, C., V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. in Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.
269. Szegedy, C., W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. in Proceedings of the IEEE conference on computer vision and pattern recognition. 2015.
270. Sharma, H. (2019). "ReLU, Leaky ReLU and Softmax basics for Neural Networks and Deep Learning." https://medium.com/@himanshuxd/activation-functions-sigmoid-relu-leaky-relu-and-softmax-basics-for-neural-networks-and-deep-8d9c70eed91e#:~:text=ReLU Accessed 19-10-2020.
271. Pektaş, A. and T. Acarman (2019) Learning to detect Android malware via opcode sequences. Neurocomputing
272. Wang, C., Q. Xu, X. Lin, and S. Liu (2019) Research on data mining of permissions mode for Android malware detection. Cluster Computing 22 (6):13337-13350
273. Hsien-De Huang, T. and H.-Y. Kao. R2-d2: Color-inspired convolutional neural network (cnn)-based android malware detections. in 2018 IEEE International Conference on Big Data (Big Data). 2018. IEEE
274. Goyal, R., A. Spognardi, N. Dragoni, and M. Argyriou. SafeDroid: a distributed malware detection service for Android. in 2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA). 2016. IEEE. doi:doi.org/10.1109/SOCA.2016.14
275. Zhang, M., Y. Duan, H. Yin, and Z. Zhao. Semantics-Aware Android Malware Classification Using Weighted Contextual API Dependency Graphs. in Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. 2014 of Conference. Scottsdale, Arizona, USA: Association for Computing Machinery. doi:10.1145/2660267.2660359
276. Yang, M. and Q. Wen. Detecting android malware by applying classification techniques on images patterns. in 2017 IEEE 2nd International Conference on Cloud Computing and Big Data Analysis (ICCCBDA). 2017. IEEE. doi:10.1109/ICCCBDA.2017.7951936
277. Karimi, A. and M.H. Moattar. Android ransomware detection using reduced opcode sequence and image similarity. in 2017 7th International Conference on Computer and Knowledge Engineering (ICCKE). 2017. IEEE
278. Wu, D.-J., C.-H. Mao, T.-E. Wei, H.-M. Lee, and K.-P. Wu. Droidmat: Android malware detection through manifest and api calls tracing. in 2012 Seventh Asia Joint Conference on Information Security. 2012. IEEE
279. Deshotels, L., V. Notani, and A. Lakhotia. Droidlegacy: Automated familial classification of android malware. in Proceedings of ACM SIGPLAN on program protection and reverse engineering workshop 2014. 2014. ACM

ÖZGEÇMİŞ

Adı Soyadı : Halit BAKIR (Khaled BAKOUR)

Yabancı Dil : İngilizce

Eğitim Durumu : Şam Üniversitesi,

Yüksek lisans (Bilgisayar mühendisliği)- 2014

Çalıştığı Kurum/Kurumlar ve Yıl/Yıllar: Software Developer Kırgaz A.Ş Doğal gaz dağıtım şirketi (22/05/2019- devam).

Yayınlar & Eserler

SCI-Expanded Indekslerinde Yer Alan Dergilerde Yayınlanan Makaleler :

Makale	Dergi
DeepVisDroid: Android malware detection by hybridizing image-based features with deep learning techniques	Neural Computing and Applications, Springer
VisDroid: Android malware classification based on local and global image features, bag of visual words and machine learning techniques	Neural Computing and Applications, Springer
A Deep Camouflage: Evaluating Android's Antimalware Systems Robustness Against Hybridization of Obfuscation Techniques with Injection Attacks	Arabian Journal for Science and Engineering. Springer.

ESci Indekslerinde Yer Alan Dergilerde Yayınlanan Makaleler:

Makale	Dergi
Android malware detection based on image-based features and machine learning techniques	SN Applied Sciences. Springer
The Android malware detection systems between hope and reality.	SN Applied Sciences. Springer

Diğer Dergilerde Yayınlanan Makaleler :

Makale	Dergi
The Android malware static analysis: Techniques, Limitations, and Open Challenges	Computer Science and Engineering (UBMK), International Conference on. IEEE, 2018.
An intrusion detection system based on a hybrid Tabu-genetic algorithm	Computer Science and Engineering (UBMK), 2017 International Conference on. IEEE, 2017.