HİBRİT ANALİZ KULLANARAK ANDROİD KÖTÜCÜL YAZILIM AİLE SINIFLANDIRMASI

ANDROID MALWARE FAMILY CLASSIFICATION BY USING HYBRID ANALYSIS

ÖMER FARUK TURAN CAVLI

Doç. Dr. SEVİL ŞEN

Tez Danışmanı

Hacettepe Üniversitesi

Lisansüstü Eğitim-Öğretim ve Sınav Yönetmeliğinin

Bilgisayar Mühendisliği Anabilim Dalı için Öngördüğü

YÜKSEK LİSANS tezi olarak hazırlanmıştır.

2021

ÖZET

HİBRİT ANALİZ KULLANARAK ANDROİD KÖTÜCÜL YAZILIM AİLE SINIFLANDIRMASI

Ömer Faruk Turan CAVLI

Yüksek Lisans, Bilgisayar Bölümü

Tez Danışmanı: Doç. Dr. Sevil ŞEN

Temmuz 2021, 124 sayfa

Mobil ve kablosuz teknolojideki gelişmelerle birlikte mobil cihazlar hayatımızın önemli bir parçası haline gelmiştir. Android işletim sistemi, mobil cihaz kullanıcı kitlesi içerisinde en çok kullanılan işletim sistemi olurken, saldırganlar tarafından da en çok hedef alınan platformdur. Literatürde Android kötücül yazılımlarının tespiti için birçok yöntem önerilmiş olsa da tespit edilen kötücül Android uygulamaların zararlı yazılım aile sınıflandırması, özellikle bu ekosistemde her gün mobil kötücül yazılım varyantlarının sayısının arttığı durumlarda büyük önem kazanmaktadır. Bu çalışmada, Android kötücül aile sınıflandırma problemi için makine öğrenmesine ve hibrit analize dayalı bir yöntem önerilmiştir. Android kötücül yazılım uygulamaları için, hibrit yaklaşım kullanılarak; ağ trafik analizi, uygulamaların cihazda gerçekleştirdiği aktivitelerin ardışık ikili sırasını içeren bilgiler ile öznitelik vektör uzayı genişletilerek kötücül yazılım aile sınıflandırması yöntemi önerilmiştir. Statik ve dinamik analizler ile çıkarılan öznitelikler üzerinde çalışılmış ve yaygın olarak kullanılan Malgenome [21], Drebin [32] ve UpDroid [55] zararlı yazılım veri kümeleri üzerinde sonuçlar elde edilerek değerlendirilmiştir.

Anahtar Kelimeler: Android, mobil güvenlik, zararlı yazılım analizi ve tespiti, aile sınıflandırması, makine öğrenmesi, statik/dinamik analiz, hibrit analiz

ABSTRACT

ANDROID MALWARE FAMILY CLASSIFICATION BY USING HYBRID ANALYSIS

Ömer Faruk Turan CAVLI

Master of Science, Department of Computer Engineering

Supervisor: Assoc. Prof. Sevil ŞEN

July 2021, 124 pages

With the developments in mobile and wireless technology, mobile devices have become an important part of our lives. While Android is the leading operating system in the market share, it is also the most targeted platform by attackers.While there have been many solutions proposed for detection of Android malware in the literature, the family classification of detected malicious applications becomes important, especially where the number of mobile malware variants increases everyday in the market. In this study, a solution based on machine learning and hybrid analysis is proposed for the Android malware familial classification problem. An extensive feature set including network-related features and activity bigrams is proposed. The effective static and dynamic analysis features are studied thoroughly and evaluated on Malgenome [21], Drebin [32] and UpDroid [55] datasets.

Keywords: Android, mobile security, malware analysis and detection, malware family classification, machine learning, static/dynamic analysis, hybrid analysis

TEŞEKKÜR

Tez çalışmam boyunca kıymetli desteklerini esirgemeyen aileme ve tez hazırlığımın bütün aşamalarına değerli bilgi ve deneyimleriyle ışık tutan danışmanım Doç. Dr. Sevil Şen'e sonsuz teşekkürlerimi sunuyorum. Ek olarak, "Updroid: Updated android malware and its familial classification" adlı çalışması ile katkılarından bu tez çalışmasına yapmış olduğu katkıdan dolayı Kürşat AKTAŞ'a teşekkürlerimi sunarım.

Bu çalışma, Türkiye Bilimsel ve Teknolojik Araştırma Kurumu (TÜBİTAK-115E150) tarafından kısmi olarak desteklenmiştir. Katkılarından dolayı TÜBİTAK'a teşekkür ederim.

İÇİNDEKİLER

ŞEKİLLER DİZİNİ

ÇİZELGELER DİZİNİ

SİMGELER VE KISALTMALAR

Kısaltmalar

IDC	International Data Corporation
AOSP	Android Open Source Project
ART	Android Run Time
APK	Android Application Package
Dex	Dalvik Executable
DT	Decision Tree
ET	Extra Trees
KNN	K-Nearest Neighbors
CNN	Convolutional Neural Network
MKL	Multiple Kernel Learning
RF	Random Forest
SVM	Support Vector Machine
NB	Naive Bayes
FP	False Positive
TP	True Positive
FN	False Negative
TN	True Negative
CDG	Component Dependency Graph
CBG	Component Behaviour Graph
CFG	Control Flow Graph
FCG	Function Call Graph
SARG	Sensitive API Call Related Graph

1. GİRİŞ

Günlük yaşantımızda, bireylerin ihtiyaçlarının birçoğunu kolaylaştırması sebebiyle mobil cihazların kullanımı günden güne artmaktadır. Son kullanıcıların mobil cihazlara yöneliminin artması, servis ve hizmet sağlayıcı firmaların sundukları hizmetleri mobil platforma taşımalarına sebep olmaktadır. Bu yaşam döngüsü geliştirilen mobil uygulamaların sayısını da günden güne arttırmaktadır.

Mobil cihazlarda üretici şirketler tarafından birbirinden farklı işletim sistemleri son kullanıcılara sunulmaktadır. Akıllı telefonlarda kullanılan bu işletim sistemleri içerisinde IDC'nin araştırmalarına göre 2020 yılında %84,8 ile pazar payının büyük oranını Android işletim sistemi oluşturmakta ve ilerleyen yıllarda bu oranın artacağı öngörülmektedir [1].

Mobil cihazlarda Android işletim sistemi, GPS, kamera, fener ve benzeri donanımları son kullanıcıların kullanımına sunan, temelde Linux çekirdeğini kullanan ara bir katmandır. Kullanılan altyapının, kütüphanelerin büyük oranda açık kaynak kodlu olması uygulama geliştiriciler için zengin bir içeriğin kullanılabilmesine olanak sağlamaktadır. Bu durum Android işletim sistemini, geliştirici kitlesi içerisinde tercihen bir adım öne taşımaktadır. Android uygulama geliştirmede, Google son yıllarda Kotlin [2] programlama dilinin kullanılmasını teşvik etse de Java [3] programlama dili daha yaygın olarak kullanımını sürdürmektedir. Geliştirici ve kullanıcı kitlelerinin bu denli geniş olması, cihaz donanımına erişimde Java programlama dili kütüphanelerinin sağladığı geniş imkanlar, kötücül yazılım geliştiricilerinin dikkatini çekmiştir.

Son kullanıcılar, finansal bilgiler, kişisel fotoğraflar ve videolar gibi kişisel verilerin birçoğunu ve banka bilgileri ile bunlara bağlı kullanıcı adları ve parolalar gibi saklı kalması gereken verileri mobil cihazlarda saklamaktadırlar. Saldırganlar bu bilgiler doğrultusunda hedef üzerinden maddi çıkar sağlamak veya hedefi maddi zarara uğratmak amacıyla geniş kitlesi olan Android işletim sistemine yönelik kötücül yazılımlar geliştirmektedirler. Kaspersky firması tarafından yayınlanan bir rapora göre 2020'nin ilk çeyreğinde tespit edilen 1.152.662 adet mobil kötücül yazılım, 93.232 adet artarak 2020 ikinci çeyrekte 1.245.894'e

ulaşmıştır [4]. Bu artış Android kötücül yazılımlar üzerinde yapılması gereken analiz yöntemlerinin önemini göstermektedir. Android kötücül yazılım analiz yöntemlerinin gelişmesi ve değişmesine bağlı olarak saldırgan kişilerin bu platforma yaymakta oldukları Android kötücül yazılımlarının sayısı ile çeşitliliğinin de her yıl arttığı görülmektedir [5, 6].

Literatürde statik ve dinamik analiz olmak üzere iki temel Android kötücül yazılım analizi ve tespit tekniği bulunmaktadır. Statik analizde şüpheli bir uygulama herhangi bir sanal/gerçek cihazda çalıştırılmadan apk dosyası üzerinde analiz edilir. Kaynak kodları ve dosyaları incelenir. Kötücül uygulamanın çalıştırılmaması statik analiz yöntemlerini bazı durumlarda dinamik analize göre avantajlı kılmaktadır. Ancak diğer yandan saldırganlar kod karıştırma veya şifreleme gibi yöntemlerle statik analiz yöntemlerine ait tespit mekanizmalarını atlatabilmektedir. Bu durumda dinamik analiz yöntemi, kötücül aktivitenin tespit edilmesinde veya sınıflandırılmasında daha etkili olabilmektedir. Bu analiz yönteminin en büyük zorluklarından birisi ise uygulamadaki kötücül aktivitenin tetiklenmesidir. Uygulamaya gönderilecek doğru girdinin bulunması ve yeterli bir süre aralığında kötücül uygulamanın çalıştırılması dinamik analiz sonuçlarına büyük etkisi olacaktır. Diğer bir sorun ise kötücül uygulamanın sanal bir ortamda çalıştığını anlayarak zararlı davranışlarını gerçekleştirmediği durumlardır. Bu bağlamda, her iki yöntemin birlikte kullanıldığı hibrit analiz yöntemlerini kullanan Ec2 [7] ve UpDroid [8] gibi çalışmalar öne çıkmaktadır. Kötücül yazılım ve yazılım ailelerinin sınıflandırılmasına yönelik çalışmalarda kötücül uygulamalar çoğunlukla tek bir yönden (ör: kaynak dosyalar, kod analizi veya AndroidManifest.xml) incelenmiştir. Bu bölümde bahsedildiği üzere artan kötücül yazılım çeşitliliği, ilişkili yaklaşımların etkisini azaltmaktadır. Bu nedenle hibrit analiz yöntemi ve bu yöntemde kullanılan öznitelik kümesinin geniş olması önem arz etmektedir.

Bu tez çalışmasında, makine öğrenmesi algoritmaları yardımıyla Android kötücül yazılımlarının statik ve dinamik analizleri üzerinden kötücül yazılımların aile sınıflandırmaları yapılmıştır. Çıkarılan özniteliklerin etkileri 3 farklı Android kötücül yazılım veri kümesi üzerinde test edilerek değerlendirilmiştir. Bu çalışmada, benzer çalışmalarda yaygın olarak kullanılan özniteliklerin yanında yeni olarak ağ tabanlı öznitelikler, uygulamanın göstermiş olduğu ardışık ikili aktivite tekrarları da kullanılmıştır. Bu çalışmada, Android kötücül uygulamalarının analizleri sırasında benzer çalışmalarda

kullanılmayan yeni öznitelikler ile daha önceki çalışmalarda -özellikle UpDroid [8] çalışmasında- kullanılan özniteliklerin bir arada kullanılarak sonuçları değerlendirilmiştir. Bu anlamda hibrit analiz yaklaşımının kullanıldığı çalışmalar ile karşılaştırma yapılarak belirli veri kümeleri ve ölçüm değerlerinde daha yüksek doğruluk oranları elde edilmiştir. Yeni eklenen özniteliklerin etkileri farklı veri kümeleri için test edilmiştir. Yoğun ağ trafiği üreten Android kötücül uygulama kümelerinde farklı ağ tabanlı öznitelikler daha etkin iken yoğun aktivite gösteren kötücül yazılım ailelerinde ise ikili aktivite tekrarlarının daha ayrıştırıcı niteliğe sahip olduğu görülmüştür. Deneyler ve sonuçlar bölümünde, bu tez çalışmasında tanıtılan yeni özniteliklerin kötücül yazılım veri kümeleri üzerindeki etkileri görülebilmektedir.

Tez içeriği genel olarak şu şekildedir. Bölüm 2'de incelemesi yapılacak olan uygulamaların çalıştığı ortam olan Android işletim sisteminden ve Android uygulamaların genel yapısından bahsedilmiştir. Ayrıca, bu tez çalışması kapsamında kullanılan makine öğrenmesi algoritmaları tanıtılmaktadır. Bölüm 3'te, Android kötücül yazılımların aile sınıflandırması üzerine literatürde yapılan çalışmalar statik/dinamik/hibrit analiz olmak üzere 3 başlık altında incelenmiştir. Bölüm 4'te, çalışmada önerilen model ve kullanılan statik/dinamik öznitelikler açıklanmıştır. Bu çalışmada önerilen modele ait deneysel çalışmalar ve sonuçları, Bölüm 5'te ayrıntılı incelenmiştir. Son olarak, Bölüm 6'da tez sonuçları özetlenmiş, gelecekte yapılabilecek çalışmalara değinilmiştir.

2. ÖN BİLGİ

Bu bölümde, Android kötücül yazılımları analiz ederken kullanılan yöntem ve özniteliklerin daha iyi anlaşılabilmesi için Android işletim sistemi ve uygulamaları ile ilgili ön bilgiler verilecektir.

2.1. Android İşletim Sistemi Mimarisi

Android işletim sistemi ilk olarak 2005 yılında Google tarafından satın alınan Android Inc. firmasının Linux çekirdeğinin değiştirilmiş halini kullanarak geliştirmesi ile ortaya çıkmıştır. Mobil bir cihaz ile birlikte ise ilk olarak 2008 yılında piyasaya sürülmüştür [9].

Android işletim sisteminin temel mimarisi Şekil 1'de verilmiştir. Android işletim sistemi; uygulamalar, uygulama çerçevesi, kütüphaneler ve Linux çekirdeği olmak üzere 4 farklı katmandan meydana gelmektedir. Bu 4 katman, aşağıda ayrıntılı açıklanmıştır.

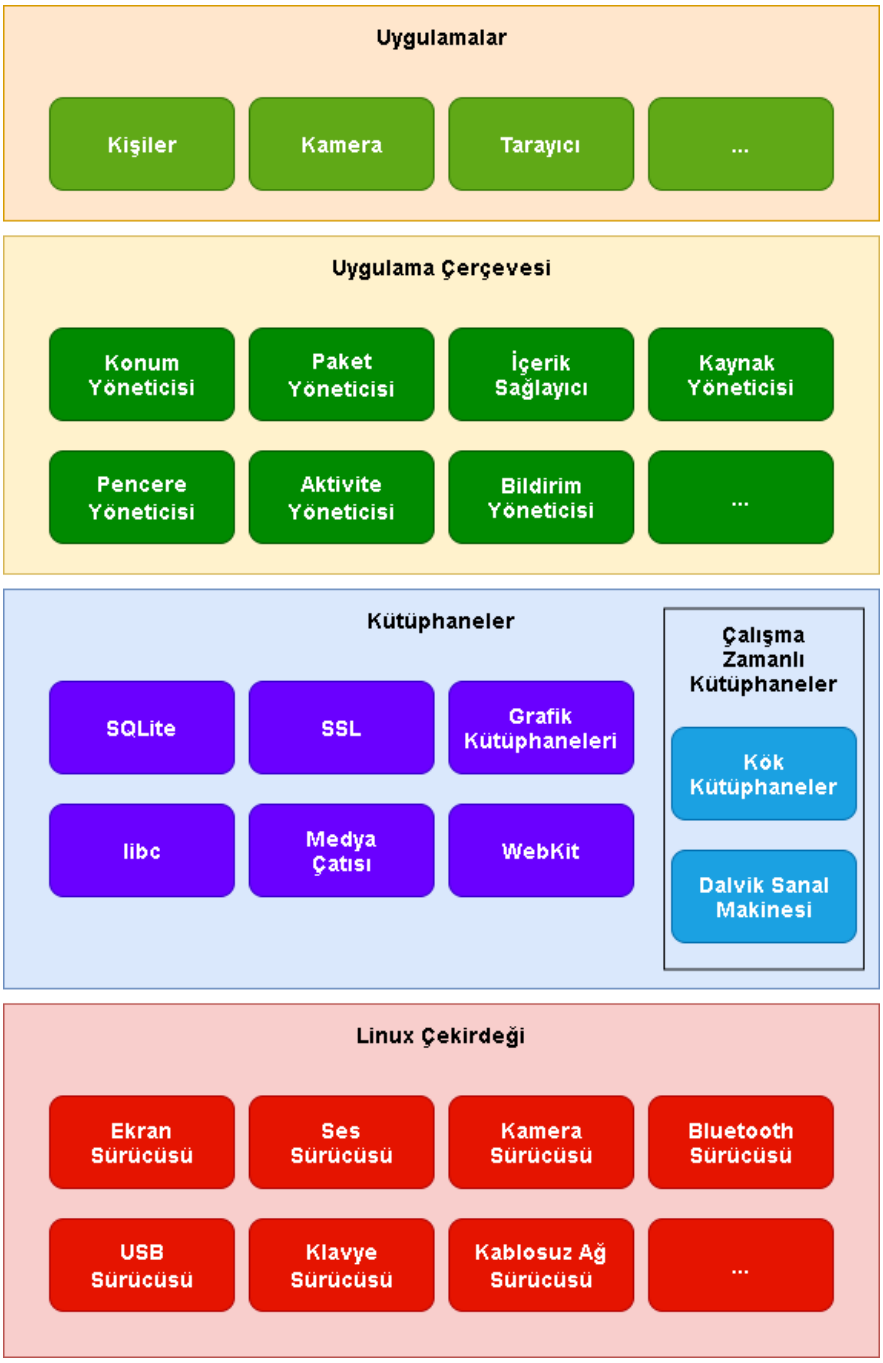
Linux çekirdeği katmanı**,** Google tarafından Linux çekirdeği üzerinde yapılan birkaç mimari değişiklik uygulanması sonrası Android işletim sisteminde en alt katmanda kullanılmaktadır. Linux Çekirdeği, işlem yönetimi, güç yönetimi, bellek/hafıza yönetimi ve kamera, tuş takımı, ekran vb. gibi donanım yönetimi gibi temel sistem işlevselliklerini sağlar. Ayrıca, Android işletim sistemi ile cihaz donanımları arasında kullanılan bir dizi donanım sürücülerinin bulunduğu katmandır.

Linux çekirdeğinin üzerinde kütüphaneler adı verilen başka bir katman bulunur. Bu katman, uygulamaların Dalvik Sanal Makineleri üzerinde birbirinden izole edilmiş ortamlarda çalışırken, kullandığı çalışma zamanlı kütüphaneleri barındırır. Android işletim sisteminin ve cihaz üzerindeki uygulamaların fonksiyonlarının çalışabilmesi veya daha verimli çalışabilmesi için TLS/SSL servislerini sağlayan OpenSSL, veritabanı hizmetini sağlayan SQLite ve görüntü hizmetini sağlayan OpenGL gibi farklı kütüphaneleri barındırır. Kütüphaneler, Android işletim sistemi için özel olarak oluşturulmuş genellikle Java tabanlı

kütüphaneleridir. Çalışma zamanlı kütüphaneler**,** Android Studio üzerinde bir Android uygulaması geliştirildiğinde, uygulama öncelikle .dex olarak bir ara baytkodu biçiminde derlenir. Uygulama daha sonra cihaza yüklendiğinde, Android Çalışma Zamanı (ART), baytkodunu cihaz işlemcisi tarafından çalıştırılabilmesi için gereken formata çevirir. Bu format, yürütülebilir biçim (.elf) olarak bilinir. Bu katman, Android işletim sistemi için özel olarak tasarlanmış ve optimize edilmiş bir tür Java Sanal Makinesi (JVM) olan Dalvik Sanal Makinesi (DVM) adlı önemli bir bileşeni barındırır. Dalvik Sanal Makinesi, her Android uygulamasının kendi Dalvik Sanal Makinesi içerisinde kendi işlem sürecinde çalışmasını sağlar. Android Çalışma Zamanı (ART) ise, Android uygulama geliştiricilerinin standart Java programlama dilini kullanarak Android uygulamaları geliştirmesine olanak tanıyan bir takım kök kütüphanelere erişim olanağı da sağlar.

Uygulama çerçevesi katmanı, Android işletim sistemi yığınındaki üçüncü katmandır. Android uygulamaları, uygulama çerçevesinde yer alan bileşenler ile doğrudan etkileşime girer. Uygulama Çerçevesi katmanı, Android uygulamalarının çalışırken kullandığı/yönettiği bileşenleri toplu olarak içeren tekrar kullanılabilir bir dizi hizmeti barındırır. Uygulama çerçevesi, kaynak yönetimi, sesli arama yönetimi, konum yöneticisi gibi Android cihazın temel işlevlerini yönetir.

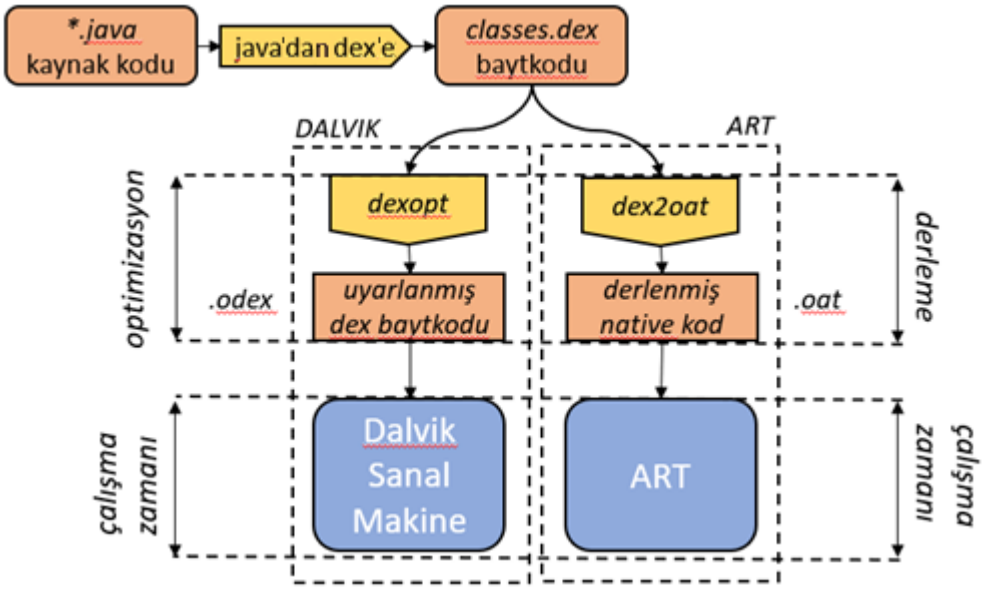
Uygulama katmanı, en üst katmanda yer alır ve uygulamaları içeri. Bunlar, hem Android işletim sistemi tarafından sağlanan ve varsayılan uygulamaları (ör: web tarayıcısı) hem de kullanıcı tarafından yüklenen üçüncü taraf uygulamaları içerir.



Şekil 1. Android İşletim Sistemi Mimarisi [9, 10].

Java programlama dili ile geliştirilen uygulamalar ilk olarak derlenir ve Java bayt koduna dönüştürülür. Ardından bellek alanının ve işlemci hızının daha verimli kullanılmasını amaçlayan .dex veya .odex formatında, Dalvik sanal makinesinin çalıştırabileceği dosyaya dönüştürülür. Dalvik sanal makine yapısı son olarak 2014 yılında Android 4.4.4 KitKat versiyonu ile kullanılmıştır. Android 4.4.4 KitKat versiyonu ile beraber uygulamaların çalışma zamanlı performansını gözle görülür şekilde iyileştiren ve yalnızca .dex dosyalarını

kullanan ART (Android Runtime) yapısı Android işletim sistemine dahil olmuştur. Android 4.4.4 KitKat sürümünde uygulamalar çalışma zamanlı olarak her iki yöntemde tercih edilebilir durumda sunulurken, Android 5.0 Lollipop sürümünden itibaren yalnızca ART yapısı kullanılmaya başlanmıştır [11]. Dalvik yapısındaki .dex dosyasında yer alan komutlar dexopt ile optimize edilerek .odex uzantılı dosyalarda saklanır ve çalıştırılmak üzere Dalvik Sanal Makinesine gönderilir. Bu durumdan farklı olarak ART yapısında .dex komutları dex2oat kullanılarak derlenir ve .oat uzantılı dosyaya dönüştürülür. Her iki yönteme ait akış şeması Şekil 2'de görülmektedir.



Şekil 2. Dalvik ve ART.

2.2. Android Uygulama Mimarisi

Uygulamalar, Android işletim sistemine .apk uzantılı ve sıkıştırılmış/paketlenmiş dosyaların çalıştırılması ile yüklenir. Bu dosya içerisinde uygulamanın çalışma zamanında kullandığı kaynaklar yer almaktadır. Bunlar kaynak kodlarının Dalvik komutları halini barındıran classes.dex dosyası, uygulamanın çalışma zamanlı kullandığı dosyalar (resim, belge, video vb.) ve AndroidManifest.xml dosyasıdır.

AndroidManifest.xml isimli dosyada, uygulama tarafından kullanılan izinler, servisler, aktiviteler ve yayın alıcıları tanımlanmıştır. İçeriğinde yer alan bazı bildirimler ile Android

işletim sisteminin, uygulamanın hangi bileşenlerini başlatabilme yetkisi olduğunu belirtir. Uygulamanın talep ettiği yazılımsal veya donanımsal izinlerin tamamı, AndroidManifest.xml dosyası içerisinde yer alır. Ek olarak, farklı uygulamaların, ana uygulama bileşenleriyle iletişim kurabilmesi için sahip olması gereken izinleri de içerir. API seviyesi 23 (Android 6.0)'ten önceki uygulamalarda izinler, uygulamanın yükleme anında istenirken; API seviye düzeyi 23 ve sonrasında, uygulama yüklendikten sonra, çalışma anında da izin talep edilebilmektedir. AndroidManifest.xml dosyası içerisinde yer alan örnek bir izin bilgisinin görünümü Şekil 3'teki gibidir.

```
<uses-permission android:name="android.permission.RECORD_AUDIO" />
```

Şekil 3. Kullanıcı İzni AndroidManifest.xml Görünümü.

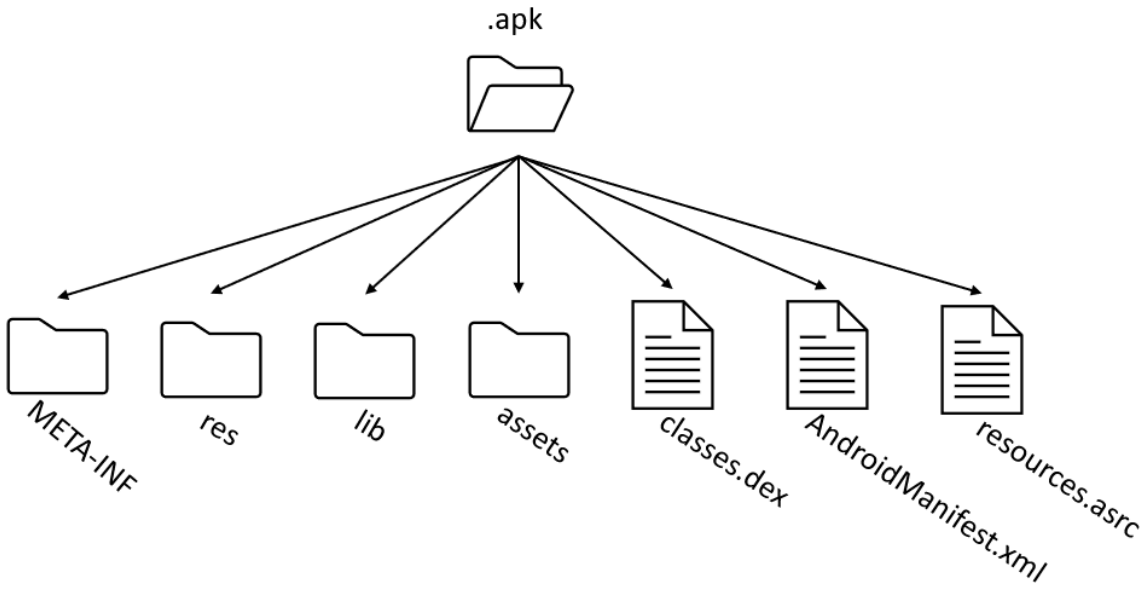
Uygulama izinleri yükleme zamanlı, çalışma zamanlı ve özel izinler olmak üzere 3 ana başlık altında incelenebilir. Yükleme zamanlı izinler ile uygulamanın kritik donanım veya verilere sınırlı bir erişim sağlanır. Uygulamayı Android işletim sistemine kuran kullanıcı tarafından yükleme sırasında ekranda beliren bir uyarı penceresi ile bu izinler aktif hale gelmektedir. Çalışma zamanlı izinler, tehlikeli izinler olarak da bilinmektedir. Bu izinler ise kullanıcının uygulamayı kullanması sırasında gerçekleştirdiği bir aktiviteye (fotoğraflar klasörüne erişim talebi vb.) bağlı olarak kişisel/kritik verilere ek erişim sağlar. Özel izinler ise platform bağımlı ve ürün üreticilerinin belirli erişimler için tanımladığı özel izinlerdir [12].

Android güvenlik mimarisinde önemli bir husus, varsayılan olarak hiçbir uygulamanın diğer uygulamaları, işletim sistemini veya kullanıcıyı olumsuz etkileyecek herhangi bir işlemi gerçekleştirme iznine sahip olmamasıdır. Buna, kullanıcının özel verilerini (kişiler veya e-postalar gibi) okuma veya yazma, başka bir uygulamanın dosyalarını okuma veya yazma, ağ erişimi gerçekleştirme, cihazı uyanık tutma vb. aktiviteler de dahildir.

Android işletim sistemi üzerinde çalışan bir Android uygulaması, izole sanal bir ortamda birbirinden farklı işletim sistemi kullanıcıları ile çalışır. Her bir uygulamaya, çalışma

zamanında Android işletim sistemi tarafından farklı bir kullanıcı kimliği (UserId) atanır. Böylelikle, uygulamalar arası veri ve kaynak erişimi daha güvenilir hale gelir [13]. Buradaki amaç uygulamaların çalışması sırasında güvenli bir ortam ile uygulamaların birbirlerini etkilememesini sağlamaktır. Birbirinden farklı kullanıcılar ile çalışan bu uygulamalar sanal ortamlar üzerinde de farklı yetkilere sahip olacak şekilde çalışmaktadırlar. Android uygulamalarında kurulum sırasında veya çalışma zamanlı olarak işlevini gerçekleştirebilmesi için kullanıcıdan izin taleplerinde bulunabilir. Talep edilen bu izinler, işletim sistemi tarafından uygulamanın sertifika bilgisine göre otomatik olarak izin verecek/vermeyecek şekilde veya kullanıcıya sorarak onay alınacak şekillerde ele alınabilir. Bu sebeple yukarıda da belirtildiği üzere bir uygulamanın kullanması gerektiği izinler o uygulamada AndroidManifest.xml dosyasında bildirilir.

Bütün Android uygulamaları için bir başka özellik de uygulamanın geliştiricisi tarafında bir sertifika ile imzalanmasıdır. Uygulama geliştirici, özel anahtarı kendisine ait bu imza ile tanımlanır. İlgili sertifikanın amacı, geliştiricileri ayırt etmek ve Android uygulamaların imza düzeyinde kontrolünün sağlanmasıdır.



Şekil 4. Android Uygulama Paket İçeriği.

Şekil 4'te bir Android uygulama paketinin içeriği şematik olarak gösterilmiştir. META-INF klasörü, .apk dosyasına ilişkin özel imza bilgisini içermektedir. lib dizini, eğer var ise, uygulamaya has derlenmiş yerel Android kütüphanelerini içermektedir. res klasörü, uygulamaya ait olan her türlü kaynak dosyalarını (resimler, dil dosyaları, vb. gibi) içermektedir. classes.dex dosyası ise derlenmiş uygulama kodlarını barındırmaktadır. Android uygulamaların kullanması gereken kaynak tabloları resources.asrc dosyasında barınmaktadır. Bu dosya uygulamanın yükleme ve çalışma sırasında kullanacağı derlenmiş kaynaklara özgü isimleri, özellikleri, tanımları, dizileri tablo formatında saklayan bir bileşendir. Uygulamanın talep ettiği izinler, aktiviteler ve servisler gibi bilgiler ile uygulamanın üst verileri bilgileri AndroidManifest.xml dosyası içerisinde belirtilmiştir.

2.3. Makine Öğrenmesi

Makine öğrenimi, bir sistemin öğrenme şeklini taklit etmek için verilerin ve algoritmaların birlikte kullanımına odaklanan bilgisayar bilimlerinde kullanılan bir yöntemdir. Yapılan çalışmada makine öğrenmesinden, örnekler üzerinden yapılan analizler ile elde edilen verilere göre sınıflandırma algoritmaları kullanılarak faydalanılmıştır. Veri, sınıflandırma işlevlerini gerçekleştirmek için daha sonra kullanılabilecek bir tür model oluşturmak için kullanılır. Her yeni model için yeni bir öğrenme gerçekleştirilir.

Makine öğrenimi algoritmaları, denetimli öğrenme (supervised learning) ve denetimsiz öğrenme (unsupervised learning) olmak üzere iki ana başlık altında incelenebilir. Gözetimli öğrenme algoritmalarında herhangi bir veri kümesi için girdi ve çıktı verisi söz konusudur. Gözetimli öğrenme başlığı altında değerlendirilen algoritmalar girdiler ve çıktılar arasında anlamlı bir ilişkilendirme modeli oluşturmayı amaçlar. Gözetimsiz öğrenme başlığı altında yer alan algoritmalarda ise genellikle veri kümesinin gruplandırılması (clustering) hedeflenir.

Gözetimli öğrenmede, veri kümesi genellikle eğitim ve test olmak üzere iki alt gruba ayrılmaktadır. İlk grupta modeli eğitebilmek için kullanılacak eğitim alt veri kümesidir. Modelin eğitilmesi sırasında eğitim verilerinin uygun şekilde hazırlanması (rastgele seçim vb.) eğitimi olumsuz etkileyebilecek doğru olmayan sonuçların ortaya çıkmaması açısından

önem arz etmektedir. İkincisi, ortaya konulan modelin test edilmesi ve modelin sonuçlarının değerlendirerek performansını iyileştirebilmek için kullanılan test alt kümesidir. İlgili veri kümeleri, ana veri kümesinden elde edilirken farklı yöntemler kullanılabilmektedir. İlk yaklaşım verinin belirlenmiş yüzdelik bir diliminin eğitim verisi olarak kalan veri kümesinin ise test verisi olarak tanımlanmasını kapsamaktadır (ör. %67 eğitim-%33 test). İlgili oranlar belirlenirken veri bağımlı olarak tercih edilebilmektedir.

Veri kümesinin eğitim-test alt kümeleri olarak bölünmesinde kullanılan diğer bir yöntem ise k-katmanlı çapraz doğrulamadır (k-fold cross validation). Bu teknik, veri setinin rastgele olarak k adet eşit büyüklükteki gruplara bölünmesini ifade eder. Her bir eğitim-test işleminin tekrarında k adet bölümden k-1 kadarı eğitim ve son kalan veri kümesi test kümesi olarak belirlenir. Bu sebeple ele alınan model k defa ana veri üzerinden belirlenmiş farklı alt veri kümeleri ile eğitilerek ve k defa farklı test kümeleri test edilir. Eğitim ve test kümesinin her seferinde değiştirilmesi modelin daha tutarlı değerlendirilmesinde önemlidir. Şekil 5'te k değerinin 5 olarak kabul edildiği bir senaryo görülebilmektedir. 5-katmanlı çapraz doğrulamada her seferinde birbirinden farklı eğitim ve test kümeleri sırasıyla seçilmiş ve bu işlem 5 kez tekrar edilmiştir. 5-katmanlı çapraz doğrulama için verinin nasıl kullanıldığı ile ilgili örnek bir gösterim Şekil 5'te bulunmaktadır.

Şekil 5. 5-Katmalı Çapraz Doğrulama Örnek Gösterim.

Android kötücül yazılımların her yıl hızla artması ve şekil değiştirmesi ile her bir uygulamanın manuel olarak analizinin yapılması ve kötücül yazılım aile bilgisinin çıkarılması gün geçtikçe zorlaşmaktadır. Bu noktada Android kötücül yazılımların aile

sınıflandırmasında makine öğrenmesinin kullanılması ihtiyaç haline gelmiştir. Yapılan çalışmalarda ailesi bilinen kötücül uygulamaların öznitelik vektörleri, uygulamaların bileşenleri incelenerek çıkarılmış ve önerilen modeller ilgili öznitelik vektörleri ile eğitilmiştir. Makine öğrenmesinde gözetimli öğrenme algoritmalarının bu anlamda kullanılarak kötücül yazılım aile sınıflandırması çalışmaları 5-6 yıl öncesine dayanmaktadır [14, 15, 16]. Çalışmalarda uygulamaların özniteliklerinin çıkarılmasında önerilen yöntemlerin farklılıklarına bağlı olarak makine öğrenmesinde kullanılan algoritmalar da farklılık göstermektedir.

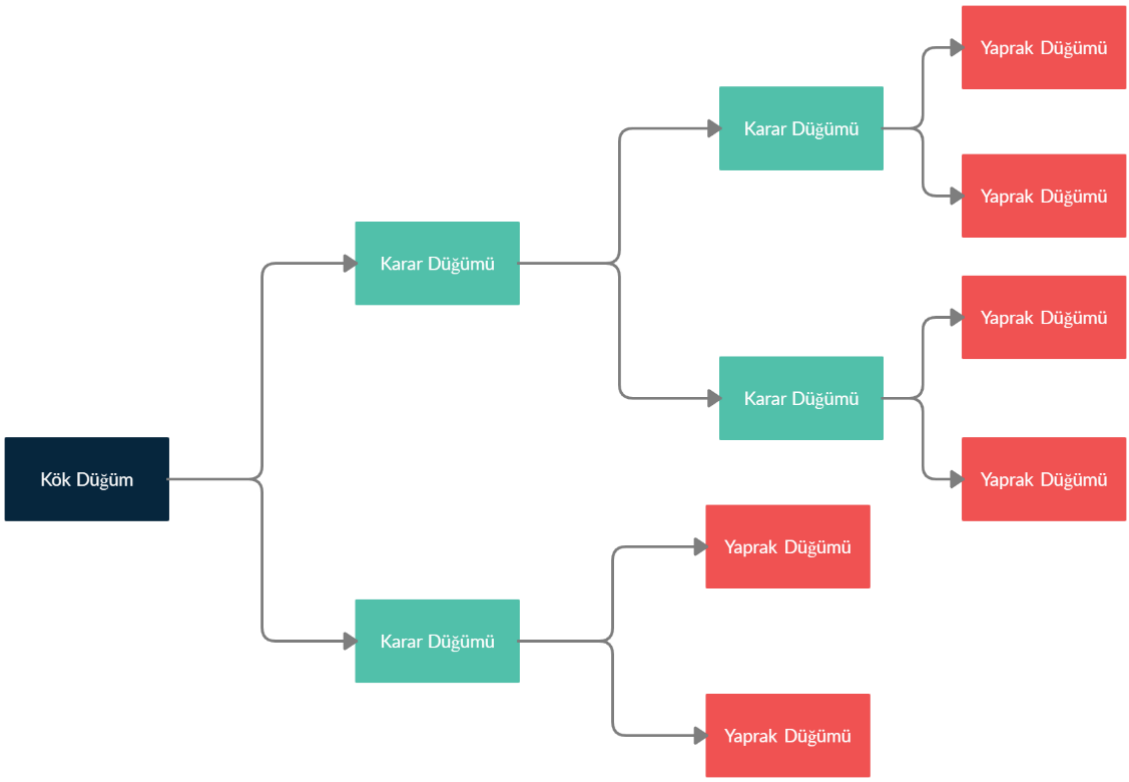
Literatürde Rastgele Orman (RF), Karar Ağaçları (DT), Naive Bayes (NB), En Yakın K Komşu (KNN), Destek Vektör Makinesi (SVM) algoritmaları Android kötücül yazılımlarının ailelerine sınıflandırılmasında sıklıkla kullanılmış olan algoritmalardır. RF ve DT algoritmaları karar ağaçları sınıfı altında değerlendirilmektedir. Karar ağacı öğrenmelerinde, veri kümelerinden alınan bilgi, ağaçlı bir yapıya aktarılır ve modellenir. Naive Bayes algoritmasında ise koşullu olasılıksal hesaplamalar üzerinden model oluşturulmaktadır. Karar ağacında tanımlanan her bir düğüm, modeldeki girdileri göstermektedir. Ağaçlı karar verme bir karar ve bir sonuç silsilesi ile devam etmesi sebebiyle diğer algoritmalara göre daha anlaşılır bir yapıdadır. Bu bağlamda bu tez çalışmasında, RF, DT, KNN, SVM algoritmaları kullanılarak değerlendirmeler ve karşılaştırmalar yapılmıştır. Aşağıda her bir algoritma hakkında genel bilgi verilmiştir.

2.3.1. Karar Ağaçları

Karar Ağacı algoritması, denetimli öğrenme (supervised learning) algoritmaları ailesinde yer almaktadır. Karar Ağacı algoritmasının kullanım amacı, eğitim verilerinden çıkarılan basit karar kurallarını öğrenerek hedef değişkenin sınıfını veya değerini tahmin etmek için kullanılabilecek bir eğitim modeli oluşturmaktır.

Karar Ağacı algoritmasında, bir test verisi için sınıflandırmada karar ağacının kök (root) düğümünden başlanır. Şekil 6'da görülebileceği üzere Kök düğüm tüm veriyi temsil eder ve kök düğüm ile birlikte her karar düğümü o anki veri kümesinin ayırt edici özelliklerine göre iki veya daha fazla homojen kümeye ayrılır. Yaprak (leaf) düğüm, kendisinden sonra

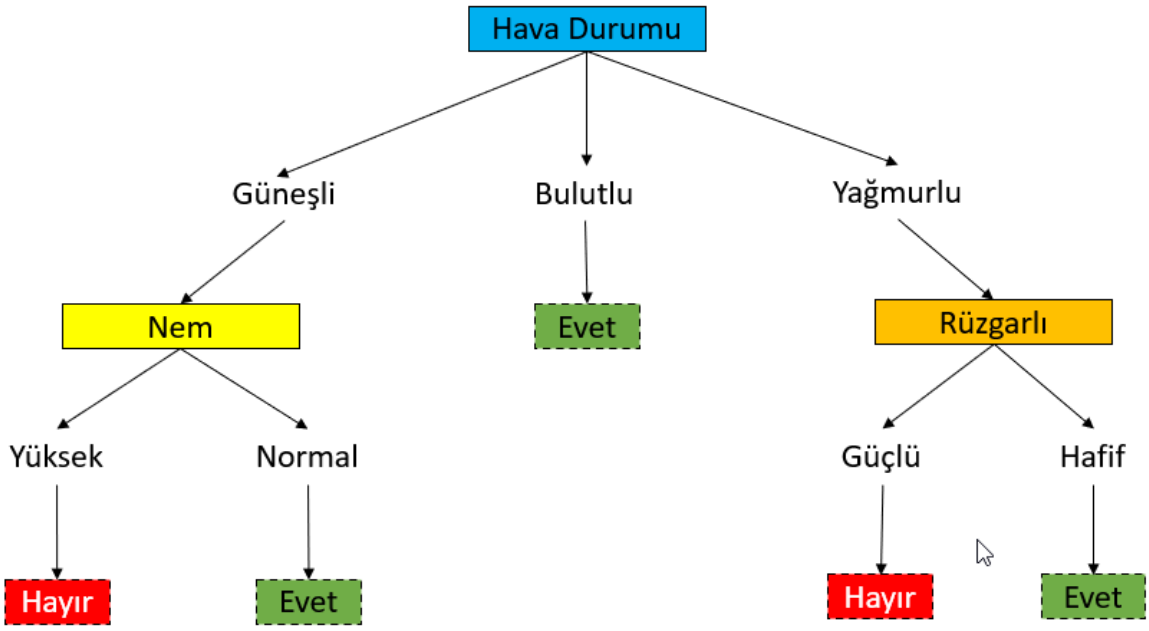
herhangi bir karar adımı olmayan ve iki veya daha fazla homojen kümeye ayrılmayan son düğümlerdir. Kök düğüm ile yaprak düğümleri arasında yer alan düğümlere ise karar düğümü denilmektedir. Testte, her bir örneklem öznitelik vektörüne göre ağaçta bir yolu izler ve ilgili yolun çıkış noktası sınıfı belirler. Şekil 6'da örnek bir karar ağacı görünümü yer almaktadır.



Şekil 6. Karar Ağacı Görüntüsü.

C4.5 ya da J.48, literatürde en bilinen ve kullanılan karar ağacı tabanlı algoritmalardan iki tanesidir. Karar ağaçları, bir veri kümesini farklı koşullara göre sınıflandırmanın yollarını tanımlayan algoritmik bir yaklaşımla oluşturulur. Denetimli öğrenme için yaygın kullanılan yöntemlerden biridir. Eldeki veri kümesinde ayırt ediciliği en yüksek olan öznitelik seçilir ve bu öznitelik baz alınarak veri kümesi sınıflandırılır. Bu işlem elde edilen her bir alt veri kümesinde yaprak (leaf) düğümlere ulaşana kadar tekrarlanarak devam eder ve bir karar ağacı oluşturulur. Karar ağaçları, örnek bir veriye ait özniteliğe göre ayrımların ve birleşimlerin oluşturduğu yapıyı temsil eder. Ağaçtaki kök düğüm ve yaprak düğüm arasındaki her bir karar düğümü, örneklemlere ait bazı öznitelikler için sorulan ayırt edici

soruları temsil etmektedir. Örneğin dinamik analizi yapılmış bir uygulamanın oluşturduğu IP paket trafiğinin büyüklüğü 1MB'tan büyük olup olmadığı oluşturulan ağaç için bir düğümü veya Android bir uygulamanın statik analizi sonucunda READ_CONTACTS iznini kullanıp kullanmadığının (1 ya da 0) farklı bir girdiyi ifade etmesi sebebiyle bir başka düğümü temsil edebilir. Bir karar düğümünden ayrılan her bir dal, sorulan soruya verilen olası cevaplardan birine karşılık gelir. Her bir yaprak düğümü ise sınıflandırma sonucunu temsil eder.



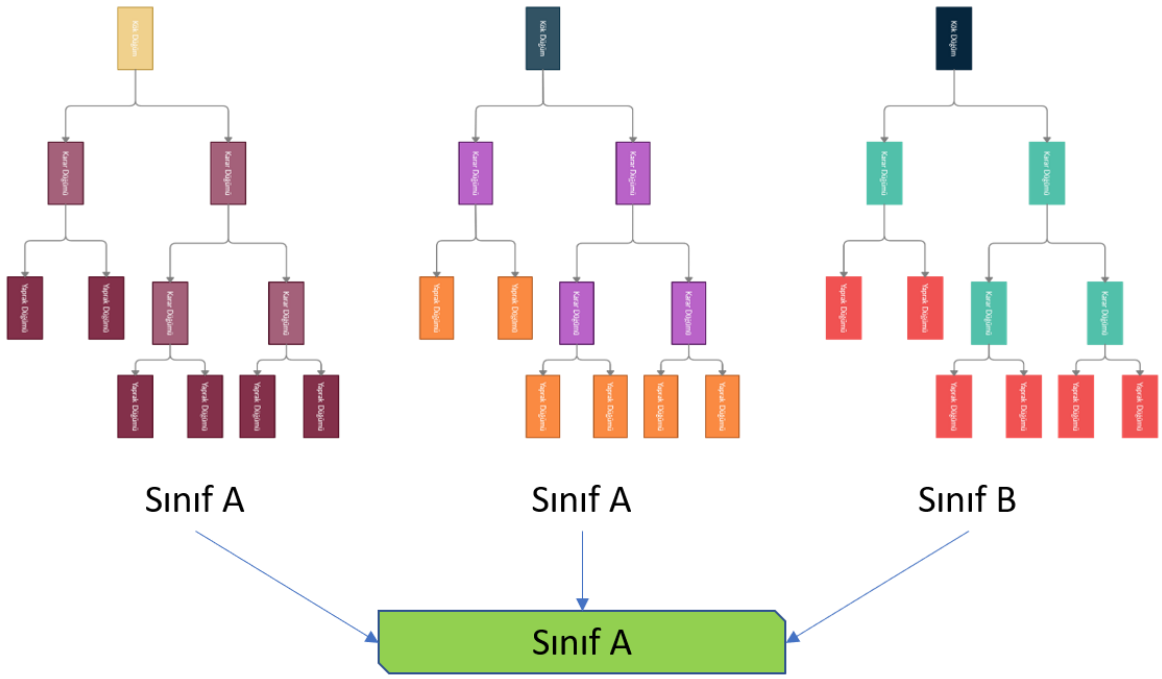
Şekil 7. Örnek bir Karar Ağacı Yapısı [17].

Yukarıdaki şekilde karar ağacı algoritmalarında sıklıkla verilen örnek olan *"Dışarıda futbol oynanabilir mi?"* sorusuna cevap bulabilmek için oluşturulan bir karar ağacı örneği görülebilir. Bu örnekte *"Hava Durumu=Güneşli, Nem=Yüksek, Rüzgar=Kuvvetli"* olumsuz bir örnek olarak sınıflandırılır. Sınıflandırmalar örneklemlere ait öznitelikler ve o özniteliklerin değerlerine göre yapılır.

2.3.2. **Rastgele Orman**

Rastgele orman, 2.3.1 başlığı altında belirtilen birbirinden farklı karar ağacı yapılarından birden fazla kullanılarak veri kümesinin sınıflandırılmasını amaçlar. Temelinde yatan

düşünce çok sayıda birbiri ile ilişkili olmayan karar ağaçlarının birlikte faaliyetinin, yalnızca tek bir karar ağacının faaliyetinden daha verimli olabileceği üzerinedir. Rastgele orman algoritmasında yer alan her bir ağaç, bir sınıf tahmini ile sonuçlanır ve eğitim-test işlemleri sonucunda ağırlıklı olarak ortaya çıkan sınıf, modelin nihai tahmini olarak belirlenir. Şekil 8'de birçok karar ağacını bir arada barındıran bir Rastgele Orman algoritmasına örnek gösterilmektedir.



Şekil 8. RF (Random Forest) Algoritmasının İşleyişine Örnek.

2.3.3. **Naive Bayes**

Naive Bayes, bir istatiksel sınıflandırma algoritmasıdır. Temelde Bayes teoremine dayanmaktadır [18]. Olasılıksal ilişkilendirmeler üzerinden verilerin sınıflandırılmasını amaçlamaktadır. Koşullu olasılıklar (Conditional Probabilitiy) üzerinden modele girdi olarak gönderilen verinin hangi sınıfa ait olduğunu tespit eder.

Bayes teoremi, koşullu olasılıkları hesaplamak için kullanılan matematiksel bir formüldür. Koşullu olasılık (conditional probability), başka bir olayın gerçekleştiği varsayılarak göz önüne alındığında, bir olayın meydana gelme olasılığının hesaplanmasıdır. Bayes teoremine

ait matematiksel formül aşağıdaki gibidir. Naive Bayes algoritmasının temelinde her bir özniteliğin sonuca eşit ve birbirinden bağımsız etkisi olduğu varsayımı yatmaktadır. Aşağıda Bayes teoreminin formülü ve formülde kullanılan değerlere ilişkin açıklamalar verilmiştir.

- P(B|A): A olayının gerçekleştiği varsayıldığı durum için B olayının gerçekleşme olasılığını ifade eder.
- P(A|B): B olayının gerçekleştiği varsayıldığı durum için A olayının gerçekleşme olasılığını ifade eder.
- P(A): A olayının gerçekleşme olasılığını ifade eder.
- P(B): B olayının gerçekleşme olasılığını ifade eder.

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)}$$

Veri kümesi, öznitelik matrisi ve yanıt vektörü olmak üzere iki bölüme ayrılmıştır [18]. Çizelge 1'de Naive Bayes algoritmasının işleyişini göstermek için küçük bir veri kümesi gösterilmektedir. Bu çizelgede;

- Öznitelik matrisini, "Hava Durumu", "Nem" ve "Rüzgarlı" değerleri oluşturmaktadır.
- Yanıt vektörünü ise öznitelik matrisinin her bir değeri için tanımlanan "Tenis Oynama" bilgisi oluşturur. Bir diğer ifade ile sınıf değişkeni adı "Tenis Oynama"dır.

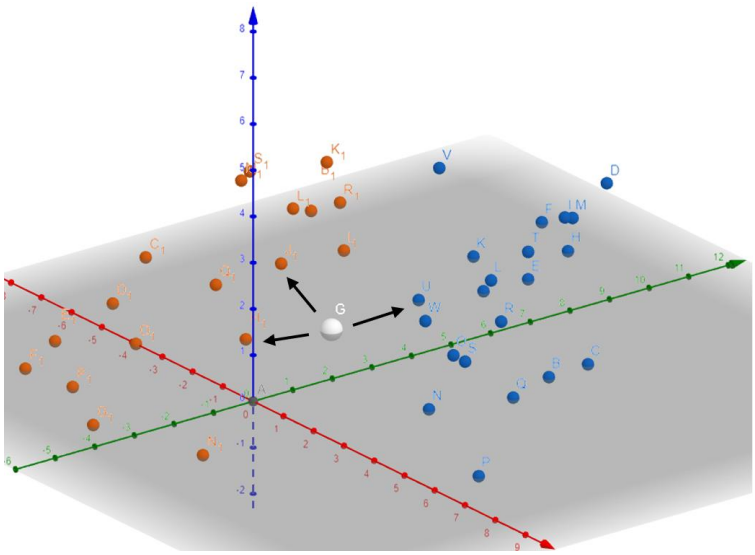
Çizelge 1. Naive Bayes Algoritmasının için Örnek bir Veri Kümesi.

Hava Durumu	Nem	Rüzgârlı	Tenis Oynama
Yağmurlu	Yüksek	Hayır	Hayır
Yağmurlu	Yüksek	Evet	Hayır
Bulutlu	Yüksek	Hayır	Evet
Güneşli	Yüksek	Hayır	Evet
Güneşli	Normal	Hayır	Evet
Güneşli	Normal	Evet	Hayır

Bulutlu	Normal	Evet	Evet
Yağmurlu	Yüksek	Hayır	Hayır
Yağmurlu	Normal	Hayır	Evet
Güneşli	Normal	Hayır	Evet
Yağmurlu	Normal	Evet	Evet
Bulutlu	Yüksek	Evet	Evet
Bulutlu	Normal	Hayır	Evet
Güneşli	Yüksek	Evet	Hayır

2.3.4. **En Yakın K Komşu**

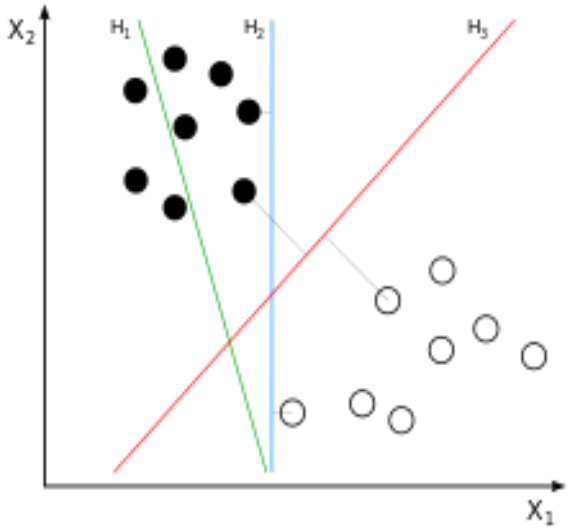
Bu algoritma, her bir örneklemi öznitelik vektörlerine göre konumlandırmaktadır. Konumlanan eğitim verileri üzerinden sınıflandırma işlemi gerçekleştirilir. Geliştirilen modele girdi olarak gelen her yeni veri bu vektör uzayında konumlandırılır ve en yakın k adet komşusuna olan uzaklığı Öklid Mesafesi ile hesaplanır. Burada tanımlanan k değeri 2 farklı sınıf ile eşit sayıda eşleşmenin önüne geçmek amacıyla tek sayı olarak tercih edilmelidir. Aşağıdaki şekilde k değerinin 3 olması koşulunda modele girdi olarak gelen G verisinin sınıflandırılmasında KNN algoritmasının çalışma şekli ifade edilmiştir.



Şekil 9. KNN Algoritmasının Vektörel Düzlemde Görünümü.

2.3.5. **Destek Vektör Makinesi**

Vektör uzayında yer alan iki farklı sınıfa ait iki farklı nokta arasında karar çizgisini tanımlayan makine öğrenmesi algoritmasıdır. Karar çizgisinin tanımlanmasında Doğrusal İlkeleme (Linear Regression) hesaplama yöntemleri kullanılır. Eğitilmiş bir modele girdi olarak gelen bir verinin sınıflandırılması bu karar çizgisi baz alınarak belirlenir. Şekil 10'da herhangi iki sınıf için çizilen 3 farklı karar çizgisi görülmektedir [19].



Şekil 10. SVM Görünümü [19].

3. İLİŞKİLİ ÇALIŞMALAR

Kötücül yazılımların bulaştıkları sistemde bıraktıkları etki, yayıldıkları sektör (finans, banka, enerji vb.) gibi benzeri sebeplerle güvenlik araştırmacıları tarafından kötücül yazılım ailelerine (ör: AnserverBot, BeanBot, BaseBridge vb.) sınıflandırılmıştır. Her geçen gün, kötücül yazılımların tespit yöntemlerini atlatmak için kullanılan yöntemlerin farklılaştığı ve giderek geliştiği görülmektedir. Bu durum bir uygulamanın kötücül olup olmadığının tespitini zorlaştırdığı gibi, halihazırda tespit edilmiş olan bir zararlı yazılımın da hangi Android kötücül yazılım ailesine dahil olduğunun tespitini de zorlaştırmaktadır. Kötücül yazılım geliştiricileri tarafından kullanılan yöntemlerin analiz edilmesi ve ilgili yazılımların sınıflandırılması için yapılan analiz yöntemleri, statik, dinamik ve her iki yönteminde kullanıldığı hibrit analiz olmak üzere üç gruba ayrılmaktadır. Her bir analiz yönteminin, analiz ortamının verimli kullanımı ve yüksek doğruluk oranlarının elde edilebilmesi açısından avantajları ve dezavantajları bulunmaktadır. Her iki analizin beraber kullanıldığı çok az çalışma bulunmaktadır. Bu çalışmaların başarımları, öznitelik tercihleri, dinamik analiz için kullandıkları yöntemler ve süreleri, kullandıkları makine öğrenmesi algoritmaları ve benzer parametrelerden etkilenmektedir. Etkili özniteliklerin bulunması, daha uzun dinamik analiz yapılması vb. gibi yaklaşımların bu tür çalışmaları geliştireceği düşünülmektedir.

Bu bölümde, özellikle Android aile sınıflandırması için literatürde önerilen yöntemler incelenecektir. Ayrıca, Android kötücül yazılımları tespit eden, aile sınıflandırması yapmayan makine öğrenmesi tabanlı çalışmalardan da bahsedilecektir.

Android kötücül yazılım ailelerinden bahsedilen bilinen ilk çalışmalardan biri birtakım Android kötücül yazılım ailelerinin incelendiği Malgenome adlı [20] çalışmadır. Bu çalışmada, yayınlanan blog içerikleri, tehdit raporları ve güvenlik uyarıları dikkate alınarak 2010 Ağustos ve 2011 Ekim tarihleri arasında 49 kötücül yazılım ailesine ait 1260 adet tespit edilmiş kötücül uygulama toplanmıştır. Toplanan uygulamaların oluşturulduğu veri kümesi Malgenome veri kümesi [21] olarak adlandırılmış ve paylaşılmıştır. Yapılan analizlerin sonucunda kötücül yazılım ailelerinin karakteristik özellikleri öne çıkarılmıştır. Temelde

hedef sisteme nasıl yüklendiği, nasıl aktif hale geldiği ve hedef cihazda gerçekleştirdiği bir dizi işlemler analiz edilmiştir. Uygulamanın hedef cihaza yüklenmesi, *repackaging*, *update attack*, *drive-by download* ve *standalone* olmak üzere 4 farklı yöntem altında incelemiştir. *Repackaging* yöntemi; var olan popüler uygulamaların saldırganlar tarafından apktool [22] benzeri araçlar ile tersine mühendislik yöntemleri ile uygulama kaynak koduna erişilerek, ek kötücül kodların uygulamaya eklenmesi yoluyla gerçekleştirildiği şeklinde tanımlamıştır. Toplanan kötücül uygulamaların %86'sının (1083 adedi) *repackaging* yöntemini kullandığı tespit edilmiştir. *Repackaging* yöntemi kullanılarak oluşturulan zararlı uygulamalar genellikle popüler uygulamaların zararlı sürümleridir. Saldırganlar yaygın kullanılan bir Android uygulamasını indirerek tersine mühendislik yöntemi ile kaynak kodunu elde eder. Ardından kötücül kodları (genellikle kötü niyetli) uygulamaya dahil ederek yeniden paketleyip yayınlar. *Update attack* zararlı kodların kötücül yazılım içerisinde barınmadığı ve belirli tetikleme yöntemlerinin sonunda ağ üzerinden cihaza indirildiği yöntemdir. *Drive-by download* kullanıcılara uygulama içi gösterilen reklamlar, bağlantılar üzerinden zararlı kodların Android cihaza indirilmesini sağlayan yöntemdir. Geriye kalan uygulamalar ise *standalone* olarak sınıflandırılmıştır. Bu uygulamalar casus yazılım (spyware), kimlik bilgilerini çalmaya yönelik işlevi olmayan, gerçeği taklit eden uygulamalar ve hedef cihazda kök yetkilerine sahip olabilmek için sömürü kodu içeren kötücül uygulamalar olmak üzere farklı grup altında incelenmiştir. Malgenome [20] çalışmasında incelenen bu kötücül yazılımlar, kuruldukları cihazda aktif hale gelirken, kullandıkları Android sistem olaylarına (BOOT, SMS, USB vb.) göre incelenmiştir. Kötücül yazılımların ulaşmak istedikleri hedef doğrultusunda cihaz üzerinde BOOT_COMPLETED, SMS_RECEIVED, PHONE_STATE gibi sistem olaylarını dinleyerek aktif hale geldikleri görülmektedir. Aktivasyonun ardından kötücül yazılımların, cihazda Android işletim sisteminin bileşenlerinde var olan açıklıklara ait sömürü kodlarını kullanarak hak yükseltmeye çalıştığı, %93'ünün (1172 adet) uzak sunucudan komutlar alarak cihazı bot haline getirdiği, cihazdaki SMS trafiğini izleme, değiştirme yoluyla finansal kayıplara yahut kişisel bilgilerin ifşasına yol açtığı görülmüştür. Bu bilgiler ışığında bazı Android kötücül yazılım ailelerinin hedef sistemde gerçekleştirdiği işlemler açısından benzerlikler gösterdiği tespit edilmiştir. Örneğin; BaseBridge ve DroidKungFuUpdate kötücül yazılım ailelerinin *update attack* yöntemi ile zararlı kodları sonradan sisteme yüklediği görülmüştür. BaseBridge ailesi, uygulama kaynak dosyaları içerisinde ikinci uygulamayı gizlerken; DroidKungFuUpdate ailesinindeki uygulamaların ağ üzerinden kötücül kodları hedef cihaza yüklenme sonrasında indirdiği görülmüştür. Çalışma

sonunda toplanan 1260 adet Android kötücül yazılım AVG Antivirus Free, Lookout Security&Antivirus, Norton Mobile Security Lite ve Trend Micro Mobile Security Personal Edition mobil anti-virüs veritabanlarında taranmış ve en yüksek olarak %79,6'lık (1003 adet) tespit oranı elde edilmiştir [20]. Ek olarak, birden çok antivirus tespit yazılımının herhangi bir kötücül yazılımın ailesi konusunda çoğunlukla hemfikir olmadığı görülmüştür.

Literatürde Android kötücül yazılımların aile sınıflandırmasında var olan anti-virüs çözümlerinin yeterli düzeyde güvenilir olmadığı farklı çalışmalar ile gösterilmiştir [23, 24, 25]. Bu anlamda bir kötücül yazılımın aile bilgisinin doğru tespit edilebilmesi veya sınıflandırılabilmesi için literatürde makine öğrenmesi algoritmaları kullanılmaya başlanmıştır.

İlişkili çalışmalar hakkında çalışma adı-çalışma yılı-analiz yöntemi başlıkları altında genel bir gösterim Çizelge 2'de görülebilmektedir. Aile bilgisinin tespitinde makine öğrenmesi algoritmalarının kullanıldığı Çizelge 2'deki çalışmalar, analiz yöntemlerine göre 3.1, 3.2 ve 3.3 numaralı başlıklar altında daha ayrıntılı incelenmektedir.

Çizelge 2. İlişkili Çalışmalara Genel Bakış.

Çalışma Adı	Çalışma Yılı	Analiz Yöntemi
DENDROID [26]	2013	Statik Analiz
DROIDSIFT [27]	2014	Statik Analiz
DROIDLEGACY [23]	2014	Statik Analiz
DROIDMINER [35]	2014	Statik Analiz
REVEALDROID [24]	2015	Statik Analiz
DROIDSCRIBE [49]	2016	Dinamik Analiz
DROIDCLASSIFIER [54]	2016	Dinamik Analiz
DROIDSIEVE [40]	2017	Statik Analiz
EC2 [7]	2017	Hibrit Analiz
FALDROID [25]	2018	Statik Analiz

UPDROID [8]	2018	Hibrit Analiz
ANDMFC [42]	2019	Statik Analiz
ANDROID MALWARE FAMILY CLASSIFICATION USING IMAGES FROM DEX FILES [44]	2019	Statik Analiz
ANDRODFA [53]	2020	Dinamik Analiz
MVIIDroid [46]	2020	Statik Analiz
IMAGE-BASED FAMILY CLASSIFICATION [45]	2020	Statik Analiz

3.1. Statik Analiz Yaklaşımları

[29] çalışmasında nihai sonuçlarda FP değerlerinin düşürülmesi ve doğru tahmin oranını arttırmak amacıyla birden fazla makine öğrenmesi algoritması (3 farklı ya da 5 farklı) kullanılmış ve sonuçlar değerlendirilmiştir. Bir uygulamanın kötücül olup olmadığına yönelik her bir makine öğrenmesi algoritmasından ayrı ayrı sonuçlar elde edilmiş ve değerlendirme yapılmıştır.

Statik analiz yaklaşımı, Android uygulamalarının çalıştırılmadan incelenmesine dayanır. Android uygulamalar, Linux tabanlı olan Android işletim sistemine *.apk* uzantılı kurulum dosyaları ile kurulmaktadır. APK (Application Package Kit) [30], uygulamaya ait kaynak dosyaları ve kodları barındıran bir arşiv dosyasıdır. Bu arşiv dosyası apktool [22] gibi çözücü araçların kullanılması ile açılabilmektedir.

Statik analiz yönteminin Android kötücül yazılımlarının tespit edilmesinde kullanıldığı örnekler ve çalışmalar bulunmaktadır [31]. Statik analiz yönteminin kullanıldığı çalışmalarda, aynı kötücül yazılım ailesine ait uygulamaların statik özniteliklerinin benzer olması gerektiği yaklaşımından yola çıkılmıştır. Statik analiz yaklaşımının kötücül yazılımların tespitine yönelik olarak kullanımı oldukça yaygındır. Yaygın olmasının en önemli sebepleri kötücül yazılımın herhangi bir cihaza bulaşmadan incelenebilmesi ve hızlı sonuç alınabilir olması olarak gösterilebilir. Ayrıca analiz maliyetinin düşük olması ve düşük kaynak tüketimi de avantajlar arasında gösterilebilir. Ancak statik analiz yaklaşımı, uygulamaların çalışma anında gerçekleştirdiği yerel (native) kod çalıştırma, ağ trafiğinin

çıkarılması vb. işlemlerin tümünü yansıtamamaktadır. Dahası, kötücül yazılım geliştiricilerinin kod içerisinde yer alan değişken, sınıf ve metot isimlerini ileri karıştırma ve bulanıklaştırma yöntemleri ile gizlemesi statik analiz yaklaşımının etkisini azaltabilmektedir. Statik analizin gerçekleştirildiği APK arşivi içerisinde yer alan uygulama bileşenleri ve anlamları Şekil 4'te gösterilmiştir.

Statik analiz yaklaşımının Android kötücül yazılım ailelerinin incelenmesi amacıyla kullanıldığı Malgenome [20] çalışması, topladığı 1260 adet kötücül yazılımı [21] analiz etmiştir. Uygulamaların kaynak kodlarında yer alan metotlar, AndroidManifest.xml dosyası içerisinde belirtilen uygulama izinleri ve yayın alıcıları incelenmiştir. Çalışmada kötücül uygulamaların kötücül olmayan uygulamalara göre READ_SMS, WRITE_SMS, RECEIVE_SMS ve SEND_SMS olmak üzere SMS ilintili izinleri daha sık talep ettiği değerlendirilmiştir [20]. Bu çıkarımı destekler nitelikte diğer bir çalışma olan Drebin çalışmasında [28], SMS mesajlarının gönderilmesi/alınması için kullanılan API çağrıları bir öznitelik olarak değerlendirilmiştir. Android kötücül yazılım tespitine yönelik makine öğrenmesi yaklaşımının kullanıldığı ilk çalışmalardan biri Drebin çalışmasıdır [28]. *Linear Support Vector Machine* algoritması kullanılmıştır. Elde edilen sonuçlar ele alınarak kötücül yazılım ailelerine göre en efektif sonuç veren öznitelikler gösterilmiştir. Bu sebeple bir uygulamanın hangi kötücül yazılım ailesine ait olduğu tespit edilirken öznitelik seçiminde yapılacak çalışmalara fikir vermiştir. Drebin [28] çalışmasında; 5560 adet kötücül yazılım içeren, Drebin olarak adlandırılan bir veri kümesi [32] kullanılmıştır. Bu veri kümesindeki uygulamaların öznitelik vektörleri çıkarılmıştır. Öznitelik vektörleri AndroidManifest.xml dosyası içerisinde tanımlanan donanım bileşenleri, izinler, aktiviteler, servisler, içerik sağlayıcılar, yayın alıcıları ve intent bilgileri kullanılarak oluşturulmuştur. Bunların yanında; uygulamanın kaynak kodlarındaki *restricted API* kullanımı, API çağrıları ile uygulama izinlerinin eşleştirilmesi, şüpheli işlemlerin yapılabileceği API çağrılarının çıkarılması (getDeviceId(), Runtime.exec(), sendTextMessage() vb.), ağ adresleri, URL ve ana makine adı bilgileri çıkarılarak vektör uzayına gömülmüştür.

Dendroid [26] çalışmasında kötücül yazılım ailelerine göre uygulamalar içerisinde yer alan kod yapılarını/bloklarını kontrol akış çizgeleri (CFG) şeklinde ifade etmiştir. Her bir kötücül uygulama için kullanılan kod blokları ve sayıları çıkarılmıştır. Bu kod blokları her bir

kötücül yazılım ailesi bazında kullanım sıklıkları çıkarılarak bir model ortaya konmuştur. Sınıflandırma modelinde KNN makine öğrenmesi algoritması kullanılmıştır. Aileler arasındaki hiyerarşik benzerliklerin çıkarılması için *single linkage hiearchical clustering* algoritması kullanılmış ve sonuçlar *dendogram* ağaçları ile ifade edilmiştir. AnserverBot ailesinin BaseBridge ailesinden türediği bilinmektedir [20, 33]. Dendroid [26] çalışmasında da bu durum karmaşıklık matrisi ile teyit edilmiştir.

DroidSIFT [27] çalışması, Android kötücül yazılımlarının aile sınıflandırılmasında ilk kez semantik tabanlı bir yöntem olarak önerilmiştir. Uygulamaların kaynak kodlarında kullandıkları metotlar ve API çağrıları, ağırlıklandırılmış bağlamsal API tabanlı çizgelerle ifade edilmiştir. API çağrıları ele alınırken güvenlik ilişkili API çağrılarının yanında, kullanıcıdan istenilen izinlerle eşleşen API çağrıları da bilhassa dikkate alınmıştır. Sınıflandırma işlemi, aile bilgisi bilinen kötücül uygulamalardan elde edilen çizgelerden oluşturulan veritabanı referans alarak yapılmıştır. Her yeni gelen uygulamadan elde edilen çizgelerin, veritabanında yer alan çizgelerle benzerlik değerleri hesaplanmıştır. Önerilen modelde Naive Bayes algoritması kullanılmıştır. Bu yaklaşım ile kötücül uygulamanın giriş noktasından başlayarak hedefine ulaştığı program semantiği çıkarılmıştır. DroidSIFT [27] çalışması, bir aileye ait kötücül bir uygulamanın hedefine ulaşırken kullandığı belirli noktalardaki metotların, API çağrılarının değişmeyeceği fikrinden faydalanmıştır.

DroidChamelon adlı çalışmada [34], kötücül yazılım geliştiricileri tarafından güvenlik çözümlerini atlatmak için, kod içerisine önemli işlevleri olmayan yahut geliştiricinin amacına direkt olarak hizmet etmeyen kod blokları/parçaları yerleştirilebilindiğinden bahsedilmiştir. Çalışmada önerilen modelin bahsedilen bir kısım kod dönüştürme yöntemlerine karşı dirençli olduğu belirtilmiştir. Diğer yandan güvenlik ilintili API çağrılarının kötücül olmayan uygulamalar tarafından da sıklıkla kullanılma ihtimali düşünüldüğünde, önerilen yöntemin kötücül uygulamaların tespitini olumsuz etkileyebileceği de bir dezavantaj olarak belirtilmiştir.

DroidMiner [35] çalışması, uygulamaların giriş noktalarından başlayıp son işleme kadar olan yapıyı iki-katmanlı davranışsal çizgelerle ifade etmiştir. DroidSIFT [27] ve Dendroid [26] çalışmalarından farklı olarak çizgeleri sırasıyla Component Dependency Graph (CDG)

ve Component Behaviour Graph (CBG) olmak üzere iki-katmanlı olarak tanımlamıştır. CDG katmanında, uygulamalardaki aktivitelerin, yayın alıcıların, servislerin ve içerik sağlayıcılarının (content://sms/inbox/, vb.) aralarındaki haberleşmeler gösterilmiştir. CBG katmanında ise uygulamanın kullandığı hassas ve uygulama izinleri ile ilintili olan API bilgilerinin, hassas kaynaklara erişimin oluşturduğu yaşam döngüsü ifade edilmiştir. Elde edilen bu çizgeler üzerinden (hassas kaynaklardan veri okuma/yazma, hassas API'lerin kullanıldığı metotların çağırılması vb.) işlem dizileri çıkarılmıştır. Bu dizilerin de alt dizileri çıkarılarak vektörler oluşturulmuştur. Önerilen model bu vektörler üzerinden eğitilmiş ve test edilmiştir. Önerilen yöntemin, kötücül yazılım geliştiricilerinin tespit edilen kod dizilerindeki hassas işlemlerin aralarına döngüler, işlevselliği olmayan kodlar ekleyerek atlatma girişimlerine karşı dirençli olduğu belirtilmiştir. Ancak statik analiz yaklaşımının kullanılması, şifrelenmiş veya Java kod yansıma (Java Reflection) yöntemi ile çağrılan metotların çizgelere dahil edilmesini zorlaştırmaktadır [35].

DroidSIFT [27] ve Dendroid [26] çalışmalarında ileri seviye karıştırma/bulanıklaştırma yöntemlerine karşı direncin yeterli düzeyde olmamasından yola çıkarak RevealDroid [24] çalışması yayınlanmıştır. RevealDroid [24] çalışması, uygulamaları hassas API'ler, ve aralarındaki haberleşmeler, intent mesajlar ve kullanılan API'lerin çağırıldığı paketler olmak üzere 4 ana başlıkta altında incelemiştir. 4 ana başlıktaki değerler sayısal olarak ele alınmıştır. API'ler arasındaki haberleşme, kaynak API ve çıkış API şeklinde ifade edilmiş ve 20 kaynak, 21 çıkış API'si ele alınmıştır. Uygulama kodunda ve AndroidManifest.xml dosyasında yer alan intent bilgileri alınmıştır. Öznitelik vektörleri boolean (1 ya da 0) olarak tanımlanmıştır. Son olarak her bir paketten çağrılan metot sayıları ele alınmıştır. Aile sınıflandırması ve kötücül yazılım tespiti için DT ve KNN olmak üzere iki farklı makine öğrenmesi algoritması kullanılmıştır. Bulanıklaştırma yöntemlerine karşı direnci ölçmek amacıyla, DroidChamelon [34] çalışmasında ProGuard [36] adlı araç kullanılmıştır. Kaynak kodda yer alan karakterlerin, dizilerin şifrelenmesi, sınıfların isimlerinin değiştirilmesi ve metot çağrılarının yönlendirilmesi işlemleri yapılmıştır. Yalnız statik analiz tabanlı incelemeler, çalışma zamanlı kod çağırılması vb. işlemlerin tespitinde yetersiz kalmaktadır. Diğer yandan RevealDroid [24] çalışması uygulamaların kaynak koduna erişilerek istenilen alanların okunmasında Soot [37] ve FlowDroid [38] araçlarını kullanmıştır. Yapılan analizler sırasında bazı uygulamalarda yükleme ve öznitelik çıkarımı sırasında uzun süreler

alması sebebiyle FlowDroid [38] ve Soot [37] araçları bu uygulamaların bir kısmını analiz edememiştir.

Dendroid [26] ve DroidSIFT [27] çalışmalarında önerilen yönteme benzer olarak GroupDroid [39] çalışmasında da uygulamaların kod benzerlikleri göz önünde bulundurulmuştur. .smali kodları içerisinde belirli bir sayıdan daha fazla işlem barındıran ve yazarların araştırmaları sonucunda belirlediği riskli API'ler olarak nitelendirilen API'leri kullanan metotlar analizler sırasında çıkarılmıştır. Bu bilgiler ışığında, Dendroid [26] çalışmasındaki gibi uygulamalara ait kontrol akış çizgeleri çıkarılmıştır. Ardından bu çizgeler 3 boyutlu düzlem üzerinde ağırlıklandırılarak (3D-CFG Centroid) ifade edilmiştir. Kodların birbirleri ile olan benzerlikleri 3D-CFG Centroid'ler arasındaki mesafe ile belirlenmiştir. Hesaplanan mesafeler baz alınarak uygulamaların gruplaması yapılmıştır. Bu çalışmada kötücül yazılım geliştiricilerinin aynı zararlı kod parçalarını, farklı kötücül yazılım ailelerine ait yazılımlarda da kullanabileceği gösterilmiştir [39].

DroidSieve [40] isimli çalışma statik analizden elde edilen öznitelikleri arttırarak daha verimli sonuçlar elde etmeye çalışmıştır. Öznitelikleri kaynak merkezli (resource-centric) ve sözdizimsel (syntactic) olmak üzere iki ana sınıfa ayırmıştır. Uygulamanın sertifika zamanı ve diğer detayları, kod içerisinde kullanılan paket adları, uygulama içerisinde gizlenmiş diğer uygulamaların (incognito app) kullandığı API bilgileri, izinler, intent, kullanılan dosyaların başlık bilgilerindeki uyumsuzluklar ve sistem çağrıları kaynak merkezli öznitelikler olarak değerlendirilmiştir. Ana uygulamadaki kullanılan API paketleri, aktiviteler, izinler, servisler, yayın alıcıları, intent, içerik sağlayıcılar, dinamik kod yükleme, kod yansıma, şifreleme kütüphanelerinin kullanımı ise sözdizimsel öznitelikler olarak değerlendirilmiştir. Makine öğrenmesi algoritması olarak Extra Trees (ET) kullanılmıştır. Kötücül yazılım geliştiricileri, uygulamanın statik analiz incelemelerini atlatabilmesi için asıl zararlı kodu uygulama kaynakları içerisinde resimler vb. dosyaların bulunduğu dikkat çekmeyecek dizinlere uzantısını ya da ismini değiştirerek konumlandırmaktadır. Bu anlamda paket/dosya isim uyumsuzlukları, dosya başlık bilgileri gizlenmiş uygulamaların da incelenmesi önemli hale gelmiştir. *repackaging* yöntemi ile geliştirilen uygulamalarda sertifika bilgilerinin tutarsız olması beklendiği için DroidSieve [39] çalışması bunu bir öznitelik olarak ele almıştır. Aile sınıflandırması için belirlenen özniteliklerin etkileri

değerlendirilmiştir. Uygulama izinlerinden CHANGE_WIFI_STATE, RECEIVE_BOOT_COMPLETED, intent olarak SIG_STR, BATTERY_CHANGED_ACTION ve PICK_WIFI_WORK kötücül yazılımları ailelerine sınıflandırırken üst sıralarda olan özniteliklerden bazılarıdır. DroidSieve [40] çalışmasında statik analiz anlamında birçok özniteliği ele almasına rağmen kötücül yazılım geliştiricilerinin bahsedilen değerlerin bir bölümünü taklit etme *(mimicry)* yöntemine karşı zayıf olduğu belirtilmiştir.

FalDroid [25], kötücül uygulamaların büyük bir bölümünün *repackaging* yöntemini kullanarak yayıldığından yola çıkarak uygulama içi kod parçalarını zararlı ve asıl uygulamaya ait kodlar olmak üzere değerlendirmiştir. Böylece analizi uygulamaya ait gerçek/zararsız kodların yan etkilerinden arındırmaya çalışmıştır. Uygulama kodunu fonksiyon çağrı çizgeleri (FCG) ile ifade ederek program semantiğini çıkarmıştır. Hassas API'ler üzerinde odaklanılmış, hassas API çağrıları tabanlı çizgeler (SARG) çıkarılmıştır. Ek olarak bu API çağrılarına TF-IDF benzeri bir yaklaşımla her bir kötücül yazılım ailesi için ağırlık değerleri tanımlanmıştır. SARG çizgeleri üzerinden *community detection* algoritmaları olan *infomap, fast greedy, fast partitioning* ve *multilevel* kullanılarak alt çizgeler çıkarılmıştır. Sınıflandırma teknikleri yardımıyla bir aile için en çok rastlanılan alt çizgeler çıkarılmış ve "fregraph" olarak adlandırılmıştır. Sık rastlanılan her bir çizgenin aileler ile olan ilintisi ağırlıklandırılmış olarak hesaplanmıştır. Bu değer üzerinden öznitelik vektörleri çıkarılmıştır. Bu çalışma ek kötücül yazılım veri kümeleri de kullanılarak genişletilmiştir [41]. Ancak yalnızca statik analiz yaklaşımı ile inceleme yapılması, kötücül yazılımların ileri atlatma/bulanıklaştırma/karıştırma yöntemlerini kullanması ile önerilen modelde tespit edilmesini zorlaştırmaktadır.

Statik analiz yönteminin kullanıldığı bir diğer çalışma ise AndMFC isimli çalışmadır [42]. Bu çalışmada ise birçok statik analiz yaklaşımlarına benzer olarak uygulamanın kaynak kod içerisinde gerçekleştirdiği API çağrıları ve uygulamanın kullandığı izinler çıkarılmıştır. Test ve değerlendirmeler sırasında AMD [42], Drebin [31] ve UpDroid [54] veri kümeleri kullanılmıştır. Standart makine öğrenmesi algoritmaları kullanılan çalışmada her üç kümesi ile yapılan testlerde en yüksek isabet oranları SVM algoritması ile elde edilmiş ve sırasıyla

Drebin [31] veri kümesinde %96,66, AMD [42] veri kümesi ile yapılan testlerde %98.86, UpDroid [54] veri kümesi ile yapılan testlerde %94,66 olarak hesaplanmıştır.

Android kötücül yazılımlarında aile sınıflandırması yapan bir diğer çalışma ise [44] çalışmasıdır. Bu çalışmada diğer çalışmalara göre farklı bir statik analiz yaklaşımı ortaya konulmuştur. Uygulamaların .dex dosyaları elde edilmiş ve görüntü işleme yöntemi kullanılarak .dex dosyasının bütünü ve .dex dosyasının yalnızca veri bölümü olmak üzere iki farklı alan ele alınmıştır. Bu çalışmada CNN algoritması kullanılarak %91'lik doğruluk oranı elde edilmiştir. Diğer bir görüntü işleme tabanlı çalışmada [45], kötücül yazılımlar ikili (binary) formata ve matrisler halinde gri-tonlamalı görüntülere dönüştürülmüştür. Bu aşamada görüntü 4 farklı görüntü filtresinden geçirilmiştir. Makine öğrenmesi algoritması için oluşturulan öznitelikler, görüntü işleme sonucu ortaya çıkan vektörlerden elde edilmiştir. Bu çalışmada statik analiz yaklaşımı kullanılmış olup 24549 adet kötücül yazılımda yapılan eğitim/test sonucunda RF algoritması ile %96,9 gibi bir doğruluk oranı ortaya konmuştur.

MVIIDroid adlı [46] çalışmada AndroidManifest.xml dosyası içerisinde tanımlanan uygulama izinleri, hassas API çağrılarının kullanım sıklığı ve Dalvik opcode kullanım dağılımı bilgileri öznitelikler olarak kullanılmıştır. Çalışmada 10 katmanlı çapraz doğrulama yöntemi ve çoklu çekirdek öğrenme algoritması (MKL) kullanılarak Drebin [32] veri kümesi üzerinde %93,1'lik bir doğruluk oranı elde edilmiştir.

3.2. Dinamik Analiz Yaklaşımları

Statik analiz araçlarının günden güne gelişmesi sebebiyle kötücül yazılım geliştiricileri uygulamalarda bulanıklaştırma (obfuscation), kod yükleme (DexClassLoad), kodda geçen sınıf/metot isimlerinin değiştirilmesi gibi yöntemlere yönelmektedirler. Bu durum, statik analizde kötücül davranışların tespit edilmesini zorlaştırmaktadır. Statik analiz yaklaşımından farklı olarak dinamik analiz yaklaşımında, uygulama belirli bir ortamda çalıştırılır ve davranışsal sonuçları gözlemlenir. Statik analize göre analiz ortamının oluşturulması yüksek maliyetlidir. Uygulamanın çalıştırıldığı süre, tetikleme noktaları vb. parametreleri göz önünde bulundurulduğunda statik analiz yaklaşımına göre kapsanan

kaynak kodun sınırlı olabileceği söylenebilir. Ancak çalışma zamanında gerçekleşen gönderilen/alınan ağ trafik verileri, telefon aramaları, dosya sisteminde yapılan okuma ve yazma işlemleri, alınan/gönderilen SMS mesajları, kod yansımaları, sonradan kod yükleme gibi işlemlerin tespitinde dinamik analiz yöntemi statik analiz yöntemine göre çok daha elverişlidir. Bu kritik işlemler uygulama öykünücü (emulator) ortamında çalışırken kayıt altına alınarak davranış verileri çıkarılmaktadır. Bu yöntem Android kötücül uygulamaların tespitinde sıklıkla kullanılmaktadır. Bu anlamda temel alınan çalışmalardan birisi TaintDroid çalışmasıdır [47]. TaintDroid [47] çalışmasından faydalanılarak geliştirilen açık kaynak kodlu dinamik analiz aracı DroidBox [48] bu anlamda sıklıkla kullanılmaktadır.

Android kötücül yazılım ailelerine sınıflandırılmasında DroidScribe adlı [49] çalışma tümüyle dinamik analiz yaklaşımını kullanan ilk çalışmadır. DroidScribe [49], uygulamaların dinamik analizini gerçekleştirmek için CopperDroid [50] çalışmasından faydalanmıştır. Android uygulamanın oluşturduğu ağ trafik boyutu, eriştiği IP, port numaraları, erişilen ve çalıştırılan dosyaların özellikleri ve alt seviye kullanılan Binder IPC metotları öznitelik değerleri olarak belirlenmiştir. Binder IPC, Android işletim sisteminin temelde kullandığı Linux çekirdeğinde yer alır ve işlemlerin aralarında haberleşmesini sağlayan metotları içermektedir. Bu anlamda uygulamanın yaptığı sistem çağrılarının temelinde burada bulunan metotların kullanılması da olasıdır. Makine öğrenmesi algoritması olarak SVM kullanılmıştır. Çalışmada her bir aile için kalite eşik değeri belirlenmiş ve bu eşik değerinin altında kalan aileler sınıflandırılmıştır. Dinamik analiz yaklaşımında yürütülen uygulamalar için yalnızca tek bir çalışma yolu (execution path) çıkarılmış ve bunun üzerinden öznitelikler elde edilmiştir. Bu durum kötücül yazılım geliştiricilerinin uygulamaların çalışma süreçleri içerisine bilinen uygulamalardaki çalışma süreçlerini taklit etmeleri ile analiz sürecini tespit edilmeden atlatmalarına sebep olabilir. Uygulamaların çalışma sürelerinin mücbir sebepler dolaysıyla (kaynak, zaman vb.) kısıtlı olması kötücül davranışların analiz süreci dışında kalmasına da neden olabilir. Çalışmada analizlerin bu kısıtlamalardan etkilenebileceği belirtilmiştir.

Dinamik analiz yaklaşımını kullanarak, ağ etkinliklerini analiz eden çalışmalar da bulunmaktadır. CREDROID adlı [51] çalışma, Malgenome [21] veri kümesinde yer alan kötücül uygulamaların ağ trafik analizini yaparak birtakım sonuçlar elde etmiştir. HTTP,

DNS ve SSL protokolleri üzerinde durmuştur. Malgenome [21] veri kümesinde yer alan uygulamaların ilgili protokoller bağlamında yalnızca %63'ü ağ trafiği oluşturmuştur. Kalan uygulamaların ya hiçbir şekilde ağ trafiği oluşturmadığı veya çalışma ortamı ile uyumsuz olması sebebiyle veri elde edilemediğinden çalışmada bahsedilmiştir [51].

Başka bir analiz çalışmasında [52], Drebin [32] veri kümesi kullanılarak ağ tabanlı bir dinamik analiz yapılmıştır. Bu çalışmada %70'ten fazla kötücül uygulamanın zararlı işlemlerini ilk beş dakika içerisinde gerçekleştirdiği görülmüştür. Analizler DNS sorguları ve HTTP istek paketleri üzerinden yapılmıştır. Yalnızca bu bilgiler ışığında sırasıyla %69,55 ve %40,89 kötücül yazılım tespit oranları elde edilmiştir.

AndroDFA [53] çalışmasında Android kötücül yazılım aile sınıflandırmasına ilişkin dinamik analiz yöntemi kullanılarak bir yaklaşım önerilmiştir. Bir Linux dağıtımı olan Android işletim sistemindeki /proc dosya sisteminden uygulamanın işletim sisteminde tüketmiş olduğu kaynaklar ele alınmıştır. Genel olarak kullanılan öznitelikler, CPU ve hafıza kullanımı, çalışan işlemlerin işletim sisteminde hayatta kalma süreleri gibi kaynak tüketimine dayalı değerlerden elde edilmiştir. Bu bilgiler dinamik olarak çalışma zamanlı her bir kötücül uygulama için elde edilmiş ve SVM algoritması kullanılarak sınıflandırma yapılmıştır. Drebin [32] ve AMD [43] olmak üzere iki veri kümesinden gelen kötücül yazılım örneklerine dayalı değerlendirme yapılmıştır.

DroidClassifier [54] isimli çalışmada dinamik analizler sırasında kötücül uygulamalar 5 dakika boyunca çalıştırılmış ve HTTP başlık bilgilerinden elde edilen "Host", "Referer", "Request-URI", "User-Agent" ve "Content-Type" olmak üzere yalnızca 5 farklı öznitelik üzerinden Android kötücül yazılım aile sınıflandırması yapılmıştır. Veri kümesi olarak toplam 1363 adet kötücül yazılım kullanılmıştır. Model 706 kötücül yazılım ile eğitilmiş ve 657 adet kötücül yazılım ile test edilmiştir. Bu çalışmada %94,33 oranında bir başarı elde edilmiştir.

3.3. Hibrit Analiz Yaklaşımları

Hibrit yaklaşımlarda genellikle birden fazla yöntemin birleştirilmesiyle oluşturulmakta ve diğer yaklaşımların dezavantajlarını en aza indirmek amaçlanır. Android kötücül yazılım aile sınıflandırmalarında bu yaklaşım ile uygulamaların hem çalışma zamanlı yaptığı işlemler dinamik analiz yöntemi ile hem de kaynak kodu/dosyaları statik analiz yöntemi ile incelenir. Elbette analizin kapsamlı olması maliyeti arttırmakta, ancak önerilen modelin daha iyi eğitilmesini ve daha doğru sonuçların alınmasını sağlar.

Hibrit analiz yaklaşımının kullanıldığı ilk çalışma, Ec2 çalışmasıdır [7]. Geliştirici adı, uygulamanın APK dosyasının boyutu, AndroidManifest.xml içerisinde yer alan aktivite, servis, içerik sağlayıcı, yayın alıcıları, intent bilgileri, yazılımsal ve donanımsal uygulama izinlerinin sayıları statik analiz öznitelikler olarak ele alınmıştır. Yazılımsal izinler, cihaz üzerinde belirli yazılım kaynaklarına erişimi sağlarken (arama kayıtları vb.); donanımsal izinler ise cihazda yer alan donanımlara erişimi sağlamaktadır (kamera, mikrofon vb.). DroidSieve [40] çalışmasında bahsedildiği üzere sertifika bilgisi de sınıflandırma açısından önemli bir bilgi olarak değerlendirilmiş ve Ec2 [7] çalışmasında kullanılmıştır. Aynı aileye ait kötücül uygulamaların aynı geliştirici tarafından yazılabileceğinden ve aynı sertifika ile imzalanabileceğinden bahsedilmiştir. AndroidManifest.xml ve kod içerisinde tanımlanan birçok bileşenin/sınıfın isimlerinin kötücül yazılım geliştiricileri tarafından kolaylıkla değiştirilebileceği düşünülerek özniteliklerin adları yerine sayıları ele alınmıştır. Ec2 [7] çalışmasında 190 statik ve 2048 adet dinamik öznitelik kullanılmıştır.

Hibrit analiz yönteminin kullanıldığı bir diğer önemli çalışma ise UpDroid [8] çalışmasıdır. Bu çalışmada güncelleme saldırısı üzerinde durulmuş, UpDroid [55] adı verilen bir kötücül uygulama veri kümesi oluşturulmuştur. İlgili veri kümesi 21 farklı kötücül yazılım ailesinden ve 2535 adet kötücül uygulamadan oluşmaktadır. Aynı zamanda yapılan çalışmada hem statik hem de dinamik öznitelikler kullanılarak bir sınıflandırma modeli önerilmiştir**.** UpDroid [8] çalışması, Ec2 [7] çalışmasında Malgenome [21] veri kümesi kullanılarak elde edilen MiF ve MiAUC performans metrikleri ile karşılaştırılmış ve UpDroid [8] çalışmasının bu bakımdan daha performanslı olduğu görülmüştür [7]. İlgili karşılaştırma Çizelge 3'te görülebilir.

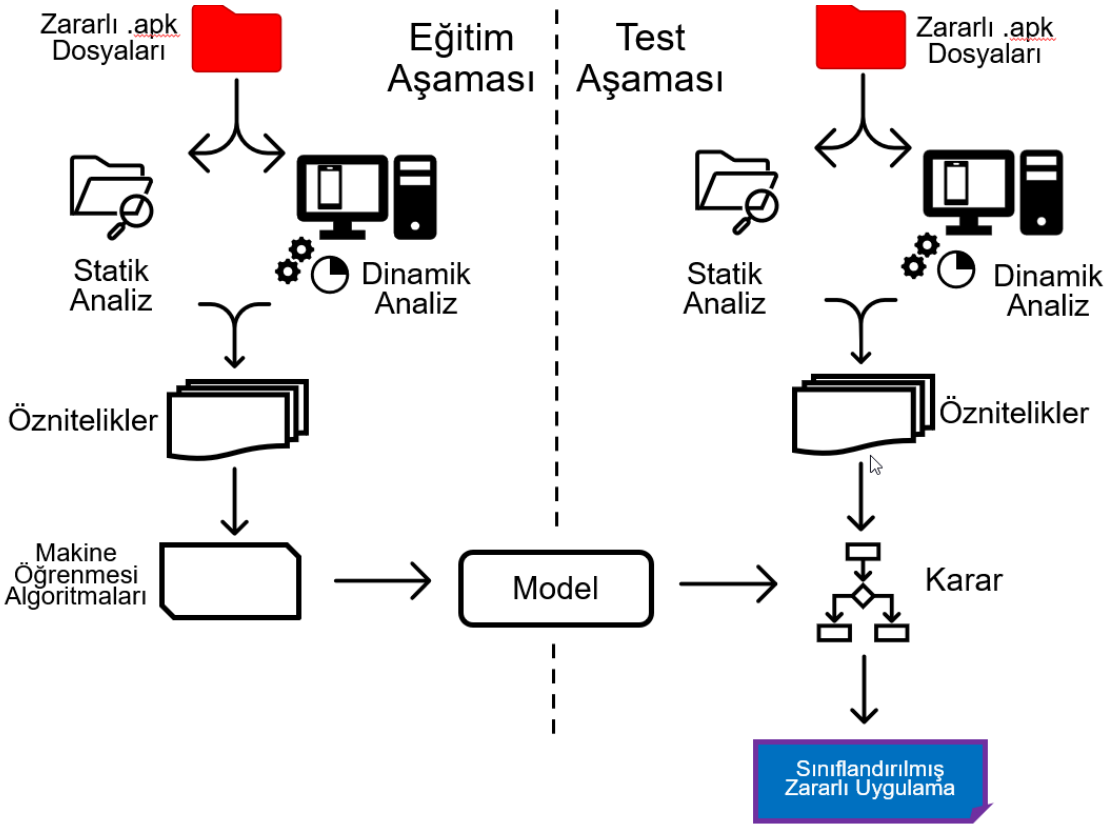
Çizelge 3. Ec2 [7] ve UpDroid [8] Çalışmalarının Karşılaştırılması.

Çalışma	Algoritma	MiF	MiAUC
Ec2 [7]	KNN	0,47	0,73
UpDroid [8]	KNN	0,96	0,98
Ec2 [7]	RF	0,95	0,97
UpDroid [8]	RF	0,94	0,99

Literatürde statik analiz yöntemleri ile gerçekleştirilen çalışmalarda kod blokları, API çağrıları ve AndroidManifest.xml dosyası içerisinden elde edilen uygulama izinleri, aktiviteleri, yayın alıcıları vb. bilgiler alınmıştır. Bu tez çalışmasında, statik analiz yöntemi dahilinde benzer çalışmalar ile ortak olarak AndroidManifest.xml dosyasındaki veriler kullanılmıştır. Yukarıda da bahsedildiği gibi güncel saldırı yöntemlerinde saldırganlar kodlarında ileri kod karıştırma tekniklerini kullanmaktadır. Bu nedenle, kötücül uygulamanın çalıştırılmasına dayalı dinamik analiz yöntemleri de önerilen yönteme dahil edilmiştir. Bu çalışmadaki amaç daha önceki çalışmalarda bir arada kullanılmamış öznitelikleri de kullanarak, başarımın arttırılmasını araştırmaktır.

4. ÖNERİLEN MODEL

Bu bölümde, bu tez çalışmasında önerilen hibrit tabanlı kötücül yazılım aile sınıflandırılması algoritması anlatılacaktır. Özellikle, bu çalışmada yeni tanıtılan öznitelikler ayrıntılı açıklanacaktır.



Şekil 11. Önerilen Model Şema Görünümü.

Yapılan çalışmada, 4.1 bölümünde bahsedilen Malgenome [21], Drebin [32] ve UpDroid [55] veri kümeleri üzerinde çalışılmıştır. Testler sırasında, bu veri kümeleri haricinde indirilen uygulamaların kötücül tespiti VirusTotal [56] uygulaması kullanılarak yapılmıştır. İlgili veri kümelerinde yer alan kötücül uygulamaların kaynak kod ve dosyaları, apktool [22] aracı ile çıkarılmıştır. Kaynak dosyaları üzerinde, öncelikle statik analiz yöntemi uygulanmış ve statik öznitelikler Python programlama dilinde Androguard [57] kütüphanesi kullanılarak yazılan bir uygulama ile çıkarılmıştır. Statik özniteliklerin ardından DroidBox [48] aracı ile kötücül uygulamalar 15 dakika boyunca çalıştırılmıştır. İncelemeler, API seviyesi 16

üzerinden yapılmıştır. Uygulama aktiviteleri DroidBox [48] aracından .json formatında, ağ trafiği ise .pcap uzantılı olarak elde edilmiştir. Şekil 11'de bu adımlara ilişkin kavramsal bir şema verilmektedir.

İlişkili çalışmalar bölümünde belirtildiği gibi her iki analizin de artıları ve eksileri bulunması sebebiyle bu çalışmada hibrit analiz yöntemi kullanılmıştır. Doğru öznitelikleri seçmek kötü amaçlı yazılım ailelerinin özelliklerini ayırt etmek için etkili sınıflandırıcıları oluşturabilmek açısından çok önemlidir. İlişkili çalışmalarda çoğunlukla kullanılan makine öğrenmesi algoritmaları, bu çalışmada da kullanılmıştır; KNN, DT, RF ve SVM. Çalışmada önerilen modeller, Weka [58] aracı kullanılarak eğitilmiş ve test edilmiştir. Weka [58] aracında yukarıda bahsedilen makine öğrenmesi algoritmaları varsayılan ayarları ile kullanılmıştır.

Bu tez çalışması ile, hibrit analiz yöntemi ile kapsamlı bir öznitelik kümesi tanımlanarak sınıflandırabilme oranın iyileştirilmesi, yeni öznitelik gruplarının, farklı veri kümelerindeki etkilerinin incelenmesi amaçlanmıştır. Android kötücül yazılımların aile sınıflandırması başlığı altında, uygulamaların statik analizi sırasında elde edilen ağ tabanlı öznitelikler olan IP ve Port bilgilerinin yanında, dinamik analiz sırasında oluşturdukları ağ trafik bilgileri (IP/TCP/HTTP/DNS paketleri vb.) de detaylı işlenerek yeni öznitelikler çıkarılmış ve sonuçları değerlendirilmiştir. Bununla beraber analizi yapılan uygulamaların çalışma zamanlı olarak gerçekleştirdiği ikili davranışların sıklık bilgisi (ör. fread-frwrite, sendnet-sendsms) üzerinde durulmuş ve bu davranışlar ile ilgili yeni öznitelikler eklenmiştir.

4.1. Veri Kümeleri

Çalışmalar, Malgenome [21] veri kümesinden 42 kötücül yazılım ailesine ait 1260, Malgenome [21] veri kümesi genişletilerek oluşturulan Drebin [32] veri kümesinden 178 aileye ait 5554 ve UpDroid [55] veri kümesinden 21 aileye ait 2535 adet kötücül yazılım üzerinde yapılmıştır. Kötücül yazılım aile sınıflandırmasındaki öznitelikler bu üç veri kümesi üzerinde değerlendirilmiştir. Malgenome [21], literatürde bu bağlamda yer alan ilk veri kümesidir. Bu veri kümesi, daha sonra Drebin [32] olarak genişletilmiştir. Malgenome [21] veri kümesindeki bütün örnekler Drebin [32] kümesinde de yer almaktadır. UpDroid

[55] veri kümesi, kendisini güncelleyen kötücül yazılımları içermekte ve çalışma zamanlı kod yükleme davranışlarının da kapsam içerisine almayı amaçlamaktadır.

Çizelge 4, Çizelge 5 ve Çizelge 6'da veri kümelerinden dinamik analiz sırasında Android uygulama öykünücü (emulator) üzerinde beklendiği gibi çalışarak çalışmada kullanılan 8.2. Ek 2 başlığında belirtilen bütün öznitelik kümeleri için hata alarak kapanmadan, sağlıklı veri üreten Malgenome [21], Drebin [32] ve Updroid [55] veri kümelerine ait uygulamaların aile bazlı listesi verilmiştir.

Çizelge 4. Malgenome [21] Veri Kümesi Aile Dağılımı.

Aile	Adet	Aile	Adet	Aile	Adet
Adrd	61	Geinimi	58	Replicator	1
BaseBridge	299	GGtrack	1	SendPay	8
BeanBot	8	Glodream	45	SerBG	9
Cosha	1	Gonca	4	Smspacem	1
CrWind	2	GPSpy	1	Spy.GoneSixty	1
Dogowar	1	Hispo	3	Tapsnake	1
DroidDream	44	Jifake	1	Typstu	9
DroidKungFu	462	Kmin	10	Xsider	14
DroidRooter	1	Pirater	1	Yzhc	21
ExploitLinuxLotoor	35	Pirates	1	Zitmo	1
FakeNefix	1	PJApps	1	Zsone	6
FakePlayer	6	Plankton	11	**Toplam**	**1137**
Gasms	1	Raden	6		

Çizelge 5. UpDroid [55] Veri Kümesi Aile Dağılımı.

Aile	Adet	Aile	Adet
Asacub	54	Marcher	21
Bankbot	24	Ogel	10
Fakebank	8	Rootnik	32
Fakeflash	4	Shedun	630
Faketoken	11	Smsreg	291
Krep	5	Smsspy	57
Ksapp	2	Sprovider	5
Leech	7	Tordow	11
Lotoor	9	Triada	1010
Malap	181	Ztorg	15
		Toplam	**2387**

Çizelge 6. Drebin [32] Veri Kümesi Aile Dağılımı.

Aile	Adet	Aile	Adet	Aile	Adet
AccuTrack	10	Gamex	6	RediAssi	3
Ackposts	2	Gapev	6	Replicator	3
Acnetdoor	1	Gappusin	46	RootSmart	5
Adrd	78	Gasms	1	RuFraud	1
Adsms	1	Geinimi	79	SafeKidZone	1
Aks	5	Generic	1	Saiva	2
Ansca	1	GGtrack	3	Sakezon	8
Antares	2	GinMaster	333	Sdisp	1
Anudow	1	Glodream	67	SeaWeth	3
Arspam	1	Gmuse	3	SendPay	56

BaseBridge	309	Gonca	5	SerBG	14
BeanBot	8	GPSpy	2	SheriDroid	2
Biige	3	Hamob	28	SmForw	1
Booster	1	Hispo	3	SMSBomber	1
Bosm	1	Iconosys	135	Smspacem	1
Boxer	25	Imlog	41	SMSreg	40
CellShark	1	Jifake	28	SMSSend	1
CellSpy	2	JSmsHider	2	SmsSpy	1
Ceshark	7	Kidlogger	4	SmsWatcher	3
Coogos	8	Kmin	86	SMSZombie	10
Copycat	12	Koomer	2	Sonus	1
Cosha	11	Ksapp	5	SpyBubble	2
CrWind	2	Lemon	6	Spy.GoneSixty	1
Dabom	2	LifeMon	3	SpyHasb	7
Dialer	2	Loicdos	1	SpyImLog	1
Dogowar	1	Loozfon	2	SpyMob	3
Dougalek	17	Luckycat	3	Spyoo	3
DroidDream	75	Lypro	1	SpyPhone	5
DroidKungFu	643	Maistealer	1	Spyset	8
DroidRooter	3	Mania	5	Ssmsp	1
DroidSheep	10	Maxit	1	Stealer	1
EICAR-Test-File	4	MMarketPay	1	Stealthcell	1
EWalls	1	Mobilespy	2	Steek	14
ExploitLinuxLotoor	66	MobileTx	67	Stiniter	4
Exploit-RageCage	1	Mobinauten	2	SuBatt	1
Faceniff	1	Mobsquz	1	Tapsnake	3
FaceNiff	5	Moghava	2	Tesbo	2

FakeDoc	130	Nandrobox	13	TheftAware	2
FakeFlash	3	Nickspy	3	Trackplus	6
FakeInstaller	911	NickyRCP	2	Trojan.SMS.Boxer.AQ	1
Fakelogo	18	Nyleaker	17	TrojanSMS.Denofow	5
FakeNefix	1	Opfake	593	TrojanSMS.Hippo	12
Fakengry	13	PdaSpy	4	Typstu	12
FakePlayer	17	Penetho	19	Updtbot	1
FakeRun	59	Pirater	1	UpdtKiller	1
FakeTimer	12	Pirates	2	Vdloader	11
Fatakr	16	PJApps	1	Vidro	5
Fidall	2	Placms	12	Whapsni	1
FinSpy	1	Plankton	545	Xsider	17
Fjcon	2	Proreso	2	YcChar	2
FoCobers	14	Qicsom	1	Yzhc	36
Foncy	2	QPlus	5	Zitmo	14
Fsm	3	Raden	10	Zsone	7
Fujacks	1	RATC	1	**Toplam**	**5091**

4.2. Öznitelik Çıkarımı

İlişkili çalışmalar bölümünde ifade edildiği gibi, statik analizler sırasında çoğunlukla kaynak kodu, dosyaları ve AndroidManifest.xml dosyasından öznitelikler çıkarılmıştır. Diğer öznitelikler ise analizi yapılan uygulamaların Android öykünücü (emulator) üzerinde çalıştırılması ile çıkarılmıştır. Bu bölümde çalışmada ele alınan statik ve dinamik öznitelikler detaylandırılacaktır.

Bu çalışmada ilk kez ağ tabanlı öznitelikler ve daha önce kullanılmamış yeni bir öznitelik grubu olarak aktivite ikili tekrarları, UpDroid [8] çalışmasında kullanılan statik ve dinamik özniteliklerle beraber kullanılarak test edilmiştir. Veri kümelerindeki her bir kötücül yazılım,

statik ve dinamik analize tabi tutulmuş ve her uygulama için bir öznitelik vektörü elde edilmiştir. 4.2.1 ve 4.2.2 başlıklarında statik ve dinamik analiz sonucu elde edilen her bir öznitelik ayrıntılı açıklanmıştır. Uygulama öznitelikleri çıkarılırken DroidBox [48] ortamı kısıtları sebebiyle uygulamalarda API seviyesi 16 ve altı baz alınmıştır.

4.2.1. Statik Öznitelikler

Analiz edilen uygulamaların tersine mühendislik yöntemi ile kaynak kodlarına ve dosyalarına erişilmiştir. Statik analiz yöntemi ile elde edilen öznitelikler genellikle AndroidManifest.xml dosyası veya doğrudan kaynak kod/dosyalar içerisinden çıkarılmıştır. İlk olarak, bir tersine mühendislik aracı olan apktool [22] aracı kullanılarak uygulama paket içeriğindeki kaynak dosyaları (assests, lib, res vb.), kaynak kodu (classes.dex), AndroidManifest.xml gibi uygulamanın dosyaları çıkarılmıştır. Burada, AndroidManifest.xml dosyasından toplanan statik özellikler UpDroid [8] çalışmasında bahsedildiği şekilde kullanılmıştır.

Arama listesi, SMS, fotoğraflar, donanım bileşenleri gibi veriler (ör. kamera, bluetooth, mikrofon) yalnızca kullanıcı izni verilerek kullanılabilir [59]. Bu sebeple uygulama izinleri, Android kötü amaçlı yazılım tespitinde en çok kullanılan öznitelikler arasında olmakla birlikte bu çalışmada da kullanılmaktadır [60]. Uygulama izinleri dışında çalışmada kullanılan statik özniteliklerin kalan kısmı AndroidManifest.xml dosyası içerisinden çıkarılmıştır. AndroidManifest.xml dosyası uygulamanın kullanacağı donanım bileşenlerini, aktiviteleri, servisleri, içerik sağlayıcıları, yayın alıcıları ve intent bilgilerini XML etiketleri ile barındırmaktadır. Şekil 12'de, Drebin [32] veri kümesinde analizi yapılan bir uygulamaya ait AndroidManifest.xml dosyasından bir örnek kesit görülmektedir.

Uygulamanız, kısıtlanmış verilere veya kısıtlanmış eylemlere erişim gerektirebilecek işlevler sunuyorsa, izin beyan etmenize gerek kalmadan bilgileri alıp alamayacağınızı veya eylemleri gerçekleştirip gerçekleştiremeyeceğinizi belirleyin. Fotoğraf çekme, medya oynatmayı duraklatma ve ilgili reklamları görüntüleme gibi birçok kullanım durumunu, herhangi bir izin beyan etmenize gerek kalmadan uygulamanızda gerçekleştirebilirsiniz.

Şekil 12. AndroidManifest.xml Örnek Görünüm.

Kullanılan aktivitelerin, intent bilgilerinin, servislerin, yayın alıcıların sayıları statik öznitelik olarak tanımlanmıştır. Android API seviyesi 16 için desteklenen 102 farklı Android uygulama izinleri çalışmaya dahil edilmiştir. Öznitelik 1 ya da 0 olarak mantıksal (boolean) formatında ele alınmıştır. AndroidManifest.xml içerisinde tanımlı olan ancak kod içerisinde kullanılmayan özel izinler, aktiviteler, intent, servisler, alıcılar üçüncü parti çalıştırılabilir dosyanın beklenen şekilde çalıştırılmasını amaçlamaktadır. Bu sebeple halihazırda analiz edilen uygulamanın AndroidManifest.xml dosyası içerisinde üçüncü taraf uygulamaya ait olabilecek bu bilgiler çalışmaya özel izinler sayısal ve aktiviteler, servisler, alıcılar ise mantıksal öznitelikler olmak üzere dahil edilmiş ve Çizelge 7'de gösterilmiştir.

Kötücül bir uygulamanın çalışma zamanlı olarak diğer kötücül uygulamayı indirdiği bir senaryo düşünüldüğünde, sharedUserId değeri bu iki uygulama arasında kaynak paylaşımında önemli olduğu değerlendirilerek statik öznitelik kümesine dahil edilmiştir. Bu çalışmada Çizelge 7'te yer alan UpDroid [8] çalışmasında da kullanılan statik özniteliklere sharedUserId özniteliği de eklenmiştir.

Ek olarak, aynı aileye ait kötücül uygulamaların kullandığı kaynakların (kod, resim vb.) boyut bakımından benzer olabileceği düşüncesinden yola çıkılarak uygulama APK paket büyüklükleri de göz önünde bulundurulmuştur.

Çizelge 7'de yapılan çalışmada kullanılan statik öznitelikler ve karşılığında adetleri gösterilmiştir. Çizelge 7'de koyu olarak belirtilen öznitelikler UpDroid [8] çalışmasında kullanılmayan ve bu çalışmada ilk kez kullanılan özniteliklerdir.

Çizelge 7. Statik Öznitelikler.

Öznitelik Adı	Değişken Türü	Adedi
APK Paket Büyüklüğü	Sayısal	1
Aktiviteler	Sayısal	1
Servisler	Sayısal	1
Yayın Alıcılar (Broadcast Receivers)	Sayısal	1
Android Uygulama İzinleri	Mantıksal	102
Özel İzinler	Sayısal	1
Ek Aktiviteler	Mantıksal	1
Ek Servisler	Mantıksal	1
Ek Yayın Alıcılar (Extra Broadcast Receivers)	Mantıksal	1
sharedUserId Kullanımı	Mantıksal	1

4.2.2. Dinamik Öznitelikler

Uygulamalar belirlenmiş bir zaman aralığı için Android uygulama öykünücü ortamında çalıştırılarak davranışları kayıt altına alınır. Kayıt altına alınan veriler (aktiviteler, ağ trafiği vb.) üzerinden dinamik öznitelikler çıkarılır. Dinamik analiz için her bir uygulama 15 dakika boyunca DroidBox [48] aracı ile Android uygulama öykünücü ortamında çalıştırılmıştır. Uygulamaların işletim sisteminde eriştiği (okuma/yazma) dosyaların bilgileri ve oluşturduğu ağ trafik bilgileri .pcap dosyası olarak toplanmıştır.

Zararlı uygulamalar, kötücül davranışlarını, uygulamanın başlatmasından belirli bir zaman sonra başlatabildiği gibi, kullanıcının uygulamaya gönderdiği komutlar ile de başlatabilmektedir. Bu sebeple uygulamalar DroidBox [48] ortamında Monkey [61] aracı ile

tetiklemeli olarak çalıştırılmıştır. Monkey [61] aracı ile uygulamaya ekran kaydırma, tıklama ve benzeri olay girdileri rastgele bir biçimde gönderilmiştir. DroidBox [48] aracı zararlı uygulamanın gerçekleştirdiği aktiviteleri .json uzantılı olarak sunmuştur. Böylelikle statik analiz aşamasında AndroidManifest.xml dosyası içerisinde tespit edilmiş olan aktivitelerin çalışma zamanlı olarak kullanılıp/kullanılmadığı bilgisi çıkarılmıştır. Bunun yanında AndroidManifest.xml dosyasında bulunmayıp çalışma zamanlı olarak sonradan indirilen kodlar vasıtasıyla çalışan aktiviteler de tespit edilmiştir.

Çizelge 8'de UpDroid [8] çalışmasında yer alan dinamik öznitelikler, bu çalışmada da hibrit analizde kullanılan öznitelik kombinasyonlarının değerlendirilmesinde ve karşılaştırılmasında kullanılmıştır. Kripto kategorisinde şifreleme, şifre çözme ve anahtar oluşturma işlemleri, kullanılan kripto algoritmaları (DES, RSA vb.), IMEI gibi DroidBox [48] aracının çıktıları arasında yer alan 23 farklı hassas verinin ifşasına ait bilgiler öznitelik kümesinde bulunmaktadır. DroidBox [48] ek olarak verilerin ifşa yöntemlerine (ağ bağlantısı, SMS vb.) ilişkin bilgileri dinamik analiz sonucunda raporlamaktadır. Çizelge 8'de yer alan özniteliklerin tamamı UpDroid [8] çalışmasında kullanılmış öznitelikler olup bu çalışmadaki modele de dahil edilmiştir.

Çizelge 8. UpDroid [8] Çalışmasında Kullanılan Dinamik Öznitelikler.

Öznitelik Adı	Değişken Türü	Adedi
Toplam Kripto Aktiviteleri	Sayısal	3
Kripto Algoritmaları	Mantıksal	13
Hassas Veri Türlerinin İfşası	Mantıksal	23
Veri İfşası Yöntemleri	Sayısal	3
Toplam Veri İfşası Adedi	Sayısal	1
Çalışma Zamanlı Kayıtlı Alıcı Adedi (Registered Receivers)	Sayısal	1
Belirli Dizinler Üzerindeki Toplam Okuma/Yazma İşlemi	Sayısal	6
Toplam Dosya Erişimi	Sayısal	1
Toplam Dosya Okuma/Yazma İşlemi	Sayısal	2

Toplam SMS Gönderimi	Sayısal	1
Toplam Telefon Görüşmeleri	Sayısal	1
DexClassLoader Sınıfı Kullanımı	Sayısal	1
Toplam Servis Başlatımı	Sayısal	1
Toplam Kripto İşlem Adedi	Sayısal	1

Ağ tabanlı özniteliklerin çıkarımında dinamik analizler sırasında Wireshark [62] ve Tshark [63] uygulamalarından Linux kabuk betikler oluşturularak yararlanılmıştır. Çalışmada kullanılan tüm ağ-tabanlı öznitelikler Çizelge 9, Çizelge 10 ve Çizelge 11'de görülmektedir. Gönderilen ve alınan ağ paketlerinin sayısı gibi ağ trafiği hakkında genel bilgilerin yanı sıra, kullanılan ağ tabanlı özniteliklerin büyük bir kısmını HTTP ve DNS protokol başlık bilgilerinden alınan bilgiler oluşturmaktadır. HTTP trafiği, SDK sürümü, IMEI ve IMSI numaraları gibi bazı hassas verileri sızdırmak için kullanılabilir. Örneğin, [52] çalışmasında Kmin ailesine ait kötücül uygulamaların Android SDK sürümünü sızdırdığı belirtilmiştir. Bu çalışmada da Shedun ailesine ait kötücül uygulamaların HTTP GET isteklerindeki ortalama URL uzunluklarında benzerlik görülmüştür. Diğer yandan ortalama URL uzunluk bilgisinin öznitelik kümesine dahil edilmesindeki neden, kötücül uygulamaların sabit uzunluğa sahip IMSI, IMEI bilgilerini HTTP trafiği üzerinden sızdırması olmuştur.

Ağ tabanlı analizlerde kötücül uygulamalar genellikle uzak kaynaktan Android işletim sistemi içerisinde çalıştırılabilir .elf, .jar uzantılı dosya veya .apk uzantılı üçüncü taraf bir uygulama indirebilmektedir. Bu anlamda HTTP trafiği dahilinde yer alan diğer öznitelikler ise .jar veya .apk uzantılı dosyaların HTTP istek URLi içerisinde varlığı bilgisidir. UpDroid [8] çalışmasında bahsedilen güncelleme saldırıları düşünüldüğünde bu bilgi sınıflandırmada önem arz etmektedir. Kötücül uygulamalar yukarıda bahsedildiği üzere IMSI, IMEI ve telefon numarası gibi benzeri kişiye veya mobil cihaza özel hassas bilgileri HTTP isteklerinde bir URL içerisinde bir parametre olarak da dışarı sızdırabilmektedir. Bu bağlamda bir URL içerisindeki ortalama parametre adedi de öznitelik kümesinde yer almaktadır.

Kötücül bir yazılımın uzak bir komuta kontrol sunucusuna alan adları ile de ulaşabildiği düşünüldüğünde HTTP protokol başlık bilgilerinin yanı sıra DNS protokolüne ait istek ve cevap adetleri, tekil alan adı adetleri, ortalama alan adı uzunlukları da öznitelik kümesine dahil edilmiştir. Bu anlamda kötücül bir uygulamanın ilk DNS isteğini gönderdiği zaman ile son DNS isteğini gönderdiği zaman ve bu zamanlar arasında geçen süre de öznitelik kümesine dahil edilmiştir.

Özetle; UpDroid [8] çalışmasında yer alan toplam başlatılan/sonlanan ağ bağlantı adetleri, toplam ağ oturum adedi, toplam network paket boyutu, toplam gönderilen/alınan ağ paket sayılarına ek olarak IP, HTTP, DNS ve UDP protokol başlıkları detaylandırılarak yeni öznitelikler eklenmiştir. Çizelge 9 ve Çizelge 10'da HTTP ve DNS protokol başlıklarına ait öznitelikler görülmektedir. Çizelge 9 ve Çizelge 10'da yer alan öznitelikler içerisinde koyu harfler ile belirtilenlerin tamamı yapılan bu tez çalışması kapsamında yeni eklenen özniteliklerdir, daha önce yapılan UpDroid [8] çalışmasında kullanılmamıştır.

Çizelge 9. HTTP Protokolü Öznitelikleri.

HTTP Protokol Öznitelikleri	Değişken Türü	Adedi
HTTP İçerik Türü	Mantıksal	6
5 dk. Periyot- HTTP Trafik Boyutu (bayt)	Sayısal	3
Toplam HTTP Trafik Boyutu (bayt)	Sayısal	1
HTTP İstek/Cevap Adedi	Sayısal	2
HTTP Frame Adedi	Sayısal	1
HTTP Ortalama Cevap Boyutu (bayt)	Sayısal	1
İlk/Son HTTP Paket Zamanı	Sayısal	2
Toplam HTTP Trafik Süresi	Sayısal	1
Toplam HTTP Sunucu Adedi	Sayısal	1
GET/POST Ortalama URL Uzunluğu	Sayısal	2
GET/POST Ortalama URL Parametre Adedi	Sayısal	2
GET/POST URL İçerisinde .apk/.jar Dosya Varlığı	Mantıksal	4

Toplam GET/POST İstek Adedi	Sayısal	2
GET/POST Tekil URL Adedi	Sayısal	2
Ortalama POST Mesaj Boyutu (bayt)	Sayısal	1

Çizelge 10. DNS Protokolü Öznitelikleri.

DNS Protokol Öznitelikleri	**Değişken Türü**	**Adedi**
İlk/Son DNS Paket Zamanı	Sayısal	2
Toplam DNS Trafik Boyutu (bayt)	Sayısal	1
DNS İsteklerinde Yer Alan Tekil Alan Adı Adedi	Sayısal	1
Ortalama DNS Alan Adı Uzunlukları	Sayısal	1
Toplam DNS İstekleri/Cevapları	Sayısal	2
Toplam DNS Paket Boyutları (bayt)	Sayısal	1

Bir Android uygulamasında yer alan zararlı kod parçası, çalışma zamanında farklı anlarda tetiklenebilmektedir [24]. Bir zararlı yazılım mobil cihaz başlatıldığında veya bir belirli bir süre sonunda çalışmaya başlayabilmektedir. Yapılan çalışmalarda kötücül bir uygulama dinamik analiz araçları ile genellikle 10 dakikalık bir zaman diliminde incelenmektedir [64]. Bu sebeple saldırganlar belirli dinamik analiz sürelerini atlatabilmek için sanal/gerçek bir cihazda zararlı kod bloklarının tetiklenmesini erteleyebilir. Bu nedenle çalışmada dinamik analiz süresi 15 dakika olarak belirlenmiştir. Toplam analiz süresi 5'er dakikalık periyotlar halinde de öznitelik kümesine dahil edilmiştir. Kötücül bir uygulama uzak sunucu ile eğer TCP protokolü üzerinden haberleşiyorsa 3'lü el sıkışma başlatmalıdır. 3'lü el sıkışma sürecinin ilki olan SYN paket gönderimleridir. Her yeni TCP bağlantısının başlangıcını oluşturması sebeple gönderilen SYN paket adedi de ağ tabanlı öznitelik kümesine eklenmiştir. Diğer yandan kötücül uygulamalar, veri sızdırma işlemi sırasında ağ tabanlı güvenlik cihazlarına yakalanmamak için SSL protokolünü kullanarak şifreli bir bağlantı oluşturabilirler. SSL kullanımı da öznitelik kümesi içerisindedir. Çizelge 11'de HTTP ve DNS protokolleri dışında diğer ağ tabanlı öznitelikler görülebilmektedir. İlgili öznitelikler de UpDroid [8] çalışmasında kullanılmamış olup, bu tez çalışmasında kullanılan yeni özniteliklerdir.

Çizelge 11. Diğer Ağ Tabanlı Öznitelikler.

Ağ Tabanlı Diğer Öznitelikler	Değişken Türü	Adedi
5 dk. Periyot – Toplam Ağ Trafik Boyutu (bayt)	Sayısal	3
Toplam Ağ Trafik Boyutu *(bayt)*	Sayısal	2
Toplam UDP Trafik Boyutu *(bayt)*	Sayısal	1
UDP Frame Adedi	Sayısal	1
SYN Paket Adedi	Sayısal	1
Tekil IP/Port Adedi	Sayısal	2
Başlatılan/Sonlanan Ağ Bağlantı Adedi	Sayısal	2
Gönderilen/Alınan Ağ Paket Adedi	Sayısal	2
SSL Kullanımı	Mantıksal	1
Ortalama IP Paket Boyutu (bayt)	Sayısal	1

Dinamik analiz çıktıları içerisinde daha çok ağ davranışları ve aktivite sıralı tekrarları üzerinde durulmuş ve kötücül uygulamaların aile sınıflandırılmasındaki etkileri incelenmiştir. Kötücül uygulamalar, kişisel ve/veya mobil cihaz bilgilerini ardışık aktiviteler gerçekleştirerek belirli desenler üzerinden sızdırabilir (ör. belirli bir dosyanın okunması ve verinin ağ üzerinden gönderilmesi). Bu sebeple uygulamalar tarafından gerçekleştirilen ardışık ikili aktivite tekrarları bu çalışmada dikkate alınmıştır. Bu çalışma ile ardışık ikili aktiviteler ilk kez Android kötücül yazılım aile sınıflandırmasında önerilmiştir. İkili aktivite tablosu oluşturulurken aşağıdaki Çizelge 12'de yer alan aktiviteler ele alınmıştır. Bu anlamda 100 (10x10) adet öznitelik, küme içerisine eklenmiştir. Bu öznitelikler ilk kez bu çalışmada kullanılmıştır. Özetle, UpDroid [8] çalışmasında verilen 175 öznitelikleri de içerecek şekilde bu çalışmada toplam 329 öznitelik kullanılmıştır.

Çizelge 12. Ardışık İkili Aktiviteler Kümesi.

Aktivite Adı	Açıklamalar
cryptousage	Android API Destekli Kriptografik Algoritma Kullanımı
dataleaks	Dosya, Ağ veya SMS Üzerinden Veri İfşası
dexclass	DexClassLoader Sınıfı Kullanılarak Dinamik Kod Yükleme
fread	Dosya Okuma
fwrite	Dosya Yazma
recvnet	Kabul Edilen Ağ Bağlantı Adedi
sendnet	Ağ Bağlantı İstek Gönderim Adedi
runtimesystemevents	Android Sistem Olayları
sendsms	SMS Gönderimi
servicestart	Başlatılan Servisler

UpDroid [8] çalışmasında kullanılan öznitelikler ile tüm öznitelikler çalışma içerisinde adlandırılmış halleri ile 7.1. Ek 1 başlığı altındaki detaylı listede görülebilmektedir.

5. DENEYLER VE SONUÇLAR

5.1. Deney Ortamı

Deneyler, Malgenome [21] veri kümesinden 42 kötücül yazılım ailesine ait 1260, Malgenome [21] veri kümesi genişletilerek oluşturulan Drebin [32] veri kümesinden 178 aileye ait 5554 ve UpDroid [55] veri kümesinden 21 aileye ait 2535 adet kötücül yazılım üzerinde yapılmıştır. Deneylerde, çok fazla örneğe sahip kötücül yazılım ailelerinin, sonuçlar üzerinde daha fazla etkisi bulunmaktadır. Bu sebeple yalnızca 10'dan veya yalnızca 20'den fazla örneğe sahip aileler değerlendirilerek testler yapılmıştır. UpDroid [8] çalışması ile yapılan karşılaştırmada UpDroid [55] veri kümesi üzerinde 10 katmanlı çapraz doğrulama yöntemi ile eğitme/test işlemleri yapılmış ve doğruluk oranı hesaplanmıştır.

Deneyler 4 farklı makine öğrenmesi algoritması ile yapılmıştır. Bu algoritmaların kullanımları sırasında Weka [58] aracında kullanılan ve önem arz eden algoritma parametreleri aşağıdaki gibidir. KNN algoritması için Weka [58] aracında kullanılan değişkenler Çizelge 13'teki gibidir. Bu değişkenler içerisinde önemli olanlardan bazıları şunlardır: sınıflandırma yaparken yalnızca en yakın 1 adet komşu aranmış ve Öklid Mesafesi denklemi kullanılmıştır.

Çizelge 13. KNN Algoritması Weka [58] Parametreleri.

KNN Parametreleri	Değer
KNN	1
crossValidate	False
debug	False
distanceWeighting	No distance weighting
meanSquared	False
nearestNeighboutSearchAlgorithm	LinearNNSearch -A "weka.core.EuclideanDistance"

windowSize	0

Çizelge 14'te DT algoritması için Weka [58] aracında kullanılan parametrelerin tamamı verilmiştir. Burada bir karar ağacı için öne çıkan parametrelerden biri, en az dal sayısını ifade eden "minNumObj"'dir. Bu deneyde bu değer 2 olarak alınmıştır. Bunun yanında "subtreeRaising" değerinin "True" olması ile ağaç yapısında ara katmanda bulunan alt ağaçlar budanır. Buradaki amaç sınıflandırma için oluşturulan ağaçtaki karar seviyelerinin optimum düzeye getirilmesidir.

Çizelge 14. DT Algoritması Weka [58] Parametreleri.

DT Parametreleri	**Değer**
binarySplits	False
confidenceFactor	0.25
debug	False
minNumObj	2
numFolds	3
reducedErrorPruning	False
saveInstanceData	False
seed	1
subtreeRaising	True
unpruned	False
useLaplace	False

RF algoritmasına ait Weka [58] aracında kullanılan parametreler, Çizelge 15'te verilmiştir. RF algoritması temelde birbirinden bağımsız birden fazla ağaç yapısının bir arada kullanılmasına dayandığı için "numTrees" değeri tabloda belirtilmiştir. Bu değer, Weka [58] aracındaki varsayılan değer olup, 10 olarak alınmıştır. Yüksek sayıda ağaç olması, çalışma süresini arttırmakta, düşük sayıda ağacın olması ise modelin doğruluk oranında azalmaya

sebep olmaktadır. “maxDepth” parametresi, ağaçlardaki maksimum derinliği vermektedir. 0 ifadesi Weka [58] aracı için sonsuz anlamına gelmektedir, dolayısıyla bir kısıt verilmemiştir. Ağaç oluşumu tamamlandıktan sonra, yaprak düğümlerden kök düğüme kadar dallar üzerinde budama işlemi yapılır. “confidenceFactor” değeri, budama işlemi sırasında referans alınacak değeri ifade etmektedir. Bu değerin düşük olması ağaç üzerinde daha fazla dalın budanması anlamına gelmektedir.

Çizelge 15. RF Algoritması Weka [58] Parametreleri.

RF Parametreleri	Değer
Debug	False
maxDepth	0
numFeatures	0
numTrees	10
seed	1

Çizelge 16’da SVM algoritması için kullanılan Weka [58] parametreleri sunulmuştur. Karmaşıklık parametresi olarak adlandırılan “c” değeri, sınıfları ayırmak için vektör düzleminde çizgi çizme sürecinin ne kadar esnek olabileceğini kontrol eder. 0 değeri daha doğrusal çizgiler ile algoritmanın kullanılmasını sağlar. “c” parametresi için Weka [58] aracında varsayılan olarak 1 alınmıştır. Sınıflara ayrıştırmada kullanılan bir diğer önemli parametre ise kullanılan çekirdek (kernel) tipidir. En basit çekirdek tipi, verileri düz bir çizgi ile ayıran bir doğrusal çekirdektir (linear kernel). Bu değer Weka [58] aracında varsayılan olarak sınıfları eğri veya doğrusal olmayan bir çizgi kullanarak ayıran polinom çekirdeğidir (PolyKernel).

Çizelge 16. SVM Algoritması Weka [58] Parametreleri.

SVM Parametreleri	Değer
buildLogisticModels	False
c	1.0

checksTurnedOff	False
debug	False
epsilon	1.0E-12
filterType	Normalize training data
kernel	PolyKernel -C 250007 -E 1.0
numFolds	-1
randomSeed	1
toleranceParameter	0.001

Modelin performansının hesaplanması sırasında deneylerde kullanılan ölçüm değerleri aşağıda açıklanan ve formülleri belirtilen denklemler ile hesaplanmıştır.

Doğruluk: Doğru şekilde sınıflandırılan uygulamaların toplam sayısının bütün veri kümesine olan oranı ile elde edilmektedir.

Kesinlik: Doğru sınıflandırılan kötücül amaçlı uygulamaların kötücül olarak sınıflandırılmış bütün uygulamalara oranı ile elde edilmektedir.

Duyarlılık: Doğru olarak sınıflandırılması gereken kötücül uygulamaların ne kadarının doğru bir şekilde sınıflandırıldığı bilgisini barındırır.

$$Doğruluk\ (Accuracy) = \frac{Tp + Tn}{Tp + Tn + Fp + Fn} \quad Kesinlik\ (Precision) = \frac{Tp}{Tp + Fp}$$

$$Duyarlılık\ (Recall) = \frac{Tp}{Tp + Fn}$$

TP, FP, FN ve TN oranları aşağıda belirtilen formüller ile hesaplanmıştır. Bu ölçüm değerlerine ait açıklamalar aşağıdaki gibidir;

- TP: Bir sınıfa ait olduğu bilinen bir verinin model tarafından doğru bir şekilde tahmin edildiği durumu ifade eder. Örneğin, Asacub ailesindeki bir kötücül yazılımın, model tarafından Asacub olarak tahmin edildiği durumdur.

- FP: Bir sınıfa ait olduğu bilinen bir verinin model tarafından yanlış bir şekilde tahmin edildiği durumu ifade eder. Örneğin, Bankbot ailesindeki bir kötücül yazılımın model tarafından yanlışlıkla Asacub olarak tahmin edildiği durum Asacub sınıfı için FP adedini ifade eder.

- TN: Bir sınıfa ait olduğu bilinen bir verinin model tarafından doğru bir şekilde tahmin edildiği durumu ifade eder. Örneğin, Asacub ailesi dışındaki doğru tahmin edilen kötücül yazılım adedi Asacub sınıfı için TN adedini ifade eder.

- FN: Bir sınıfa ait olduğu bilinen bir verinin model tarafından yanlış bir şekilde tahmin edildiği durumu ifade eder. Örneğin, Asacub ailesindeki bir kötücül yazılımın model tarafından yanlışlıkla Bankbot olarak tahmin edildiği durum Asacub sınıfı için FN adedini ifade eder.

$$TP\ Oranı = \frac{Tp}{Tp + Fn} \qquad FP\ Oranı = \frac{Fp}{Fp + Tn}$$

$$TN\ Oranı = \frac{Tn}{Fp + Tn} \qquad FN\ Oranı = \frac{Fn}{Fn + Tp}$$

Ec2 [7] çalışması ile yapılan karşılaştırmada MiF ve MiAUC metrikleri kullanılmıştır. MiF değeri, her bir kötücül yazılım ailesi için hesaplanan kesinlik ve duyarlılık değerlerinin ortalaması üzerinden aşağıdaki formüller ile hesaplanır. Bazı çalışmalarda, Ec2 [7] çalışmasında da olduğu gibi önerilen modelin performansı doğruluk (accuracy) yerine MiF ve MiAUC ölçüm değerleri üzerinden hesaplanmıştır. Her bir sınıftaki veri miktarının homojen dağılmaması, bu metrik üzerinden değerlendirme yapılmasının sebeplerinden biridir.

$$MiP = \frac{\sum_{i=1}^{n} TPi}{\sum_{i=1}^{n} TPi + FPi} \qquad MiR = \frac{\sum_{i=1}^{n} TPi}{\sum_{i=1}^{n} TPi + FNi}$$

$$Micro\ F-Score\ (MiF) = 2\ x\ \frac{MiP\ x\ MiR}{MiP + MiR}$$

5.2. Kötücül Yazılım Aile Sınıflandırması Deney Sonuçları

UpDroid [8] çalışmasında kullanılan öznitelikler ile tüm öznitelikler ile yapılan testlerin sonuçları Çizelge 17'de görülebilmektedir. Çizelge 17'de elde edilen sonuçlarda 10 katmanlı çapraz doğrulama kullanılmıştır. Karşılaştırma için 3 farklı metrik kullanılmıştır; doğruluk, TP veya FP oranları. Elde edilen sonuçlar tüm öznitelikler kullanılarak elde edilmiştir. En yüksek doğruluk oranı, KNN algoritması ile elde edilmiştir.

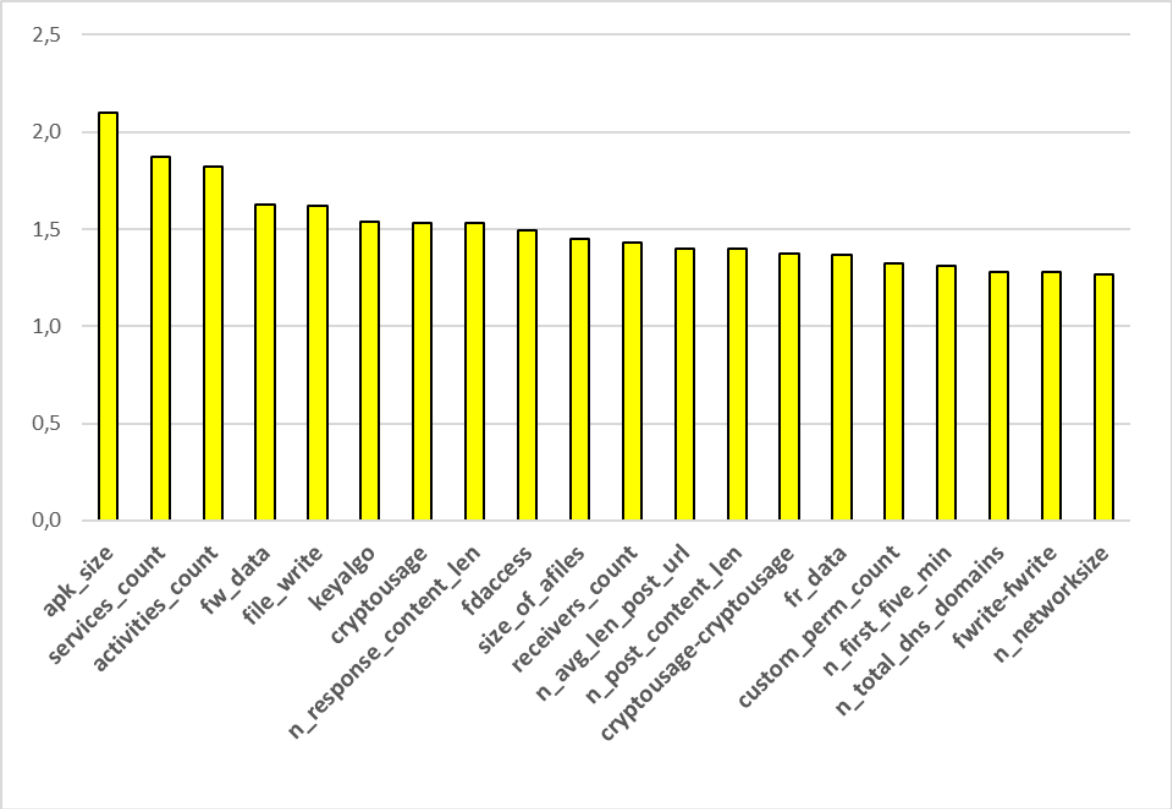
Çizelge 17. UpDroid [8] Çalışması ile Karşılaştırma.

Çalışma	Algoritma	Doğruluk %	TP %	FP %
Önerilen Model	KNN	**98,04**	98,00	0,40
UpDroid [8]	KNN	96,85	96,40	0,40

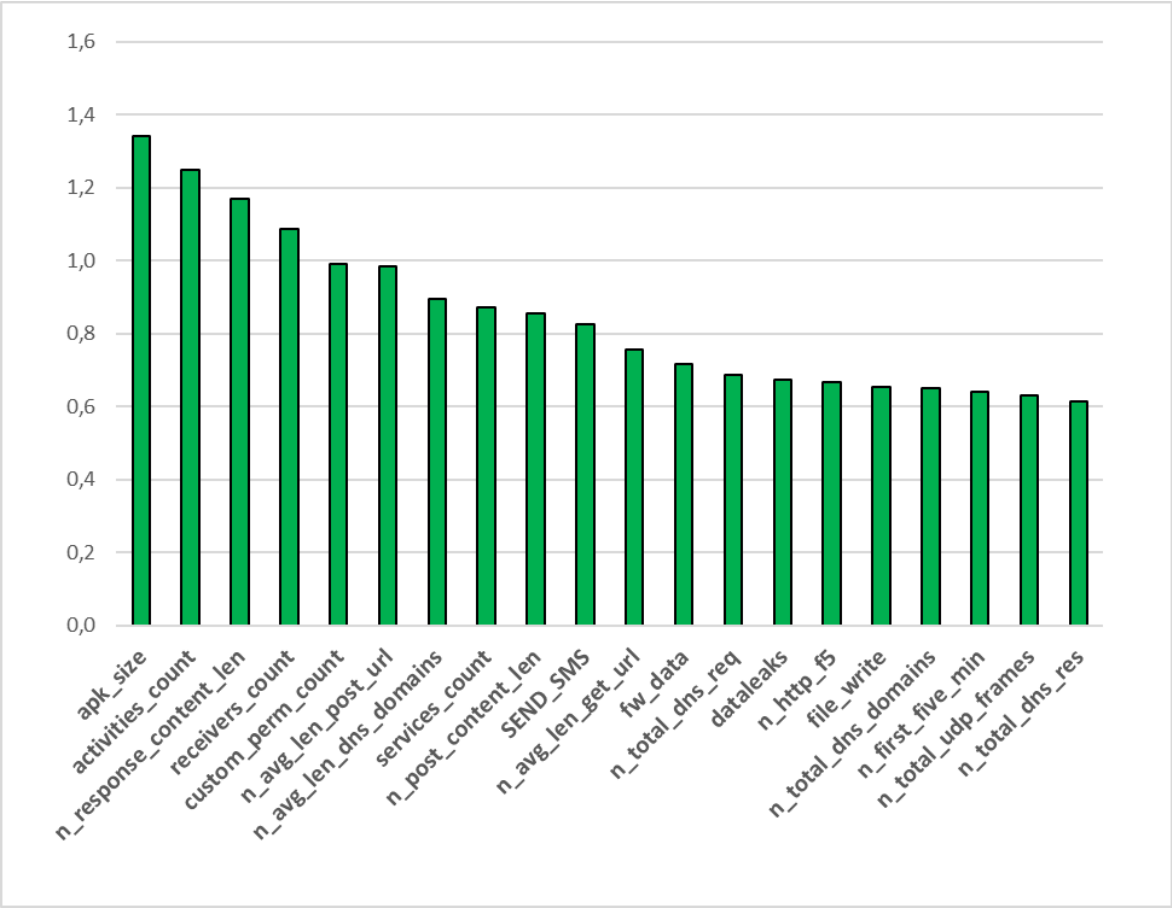
Model içerisinde kullanılan özniteliklerin tamamının 3 farklı veri kümesi içinde bilgi kazanımı (information gain) tabanında ayırt edicilik değerleri karşılaştırılmıştır. Hesaplanan değerler 0 (bilgi yok) ila 1 (maksimum bilgi) arasında değişmektedir. Bir sınıfın tespit edilmesinde daha fazla bilgi ile katkıda bulunan öznitelikler daha yüksek bir bilgi kazanım değerine sahiptir. İlgili değerlerin hesaplanabilmesi için Weka [58] uygulamasında, "InfoGainAttributeEval" öznitelik değerlendiricisi kullanılmış ve öznitelikler bilgi kazanımına göre en yüksekten en düşüğe kadar sıralanmıştır.

Şekil 13, Şekil 14 ve Şekil 15'de testler sırasında sınıflandırmada en etkili 20 adet öznitelik UpDroid [55], Malgenome [21] ve Drebin [32] veri kümeleri için ayrı ayrı gösterilmiştir. "n_" ifadesi ile başlayan öznitelikler ağ tabanlı öznitelikleri temsil etmektedir. Bütün veri kümeleri için ilk 20'de ağ tabanlı öznitelikler içerisinden örnekler önemli sayıda yer almaktadır. Kötücül uygulamaların dinamik analizi esnasında ilk 5 dakika içerisinde ürettiği trafik boyutunu ifade eden "n_first_five_min" özniteliğinin her üç veri kümesi dahilinde yapılan testlere göre ilk 20'de yer aldığı görülmektedir. Bu durumdan, kötücül yazılımların

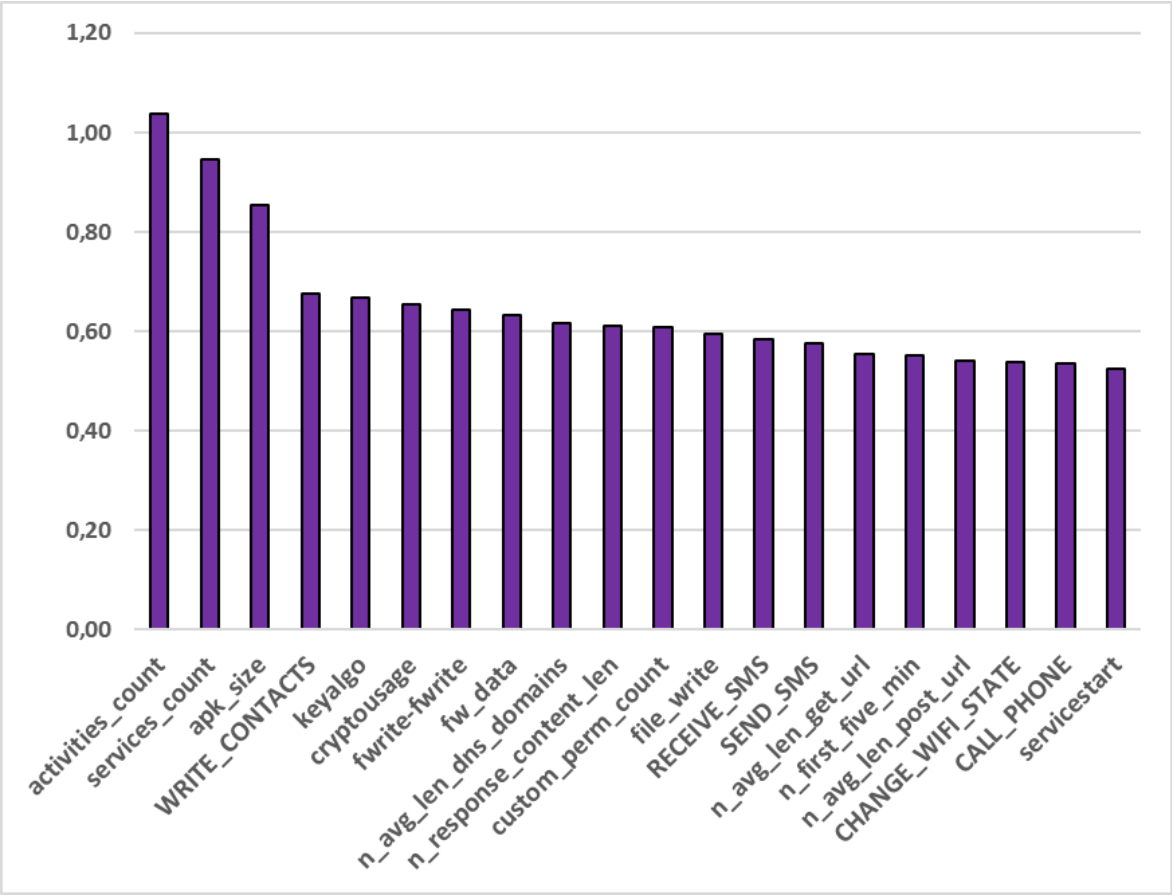
karakteristik davranışlarını genellikle ilk 5 dakika içerisinde ortaya çıkardığı sonucuna varılabilir. Bunun yanında GET ve POST isteklerinde ortalama URL uzunluğu, toplam DNS sorgusu yapılan alan adı sayısı bilgisi ilk 20 sınıflandırıcı öznitelik arasında yer almaktadır. Ağ tabanlı özniteliklerin yanında UpDroid [55] ve Malgenome [21] veri kümeleri için ardışık ikili aktivite tekrarlarından kötücül uygulamanın işletim sistemi içerisinde ardışık zamanda 2 kez art arda dosya yazma işlemi yaptığını ifade eden "fwrite-fwrite" özniteliği de ilk 20 öznitelik arasında görülmektedir.



Şekil 13. UpDroid [55] Veri Kümesi için En Etkili 20 Öznitelik.



Şekil 14. Drebin [32] Veri Kümesi için En Etkili 20 Öznitelik.



Şekil 15. Malgenome [21] Veri Kümesi için En Etkili 20 Öznitelik.

Çizelge 18'de, Çizelge 17'de elde edilen sonuca ait olup, UpDroid [8] veri kümesi için yapılan testlerdeki karmaşıklık matrisini göstermektedir. Mavi renkli kutular doğru bir şekilde sınıflandırması yapılmış örnek sayılarını aile bazında göstermektedir. Karmaşıklık matrisi testler sonucunda sınıflar arası geçişleri göstermektedir. Çizelge 18'de görüleceği üzere, UpDroid [55] veri kümesinde hatalı sınıflandırma ile karşılaşılmayan, en yüksek TP oranına Tordow kötücül yazılım ailesinde ulaşılmıştır. Tordow ailesine ait kötücül yazılımlar genellikle hedef cihaza yüklenmeleri sonrası yeni kötücül uygulamalar indirmektedir [65]. UpDroid [55] veri kümesine en çok adede sahip aileler Shedun ve Triada'dır, bu ailelerde yüksek TP değerlerine ulaşılmıştır. Çizelge 18'de göze çarpan geçişler Triada ve Smsreg ailelerinde olmuştur. [66] ve [67] çalışmalarında belirtildiği üzere

her iki ailede yer alan kötücül uygulamalar yüklendikleri mobil cihaza zararsız reklam uygulamaları (adware) yüklemektedir.

Çizelge 18. UpDroid [55] Veri Kümesi için Karmaşıklık (Confusion) Matrisi.

a	b	c	d	e	f	g	h	i	j	k	l	m	← Sınıflandırma
51	1	1	1	0	0	0	0	0	0	0	0	0	**a = asacub**
3	**20**	0	0	1	0	0	0	0	0	0	0	0	**b = bankbot**
2	0	**9**	0	0	0	0	0	0	0	0	0	0	**c = faketoken**
0	0	0	**178**	0	1	0	1	1	0	0	0	0	**d = malap**
0	1	0	0	**20**	0	0	0	0	0	0	0	0	**e = marcher**
0	0	0	1	0	**8**	1	0	0	0	0	0	0	**f = ogel**
0	0	0	0	0	0	**27**	0	2	0	0	3	0	**g = rootnik**
0	0	0	0	0	0	0	**629**	1	0	0	0	0	**h = shedun**
0	0	0	1	2	0	1	0	**282**	0	0	5	0	**i = smsreg**
3	0	0	0	1	0	0	0	0	**53**	0	0	0	**j = smsspy**
0	0	0	0	0	0	0	0	0	0	**11**	0	0	**k = tordow**
0	0	0	1	0	0	1	0	6	1	0	**1000**	1	**l = triada**
0	0	0	0	1	0	0	0	0	0	0	1	**13**	**m = ztorg**

Çizelge 20'de üç ayrı veri kümesi içerisinde 20 ve üzeri örnekleme sahip ailelerde dinamik analiz sırasında elde edilen ağ trafik büyüklükleri hakkında bilgi verilmiştir**.** Çizelge 20'de verilen değerler bayt cinsindendir. Drebin [32] ve Malgenome [21] veri kümeleri için en az bir örneklem için tekil olarak en çok yaklaşık 13 kilobaytlık bir ağ trafiği gözlemlenirken UpDroid [55] veri kümesinde bu değer 40 katın üzerinde olup yaklaşık 521 kilobayttır. Aynı şekilde UpDroid [55] veri kümesinde her örneklem için ortalama yaklaşık 5,8 kilobaytlık bir ağ trafiği gözlemlenirken, Drebin [32] ve Malgenome [21] veri kümelerinde bu değer yalnızca sırasıyla yaklaşık 161 (~40 kat) ve 206 (~30 kat) bayt olarak tespit edilmiştir. Bu çalışmada yer alan yeni özniteliklerin etkileri farklı kombinasyonlar uygulanarak 3 veri

kümesinde de test edilmiştir. UpDroid [55] veri kümesindeki örnekler Drebin [32] veri kümesindeki örneklere göre güncelleme saldırılarını, dinamik kod yükleme aktivitelerini daha fazla içermeleri sebebiyle daha yoğun ağ trafiği oluşturmakta ve yalnızca ağ tabanlı öznitelikler ile yapılan testlerde bu bağlamda UpDroid [55] veri kümesi üzerinde daha olumlu sonuçlar elde edildiği Çizelge 22'de görülmektedir. Çizelge 20'deki testler sırasında, Çizelge 17'de görüldüğü üzere daha yüksek sonuç üreten KNN makine öğrenmesi algoritması ve eğitme/test için 10 katmanlı çapraz doğrulama yöntemi kullanılmıştır. Sonuçlara göre yalnızca 55 farklı ağ tabanlı öznitelik kullanılarak yapılan testlerde %90'nın üzerinde doğruluk değeri elde edilmiştir. Çizelge 20'de UpDroid [55] veri kümesindeki kötücül uygulamaların Drebin [32] ve Malgenome [21] veri kümelerindeki kötücül uygulamalardan daha yoğun ağ etkinliklerine sahip olduğu gösterilmiştir. Benzer şekilde, yeni eklenen öznitelikler olan ardışık ikili aktivite tekrarları da UpDroid [55] veri kümesi üzerinde diğer veri kümelerine göre daha olumlu etki göstermiştir. UpDroid [55] ve Drebin [32] veri kümesinde en çok rastlanan ardışık ikili aktivite tekrarı "fread-fread" olurken Malgenome [21] veri kümesinde "fwrite-fwrite" olduğu görülmüştür. Çizelge 19'da veri kümelerine göre en çok rastlanılan ilk 3 ardışık ikili aktivite tekrarları belirtilmiştir. Ağ trafiğinin daha yoğun olması sebebiyle "recvnet-recvnet" özniteliği UpDroid [55] veri kümesinde ikinci sırada bulunmaktadır.

Çizelge 19. En Çok Rastlanılan İlk 3 Ardışık İkili Aktiviteler.

Veri Kümesi	**1.**	**2.**	**3.**
UpDroid [55]	fread-fread	recvnet- recvnet	fwrite-fwrite
Drebin [32]	fread-fread	fwrite-fwrite	recvnet- recvnet
Malgenome [21]	fwrite-fwrite	fread-fread	recvnet- recvnet

Çizelge 20. Veri Kümelerinde Ağ Trafik Büyüklüğü Değerleri.

Veri Kümesi	**Minimum (Bayt)**	**Maksimum (Bayt)**	**Ortalama (Bayt)**	**Standart Sapma (Bayt)**
UpDroid [55]	0	520.495	5.799,00	29.345,69

Drebin [32]	0	12.932	161,46	625,04
Malgenome [21]	0	12.932	206,59	855,46

Çizelge 21. Bütün Öznitelikler ile Yapılan Testlerin Sonuçları.

	Veri Kümeleri	**KNN**	**RF**	**DT**	**SVM**
Doğruluk %	Malgenome [21]	97,38	93,49	96,89	97,83
	UpDroid [55]	98,04	97,44	97,53	97,44
	Drebin [32]	96,40	92,33	93,58	95,77
TP %	Malgenome [21]	97,40	93,50	96,90	97,80
	UpDroid [55]	98,00	97,40	97,50	97,40
	Drebin [32]	96,40	92,30	93,60	95,80
FP %	Malgenome [21]	1,40	3,90	1,30	0,80
	UpDroid [55]	0,40	0,60	0,40	0,80
	Drebin [32]	0,30	0,90	0,50	0,40

Çizelge 22. Farklı Öznitelik Gruplarının Başarıma Oranı

	VERİ KÜMELERİ	**Ağ Tabanlı**	**İkili Aktivite Tekrarları**	**Ağ Tabanlı + İkili Aktivite Tekrarları**	**UpDroid Öznitelikleri + Ağ Tabanlı**	**UpDroid Öznitelikleri + İkili Aktivite Tekrarları**	**Tüm Öznitelikler**
Doğruluk %	Malgenome [21]	73,99	75,86	79,59	97,45	97,73	97,38
	UpDroid [55]	91,72	92,61	92,78	98,17	98,04	98,04
	Drebin [32]	61,38	70,59	77,11	95,92	96,32	96,40
TP %	Malgenome [21]	74,00	75,90	79,60	97,50	97,70	97,40
	UpDroid [55]	91,70	92,60	92,80	98,20	98,00	98,00
	Drebin [32]	61,40	70,60	77,10	95,90	96,30	96,40

FP %	Malgenome [21]	8,80	8,60	8,10	1,30	1,10	1,40
	UpDroid [55]	2,30	2,00	2,00	0,40	0,30	0,40
	Drebin [32]	4,30	2,60	2,10	0,30	0,30	0,30

Son olarak önerilen yaklaşım, Android kötücül yazılım aile sınıflandırması için hibrit öznitelikler kullanan iki çalışma olan Ec2 [7] ve UpDroid [8] çalışmaları ile aynı performans metrikleri kullanılarak karşılaştırılmıştır. Adil bir karşılaştırma yapmak için ilgili çalışmalarda kullanılan Drebin [32] veri kümesi kullanılmıştır. Ec2 [7] çalışmasında performans değerlendirmesi için sırasıyla MiF ve MiAUC metrikleri kullanılmıştır. Aynı şekilde UpDroid [8] çalışmasında da karşılaştırma için bu metrikler kullanılmıştır. Bu çalışmada önerilen modelin, Ec2 [7] çalışmasından daha iyi performans gösterdiği ve UpDroid [8] çalışması ile benzer sonuçlara ulaştığı Çizelge 23'te gösterilmiştir.

Çizelge 23. UpDroid [8], Ec2 [7] Çalışmaları ile Karşılaştırma.

Çalışma	**Algoritma**	**MiF**	**MiAUC**
Ec2 [7]	KNN	0,47	0,73
UpDroid [8]	KNN	0,96	0,98
Önerilen Model	KNN	**0,96**	**0,98**
Ec2 [7]	RF	0,95	0,97
UpDroid [8]	RF	0,94	0,99
Önerilen Model	RF	0,94	0,99

Deneyler sırasında yapılan bir diğer değerlendirme, Çizelge 26'da görülebilmektedir. Drebin [32], Malgenome [21], UpDroid [55] olmak üzere bütün veri kümeleri ile beraber 10 ve 20 katmanlı çapraz doğrulama ve 4 farklı algoritma kullanılarak eğitme ve test adımları gerçekleştirilmiştir. Malgenome [21] veri kümesi tümüyle Drebin [32] veri kümesi içerisinde de bulunmaktadır. Bunun yanında Drebin [32] ve UpDroid [55] veri kümelerinde gerçekleştirilen testler sırasında ele alınan kötücül yazılım aileleri arasında SMSreg ve SmsSpy olmak üzere yalnızca iki ailenin ortak olduğu görülmüştür. Çizelge 24'te yapılan

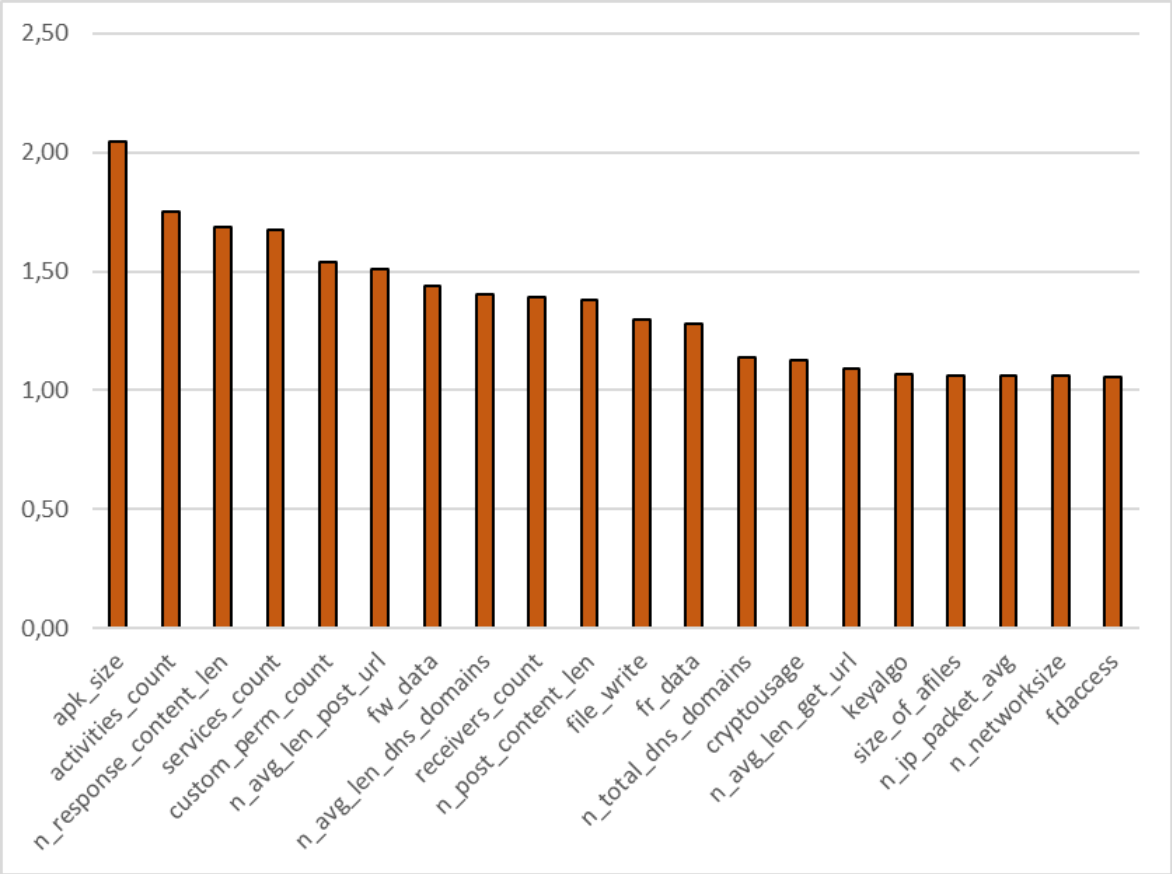
eğitme ve testler için kullanılan veri kümesinin aile dağılımı görülebilmektedir. Bir arada yapılan testte, bu ailelere ait TP değerlerinin arttığı görülmüştür.

Çizelge 24. Drebin [32], UpDroid [55] ve Malgenome [21] Veri Kümelerinin Birleşiminden Oluşan Veri Kümesi.

Aile	**Adet**	**Aile**	**Adet**
Adrd	78	Iconosys	135
Asacub	54	Imlog	41
Bankbot	24	Jifake	28
BaseBridge	309	Kmin	86
Boxer	25	Malap	181
DroidDream	75	Marcher	21
DroidKungFu	643	MobileTx	67
ExploitLinuxLotoor	66	Opfake	593
FakeDoc	130	Plankton	545
FakeInstaller	911	Rootnik	32
FakeRun	59	SendPay	56
Gappusin	46	Shedun	630
Geinimi	79	SMSreg	331
GinMaster	333	Smsspy	58
Glodream	67	Triada	1010
Hamob	28	Yzhc	36
		Toplam	**6777**

Çizelge 25'te görüldüğü üzere, Çizelge 24'te belirtilen örneklemleri içeren bütün veri kümeleri üzerinde yapılan testlerde en yüksek doğruluk oranına 20 katmanlı çapraz doğrulamada KNN algoritması ile ulaşılmıştır. Çizelge 25'te görüldüğü üzere, bu veri kümesi ile %97,40'luk doğruluk oranı elde edilmiştir. Bu teste ait aileler arası geçişleri ifade

eden karmaşıklık matrisi, Çizelge 27'de görülebilmektedir. Bu veri kümesi ile yapılan testler sırasında kullanılan özniteliklerin, Weka [58] aracında "InfoGainAttributeEval" yöntemi ile bilgi kazanımlarına göre yapılan sıralaması Şekil 16'da verilmiştir. Şekil 13, Şekil 14 ve Şekil 15'teki sonuçlara benzer şekilde, ağ tabanlı öznitelikler ilk 20 öznitelik içerisinde görülmektedir.



Şekil 16. Drebin [32], Malgenome [21] ve UpDroid [55] Birleşiminden Oluşan Veri Kümesindeki En Etkili 20 Öznitelik.

Çizelge 25. Bütün Veri Kümeleri ile Tek Seferde Yapılan Eğitme/Test Sonuçları.

	10 Kat. Çapraz Doğ.			**20 Kat. Çapraz Doğ.**		
Algoritmalar	**Doğruluk %**	**TP %**	**FP %**	**Doğruluk %**	**TP %**	**FP %**
KNN	**96,85**	96,90	0,20	**97,40**	97,40	0,20
RF	93,66	93,70	0,50	95,16	95,20	0,40

DT	94,33	94,30	0,30	95,68	95,70	0,30
SVM	96,08	96,10	0,30	96,74	96,70	0,30

Çizelge 26. Bütün Veri Kümeleri ile Tek Seferde Yapılan Eğitme/Test Sonuçlarının Öznitelik Bazlı Değerlendirilmesi.

Algoritmalar		**Ağ Tabanlı**	**İkili Aktivite Tekrarları**	**Ağ Tabanlı + İkili Aktivite Tekrarları**	**UpDroid Öznitelikleri + Ağ Tabanlı**	**UpDroid Öznitelikleri + İkili Aktivite Tekrarları**
KNN	**Doğruluk %**	74,53	79,57	84,75	**97,19**	**97,48**
	TP %	74,50	79,60	84,80	97,20	97,50
	FP %	2,20	1,40	1,10	0,20	0,20
DT	**Doğruluk %**	**78,14**	79,77	84,81	95,38	96,00
	TP %	78,10	79,80	84,80	95,40	96,60
	FP %	2,20	1,40	1,10	0,30	0,30
RF	**Doğruluk %**	76,97	**82,30**	**86,90**	95,57	95,78
	TP %	77,00	82,30	86,60	95,60	95,80
	FP %	2,10	1,50	1,20	0,40	0,40
SVM	**Doğruluk %**	63,01	38,75	67,57	96,63	96,56
	TP %	63,00	38,70	67,60	96,60	96,60
	FP %	4,90	9,60	4,30%	0,30	0,30

Çizelge 27. Drebin [32], Malgenome [21] ve UpDroid [55] Birleşiminden Oluşan Veri Kümesinde Yapılan Teste ait Karmaşıklık (Confusion) Matrisi.

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	aa	ab	ac	ad	ae	af	<-- sınıflandırma
73	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	a=Adrd
0	52	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	b=Asacub
0	2	20	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	c=Bankbot
0	0	0	295	1	1	1	3	0	0	0	2	0	0	0	3	0	1	0	0	0	0	0	0	2	0	0	0	0	0	0	0	d=BaseBridge
0	0	0	1	14	0	0	0	0	6	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	e=Boxer
0	0	0	0	0	69	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	0	f=DroidDream
0	0	0	0	0	0	635	0	0	0	0	2	0	2	0	0	0	0	0	0	0	0	0	0	3	0	0	0	0	0	0	1	g=DroidKungFu
0	0	0	1	0	0	4	52	1	0	0	3	0	1	0	1	0	0	2	0	0	0	0	0	1	0	0	0	0	0	0	0	h=ExploitLinuxLotoor
0	1	0	0	0	0	0	0	129	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	i=FakeDoc
1	0	0	0	6	0	0	0	0	897	0	0	0	0	0	0	0	0	0	0	0	0	0	6	0	0	0	0	1	0	0	0	j=FakeInstaller
0	0	0	0	0	0	0	0	0	0	58	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	k=FakeRun
0	0	0	1	0	1	4	0	0	0	0	37	0	2	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	l=Gappusin
0	0	0	0	0	0	0	0	0	0	0	0	79	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	m=Geinimi
0	0	0	0	0	0	0	1	0	0	0	1	0	329	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	n=GinMaster
0	0	0	0	0	0	0	0	0	0	0	0	0	1	64	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	o=Glodream
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	28	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	p=Hamob
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	134	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	q=Iconosys
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	40	0	0	0	0	0	0	0	0	0	0	0	0	0	0	r=Imlog
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	26	0	0	0	0	1	0	0	0	0	0	0	0	0	s=Jifake
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	86	0	0	0	0	0	0	0	0	0	0	0	0	t=Kmin
0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0	0	0	176	0	0	0	0	0	2	0	1	0	0	0	u=Malap
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	20	0	0	0	0	0	0	0	0	0	0	v=Marcher
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	67	0	0	0	0	0	0	0	0	0	w=MobileTx
0	0	0	0	0	0	0	0	0	9	0	0	0	0	0	0	0	0	1	0	0	0	0	583	0	0	0	0	0	0	0	0	x=Opfake
0	0	0	0	0	0	0	1	0	0	4	0	0	1	1	1	0	0	0	0	0	0	0	0	528	0	0	0	0	0	1	0	y=Plankton
0	0	0	2	0	0	6	1	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0	27	0	0	2	0	2	0	z=Rootnik
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	629	0	1	0	0	0	aa=Shedun
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	55	0	0	0	0	ab=SendPay
0	0	0	1	2	0	0	0	0	2	0	0	0	1	0	0	0	0	0	0	0	2	0	0	3	1	0	0	313	0	6	0	ac=SMSreg
1	1	1	0	0	0	0	0	0	3	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	51	0	0	ad=SmsSpy
0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	0	0	7	0	999	0	ae=Triada
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	36	af=Yzhc

5.3. Kötücül Olmayan Yazılımlar ile Gerçekleştirilen Deneylerin Sonuçları

Android kötücül yazılım tespiti bağlamında bu çalışmada kullanılan uygulama izinleri, kaynak dosyaları, DexClassLoader kullanımı [68] gibi özniteliklerin kullanıldığı çalışmalar bulunmaktadır [69]. Bu bağlamda bir değerlendirme yapmak amacıyla çalışmada önerilen yeni öznitelik kümeleri ve tüm öznitelikler kötücül olmayan uygulamalar kullanılarak kötücül yazılım tespiti deneylerinde test edilmiştir. İlgili deney sırasında API seviyesi 16 olan Google Play [70] üzerinden indirilen 1049 adet uygulama VirusTotal [56] servisinde taranmış ve hiçbir anti virüs yazılımının tespit etmediği uygulamalar zararsız olarak kabul edilmiştir.

testler yapılmıştır. Buradaki nihai Kötücül olmayan yazılım testleri Malgenome [21], Drebin [32] ve UpDroid [55] veri kümeleri için ayrı yapılmasının yanında bütün veri kümeleri birleştirilerek te yapılmıştır. Bu veri kümeleri alınan kötücül uygulamalar için herhangi bir filtre uygulanmamış ve her bir test için veri kümelerinin tamamı alınarak testler yapılmıştır. Testler 6 farklı öznitelik kombinasyonu ve KNN, DT, RF ve SVM olmak üzere 4 farklı makine öğrenmesi algoritması kullanılarak 10 katmanlı çapraz doğrulama yöntemi ile amaç her bir test durumu için özniteliklerin ve makine öğrenmesi algoritmalarının doğruluk oranına etkisini görebilmenin yanında FP oranına etkisini görmektir.

Çizelge 28'de Malgenome [21] veri kümesi üzerinde yapılan testleri göstermektedir. En yüksek doğruluk oranı ve en düşük FP oranı SVM algoritması ile tüm özniteliklerin kullanıldığı test durumunda sırasıyla 99,77% ve 0,20% olarak elde edilmiştir. Çizelge 28'de bu test durumu kalınlaştırılmış yazım ile belirtilmiştir. Yalnızca ağ tabanlı özniteliklerin kullanıldığı RF algoritması ile yapılan test durumuna ait sonuçların yine RF algoritmasının ve tüm özniteliklerin kullanıldığı test durumundaki sonuçlara göre daha yüksek doğruluk oranı ve daha düşük FP oranı elde edildiği görülmektedir. Yalnızca ağ tabanlı öznitelikler ile yapılan testlerde FP oranı ve doğruluk oranları oldukça yüksek olduğu görülürken ikili aktivite tekrarları ile yapılan testte çok yüksek FP oranı elde edilmiştir. Ancak yalnızca ağ tabanlı özniteliklerin kullanıldığı test sonuçları ardışık ikili aktivite tekrarları da eklendiğinde 4 farklı algoritma için de FP oranının düştüğü doğruluk oranlarının ise arttığı görülmüştür.

Çizelge 28. Malgenome [21] Veri Kümesi ile Yapılan Kötücül Olmayan Uygulama Testleri Sonuçları.

Algoritmalar		Ağ Tabanlı	İkili Aktivite Tekrarları	Ağ Tabanlı + İkili Aktivite Tekrarları	UpDroid Öznitelikleri + Ağ Tabanlı	UpDroid Öznitelikleri + İkili Aktivite Tekrarları	Tüm Öznitelikler
KNN	Doğruluk %	98,32	84,91	98,54	98,86	97,93	98,76
	TP %	98,30	84,90	98,50	98,90	97,90	98,80
	FP %	1,70	15,30	1,50	1,20	2,10	1,30
DT	Doğruluk %	99,10	87,32	99,23	99,59	98,81	99,45
	TP %	99,10	87,30	99,20	99,60	98,80	99,40
	FP %	0,90	12,80	0,70	0,40	1,20	0,50
RF	Doğruluk %	98,99	99,27	98,99	99,31	98,99	98,76
	TP %	99,00	91,30	99,00	99,30	99,00	98,80
	FP %	1,00	9,10	1,0	0,70	1,00	1,30
SVM	Doğruluk %	98,36	67,76	98,45	99,72	98,12	**99,77**
	TP %	98,40	67,80	98,50	99,70	98,10	99,80
	FP %	1,70	34,70	1,60	0,30	1,80	**0,20**

Çizelge 29'de Drebin [32] veri kümesi üzerinde yapılan testleri göstermektedir. Çizelge 28'de Malgenome [21] veri kümesi ile yapılan testlere göre yalnızca ardışık ikili aktivite tekrarları ile yapılan testlerdeki FP oranları daha yüksek ancak farklı olarak doğruluk oranlarının da RF algoritması dışında daha yüksek hesaplanmıştır. En yüksek doğruluk oranı Çizelge 28'de Malgenome [21] sonuçları ile benzer olarak DT algoritması ve UpDroid [8] çalışmasındaki öznitelikler ile birlikte ağ tabanlı özniteliklerin kullanıldığı test durumunda 99,70% olarak ve en düşük FP oranı ise tüm özniteliklerin kullanıldığı DT algoritması ile 0,40% olarak elde edilmiş ve Çizelge 29'da kalın yazım ile belirtilmiştir.

Çizelge 29. Drebin [32] Veri Kümesi ile Yapılan Kötücül Olmayan Uygulama Testleri Sonuçları.

Algoritmalar		Ağ Tabanlı	İkili Aktivite Tekrarları	Ağ Tabanlı + İkili Aktivite Tekrarları	UpDroid Öznitelikleri + Ağ Tabanlı	UpDroid Öznitelikleri + İkili Aktivite Tekrarları	Tüm Öznitelikler
KNN	Doğruluk %	98,61	90,97	98,72	98,96	98,21	98,94
	TP %	98,60	91,00	98,70	99,00	98,20	98,90
	FP %	1,70	27,60	3,70	3,40	5,70	3,30
DT	Doğruluk %	98,70	92,15	99,06	**99,70**	99,43	99,68
	TP %	98,70	92,20	99,10	99,70	99,40	99,70
	FP %	1,30	24,00	1,90	0,50	1,50	**0,40**
RF	Doğruluk %	98,70	93,66	99,14	99,41	98,92	99,51
	TP %	98,70	93,70	99,10	99,40	98,90	99,50
	FP %	1,50	26,50	2,30	1,70	5,00	1,20
SVM	Doğruluk %	98,54	85,79	98,56	99,69	98,38	99,69
	TP %	98,50	85,80	98,60	99,70	98,40	99,70
	FP %	2,40	67,40	2,40	1,00	5,80	0,80

Çizelge 30, UpDroid [55] veri kümesi ile yapılan testlerin sonuçlarını içermektedir. DT algoritması ve tüm öznitelikler ile yapılan testte 99,94% doğruluk oranı ile en yüksek ve 0,10% FP oranı ile en düşük değer elde edilmiştir. Bu sonucun yanında ardışık ikili aktiviteler ile yapılan testte Çizelge 28 ve Çizelge 29'daki sonuçlara göre daha düşük FP oranları ve daha yüksek doğruluk oranları elde edilmiştir. Bu anlamda bu öznitelik kümesinin UpDroid [55] veri kümesindeki ayrıt edici niteliği Malgenome [21] ve Drebin [32] veri kümelerine göre daha iyi durumda olduğu belirtilebilir.

Çizelge 30. UpDroid [55] Veri Kümesi ile Yapılan Kötücül Olmayan Uygulama Testleri Sonuçları.

Algoritmalar		Ağ Tabanlı	İkili Aktivite Tekrarları	Ağ Tabanlı + İkili Aktivite Tekrarları	UpDroid Öznitelikleri + Ağ Tabanlı	UpDroid Öznitelikleri + İkili Aktivite Tekrarları	Tüm Öznitelikler
KNN	Doğruluk %	98,81	94,08	99,39	99,56	99,45	99,67
	TP %	98,80	94,10	99,40	99,60	99,40	99,70
	FP %	1,20	9,20	0,90	0,50	0,70	0,30
DT	Doğruluk %	98,90	94,13	99,45	99,88	99,59	**99,94**
	TP %	98,90	94,10	99,40	99,90	99,60	99,90
	FP %	0,60	8,80	0,70	0,20	0,70	**0,10**
RF	Doğruluk %	98,93	96,17	99,68	99,71	99,30	99,65
	TP %	98,90	96,20	99,70	99,70	99,30	99,60
	FP %	1,00	4,30	0,20	0,50	1,00	0,70
SVM	Doğruluk %	97,94	80,69	97,99	99,59	98,95	99,56
	TP %	97,90	80,70	98,00	99,60	98,90	99,60
	FP %	2,20	43,20	2,20	0,60	1,20	0,70

Çalışmada kullanılan Malgenome [21], Drebin [32] ve UpDroid [55] veri kümelerinin birleştirilmesi ile oluşturulan veri kümesi ile yapılan testlere ilişkin sonuçlar Çizelge 31'de gösterilmiştir. Bu testlerde en yüksek doğruluk oranı UpDroid [8] çalışmasındaki öznitelikler ile ağ tabanlı özniteliklerin birlikte kullanıldığı test durumunda SVM algoritması ile 99,71% olarak elde edilmiştir. Tüm veri kümesi ile yapılan testlerde Çizelge 31'den görüleceği üzere en düşük FP oranı ise DT algoritması ile tüm özniteliklerin kullanıldığı ve UpDroid [8] çalışmasındaki özniteliklerin ağ tabanlı öznitelikler ile birlikte kullanıldığı iki farklı test durumunda 1,00% olarak hesaplanmıştır.

Çizelge 31. Tüm Veri Kümesi ile Yapılan Kötücül Olmayan Uygulama Testleri Sonuçları.

Algoritmalar		Ağ Tabanlı	İkili Aktivite Tekrarları	Ağ Tabanlı + İkili Aktivite Tekrarları	UpDroid Öznitelikleri + Ağ Tabanlı	UpDroid Öznitelikleri + İkili Aktivite Tekrarları	Tüm Öznitelikler
KNN	Doğruluk %	98,58	93,25	98,98	99,11	98,68	98,99
	TP %	98,60	93,20	99,00	99,10	98,70	99,00
	FP %	2,20	29,60	4,60	3,80	6,00	4,70
DT	Doğruluk %	98,72	93,72	99,17	99,68	99,45	99,66
	TP %	98,70	93,70	99,20	99,70	99,40	99,70
	FP %	1,10	28,60	2,50	**1,00**	2,00	**1,00**
RF	Doğruluk %	98,68	94,97	99,26	99,65	99,34	99,50
	TP %	98,70	95,00	99,30	99,60	99,30	99,50
	FP %	1,90	31,20	2,30	1,30	4,20	2,20
SVM	Doğruluk %	98,52	89,27	98,53	**99,71**	98,76	99,65
	TP %	98,50	89,30	98,50	99,70	98,80	99,50
	FP %	2,60	75,20	2,60	1,30	6,50	1,50

6. SONUÇLAR

Bu tez çalışmasında, Android kötücül yazılım aile sınıflandırması için hibrit bir yaklaşım önerilmiştir. UpDroid [8] çalışmasında kullanılan özniteliklerin yanı sıra, daha önceki çalışmalarda kullanılmamış iki yeni öznitelik grubunun, ağ tabanlı özniteliklerin ve ikili aktivite tekrarlarının etkileri Drebin [32], Malgenome [21] ve UpDroid [55] veri kümeleri üzerinde incelenmiştir. Bilindiği kadarıyla, bu iki öznitelik grubu ilk kez kötücül yazılım aile sınıflandırmasında kullanılmaktadır.

İlişkili çalışmalarda daha önce kullanılmamış ardışık ikili aktivite tekrarlarını ifade eden öznitelik kümesi ilk kez bu çalışmada kullanılmıştır. Uygulamalar dinamik analiz sırasında birtakım aktiviteler (dosya okuma/yazma, SMS gönderimi vb.) gerçekleştirmektedir. Bu çalışmada dinamik analizler sırasında gerçekleşen aktiviteler gerçekleşme zamanına göre ardışık ikililer şeklinde sıralanmış ve ikili olarak tekrarlama adetleri ele alınmıştır (ör. fileread-sendsms ardışık ikilisinin 3 kez tekrar etmesi vb.). Çalışmada kullanılan ağ tabanlı öznitelikler ilk bu çalışmada önerilmiştir. Ağ tabanlı öznitelikler bağlamında genel ağ trafik paket büyüklüğü, IP ve Port sayısı, DNS ve HTTP protokollerinin başlık bilgileri kullanılarak detaylandırılmıştır. Ağ tabanlı, ardışık ikili aktivite tekrarları ilk kez bu çalışmada bir arada kullanılmış ve yine literatürde sıkça kullanılan Drebin [32], Malgenome [21] ve UpDroid [55] veri kümelerinde test edilmiştir.

Literatürde Android kötücül yazılım aile sınıflandırmasında hibrit analiz yönteminin önerildiği bilindiği kadarıyla Ec2 [7] ve UpDroid [8] çalışmaları bulunmaktadır. Ec2 [7] çalışmasında önerilen modelde Drebin [31] veri kümesi üzerinde yapılan testlerde elde edilen MiF ve MiAUC değerlerine göre bu çalışmada önerilen model daha yüksek performans göstermiştir. Buna ek olarak, UpDroid [8] çalışmasında UpDroid [55] veri kümesi ile KNN algoritması kullanılarak 10 katmanlı çapraz doğrulamada %96,85 olarak hesaplanırken önerilen çalışmada %98,04 olarak daha yüksek bir doğruluk oranına ulaşılmıştır. Bu anlamda UpDroid [8] çalışmasındaki sonuçlar yeni eklenen öznitelik kümeleri ile geliştirilmiştir. UpDroid [8] çalışmasındaki doğruluk oranı eklenen yeni öznitelikler ile beraber yapılan test ile iyileştirilmiştir.

Ağ trafiğinin yoğun olduğu aileleri içerdiği bilinen UpDroid [55] veri kümesinde de bu öznitelik grubu ile yapılan testlerde olduğu gibi %90'ın üzerinde sonuçlara ulaşılırken Drebin [32] ve Malgenome [21] veri kümelerinde ağ tabanlı öznitelikler aynı etkiyi göstermemiştir. Bunun yanında yalnızca ardışık ikili aktivite tekrarlarını içeren öznitelikler ile yapılan testlerde yalnızca ağ tabanlı öznitelikler ile yapılan testlere göre Drebin [32], Malgenome [21] ve UpDroid [55] veri kümelerinde daha yüksek doğruluk oranları elde edilmiş ancak yalnızca UpDroid [55] veri kümesinde %90'ın üzerinde bir doğruluk oranı elde edilmiştir. Bu durum UpDroid [55] veri kümesindeki Android kötücül yazılım uygulamalarının, Malgenome [21] ve Drebin [32] veri kümelerindeki örneklerden daha fazla ağ etkinliğine sahip olmasından kaynaklanmaktadır.

Son olarak yeni öznitelikler kötücül uygulama tespitinde kullanılmıştır. Her bir veri kümesi için ayrı ayrı yapılan testlerin sonucunda FP oranları ve doğruluk oranları analiz edilmiştir. Yalnızca ağ tabanlı özniteliklerin kullanıldığı test durumları için FP oranı 3%'ün üzerine çıkmamış, doğruluk oranı ise %98'in üzerinde olarak hesaplanmıştır. Ancak ardışık ikili aktivite tekrarları ile yapılan testlerin yalnızca ağ tabanlı öznitelik kümesi ile yapılan testlere göre FP oranlarının aynı algoritma ve aynı öznitelik kümesinin kullanıldığı test durumlarında çok daha yüksek olduğu görülmüştür. Bu sonuçlar ile kullanılan zararlı ve zararsız veri kümeleri arasında ardışık ikili aktiviteler arasında benzerlikler diğer özniteliklere oranla daha fazla olduğu ifade edilebilir. Bunun yanında ağ tabanlı öznitelik kümesine ardışık ikili aktivite tekrarlarının eklenmesinin sonuçları iyileştirdiği ve ayırt ediciliği arttırdığı görülmüştür.

6.1. Gelecek Çalışmalar

Bu çalışmada, DroidBox [48] aracının belirtilen kısıtlarından dolayı API seviyesi 16 ve daha düşük uygulamalarda testler gerçekleştirilmiştir. Fakat, yeni uygulamalar daha yüksek API seviyelerini desteklemektedir. Önerilen yöntemin, daha büyük ve Drebin [32], Malgenome [21], UpDroid [55] veri kümelerinden farklı olarak API seviyesi yüksek olan güncel büyük boyutlu Android kötücül yazılım veri kümelerinde etkisi gözlemlenebilir. Saldırı yüzeyinin ve saldırganların hem cihaz ve market üzerindeki koruma önlemlerinden hem de ağ düzeyindeki saldırı tespit cihazlarından kaçınma yöntemlerinin (tünelleme, şifreleme vb.)

günden güne geliştiği düşünüldüğünde öznitelik kümesine eklemeler yapılabilir. Bu anlamda, eklenebilecek bazı özniteliklere yönelik birkaç öneri aşağıda verilmiştir.

Bu çalışmada şifreli ağ trafikleri çözülmeden modele dahil edilmiştir. Bu çalışma kapsamında eklenen ağ tabanlı öznitelikler açısından değerlendirildiğinde SSL/TLS kullanımının yanında, güncel kötücül yazılımların oluşturduğu şifreli ağ trafiği Android öykünücü üzerinde bir ek yazılımlar kullanılarak Haystack [71], henüz şifrelenmeden elde edilerek değerlendirilerek öznitelik kümesine dahil edilebilir [54]. Bunun yanında çalışmada ağ trafiği oluşturan örneklemlerin veri gönderim/alım işlemi yaptığı hedef sunucuların çalışır halde olup olmadığı ve her bir örneklem için ağ trafik bilgisinden çıkarılan IP ve alan adı bilgilerinin VirusTotal [56] ve benzeri kara liste (blacklist) veri tabanların yer alıp almadığı bilgileri de çalışma içerisine dahil edilebilir.

Ağ trafiği içerisinde uzak sunucular ile kurulan bağlantı sonrası mobil cihazlara gönderilen olası komutların nitelik veya nicelik olarak incelenmesi, çalışmaya dahil edilebilir. Bunun yanında kötücül yazılımın bulaşmış olduğu kurban cihazın uzak sunucudan indirdiği dosyalar özelinde öznitelikler çalışma içerisinde değerlendirilebilir. Örneğin, zafiyeti olduğu bilinen bir çekirdek sürümüne sahip Android cihaza indirilen bir sömürü kodunu içeren dosyaların uzantıları ve bu dosyaların tekil sağlama değerleri (file hash) çıkarılarak öznitelik kümesine eklenebilir. Aynı şekilde bu çalışmada bir HTTP isteği içerisinde bir apk dosyasının olup olmadığı bilgisi kullanılmıştır. Bu anlamda bu apk dosyalarının tekilliği yani kötücül uygulamanın kaç farklı apk dosyasını indirdiği öznitelik kümesine dahil edilebilir.

7. KAYNAKLAR

[1] Anonim, IDC Smartphone Market Share https://www.idc.com/promo/smartphone-market-share/os (Erişim tarihi: **12 Haziran 2021**).

[2] Kotlin Programlama Dili, https://kotlinlang.org/ (Erişim tarihi: **12 Haziran 2021**)

[3] Java Programlama Dili, https://www.java.com/en (Erişim tarihi: **12 Haziran 2021**)

[4] V. Chebyshev, IT threat evolution Q2 2020. Mobile statistics. https://securelist.com/it-threat-evolution-q2-2020-mobile-statistics/98337/ (Erişim tarihi: **12 Haziran 2021**)

[5] R. Samani, G. Davis, McAfee Advanced Threat Research and Mobile Malware Research team, McAfee Mobile Threat Report Q1 2019, https://www.mcafee.com/enterprise/en-us/assets/reports/rp-mobile-threat-report-2019.pdf (Erişim tarihi: **12 Haziran 2021**)

[6] R. Samani, McAfee Advanced Threat Research and Mobile Malware Research team, McAfee Mobile Threat Report Q1 2020, https://www.mcafee.com/content/dam/consumer/en-us/docs/2020-Mobile-Threat-Report.pdf (Erişim tarihi: **12 Haziran 2021**)

[7] T. Chakraborty, F. Pierazzi and V. S. Subrahmanian, Ec2: Ensemble clustering and classification for predicting android malware families, in IEEE Transactions on Dependable and Secure Computing, 17(2), pp. 262–277, **2017**

[8] K. Aktas, S. Sen, Updroid: Updated android malware and its familial classification, in Nordic Conference on Secure IT Systems, pp. 352– 368, **2018**

[9] Android Operating System https://en.wikipedia.org/wiki/Android_(operating_system) (Erişim Tarihi: **12 Haziran 2021**).

[10] Android Operating System: Architecture, Security Challenges and Solutions, https://www.researchgate.net/publication/299394606_Android_Operati g_System_Architecture_Security_Challenges_and_Solutions (Erişim Tarihi: **12 Haziran 2021**).

[11] Dalvik (Programming) https://en.wikipedia.org/wiki/Dalvik_(software) (Erişim

tarihi: **12 Haziran 2021**)

[12] Permissions on Android, https://developer.android.com/guide/topics/permissions/overview, (Erişim Tarihi: **05 Temmuz 2021**)

[13] AndroidManifest, sharedUserId https://developer.android.com/guide/topics/manifest/manifest-element#uid (Erişim Tarihi: **12 Haziran 2021**)

[14] B. Kang, S. Y. Yerima, K. McLaughlin, S. Sezer, N-opcode analysis for android malware classification and categorization, in 2016 International conference on cyber security and protection of digital services, cyber security, pp. 1-7, **2016**

[15] A. Pektaş, M. Çavdar, T. Acarman, Android malware classification by applying online machine learning, in International Symposium on Computer and Information Sciences, pp. 72-80, **2016**

[16] B. Kang,, B. Kang, J. Kim, E. G. Im, Android malware classification method: Dalvik bytecode frequency analysis, in Proceedings of the 2013 research in adaptive and convergent systems, pp. 349-350, 2013

[17] Decision Tree, https://web.cs.hacettepe.edu.tr/~ilyas/Courses/BIL712/lec02-DecisionTree.pdf (Erişim Tarihi: **12 Haziran 2021**).

[18] Naive Bayes Algorithm, https://en.wikipedia.org/wiki/Bayes%27_theorem (Erişim Tarihi: **12 Haziran 2021**)

[19] Support Vector Machines, https://en.wikipedia.org/wiki/Support-vector_machine (Erişim Tarihi: **12 Haziran 2021**).

[20] Y. Zhou, X. Jiang, Dissecting android malware: Characterization and evolution, in 2012 IEEE symposium on security and privacy, pp. 95– 109, **2012**

[21] Android Malware Genome Project, http://www.malgenomeproject.org/ (Erişim Tarihi: **12 Haziran 2021**)

[22] Apktool, https://ibotpeaches.github.io/Apktool/ (Erişim Tarihi: **12 Haziran 2021**).

[23] L. Deshotels, V. Notani, and A. Lakhotia, DroidLegacy: Automated familial classification of Android malware, in Proceedings of ACM SIGPLAN on program protection and reverse engineering workshop, pp. 1-12, **2014**

[24] J. Garcia, M. Hammad, S. Malek, Lightweight, Obfuscation-Resilient Detection and Family Identification of Android Malware, in ACM Transactions on Software Engineering and Methodology, 26(3), pp. 1-29, **2018**

[25] M. Fan, J. Liu, X. Luo, K. Chen, T. Chen, Z. Tian, X. Zhang, Q. Zheng, T. Liu, Frequent subgraph based familial classification of android malware, in 2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE), pp. 24-35, **2016**

[26] G. Suarez-Tangil, J.E Tapiador, P. Peris-Lopez, J. Blasco, Dendroid: A text mining approach to analyzing and classifying code structures in android malware families, in Expert Systems with Applications, 41(4), pp. 1104–1117, **2014**

[27] M. Zhang, Y. Duan, H. Yin, 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**

[28] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, and K. Rieck, DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket, in NDSS, San Diego, CA, USA, **2014**.

[29] N. Milosevic, A. Dehghantanha, K. K. R. Choo, Machine learning aided Android malware classification, Computers & Electrical Engineering, 61, pp. 266-274, **2017**

[30] Android Application Package, https://en.wikipedia.org/wiki/Android_application_package (Erişim Tarihi: 12 Haziran 2021)

[31] A. T. Kabakus, I. A. Dogru, C. Aydin, APK Auditor: Permission-based Android malware detection system, Digital Investigation, 13, pp. 1-14, **2015**

[32] The Drebin Dataset, https://www.sec.cs.tu-bs.de/~danarp/drebin/ (Erişim Tarihi: **12 Haziran 2021**).

[33] Y. Zhou, X. Jiang, An Analysis of the AnserverBot Trojan, https://www.csc2.ncsu.edu/faculty/xjiang4/pubs/AnserverBot_Analysis.pdf, (Erişim Tarihi: **12 Haziran 2021**)

[34] V. Rastogi, Y. Chen, X Jiang, Droidchameleon: evaluating android anti-malware against transformation attacks, in Proceedings of the 8th ACM SIGSAC symposium

on Information, computer and communications security, pp. 329-334, **2013**

[35] C. Yang, Z. Xu, G. Gu, V. Yegneswaran, P. Porras, Droidminer: Automated mining and characterization of fine-grained malicious behaviors in android applications, in European symposium on research in computer security. pp. 163–182, Springer, Cham, **2014**

[36] ProGuard https://www.guardsquare.com/proguard (Erişim Tarihi: **12 Haziran 2021**)

[37] Soot - A framework for analyzing and transforming Java and Android applications, http://soot-oss.github.io/soot/ (Erişim Tarihi: **12 Haziran 2021**).

[38] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, P. McDaniel, Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps, Acm Sigplan Notices, 49(6), pp. 259-269, **2014**

[39] N. Marastoni, A. Continella, D. Quarta, S. Zanero, M. Dalla Preda, Groupdroid: Automatically grouping mobile malware by extracting code similarities, in Proceedings of the 7th Software Security, Protection, and Reverse Engineering / Software Security and Protection Workshop, pp. 1-12, **2017**

[40] G. Suarez-Tangil, S. K. Dash, M. Ahmadi, J. Kinder, G. Giacinto, and L. Cavallaro, Droidsieve: Fast and accurate classification of obfuscated android malware, in Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy, pp. 309–320, **2017**

[41] M. Fan, J. Liu, X. Luo, K. Chen, Z. Tian, Q. Zheng, T. Liu, Android malware familial classification and representative sample selection via frequent subgraph analysis, in IEEE Transactions on Information Forensics and Security, 13(8), pp. 1890-1905, **2018**

[42] S. Türker, A. B. Can, Andmfc: Android malware family classification framework, in 2019 IEEE 30th International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC Workshops), pp. 1-6, **2019**

[43] AMD dataset, Available at: http://amd.arguslab.org/ , (Erişim Tarihi: **25 Nisan 2019**)

[44] M. Kang, J. Park, S. Park, S. Cho, M. Park, Android Malware Family Classification using Images from Dex Files, in The 9th International Conference on Smart Media and Applications, **2020**

[45] G. Iadarola, F. Martinelli, F. Mercaldo, A. Santone, Image-based Malware Family Detection: An Assessment between Feature Extraction and Classification Techniques, in 5th International Conference on Internet of Things, Big Data and Security, pp. 499-506, **2020**

[46] Q. Wu, M. Li, X. Zhu, B. Liu, Mviidroid: A multiple view information integration approach for android malware detection and family identification, in IEEE MultiMedia 27(4), pp. 48-57, **2020**

[47] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B. G. Chun, L. P. Cox, P. McDaniel, A. N. Sheth, Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones, in ACM Transactions on Computer Systems (TOCS), 32(2), pp. 1-29, **2014**

[48] Droidbox, https://github.com/pjlantz/droidbox , (Erişim Tarihi: **12 Haziran 2021**).

[49] S. K. Dash, G. Suarez-Tangil, S. Khan, K. Tam, M. Ahmadi, J. Kinder, L. Cavallaro, Droidscribe: Classifying android malware based on runtime behavior, in 2016 IEEE Security and Privacy Workshops (SPW), pp. 252-261, **2016**

[50] K. Tam, S. J. Khan, A. Fattori, L. Cavallaro, Copperdroid: automatic reconstruction of android malware behaviors, in The Network and Distributed System Security Symposium, San Diego, CA, USA, **2015**

[51] J. Malik, R. Kaushal, Credroid: Android malware detection by network traffic analysis, in Proceedings of the 1st ACM Workshop on Privacy-Aware Mobile Computing, pp. 28–36, **2016**

[52] Z. Chen, H. Han, Q. Yan, B. Yang, L. Peng, L. Zhang, J. Li, A first look at android malware traffic in first few minutes, in 14th IEEE International Conference on Trust, Security and Privacy in Computing and Communications, pp. 206–213, **2015**

[53] L. Massarelli, L. Aniello, C. Ciccotelli, L. Querzoni, D. Ucci, R. Baldoni, AndroDFA: Android Malware Classification Based on Resource Consumption, in Information 11(6), **2020**

[54] Z. Li, L. Sun, Q. Yan, W. Srisa-An, Z. Chen, Droidclassifier: Efficient adaptive mining of application-layer header for classifying android malware, in International Conference on Security and Privacy in Communication Systems, pp. 597-616, **2016**

[55] Updroid, https://wise.cs.hacettepe.edu.tr/projects/updroid/dataset/ , (Erişim Tarihi:

12 Haziran 2021)

[56] Virus Total, https://www.virustotal.com/gui/ (Erişim Tarihi: **12 Haziran 2021**)

[57] Androguard, https://github.com/androguard/androguard (Erişim Tarihi: **12 Haziran 2021**)

[58] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I. H. Witten, The weka data mining software: an update, in ACM SIGKDD explorations newsletter, 11(1), pp. 10–18, **2009 https://www.cs.waikato.ac.nz/ml/weka/**

[59] Manifest.Permission, https://developer.android.com/reference/android/Manifest.permission (Erişim Tarihi: **12 Haziran 2021**)

[60] A. Feizollah, N. B. Anuar, R. Salleh, A. W. A. Wahab, A review on feature selection in mobile malware detection, Digital Investigation 13, pp. 22–37, **2015**

[61] S. R. Choudhary, A. Gorla, A. Orso, Automated test input generation for android: are we there yet?(e), in 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 429–440, **2015**

[62] Wireshark, https://www.wireshark.org/ (Erişim Tarihi: **12 Haziran 2021**)

[63] Tshark, https://www.wireshark.org/docs/man-pages/tshark.html (Erişim Tarihi: **12 Haziran 2021**)

[64] D. Maier, M. Protsenko, T. Muller, A game of droid and mouse: The ¨threat of split-personality malware on android, Computers & Security, 54, pp. 2–15, **2015**

[65] Comodo: Comodo threat research labs warns android users of tordow v2.0 outbreak, https://blog.comodo.com/comodo-news/comodo-warns-android-users-of-tordow-v2-0-outbreak/ (Erişim Tarihi: **12 Haziran 2021**)

[66] G DATA Software AG, Analysis of Xafecopy & Android.Application.SMSreg.A. https://file.gdatasoftware.com/web/en/documents/whitepaper/G_Data_Analysis_Xafecopy.pdf (Erişim Tarihi: **12 Haziran 2021**)

[67] Snow, J.: Triada: organized crime on android. https://www.kaspersky.com/blog/triada-trojan/11481/ (Erişim Tarihi: **12 Haziran 2021**)

[68] A.I. Aysan, F. Sakiz, S. Sen, Analysis of dynamic code updating in Android with security perspective, in IET Information Security, 13(3), pp.269-277, **2019**

[69] S. Sen, A. I. Aysan, and J. A. Clark, SAFEDroid: using structural features for detecting android malwares, in International Conference on Security and Privacy in Communication Systems, pp. 255-270, Springer, Cham, **2017**

[70] Google Play Store, https://play.google.com/store, (Erişim Tarihi: **12 Haziran 2021**)

[71] A. Razaghpanah, N. Vallina-Rodriguez, S. Sundaresan, C. Kreibich, P. Gill, M. Allman, and V. Paxson, Haystack: in situ mobile traffic analysis in user space, in arXiv preprint arXiv:1510.01419, **2015**

8. EKLER

8.1. Ek 1

Sıra	Uygulama/Özel İzinler	Açıklamalar
1	ACCESS_CHECKIN_PROPERTIES	https://developer.android.com/reference/android/Manifest.permission#ACCESS_CHECKIN_PROPERTIES
2	ACCESS_COARSE_LOCATION	https://developer.android.com/reference/android/Manifest.permission#ACCESS_COARSE_LOCATION
3	ACCESS_FINE_LOCATION	https://developer.android.com/reference/android/Manifest.permission#ACCESS_FINE_LOCATION
4	ACCESS_LOCATION_EXTRA_COMMANDS	https://developer.android.com/reference/android/Manifest.permission#ACCESS_LOCATION_EXTRA_COMMANDS
5	ACCESS_NETWORK_STATE	https://developer.android.com/reference/android/Manifest.permission#ACCESS_NETWORK_STATE
6	ACCESS_WIFI_STATE	https://developer.android.com/reference/android/Manifest.permission#ACCESS_WIFI_STATE
7	ACCOUNT_MANAGER	https://developer.android.com/reference/android/Manifest.permission#ACCOUNT_MANAGER
8	ADD_VOICEMAIL	https://developer.android.com/reference/android/Manifest.permission#ADD_VOICEMAIL
9	BATTERY_STATS	https://developer.android.com/reference/android/Manifest.permission#BATTERY_STATS
10	BIND_ACCESSIBILITY_SERVICE	https://developer.android.com/reference/android/Manifest.permission#BIND_ACCESSIBILITY_SERVICE
11	BIND_APPWIDGET	https://developer.android.com/reference/android/Manifest.permission#BIND_APPWIDGET
12	BIND_DEVICE_ADMIN	https://developer.android.com/reference/android/Manifest.permission#BIND_DEVICE_ADMIN
13	BIND_INPUT_METHOD	https://developer.android.com/reference/android/Manifest.permission#BIND_INPUT_METHOD

14	BIND_REMOTEVIEWS	https://developer.android.com/reference/android/Manifest.permission#BIND_REMOTEVIEWS
15	BIND_TEXT_SERVICE	https://developer.android.com/reference/android/Manifest.permission#BIND_TEXT_SERVICE
16	BIND_VPN_SERVICE	https://developer.android.com/reference/android/Manifest.permission#BIND_VPN_SERVICE
17	BIND_WALLPAPER	https://developer.android.com/reference/android/Manifest.permission#BIND_WALLPAPER
18	BLUETOOTH	https://developer.android.com/reference/android/Manifest.permission#BLUETOOTH
19	BLUETOOTH_ADMIN	https://developer.android.com/reference/android/Manifest.permission#BLUETOOTH_ADMIN
20	BROADCAST_PACKAGE_REMOVED	https://developer.android.com/reference/android/Manifest.permission#BROADCAST_PACKAGE_REMOVED
21	BROADCAST_SMS	https://developer.android.com/reference/android/Manifest.permission#BROADCAST_SMS
22	BROADCAST_STICKY	https://developer.android.com/reference/android/Manifest.permission#BROADCAST_STICKY
23	BROADCAST_WAP_PUSH	https://developer.android.com/reference/android/Manifest.permission#BROADCAST_WAP_PUSH
24	CALL_PHONE	https://developer.android.com/reference/android/Manifest.permission#CALL_PHONE
25	CALL_PRIVILEGED	https://developer.android.com/reference/android/Manifest.permission#CALL_PRIVILEGED
26	CAMERA	https://developer.android.com/reference/android/Manifest.permission#CAMERA
27	CHANGE_COMPONENT_ENABLED_STATE	https://developer.android.com/reference/android/Manifest.permission#CHANGE_COMPONENT_ENABLED_STATE
28	CHANGE_CONFIGURATION	https://developer.android.com/reference/android/Manifest.permission#CHANGE_CONFIGURATION
29	CHANGE_NETWORK_STATE	https://developer.android.com/reference/android/Manifest.permission#CHANGE_NETWORK_STATE

30	CHANGE_WIFI_MULTICAST_STATE	https://developer.android.com/reference/android/Manifest.permission#CHANGE_WIFI_MULTICAST_STATE
31	CHANGE_WIFI_STATE	https://developer.android.com/reference/android/Manifest.permission#CHANGE_WIFI_STATE
32	CLEAR_APP_CACHE	https://developer.android.com/reference/android/Manifest.permission#CLEAR_APP_CACHE
33	CONTROL_LOCATION_UPDATES	https://developer.android.com/reference/android/Manifest.permission#CONTROL_LOCATION_UPDATES
34	DELETE_CACHE_FILES	https://developer.android.com/reference/android/Manifest.permission#DELETE_CACHE_FILES
35	DELETE_PACKAGES	https://developer.android.com/reference/android/Manifest.permission#DELETE_PACKAGES
36	DIAGNOSTIC	https://developer.android.com/reference/android/Manifest.permission#DIAGNOSTIC
37	DISABLE_KEYGUARD	https://developer.android.com/reference/android/Manifest.permission#DISABLE_KEYGUARD
38	DUMP	https://developer.android.com/reference/android/Manifest.permission#DUMP
39	EXPAND_STATUS_BAR	https://developer.android.com/reference/android/Manifest.permission#EXPAND_STATUS_BAR
40	FACTORY_TEST	https://developer.android.com/reference/android/Manifest.permission#FACTORY_TEST
41	GET_ACCOUNTS	https://developer.android.com/reference/android/Manifest.permission#GET_ACCOUNTS
42	GET_PACKAGE_SIZE	https://developer.android.com/reference/android/Manifest.permission#GET_PACKAGE_SIZE
43	GET_TASKS	https://developer.android.com/reference/android/Manifest.permission#GET_TASKS
44	GLOBAL_SEARCH	https://developer.android.com/reference/android/Manifest.permission#GLOBAL_SEARCH
45	INSTALL_LOCATION_PROVIDER	https://developer.android.com/reference/android/Manifest.permission#INSTALL_LOCATION_PROVIDER
46	INSTALL_PACKAGES	https://developer.android.com/reference/android/Manifest.permission#INSTALL_PACKAGES
47	INTERNET	https://developer.android.com/reference/android/Manifest.permission#INTERNET

48	KILL_BACKGROUND_PROCESSES	https://developer.android.com/reference/android/Manifest.permission#KILL_BACKGROUND_PROCESSES
49	MASTER_CLEAR	https://developer.android.com/reference/android/Manifest.permission#MASTER_CLEAR
50	MODIFY_AUDIO_SETTINGS	https://developer.android.com/reference/android/Manifest.permission#MODIFY_AUDIO_SETTINGS
51	MODIFY_PHONE_STATE	https://developer.android.com/reference/android/Manifest.permission#MODIFY_PHONE_STATE
52	MOUNT_FORMAT_FILESYSTEMS	https://developer.android.com/reference/android/Manifest.permission#MOUNT_FORMAT_FILESYSTEMS
53	MOUNT_UNMOUNT_FILESYSTEMS	https://developer.android.com/reference/android/Manifest.permission#MOUNT_UNMOUNT_FILESYSTEMS
54	NFC	https://developer.android.com/reference/android/Manifest.permission#NFC
55	PERSISTENT_ACTIVITY	https://developer.android.com/reference/android/Manifest.permission#PERSISTENT_ACTIVITY
56	PROCESS_OUTGOING_CALLS	https://developer.android.com/reference/android/Manifest.permission#PROCESS_OUTGOING_CALLS
57	READ_CALENDAR	https://developer.android.com/reference/android/Manifest.permission#READ_CALENDAR
58	READ_CALL_LOG	https://developer.android.com/reference/android/Manifest.permission#READ_CALL_LOG
59	READ_CONTACTS	https://developer.android.com/reference/android/Manifest.permission#READ_CONTACTS
60	READ_EXTERNAL_STORAGE	https://developer.android.com/reference/android/Manifest.permission#READ_EXTERNAL_STORAGE
61	READ_FRAME_BUFFER	https://developer.android.com/reference/android/Manifest.permission#READ_FRAME_BUFFER
62	READ_INPUT_STATE	https://developer.android.com/reference/android/Manifest.permission#READ_INPUT_STATE
63	READ_LOGS	https://developer.android.com/reference/android/Manifest.permission#READ_LOGS

64	READ_PHONE_STATE	https://developer.android.com/reference/android/Manifest.permission#READ_PHONE_STATE
65	READ_SMS	https://developer.android.com/reference/android/Manifest.permission#READ_SMS
66	READ_SYNC_SETTINGS	https://developer.android.com/reference/android/Manifest.permission#READ_SYNC_SETTINGS
67	READ_SYNC_STATS	https://developer.android.com/reference/android/Manifest.permission#READ_SYNC_STATS
68	REBOOT	https://developer.android.com/reference/android/Manifest.permission#REBOOT
69	RECEIVE_BOOT_COMPLETED	https://developer.android.com/reference/android/Manifest.permission#RECEIVE_BOOT_COMPLETED
70	RECEIVE_MMS	https://developer.android.com/reference/android/Manifest.permission#RECEIVE_MMS
71	RECEIVE_SMS	https://developer.android.com/reference/android/Manifest.permission#RECEIVE_SMS
72	RECEIVE_WAP_PUSH	https://developer.android.com/reference/android/Manifest.permission#RECEIVE_WAP_PUSH
73	RECORD_AUDIO	https://developer.android.com/reference/android/Manifest.permission#RECORD_AUDIO
74	REORDER_TASKS	https://developer.android.com/reference/android/Manifest.permission#REORDER_TASKS
75	RESTART_PACKAGES	https://developer.android.com/reference/android/Manifest.permission#RESTART_PACKAGES
76	SEND_SMS	https://developer.android.com/reference/android/Manifest.permission#SEND_SMS
77	SET_ALARM	https://developer.android.com/reference/android/Manifest.permission#SET_ALARM
78	SET_ALWAYS_FINISH	https://developer.android.com/reference/android/Manifest.permission#SET_ALWAYS_FINISH
79	SET_ANIMATION_SCALE	https://developer.android.com/reference/android/Manifest.permission#SET_ANIMATION_SCALE
80	SET_DEBUG_APP	https://developer.android.com/reference/android/Manifest.permission#SET_DEBUG_APP
81	SET_PREFERRED_APPLICATIONS	https://developer.android.com/reference/android/Manifest.permission#SET_PREFERRED_APPLICATIONS

82	SET_PROCESS_LIMIT	https://developer.android.com/reference/android/Manifest.permission#SET_PROCESS_LIMIT
83	SET_TIME	https://developer.android.com/reference/android/Manifest.permission#SET_TIME
84	SET_TIME_ZONE	https://developer.android.com/reference/android/Manifest.permission#SET_TIME_ZONE
85	SET_WALLPAPER	https://developer.android.com/reference/android/Manifest.permission#SET_WALLPAPER
86	SET_WALLPAPER_HINTS	https://developer.android.com/reference/android/Manifest.permission#SET_WALLPAPER_HINTS
87	SIGNAL_PERSISTENT_PROCESSES	https://developer.android.com/reference/android/Manifest.permission#SIGNAL_PERSISTENT_PROCESSES
88	STATUS_BAR	https://developer.android.com/reference/android/Manifest.permission#STATUS_BAR
89	SYSTEM_ALERT_WINDOW	https://developer.android.com/reference/android/Manifest.permission#SYSTEM_ALERT_WINDOW
90	TAINT_ACCELEROMETER	https://developer.android.com/reference/android/Manifest.permission#TAINT_ACCELEROMETER
91	TAINT_ACCOUNT	https://developer.android.com/reference/android/Manifest.permission#TAINT_ACCOUNT
92	TAINT_BROWSER	https://developer.android.com/reference/android/Manifest.permission#TAINT_BROWSER
93	TAINT_CALENDAR	https://developer.android.com/reference/android/Manifest.permission#TAINT_CALENDAR
94	TAINT_CALL_LOG	https://developer.android.com/reference/android/Manifest.permission#TAINT_CALL_LOG
95	TAINT_CAMERA	https://developer.android.com/reference/android/Manifest.permission#TAINT_CAMERA
96	TAINT_CONTACTS	https://developer.android.com/reference/android/Manifest.permission#TAINT_CONTACTS
97	TAINT_DEVICE_SN	https://developer.android.com/reference/android/Manifest.permission#TAINT_DEVICE_SN
98	TAINT_EMAIL	https://developer.android.com/reference/android/Manifest.permission#TAINT_EMAIL

99	TAINT_FILECONTENT	https://developer.android.com/reference/android/Manifest.permission#TAINT_FILECONTENT
100	TAINT_ICCID	https://developer.android.com/reference/android/Manifest.permission#TAINT_ICCID
101	TAINT_IMEI	https://developer.android.com/reference/android/Manifest.permission#TAINT_IMEI
102	TAINT_IMSI	https://developer.android.com/reference/android/Manifest.permission#TAINT_IMSI
103	TAINT_LOCATION	https://developer.android.com/reference/android/Manifest.permission#TAINT_LOCATION
104	TAINT_LOCATION_GPS	https://developer.android.com/reference/android/Manifest.permission#TAINT_LOCATION_GPS
105	TAINT_LOCATION_LAST	https://developer.android.com/reference/android/Manifest.permission#TAINT_LOCATION_LAST
106	TAINT_LOCATION_NET	https://developer.android.com/reference/android/Manifest.permission#TAINT_LOCATION_NET
107	TAINT_MIC	https://developer.android.com/reference/android/Manifest.permission#TAINT_MIC
108	TAINT_OTHERDB	https://developer.android.com/reference/android/Manifest.permission#TAINT_OTHERDB
109	TAINT_PACKAGE	https://developer.android.com/reference/android/Manifest.permission#TAINT_PACKAGE
110	TAINT_PHONE_NUMBER	https://developer.android.com/reference/android/Manifest.permission#TAINT_PHONE_NUMBER
111	TAINT_SETTINGS	https://developer.android.com/reference/android/Manifest.permission#TAINT_SETTINGS
112	TAINT_SMS	https://developer.android.com/reference/android/Manifest.permission#TAINT_SMS
113	UPDATE_DEVICE_STATS	https://developer.android.com/reference/android/Manifest.permission#UPDATE_DEVICE_STATS
114	USE_SIP	https://developer.android.com/reference/android/Manifest.permission#USE_SIP
115	VIBRATE	https://developer.android.com/reference/android/Manifest.permission#VIBRATE
116	WAKE_LOCK	https://developer.android.com/reference/android/Manifest.permission#WAKE_LOCK

117	WRITE_APN_SETTINGS	https://developer.android.com/reference/android/Manifest.permission#WRITE_APN_SETTINGS
118	WRITE_CALENDAR	https://developer.android.com/reference/android/Manifest.permission#WRITE_CALENDAR
119	WRITE_CALL_LOG	https://developer.android.com/reference/android/Manifest.permission#WRITE_CALL_LOG
120	WRITE_CONTACTS	https://developer.android.com/reference/android/Manifest.permission#WRITE_CONTACTS
121	WRITE_EXTERNAL_STORAGE	https://developer.android.com/reference/android/Manifest.permission#WRITE_EXTERNAL_STORAGE
122	WRITE_GSERVICES	https://developer.android.com/reference/android/Manifest.permission#WRITE_GSERVICES
123	WRITE_SECURE_SETTINGS	https://developer.android.com/reference/android/Manifest.permission#WRITE_SECURE_SETTINGS
124	WRITE_SETTINGS	https://developer.android.com/reference/android/Manifest.permission#WRITE_SETTINGS
125	WRITE_SYNC_SETTINGS	https://developer.android.com/reference/android/Manifest.permission#WRITE_SYNC_SETTINGS
	Diğer Statik Öznitelikler	
126	receivers_count	AndroidManifest dosyasında alıcı sayısı (https://developer.android.com/guide/topics/manifest/receiver-element)
127	services_count	AndroidManifest dosyasında servis alıcı sayısı (https://developer.android.com/guide/topics/manifest/service-element)
128	activities_count	AndroidManifest dosyasında servis alıcı sayısı (https://developer.android.com/guide/topics/manifest/activity-element)
129	extra_act	AndroidManifest dosyasında belirtilen ancak uygulama smali dosyasında bulunmayan aktivite sayısı
130	extra_rec	AndroidManifest dosyasında belirtilen ancak uygulama smali dosyasında bulunmayan alıcı sayısı

131	extra_ser	AndroidManifest dosyasında belirtilen ancak uygulama smali dosyasında bulunmayan servis sayısı
132	custom_perm_count	1-125 sıralamasında belirtilen Android izinleri dışında AndroidManifest.xml dosyasında belirtilen izinlerin sayısı
133	shared_uid	AndroidManifest dosyasında sharedUid kullanımı (https://developer.android.com/guide/topics/manifest/manifest-element#uid)
134	size_of_afiles	Uygulama kaynak dosyaları boyutu
135	jar_exists	Uygulama kaynak dosyalarında .jar uzantılı dosya varlığı
136	apk_exists	Uygulama kaynak dosyalarında .apk uzantılı dosya varlığı
137	apk_size	Uygulama paket büyüklüğü
	Kripto Kullanımı	
138	AES	https://developer.android.com/reference/javax/crypto/Cipher
139	ARC4	https://developer.android.com/reference/javax/crypto/Cipher
140	RSA	https://developer.android.com/reference/javax/crypto/Cipher
141	PBEwithHmacSHA	https://developer.android.com/reference/javax/crypto/Cipher
142	PBEwithHmacSHA1	https://developer.android.com/reference/javax/crypto/Cipher
143	HmacMD5	https://developer.android.com/reference/javax/crypto/Cipher
144	HmacSHA1	https://developer.android.com/reference/javax/crypto/Cipher
145	HmacSHA256	https://developer.android.com/reference/javax/crypto/Cipher
146	HmacSHA384	https://developer.android.com/reference/javax/crypto/Cipher
147	HmacSHA512	https://developer.android.com/reference/javax/crypto/Cipher
148	DES	https://developer.android.com/reference/javax/crypto/Cipher

149	DESede	https://developer.android.com/reference/javax/crypto/Cipher
150	BLOWFISH	https://developer.android.com/reference/javax/crypto/Cipher
	Ağ Tabanlı Öznitelikler	
151	n_avg_len_dns_domains	DNS isteklerinde yer alan adlarının ortalama uzunluğu
152	n_avg_len_get_url	GET istekleri ortalama URL uzunluğu
153	n_avg_len_post_url	POST istekleri ortalama URL uzunluğu
154	n_avg_parameters_get	GET isteklerinde yer alan ortalama parametre sayısı
155	n_avg_parameters_post	POST isteklerinde yer alan ortalama parametre sayısı
156	n_content_app	Uygulama türü HTTP içerik adedi
157	n_content_audio	Ses türü HTTP içerik adedi
158	n_content_image	Resim türü HTTP içerik adedi
159	n_content_multi	Çoklu türde HTTP içerik adedi
160	n_content_text	Metin türü HTTP içerik adedi
161	n_content_video	Video türü HTTP içerik adedi
162	n_first_five_min	Uygulama ilk 5 dakikalık ağ trafik boyutu
163	n_get_url_contains_apk	GET isteği içerisinde .apk dosya varlığı
164	n_get_url_contains_jar	GET isteği içerisinde .jar dosya varlığı
165	n_http_f5	İlk 5 dakikalık HTTP trafik boyutu
166	n_http_l5	İkinci 5 dakikalık HTTP trafik boyutu
167	n_http_s5	Üçüncü 5 dakikalık HTTP trafik boyutu
168	n_ip_packet_avg	Ortalama IP paket boyutu
169	n_ipcount	Ağ paketi içerisinde tekil IP adedi
170	n_last_five_min	Son 5 dakikalık trafik boyutu
171	n_netfiltered	Yalnızca HTTP ve DNS paketlerini içeren filtrelenmiş ağ paket boyutu
172	n_networksize	Filtrelenmemiş ağ paket boyutu
173	n_num_of_get_req	GET isteği toplam sayısı
174	n_num_of_http_frame	Toplam HTTP frame adedi
175	n_num_of_http_req	Toplam HTTP istek adedi
176	n_num_of_http_res	Toplam HTTP cevap adedi
177	n_num_of_post_req	Toplam POST isteği adedi
178	n_number_of_syn	Toplam SYN paketi adedi
179	n_portcount	Ağ paketi içerisinde tekil Port adedi
180	n_post_content_len	POST isteklerindeki ortalama içerik büyüklüğü

181	n_post_url_contains_apk	POST isteği içerisinde .apk dosya varlığı
182	n_post_url_contains_jar	POST isteği içerisinde .jar dosya varlığı
183	n_response_content_len	Ortalama cevap paketi büyüklüğü
184	n_second_five_min	Uygulama ikinci 5 dakikalık ağ trafik boyutu
185	n_size_of_http_traffic	Toplam HTTP ağ trafik boyutu
186	n_ssl_usage	SSL kullanımı
187	n_time_eplsd_dns_traffic	İlk ve son DNS paketi arasında geçen süre
188	n_time_eplsd_http_traffic	İlk ve son HTTP paketi arasında geçen süre
189	n_time_first_dns_packet	İlk DNS paketi zamanı
190	n_time_first_http_packet	İlk HTTP paketi zamanı
191	n_time_last_dns_packet	Son DNS paketi zamanı
192	n_time_last_http_packet	Son HTTP paketi zamanı
193	n_total_dns_bytes	Toplam DNS ağ trafik boyutu
194	n_total_dns_domains	DNS isteklerindeki tekil alan adı adedi
195	n_total_dns_req	Toplam DNS istek adedi
196	n_total_dns_res	Toplam DNS cevap adedi
197	n_total_udp_bytes	Toplam UDP ağ trafik boyutu
198	n_total_udp_frames	Toplam UDP frame adedi
199	n_uniq_get_url	Tekil GET istek URL adedi
200	n_uniq_http_host	Tekil HTTP isteği gönderilen ana makine adedi
201	n_uniq_post_url	Tekil POST istek URL adedi
	DroidBox Çalışma Zamanlı Aktiviteler	
202	closenet	Sonlandırılan ağ bağlantıları
203	cryptousage	Android API destekli olan kriptografik algoritmaların kullanımları
204	dataleaks	Dosya, ağ veya SMS ile veri sızması durumları
205	decryption	Şifre çözme işlemleri
206	dexclass	DexClassLoader sınıfı kullanılarak dinamik kod yükleme durumları
207	encryption	Şifreleme işlemleri
208	enfperm	AndroidManifest içerisinde belirtilen izinler dışında kullanılan izinler
209	fdaccess	Dosya erişimleri
210	file_read	Dosya okumaları
211	file_write	Dosya yazmaları
212	fileleak	Dosya ifşaları
213	fr_cmdline	“cmdline” dosyasının okumaları
214	fr_data	“data” dizini altından dosya okuma işlemleri
215	fr_meminfo	“meminfo” dosyasının okumaları

216	fr_urandom	Rastgele dosya okuma işlemi
217	fw_data	“data” dizininde yazma işlemleri
218	fw_eventX	“event” dosyasına yazma işlemleri
219	networkleak	Ağ tabanlı bilgi ifşaları
220	opennet	Yeni açılan ağ bağlantıları
221	phonecalls	Telefon aramaları
222	recvnet	Kabul edilen ağ bağlantıları
223	runtimesystemevents	Kullanılan Android sistem olayları
224	sendnet	Kurulu bir ağ bağlantı üzerinden istek gönderimleri
225	sendsms	SMS gönderimleri
226	servicestart	Servis başlatımları
227	smsleak	SMS yolu ile bilgi ifşaları
228	keyalgo	Şifreleme sırasında kullanılan anahtarların algoritma bilgileri
229	sswrite	SSL kullanımları
	İkili Aktivite Öznitelikleri (“DroidBox Çalışma Zamanlı Aktiviteler” tablosu içerisinden tercih edilen aktivitelerden oluşturulmuştur.)	
230	recvnet-cryptousage	Tez Metni Bölüm 4.2.2
231	recvnet-dataleaks	Tez Metni Bölüm 4.2.2
232	recvnet-dexclass	Tez Metni Bölüm 4.2.2
233	recvnet-fread	Tez Metni Bölüm 4.2.2
234	recvnet-fwrite	Tez Metni Bölüm 4.2.2
235	recvnet-recvnet	Tez Metni Bölüm 4.2.2
236	recvnet-runtimesystemevents	Tez Metni Bölüm 4.2.2
237	recvnet-sendnet	Tez Metni Bölüm 4.2.2
238	recvnet-sendsms	Tez Metni Bölüm 4.2.2
239	recvnet-servicestart	Tez Metni Bölüm 4.2.2
240	runtimesystemevents-cryptousage	Tez Metni Bölüm 4.2.2
241	runtimesystemevents-dataleaks	Tez Metni Bölüm 4.2.2
242	runtimesystemevents-dexclass	Tez Metni Bölüm 4.2.2
243	runtimesystemevents-fread	Tez Metni Bölüm 4.2.2
244	runtimesystemevents-fwrite	Tez Metni Bölüm 4.2.2
245	runtimesystemevents-recvnet	Tez Metni Bölüm 4.2.2
246	runtimesystemevents-runtimesystemevents	Tez Metni Bölüm 4.2.2
247	runtimesystemevents-sendnet	Tez Metni Bölüm 4.2.2
248	runtimesystemevents-sendsms	Tez Metni Bölüm 4.2.2

249	runtimesystemevents-servicestart	Tez Metni Bölüm 4.2.2
250	sendnet-cryptousage	Tez Metni Bölüm 4.2.2
251	sendnet-dataleaks	Tez Metni Bölüm 4.2.2
252	sendnet-dexclass	Tez Metni Bölüm 4.2.2
253	sendnet-fread	Tez Metni Bölüm 4.2.2
254	sendnet-fwrite	Tez Metni Bölüm 4.2.2
255	sendnet-recvnet	Tez Metni Bölüm 4.2.2
256	sendnet-runtimesystemevents	Tez Metni Bölüm 4.2.2
257	sendnet-sendnet	Tez Metni Bölüm 4.2.2
258	sendnet-sendsms	Tez Metni Bölüm 4.2.2
259	sendnet-servicestart	Tez Metni Bölüm 4.2.2
260	sendsms-cryptousage	Tez Metni Bölüm 4.2.2
261	sendsms-dataleaks	Tez Metni Bölüm 4.2.2
262	sendsms-dexclass	Tez Metni Bölüm 4.2.2
263	sendsms-fread	Tez Metni Bölüm 4.2.2
264	sendsms-fwrite	Tez Metni Bölüm 4.2.2
265	sendsms-recvnet	Tez Metni Bölüm 4.2.2
266	sendsms-runtimesystemevents	Tez Metni Bölüm 4.2.2
267	sendsms-sendnet	Tez Metni Bölüm 4.2.2
268	sendsms-sendsms	Tez Metni Bölüm 4.2.2
269	sendsms-servicestart	Tez Metni Bölüm 4.2.2
270	servicestart-cryptousage	Tez Metni Bölüm 4.2.2
271	servicestart-dataleaks	Tez Metni Bölüm 4.2.2
272	servicestart-dexclass	Tez Metni Bölüm 4.2.2
273	servicestart-fread	Tez Metni Bölüm 4.2.2
274	servicestart-fwrite	Tez Metni Bölüm 4.2.2
275	servicestart-recvnet	Tez Metni Bölüm 4.2.2
276	servicestart-runtimesystemevents	Tez Metni Bölüm 4.2.2
277	servicestart-sendnet	Tez Metni Bölüm 4.2.2
278	servicestart-sendsms	Tez Metni Bölüm 4.2.2
279	servicestart-servicestart	Tez Metni Bölüm 4.2.2
280	cryptousage-cryptousage	Tez Metni Bölüm 4.2.2
281	cryptousage-dataleaks	Tez Metni Bölüm 4.2.2
282	cryptousage-dexclass	Tez Metni Bölüm 4.2.2
283	cryptousage-fread	Tez Metni Bölüm 4.2.2
284	cryptousage-fwrite	Tez Metni Bölüm 4.2.2
285	cryptousage-recvnet	Tez Metni Bölüm 4.2.2
286	cryptousage-runtimesystemevents	Tez Metni Bölüm 4.2.2
287	cryptousage-sendnet	Tez Metni Bölüm 4.2.2

288	cryptousage-sendsms	Tez Metni Bölüm 4.2.2
289	cryptousage-servicestart	Tez Metni Bölüm 4.2.2
290	dataleaks-cryptousage	Tez Metni Bölüm 4.2.2
291	dataleaks-dataleaks	Tez Metni Bölüm 4.2.2
292	dataleaks-dexclass	Tez Metni Bölüm 4.2.2
293	dataleaks-fread	Tez Metni Bölüm 4.2.2
294	dataleaks-fwrite	Tez Metni Bölüm 4.2.2
295	dataleaks-recvnet	Tez Metni Bölüm 4.2.2
296	dataleaks-runtimesystemevents	Tez Metni Bölüm 4.2.2
297	dataleaks-sendnet	Tez Metni Bölüm 4.2.2
298	dataleaks-sendsms	Tez Metni Bölüm 4.2.2
299	dataleaks-servicestart	Tez Metni Bölüm 4.2.2
300	dexclass-cryptousage	Tez Metni Bölüm 4.2.2
301	dexclass-dataleaks	Tez Metni Bölüm 4.2.2
302	dexclass-dexclass	Tez Metni Bölüm 4.2.2
303	dexclass-fread	Tez Metni Bölüm 4.2.2
304	dexclass-fwrite	Tez Metni Bölüm 4.2.2
305	dexclass-recvnet	Tez Metni Bölüm 4.2.2
306	dexclass-runtimesystemevents	Tez Metni Bölüm 4.2.2
307	dexclass-sendnet	Tez Metni Bölüm 4.2.2
308	dexclass-sendsms	Tez Metni Bölüm 4.2.2
309	dexclass-servicestart	Tez Metni Bölüm 4.2.2
310	fread-cryptousage	Tez Metni Bölüm 4.2.2
311	fread-dataleaks	Tez Metni Bölüm 4.2.2
312	fread-dexclass	Tez Metni Bölüm 4.2.2
313	fread-fread	Tez Metni Bölüm 4.2.2
314	fread-fwrite	Tez Metni Bölüm 4.2.2
315	fread-recvnet	Tez Metni Bölüm 4.2.2
316	fread-runtimesystemevents	Tez Metni Bölüm 4.2.2
317	fread-sendnet	Tez Metni Bölüm 4.2.2
318	fread-sendsms	Tez Metni Bölüm 4.2.2
319	fread-servicestart	Tez Metni Bölüm 4.2.2
320	fwrite-cryptousage	Tez Metni Bölüm 4.2.2
321	fwrite-dataleaks	Tez Metni Bölüm 4.2.2
322	fwrite-dexclass	Tez Metni Bölüm 4.2.2
323	fwrite-fread	Tez Metni Bölüm 4.2.2
324	fwrite-fwrite	Tez Metni Bölüm 4.2.2
325	fwrite-recvnet	Tez Metni Bölüm 4.2.2
326	fwrite-runtimesystemevents	Tez Metni Bölüm 4.2.2

327	fwrite-sendnet	Tez Metni Bölüm 4.2.2
328	fwrite-sendsms	Tez Metni Bölüm 4.2.2
329	fwrite-servicestart	Tez Metni Bölüm 4.2.2

8.2. Ek 2

Familial Classification of Android Malware using Hybrid Analysis

1st Omer Faruk Turan Cavli
WISE Lab., Department of Computer Engineering
Hacettepe University
Ankara, Turkey
turan.cavli@hacettepe.edu.tr

2nd Sevil Sen
WISE Lab., Department of Computer Engineering
Hacettepe University
Ankara, Turkey
ssen@cs.hacettepe.edu.tr

***Abstract*—With the developments in mobile and wireless technology, mobile devices have become important part of our lives. While Android is the leading operating system in the market share, it is also the most targeted platform by attackers. While there have been many solutions proposed for detection of Android malware in the literature, the family classification of detected malicious applications becomes important, especially where the number of mobile malware variants increases every day in the market. In this study, a solution based on machine learning and hybrid analysis is proposed for the Android malware familial classification problem. An extensive feature set including network-related features and activity bigrams is proposed. The effective static and dynamic analysis features are studied thoroughly and evaluated on Malgenome [1], Drebin [2], and UpDroid [3] datasets.**

***Index Terms*—Android, mobile security, malware analysis and detection, malware family classification, machine learning, static/dynamic analysis, hybrid analysis**

I. Introduction

According to the research of International Data Corporation (IDC), Android has constituted the biggest share of mobile market with 86.1% in 2019 and this share is predicted to increase in the following years [4]. Hence it becomes the most targeted mobile platform by attackers. Malicious applications can harm end users in many ways such as stealing their private data, spamming, sending sms to premium rate numbers. Nowadays, the main motivation of attackers are financial gain.

Every day new mobile malware or new variations of existing malware are introduced by malicious attackers and uploaded to market stores. According to the McAfee Mobile Threat Report Q1 2020, the number of mobile malware increases rapidly every year. It is also stated that evasion techniques such as obfuscation, updating in order to bypass security solutions are used more than before [5]. According to another report published by Kaspersky [6], the number of 1,152,662 mobile malware in the first quarter of 2020 increased by 93,232 and reached 1,245,894 in the second quarter of 2020. Because of increasing number of mobile malware that bypass analysis methods, hybrid analysis techniques become more important for malware detection and familial classification, since an evasion technique that evades one type of analysis could be caught by another type of analysis.

There are two main malware analysis and detection techniques in the literature: static and dynamic analysis. In static analysis, a suspicious application is analyzed without running it on any device and only the apk file is analyzed. Source code and other components of the application such as manifest file are analyzed. On one hand, since the application does not need to be run on any devices, static analysis is usually more efficient than dynamic analysis. On the other hand, attackers could easily bypass static analysis by using evasive techniques such as obfuscation, dynamic code loading and encryption. In dynamic analysis, suspicious application is run on a virtual or real device for the purpose of monitoring run time behaviours of the application such as API/system calls, network traffic, file operations. While dynamic analysis is robust to obfuscation techniques, the main problem here is to trigger malicious behaviour during the analysis, which is carried out for a limited time. Attacker could bypass dynamic analysis by waiting for a specific time or an action in order to run the malicious code. Moreover, if the attacker understands that it is run in an emulator, it may not trigger the malicious code. Hybrid analysis aims to take advantage of both techniques.

As a result of the arms race between attackers and security solutions, while attackers always develop their techniques in order to bypass security mechanisms, in a similar way, security developers improve their solutions against new malware. Besides new malware, attackers commonly introduce new variants of existing malware. Therefore, not only detection of malicious applications, but also their familial classification has become significantly important. When a malicious application is detected, if its family is identified, its effect on the device could be minimized with taking predetermined actions. Moreover, the familial classification helps to reduce the manual analysis time of malware.

In this study, a new multi-class classification algorithm is proposed in order to group malware into their families. Although we have applied well-known machine learning approaches such as k-NN, SVM, Random Forest and Decision Tree algorithms, the main contribution of the study is to extensively analyze effective static and dynamic features on mobile malware familial classification. Few studies in the literature, namely UpDroid [3] and Ec2 [7], have also employed hybrid analysis for malware family classification. In this study, in addition to the hybrid features that are employed in those previous studies, new feature groups such as network-related

features, activity bigrams are firstly employed for malware family classification and thoroughly analyzed. We focus on update attacks that are emphasized in UpDroid [3] study. Besides UpDroid, the proposed features are evaluated on Malgenome [1] and Drebin [2] datasets. It is shown that even only using network-related features can be very effective for familial classification on the UpDroid [3] dataset and high accuracy is obtained. In general, newly added features have positive effects on malware familial classification and has shown to produce better results than Ec2 [7] and UpDroid [3].

The rest of the paper is organized as follows: Section II summarizes the related approaches in the literature. The proposed approach, particularly on how features are selected and extracted, are given in details in Section III. In Section IV, the datasets used in the experiments are presented, and the experimental results are discussed. The effect of each feature group on malware family classification is extensively analyzed. Finally, the study is concluded in Section V.

II. Related Work

Machine learning-based techniques have already been proposed to classify Android malware by using static, dynamic or hybrid approaches. However, most of these approaches employ either only static features or only dynamic features. There are also a few study that employ hybrid features. In this section, these approaches are reviewed.

Malgenome [1] categorizes Android malware under 4 groups according to methods they applied: repackaging, update attack, drive-by download and standalone. They have collected 1260 Android malware sample belonging to 49 different family from different app markets. They analyzed these 1260 Android malware samples manually over a year. In Drebin [2], machine learning algorithms were used for classifying Android malware into families. They have used only static features because of the limited resources for dynamic analysis. The feature vector that they have used consisted of Hardware Components, Permissions, Intents, Activities, Services, Content Providers and Broadcast Receivers that was given in AndroidManifest.xml file. In addition, they have extracted restricted API calls, IP addresses, URLs, hostnames from the source code of an application. The Malgenome and Drebin datasets are extensively used in the studies in order to compare their approaches with other proposals in the literature.

Dendroid [8] employs machine learning algorithms on Android code blocks/structures. These code blocks are represented as control flow graphs. For each malware family, the commonly used code blocks and their count in the application code were extracted. Droidminer [9] uses behavioural graphs of mobile applications. DroidSIFT [10] is a semantic-based approach that classifies Android malware via dependency graphs obtained from source codes of applications. Another work on familial classification study is Droidlegacy [11]. They extract the most common malicious code blocks and Android API calls in a family. It is tested only with 11 families and achieved to get an accuracy of 98%. On the other hand, attackers can mislead the model by using the most common libraries.

As attackers use code obfuscation and code loading techniques more than ever, for some cases static analysis approach has remained incapable of classifying or detecting malware samples accurately. Therefore, a dynamic analysis based approach has been proposed to address the issue. In DroidScribe [12], the features such as total network traffic size, accessed IP/Port numbers, files and Binder-IPC methods are extracted for malware familial classification. To the best of our knowledge, Ec2 [7] and UpDroid [3] are the only studies that employ hybrid approach for analyzing and classifying Android malware samples by using machine learning techniques. UpDroid also presents a dataset of malware that use updating techniques for downloading malicious code at runtime.

There are also few studies that analyze network activities of malicious applications. CREDROID [13] analyzes the network activities of the samples in the Malgenome dataset. They extract information related to the following protocols: HTTP, DNS and SSL protocols. It is noticed that 63% of the samples generate network traffic. Rest of them either do not generate network traffic or are not compatible with the environment. In the current study, we employ also HTTP and DNS traffic related features for malware familial classification. Chen et al. [14] analyzes the network activities of the samples in the Drebin dataset. They obtain similar results with the current study that more than 70% malware samples generates malicious traffic in their first five minutes. By using information collected from DNS query and HTTP request packets, they obtained 69.55% and 40.89% detection rates respectively.

III. Hybrid Analysis

The increasing variety of Android malware has caused to increase the number of families. So researchers have recently focused on the malware family classification. In these approaches, mostly static analysis are employed, there are only few studies based on dynamic analysis [12]. However, as noted above, both analysis have pros and cons, therefore hybrid analysis is employed here. Choosing the right features is very important for producing effective classifiers and hence distinguishing the characteristics of malware families. This is the main of this study. By using features collected from both static and dynamic analysis, the effective features on malware familial classification are explored. The general overview of the proposed approach is given in Figure 1.

The features employed in this study is represented in detail in the subsequent section. Well-known machine learning algorithms, namely k-nearest neighbors (k-NN), decision tree, random forest and Support Vector Machine (SVM), has been applied on these features. Here, Weka tool [15] is used to implement the machine learning algorithms and the default setting of these algorithms as given in Weka is employed.

A. Feature extraction

In static analysis, features are usually extracted from the manifest file or directly from the code. Firstly, a reverse engi-

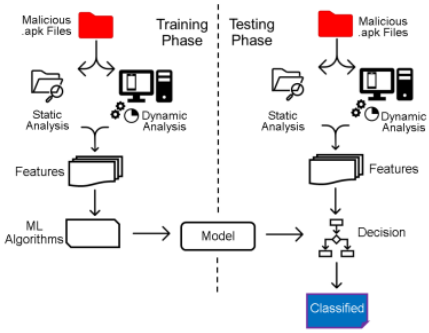


Fig. 1. Simplified schema of the proposed approach

neering tool called Apktool [16] is used to obtain the source files of an application such as assests, source code, manifest file. Here, the static features collected from the manifest file is employed as given in UpDroid [3]. Accessing to personal data such as call list, sms, gallery, hardware components (e.g. camera, bluetooth, microphone) can only be used by granting permission by the user. Therefore, the permissions, which are among the most used features in Android malware detection [17], are employed in this study. Other static features are the number of services, activities, receivers, extra components collected from the manifest file and the apk size.

As noted above, features collected from both static and dynamic analysis are explored in this study. Besides features given in UpDroid [3], two new feature groups collected from dynamic analysis are introduced in this study: network-related features and activity bigrams. In UpDroid [3], 175 features are employed in total, where most of them (110) are dynamic features since the main focus is to classify update attacks. The majority of static features are obtained from the manifest file such as number of services, activities, receivers and permissions. Dynamic features obtained from the output of DroidBox [18] mainly consist of the number of operations at runtime such as file operations, sent SMSs, phone conversations, cryptographic and dynamic code loading operations, data leakage.

Besides general information about network traffic such as the number of sent and received network packets given in Table III, most of the network-related features used in this study have been extracted from HTTP and DNS protocol headers. HTTP traffic could be used for leaking some important data such as SDK version, IMEI and IMSI numbers. For example, Kmin family could leak the SDK version over HTTP traffic [14]. Moreover, it is observed that samples in a same family can show similarity in their HTTP and DNS traffic patterns. For example, most of the samples belonging to the shedun family has the same URL length in GET requests. Some malware families try to leak fixed sized IMSI and IMEI numbers of victim devices in HTTP GET requests. In this study, the existence of the following extensions .apk and .jar in HTTP request/reply messages is also included in the features. Especially update attacks can download their malicious code at runtime, these features could be indicators for malicious activity. Therefore, these newly added features control whether the application has downloaded a file successfully that has the extension of .jar or .apk. The content types of HTTP responses (app, text, video, audio, multi, and image) are also considered. Besides IMSI/IMEI information of a device, malicious applications can send other important data such contact lists, phone numbers via HTTP requests as URL parameters. Therefore, the average number of parameters in a URL is added into the features. Besides HTTP traffic, features related to DNS requests/responses are included in the feature set as listed in Table II. To sum up, there are overlapping network-related features with UpDroid [3] such as total number of opened/closed connections, total number of network connections, total size of network packets and number of sent/received network packets. In addition, we analyse network packets more deeply and extract new features from the headers of the following protocols: IP, TCP, UDP, HTTP, DNS.

TABLE I
FEATURES RELATED TO HTTP PROTOCOL

HTTP Features	Type	Count
Content type of HTTP traffic	Boolean	6
Size of HTTP traffic in each time interval (byte)	Numeric	3
Total size of HTTP traffic (byte)	Numeric	1
Number of HTTP requests/responses	Numeric	2
Number of HTTP frames	Numeric	1
Average content length in HTTP responses (byte)	Numeric	1
Time of the first/last HTTP packet	Numeric	2
Total time period of HTTP traffic	Numeric	1
Number of unique HTTP servers	Numeric	1
Average URL length in GET/POST messages	Numeric	2
Average number of parameters in GET/POST messages	Numeric	2
Existence of .apk/.jar extension in GET/POST messages	Boolean	4
Number of GET/POST requests	Numeric	2
Number of unique URLs in GET/POST messages	Numeric	2
Size of POST messages (byte)	Numeric	1

TABLE II
FEATURES RELATED TO DNS PROTOCOL

DNS Features	Type	Count
Time of the first/last DNS packet	Numeric	2
Total time period of DNS traffic	Numeric	1
Total number of unique domains in DNS queries	Numeric	1
Average length of DNS domain names	Numeric	1
Total number of DNS queries/responses	Numeric	2
Total size of DNS packets (byte)	Numeric	1

Malicious code part of an Android application could be triggered at different times [19]. A malware can run its malicious code when the mobile device is booted or after a certain period of time passed. The dynamic analysis tools in Android generally run for 10 minutes [20]. Hence, attackers can postpone triggering malicious code in a virtual/real device. Therefore, in this study, we run each application for 15 minutes and divide the analysis time into three equal parts (each for

5 minutes). Some features collected separately for each time period, since malicious code might be active in only one period. Here, Droidbox is used for collecting data produced at runtime.

TABLE III
OTHER NETWORK-RELATED FEATURES

Other Network-Based Features	Type	Count
Total network traffic size in each time period (byte)	Numeric	3
Total network traffic size (byte)	Numeric	2
Total UDP traffic size (byte)	Numeric	1
Number of UDP frames	Numeric	1
Number of SYN packets	Numeric	1
Number of unique IP/Port	Numeric	2
Number of opened/closed network connections	Numeric	2
Number of sent/received network packets	Numeric	2
SSL usage in network traffic	Boolean	1
Average size of IP packets (byte)	Numeric	1

Malware can leak personal and/or device information with different activity sequences (e.g. reading a specific file on the device device and sending this information over the network). Therefore, consecutive activities performed by applications are taken into account in this study. Here, the bigrams of the activities given in Table IV are used and the number of bigrams encountered in the application are given to the model as features. Hence, 100 (10x10) features are collected for this feature group. All these 100 features are firstly employed in this study. So, together with 175 features given in UpDroid, in total, 329 features are used in this study.

TABLE IV
ACTIVITIES USED IN BIGRAMS

Activity Name	Description
cryptousage	Usage of cryptographic alg. with support of Android API
dataleaks	Information leakage via SMS, file or network
dexclass	Code loading by using DexClassLoader
fread	Reading files
fwrite	Writing files
recvnet	Network connections received
sendnet	Network connections requested
runtimesystemevents	Usage of Android system events
sendsms	Sending SMS
servicestart	Starting services

IV. EXPERIMENTAL RESULTS

In this section, features on malware family classification are evaluated on three datasets, namely Malgenome [1], Drebin [2] and UpDroid [3], and discussed. Malgenome is the first dataset introduced and employed for comparison in the literature. Malgenome has 1,260 applications belonging to 49 families. This dataset is extended in Drebin [2], which has 5554 applications belonging to 178 families. UpDroid is one of the recently proposed datasets. It consists of 2535 update attacks belonging to 21 malware families. An update attack load its malicious code at runtime. Since network-based features could be more effective on update attacks, this dataset is included in the experiments. Moreover, the feature set employed in the UpDroid study is extended in the current study.

In the experiments, families that have larger samples have more impact on results. Therefore, families that have more than 10 samples are included in the evaluations. 10 fold cross-validation method is employed for comparisons as in UpDroid. Table V shows the results. Three metrics are employed for comparison: accuracy, true positives (TP) or detection rate in other words and, false positives (FP). In this table, all features are included. The best results are obtained with the kNN algorithm, so unless specified differently, the results of this algorithm are given in the subsequent results.

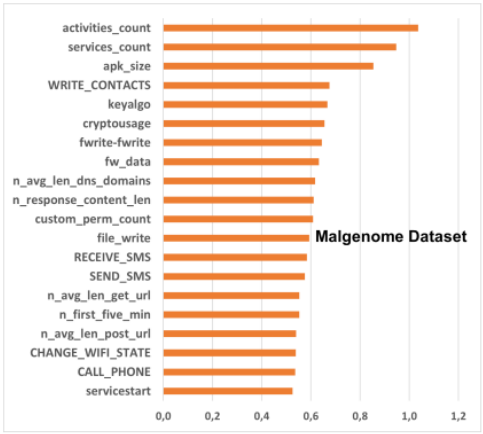


Fig. 2. Most effective 20 features for the Malgenome dataset

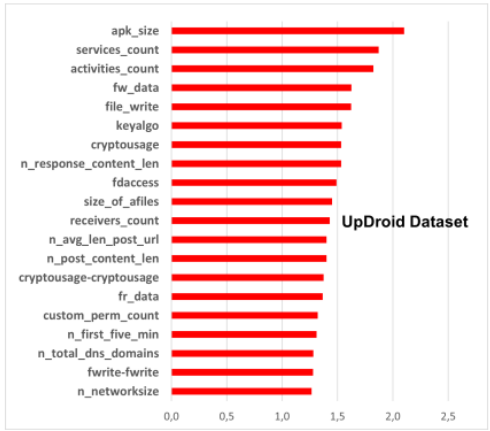


Fig. 3. Most effective 20 features for the UpDroid dataset

Figure 2, Figure 3 and Figure 4 show the most effective top twenty features on datasets. Features starting with "n" are representing network-related features. It is observed that quite

TABLE V
EFFECTS OF ALL FEATURES ON MALGENOME [1], DREBIN [2] AND UPDROID [3]

Algorithm	Accuracy %			TP %			FP %		
	Malgenome	UpDroid	Drebin	Malgenome	UpDroid	Drebin	Malgenome	UpDroid	Drebin
kNN	97.38	98.04	96.40	97.4	98.0	96.4	1.4	0.4	0.3
Random Forest	93.49	97.44	92.33	93.5	97.4	92.3	3.9	0.6	0.9
Decision Tree	96.89	97.53	93.58	96.9	97.5	93.6	1.3	0.4	0.5
SVM	97.83	97.44	95.77	97.8	97.4	95.8	0.8	0.8	0.4

number of network-related features are in top 20 for all three datasets. As seen in the figure, the size of the network traffic collected in the first five minutes is the common feature in all datasets. It is observed that malicious applications mostly generate network traffic in their first five minutes. In addition, the average length of URL of GET and POST requests are among the most distinguishing features in all datasets.

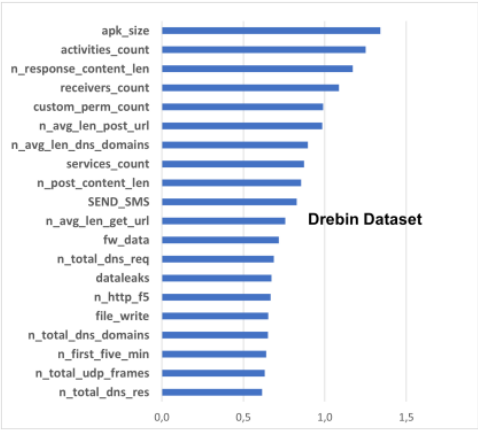


Fig. 4. Most effective 20 features for the Drebin dataset

The comparison with UpDroid is given in Table VI. As shown in the results, the proposed features have positive effects on the results. By using all three machine learning algorithms employed in UpDroid [3], the proposed approach produces higher accuracy. Please note that the results are evaluated on the UpDroid dataset.

TABLE VI
COMPARISON WITH UPDROID [3] ON THE UPDROID DATASET

Study	Algorithm	Accuracy %	TP %	FP %
Our Approach	kNN	**98.04**	98.0	0.4
UpDroid	kNN	96.85	96.4	0.4

Figure 5 shows the confusion matrix of the results given in Table VI. The comparison results on the Drebin dataset [2] are given in Table VII. Here, the proposed approach produces comparable results with UpDroid.

TABLE VII
COMPARISON WITH UPDROID [3] ON THE DREBIN DATASET

Study	Algorithm	Accuracy %	TP %	FP %
UpDroid	kNN	**96.85**	**96.8**	**0.3**
Our Approach	kNN	96.39	96.4	0.3

```
 a   b   c   d   e   f   g   h   i   j   k   l   m   <-- classified as
51   1   1   1   0   0   0   0   0   0   0   0   0 |   a = asacub
 3  20   0   0   1   0   0   0   0   0   0   0   0 |   b = bankbot
 2   0   9   0   0   0   0   0   0   0   0   0   0 |   c = faketoken
 0   0   0 178   0   1   0   1   1   0   0   0   0 |   d = malap
 0   1   0   0  20   0   0   0   0   0   0   0   0 |   e = marcher
 0   0   0   1   0   8   1   0   0   0   0   0   0 |   f = ogel
 0   0   0   0   0   0  27   0   2   0   0   3   0 |   g = rootnik
 0   0   0   0   0   0   0 629   1   0   0   0   0 |   h = shedun
 0   0   0   1   2   0   1   0 282   0   0   5   0 |   i = smsreg
 3   0   0   0   1   0   0   0   0  53   0   0   0 |   j = smsspy
 0   0   0   0   0   0   0   0   0   0  11   0   0 |   k = tordow
 0   0   0   1   0   0   1   0   6   1   0 1000   1 |   l = triada
 0   0   0   0   1   0   0   0   0   0   0   1  13 |   m = ztorg
```

Fig. 5. Confusion matrix for the UpDroid Dataset

As it is shown in Figure 5, the highest TP rate is obtained without misdetection in Tordow malware family which has only 11 samples. Malicious applications belong to Tordow family usually download other application files to the device at runtime [21]. Triada ans Shedun families, whose have high number of samples in the dataset, also have high TP rates. Despite their high TP rates, most of confusions are obtained between Triada and Smsreg families. As it is stated in [22] and [23] that both families have adware purposes and, download harmless adware applications after being installed on a victim device.

The effects of new features are represented in Table VIII. Since the samples in the UpDroid dataset generates more network traffic than the samples in the Drebin dataset, network-based features have more positive impact on the UpDroid dataset as shown in Table VIII. It can be inferred from the table that over 90% of accuracy is achieved with only 55 different network-related feature on the UpDroid dataset. 10-fold cross validation is used for obtaining results in Table VIII. Majority of malicious applications in the UpDroid dataset have higher network activity than applications in Drebin and Malgenome datasets. Similarly, newly added activity bigrams features have more positive effect on the UpDroid dataset.

Finally, the proposed approach is compared with Ec2 [7] and UpDroid [3], two studies that use hybrid features for malware familial classification. Drebin dataset [2] is used for a fair comparison. In Ec2 [7], the following metrics are used for the performance evaluation: MiF and MiAUC scores. Therefore, the same metrics are employed in Table IX. It is shown that our

TABLE VIII
EFFECTS OF NEW FEATURES

	Accuracy %			TP %			FP %		
FEATURES / DATASETS	**Malgenome**	**UpDroid**	**Drebin**	**Malgenome**	**UpDroid**	**Drebin**	**Malgenome**	**UpDroid**	**Drebin**
Network-Related	73.99	91.72	61.38	74.0	91.7	61.4	8.8	2.3	4.3
Activity Bigrams	75.86	92.61	70.59	75.9	92.6	70.6	8.6	2.0	2.6
Network-Related + Activity Bigrams	79.59	92.78	77.11	79.6	92.8	77.1	8.1	2.0	2.1
UpDroid Features + Network-Related	97.45	98.17	95.92	97.5	98.2	95.9	1.3	0.4	0.3
UpDroid Features + Activity Bigrams	97.73	98.04	96.32	97.7	98.0	96.3	1.1	0.3	0.3
All Features	97.36	98.04	96.40	97.4	98.0	96.4	1.4	0.4	0.3

approach outperforms than Ec2 [7] and shows similar results with UpDroid [3] on these metrics.

TABLE IX
COMPARISON WITH EC2 [7] AND UPDROID [3] ON THE DREBIN DATASET

Study	**Algorithm**	**MiF**	**MiAUC**
Ec2	kNN	0,47	0,73
UpDroid	kNN	0,96	0,98
Our Approach	kNN	0,96	0,98
Ec2	Random Forest	0,95	0,97
UpDroid	Random Forest	0,94	0,99
Our Approach	Random Forest	0,94	0,99

V. CONCLUSION

In this study, a hybrid approach is proposed for malware familial classification. Besides features given in UpDroid [3], the effects of two new feature groups, network-based features and activity bigrams, are explored. To the best of the authors' knowledge, these two feature groups are firstly used on malware familial classification. The results show the positive effect of these features, particularly on the UpDroid dataset [3], since malicious applications in this dataset have more network activity than samples in other datasets, Malgenome and Drebin.

Acknowledgements This study is partly supported by the Scientific and Technological Research Council of Turkey (TUBITAK-115E150). We thank TUBITAK for its support. We also would like to thank Kursat Aktas for his help on the study.

REFERENCES

[1] Zhou, Y., Jiang, X.: Dissecting android malware: Characterization and evolution. In: 2012 IEEE symposium on security and privacy. pp. 95–109. IEEE (2012)
[2] Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K., Siemens, C.: Drebin: Effective and explainable detection of android malware in your pocket. In: Ndss. vol. 14, pp. 23–26 (2014)
[3] Aktas, K., Sen, S.: Updroid: Updated android malware and its familial classification. In: Nordic Conference on Secure IT Systems. pp. 352–368. Springer (2018)
[4] IDC: Smartphone market share. https://www.idc.com/promo/smartphone-market-share/os (2020)
[5] McAfee: Mcafee mobile threat report q1-2020. https://www.mcafee.com/content/dam/consumer/en-us/docs/2020-Mobile-Threat-Report.pdf (2020)
[6] Chebyshev, V.: It threat evolution q2 2020. mobile statistics. https://securelist.com/it-threat-evolution-q2-2020-mobile-statistics/98337/ (2020)
[7] Chakraborty, T., Pierazzi, F., Subrahmanian, V.: Ec2: Ensemble clustering and classification for predicting android malware families. IEEE Transactions on Dependable and Secure Computing 17(2), 262–277 (2017)
[8] Suarez-Tangil, G., Tapiador, J.E., Peris-Lopez, P., Blasco, J.: Dendroid: A text mining approach to analyzing and classifying code structures in android malware families. Expert Systems with Applications 41(4), 1104–1117 (2014)
[9] Yang, C., Xu, Z., Gu, G., Yegneswaran, V., Porras, P.: Droidminer: Automated mining and characterization of fine-grained malicious behaviors in android applications. In: European symposium on research in computer security. pp. 163–182. Springer (2014)
[10] Zhang, M., Duan, Y., Yin, H., Zhao, Z.: Semantics-aware android malware classification using weighted contextual api dependency graphs. In: Proceedings of the 2014 ACM SIGSAC conference on computer and communications security. pp. 1105–1116 (2014)
[11] Deshotels, L., Notani, V., Lakhotia, A.: Droidlegacy: Automated familial classification of android malware. In: Proceedings of ACM SIGPLAN on program protection and reverse engineering workshop 2014. pp. 1–12 (2014)
[12] Dash, S.K., Suarez-Tangil, G., Khan, S., Tam, K., Ahmadi, M., Kinder, J., Cavallaro, L.: Droidscribe: Classifying android malware based on runtime behavior. In: 2016 IEEE Security and Privacy Workshops (SPW). pp. 252–261. IEEE (2016)
[13] Malik, J., Kaushal, R.: Credroid: Android malware detection by network traffic analysis. In: Proceedings of the 1st acm workshop on privacy-aware mobile computing. pp. 28–36 (2016)
[14] Chen, Z., Han, H., Yan, Q., Yang, B., Peng, L., Zhang, L., Li, J.: A first look at android malware traffic in first few minutes. In: 2015 IEEE Trustcom/BigDataSE/ISPA. vol. 1, pp. 206–213. IEEE (2015)
[15] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The weka data mining software: an update. ACM SIGKDD explorations newsletter 11(1), 10–18 (2009)
[16] Connor Tumbleson, R.W.: A tool for reverse engineering android apk files. https://ibotpeaches.github.io/Apktool/ (2010)
[17] Feizollah, A., Anuar, N.B., Salleh, R., Wahab, A.W.A.: A review on feature selection in mobile malware detection. Digital investigation 13, 22–37 (2015)
[18] pjlantz: Droidbox: Dynamic analysis of android apps. https://github.com/pjlantz/droidbox (2018)
[19] Garcia, J., Hammad, M., Malek, S.: Lightweight, obfuscation-resilient detection and family identification of android malware. ACM Transactions on Software Engineering and Methodology (TOSEM) 26(3), 1–29 (2018)
[20] Maier, D., Protsenko, M., Müller, T.: A game of droid and mouse: The threat of split-personality malware on android. Computers & Security 54, 2–15 (2015)
[21] Comodo: Comodo threat research labs warns android users of tordow v2.0 outbreak. https://blog.comodo.com/comodo-news/comodo-warns-android-users-of-tordow-v2-0-outbreak/ (March 2018)
[22] Data, G.: Analysis of xafecopy android.application.smsreg.a. https://file.gdatasoftware.com/web/en/documents/whitepaper/G_Data_Analysis_Xafecopy.pdf (2017)
[23] Snow, J.: Triada: organized crime on android. https://www.kaspersky.com/blog/triada-trojan/11481/ (2016)