T.C.
SÜLEYMAN DEMİREL ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

ANDROID İŞLETİM SİSTEMİ İÇİN DERİN ÖĞRENME TABANLI KÖTÜ AMAÇLI YAZILIM TESPİT ARACI GELİŞTİRME

Mahmut TOKMAK

Danışman
Prof. Dr. Ecir Uğur KÜÇÜKSİLLE

DOKTORA TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI
ISPARTA - 2020

TEZ ONAYI

Mahmut TOKMAK tarafından hazırlanan **"Android İşletim Sistemi İçin Derin Öğrenme Tabanlı Kötü Amaçlı Yazılım Tespit Aracı Geliştirme"** adlı tez çalışması aşağıdaki jüri üyeleri önünde Süleyman Demirel Üniversitesi Fen Bilimleri Enstitüsü **Bilgisayar Mühendisliği Anabilim Dalı**'nda **DOKTORA TEZİ** olarak başarı ile savunulmuştur.

Danışman	**Prof. Dr. Ecir Uğur KÜÇÜKSİLLE** Süleyman Demirel Üniversitesi	
Jüri Üyesi	**Prof. Dr. Tuncay AYDOĞAN** Isparta Uygulamalı Bilimler Üniversitesi	
Jüri Üyesi	**Dr. Öğr. Üyesi Kubilay TAŞDELEN** Isparta Uygulamalı Bilimler Üniversitesi	
Jüri Üyesi	**Prof. Dr. Harun UĞUZ** Konya Teknik Üniversitesi	
Jüri Üyesi	**Doç. Dr. Halife KODAZ** Konya Teknik Üniversitesi	

Enstitü Müdürü **Doç. Dr. Şule Sultan UĞUR**

TAAHHÜTNAME

Bu tezin akademik ve etik kurallara uygun olarak yazıldığını ve kullanılan tüm literatür bilgilerinin referans gösterilerek tezde yer aldığını beyan ederim.

Mahmut TOKMAK

İÇİNDEKİLER

Sayfa

ÖZET

Doktora Tezi

ANDROID İŞLETİM SİSTEMİ İÇİN DERİN ÖĞRENME TABANLI KÖTÜ AMAÇLI YAZILIM TESPİT ARACI GELİŞTİRME

Mahmut TOKMAK

Süleyman Demirel Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

Danışman: Prof. Dr. Ecir Uğur KÜÇÜKSİLLE

Teknoloji çağı olarak adlandırılan günümüz dünyasında, akıllı telefonlar hayatın bir parçası haline gelmiştir. Akıllı telefonlar iletişim dışında internet kullanımı, sosyal medya kullanımı, bankacılık işlemleri, e-posta işlemleri gibi birçok alanda kullanıcıların vazgeçilmezi olmuştur. Android işletim sistemi de günümüzde akıllı telefon ve tabletlerde kullanılan en popüler işletim sistemidir. Böylesine popüler ve çok kullanılan bir platform zararlı yazılımların hedefi haline gelmiştir. Zararlı yazılımlar kullanıcılara ciddi zararlar verebilmektedirler.

Bu tez çalışmasında akıllı telefonları hedef alan zararlı yazılımlar statik, dinamik, hibrit analiz yöntemleri kullanılarak tespit edilmiştir. 38.044 adet zararlı olmayan yazılım ve 24.503 adet zararlı yazılım ile oluşturulan veri seti statik analize tabi tutulmuştur. Statik analizde 9 ayrı kategoride öznitelik çıkarımı yapılmıştır. Bu öznitelikler; istenen izinler, amaçlar, Android bileşenleri, Android uygulama çağrıları, kullanılan izinler, kullanılmayan izinler, şüpheli Android uygulama çağrıları, sistem komutları, internet adresleri başlıkları altında kategorize edilmiştir. Elde edilen öznitelikler temel bileşen analizi ile boyut indirgemeye tabi tutulmuş ve derin sinir ağı modeline girdi olarak kullanılmıştır. Kurulan modelde veri setinin %80'i eğitim için, %20'si test için ayrılmıştır. Yapılan statik analiz sonucunda; eğitim setinde %99,74 doğruluk oranı, %99,73 F1 skoru, %99,69 kesinlik, %99,77 duyarlılık değerleri elde edilmiştir. Test veri setinde ise; %99,38 doğruluk oranı, %99,36 F1 skoru, %99,32 kesinlik, %99,39 duyarlılık değerleri elde edilmiştir.

Çalışmanın dinamik analiz kısmında uygulamalar sanal bir akıllı telefonda çalıştırılmış, stratejik öneme sahip Android uygulama çağrıları kancalama yapılarak elde edilmiştir. 20.937 adet zararlı olmayan yazılım ve 14.205 adet zararlı yazılımdan oluşan veri seti kullanılmıştır. Dinamik olarak elde edilen öznitelikler ile aynı uygulamalara ait statik öznitelikler birleştirilerek hibrit analiz olarak adlandırılan yöntem uygulanmıştır. Bütün öznitelikler temel bileşen analizi ile boyut indirgemeye tabi tutulmuş ve derin sinir ağı modeline girdi olarak kullanılmıştır. Kurulan modelde veri setinin %80'i eğitim için, %20'si test için ayrılmıştır. Yapılan hibrit analiz sonucunda; eğitim setinde %99,43 doğruluk oranı, %99,41 F1 skoru, %99,42 kesinlik, %99,4 duyarlılık değerleri elde

edilmiştir. Test veri setinde ise; %96,94 doğruluk oranı, %96,78 F1 skoru, %96,99 kesinlik, %96,59 duyarlılık değerleri elde edilmiştir.

Anahtar Kelimeler: Android zararlı yazılım analizi, statik analiz, dinamik analiz, hibrit analiz, derin öğrenme.

2020, 81 sayfa

ABSTRACT

Phd. Thesis

DEVELOPMENT OF DEEP LEARNING BASED MALWARE DETECTION TOOL FOR ANDROID OPERATING SYSTEM

Mahmut TOKMAK

Süleyman Demirel University
Graduate School of Natural and Applied Sciences
Department of Computer Engineering

Supervisor: Prof. Dr. Ecir Uğur KÜÇÜKSİLLE

In today's world, which is called the age of technology, smartphones have become a part of life. Smart phones have become indispensable for users in many areas such as internet use, social media usage, banking transactions, e-mail transactions other than communication. The Android operating system is also the most popular operating system used on smartphones and tablets today. Such a popular and widely used platform has become the target of malicious software. Malware can seriously harm users. Efforts have been made to protect smart phones from these malicious software and they are still underway.

In this thesis, malicious software targeting smart phones were determined by using static, dynamic and hybrid analysis methods. Data set created with 38.044 non-malicious software and 24.503 malware were subjected to static analysis. In static analysis, feature extraction was made in 9 different categories. These attributes are; The desired permissions are categorized under the headings, Android components, Android application calls, used permissions, unused permissions, suspicious Android application calls, system commands, internet addresses. Obtained attributes were subjected to size reduction by principal component analysis and used as input to deep neural network model. In the established model, 80% of the data set is reserved for education and 20% for the test. As a result of the static analysis; 99,74% accuracy rate, 99,73% F1 score, 99,69% precision and 99,77% recall values were obtained in the training set. In the test dataset; 99,38% accuracy rate, 99,36% F1 score, 99,32% precision, 99,39% recall values were obtained.

In the dynamic analysis part of the study, applications were run on a virtual smartphone, and Android application calls with strategic importance were obtained by hooking. A data set consisting of 20.937 non-malicious software and 14.205 malware is used. The method called hybrid analysis was applied by combining the dynamically obtained features with the static features of the same applications. All the attributes were subjected to size reduction with principal component analysis and used as input to the deep neural network model. In the established model, 80% of the data set is reserved for education and 20% for the test. As a result of the hybrid analysis; 99.43% accuracy rate, 99,41% F1 score,

99,42% precision and 99,4% recall values were obtained in the training set. In the test dataset; 96.94% accuracy rate, 96.78% F1 score, 96.99% precision, 96.59% recall values were obtained.

Keywords: Android malware analysis, static analysis, dynamic analysis, hybrid analysis, deep learning.

2020, 81pages

TEŞEKKÜR

Bu araştırma için beni yönlendiren, karşılaştığım zorlukları bilgi ve tecrübesi ile aşmamda yardımcı olan değerli Danışman Hocam Prof. Dr. Ecir Uğur KÜÇÜKSİLLE'ye teşekkürlerimi sunarım. Çalışmalarım esnasında desteklerini esirgemeyen değerli hocalarım Prof.Dr. Tuncay AYDOĞAN ve Dr. Öğretim Üyesi Kubilay TAŞDELEN'e teşekkür ederim.

Araştırmanın yürütülmesinde fikirlerine başvurduğum, destek aldığım Bilgisayar Mühendisi Sinan DURGUT'a teşekkür ederim.

4537-D1-16 No`lu Proje ile tezimi maddi olarak destekleyen Süleyman Demirel Üniversitesi Bilimsel Araştırma Projeleri Yönetim Birimi Başkanlığı'na teşekkür ederim.

Tezimin her aşamasında beni yalnız bırakmayan eşime, çocuklarıma ve aileme, sonsuz sevgi ve saygılarımı sunarım.

Mahmut TOKMAK
ISPARTA, 2020

ŞEKİLLER DİZİNİ

Sayfa

ÇİZELGELER DİZİNİ

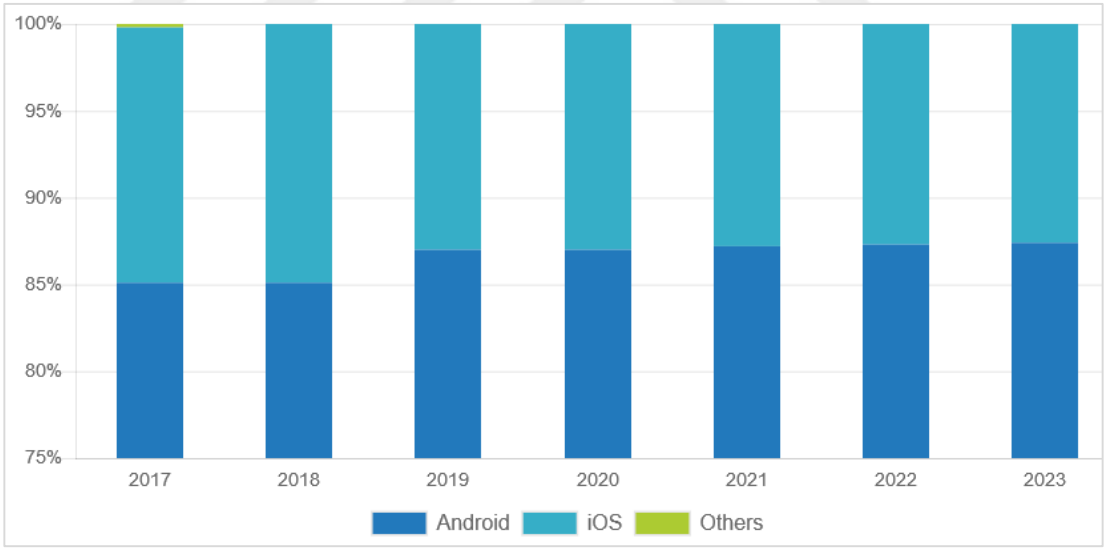
Sayfa

SİMGELER VE KISALTMALAR DİZİNİ

AMD	Android malware dataset
ADB	Android hata ayıklama köprüsü (Android debug bridge)
AOT	Zaman öncesi (Ahead of time)
AVD	Android sanal aygıt (Android virtual device)
API	Uygulama programlama ara yüzü (Application programming interface)
APK	Android uygulama paketi (Android application package)
ART	Android çalışma zamanı (Android runtime)
DL	Derin öğrenme (Deep learning)
DNN	Derin sinir ağı (Deep neural network)
DVM	Dalvik sanal makine (Dalvik virtual machine)
HAL	Donanım soyutlama katmanı (Hardware abstraction layer)
I	Öznitelik fonksiyonu
IDC	International data corporation
IPCA	Artırımlı temel bileşen analizi (Incremental principal componen analysis)
JVM	Java sanal makine (Java virtual machine)
Ö	Özellik seti
PCA	Temel bileşen analizi (Principal component analysis)
SMS	Kısa mesaj servisi (Short message service)
SVM	Destek vektör makineleri (Support vector machine)
φ	Öznitelik vektörü

1. GİRİŞ

Günümüzde akıllı telefonlar birçok kullanıcının hayatında önemli bir yer tutarak onların vazgeçilmezi haline gelmiştir. Çünkü akıllı telefonlar yalnızca telefon olarak değil, aynı zamanda kısa mesaj servisi (Short Message Service: SMS), e-posta, internet, sosyal medya, haritalar, GPS, bankacılık uygulamaları, oyunlar gibi çeşitli hizmetler sunan taşınabilir bilgisayarlar olarak da işlev görmektedir. International Data Corporation (IDC) firmasının 2019 Ekim ayında güncellediği verilerine göre; dünya akıllı telefon pazarında 2019 yılında %2,2'lik bir düşüş görülmüş ve satılan akıllı telefon adedi 1.402 milyardan 1.371 milyara gerilemiştir. Ancak yine firmanın tahminine göre 2023 yılına kadar pazarın büyümesi ve akıllı telefon pazarında satışların 1.484 milyar adede ulaşması beklenmektedir. Şekil 1.1'deki işletim sistemlerinin oranları incelendiğinde Android'in bu pazardaki yeri 2019 yılında %87 dir ve firmanın tahminine göre 2023 yılında %87,4 oranında olması beklenmektedir (IDC, 2019).



Şekil 1.1. Dünya çapında akıllı telefon pazar payı tahmini (IDC, 2019)

Statista'nın, 2009 yılından 2019 yılı Eylül ayına kadar olan Google Play mağazası, uygulama sayılarının paylaşıldığı raporuna göre; 2019 Eylül ayı itibariyle resmi Android yazılım mağazası olan Google Play, bünyesinde yaklaşık 2,8 milyon Android yazılımı bulundurmaktadır (Statista, 2019).

Açık kaynak kodlu ve dünya çapında yaygın olarak tercih edilen bir teknoloji olarak Android, zararlı yazılımların hedefi haline gelmiştir. Bu zararlı yazılımlar kullanıcının onayı olmadan özel ücret numaralarına metin mesajları gönderme, kişisel verilere erişim sağlama ve hatta kullanıcının cihazına ek zararlı yazılım indirebilecek ve uygulayabilecek kod yükleme yeteneğine sahiptir. Zararlı yazılımlar aynı zamanda mobil botnetleri oluşturmak için de kullanılabilmektedir (Anagnostopoulos vd., 2016; Alzaylaee vd., 2020). Son birkaç yılda, Android platformunu hedef alan zararlı amaçlı yazılım örneklerinin sayısı önemli ölçüde artmıştır. McAfee'nin 2018 tarihli bir raporuna göre, 2017'de 2,5 milyondan fazla yeni Android zararlı yazılım uygulaması tespit edilmiş, böylece 2017'de zararlı yazılım örneklerinin sayısı yaklaşık 25 milyona yükselmiştir (McAfee, 2018; Alzaylaee vd., 2020).

Android'in yaygın kullanımı ve zararlı yazılımların sayılarındaki artış, zararlı yazılımların tespiti konusunda çalışmaları da zorunlu kılmaktadır. Bu bağlamda Google Play zararlı yazılımların yayılmasını engellemek için 2012 yılında Bouncer adlı bir tespit mekanizmasını tanıtmışlardır. Bouncer herhangi bir zararlı yazılım davranışını tespit etmek için; uygulama sanal bir alanda beş dakika çalıştırılmış ve teste tabi tutularak zararlı davranışlar tespit edilmeye çalışılmıştır. Ancak bu tespit sisteminin zararlı yazılımlar tarafından atlatılabileceği gösterilmiştir (Alzaylaee vd., 2020). Bunun yanı sıra Google 2017 yılında Google Play Protect'i duyurmuştur. Google Play Protect, verileri ve uygulamaları güvende tutmak için sürekli çalışan ve cihazı otomatik olarak tarayarak mobil güvenliği sağlamaya çalışan bir hizmettir. Aynı zamanda Google bu hizmet ile nereden indirildiklerine bakılmaksızın her gün 50 milyardan fazla uygulamanın tarandığını belirtmiştir (Google Play, 2018). Ancak, McAfee yayınladığı raporda, son 90 gün içinde tespit edilen zararlı yazılımlara karşı test edildiğinde Google Play Protect hizmetinin başarısız olduğunu belirtmişlerdir. (McAfee, 2018). Görüldüğü gibi Google Play'in çalışmalarına ek olarak zararlı yazılımlarla mücadele etmek için çeşitli yaklaşım ve çalışmalara halen ihtiyaç duyulmaktadır.

Android zararlı yazılımlarını tespit etmek amacıyla önceki çalışmalarda çeşitli yaklaşımlar önerilmiştir. Bu yaklaşımlar statik analiz, dinamik analiz veya hibrit analiz olarak adlandırılmaktadır. Statik analiz olarak adlandırılan yöntemde uygulama çalıştırılmadan, yazılmış olan kodun analizi ilkesine dayanmaktadır. Dinamik analiz, uygulamayı emülatör (Android Virtual Device-Android Sanal Aygıt "AVD") gibi kontrollü bir ortamda veya gerçek bir cihazda çalıştırılarak uygulamanın davranışlarını belirleme ilkesine dayanmaktadır. Hibrit analiz yöntemi ise statik ve dinamik analiz yönteminin birlikte kullanılması ilkesine dayanmaktadır.

Android uygulamalarının analizleri sonucunda elde edilen, uygulamalara ait öznitelikler makine öğrenme algoritmalarına girdi olarak kullanılıp test edildikten sonra zararlı uygulamaların tespiti gerçekleştirilmeye çalışılmaktadır. Destek vektör makineleri (Support vector machine: SVM), Bayes sınıflandırıcı, J48, Rastgele Orman, derin öğrenme algoritmaları ile yapılan çalışmalar bunlara örnek verilebilmektedir. Literatüre bakıldığında bu çalışmalara girdi olarak kullanılan özniteliklerin çoğunlukla Android izinleri ve API (Android programlama ara yüzü: Application Programming Interface) çağrıları oldukları gözlemlenmektedir.

Bu tez çalışmasında; Android platformunu tehdit eden zararlı yazılımların tespit edilmesi hedeflenmiştir. Bu amaçla diğer çalışmalara göre çok geniş bir yelpazede uygulama analizlere dahil edilmiştir. Toplam olarak 62.547 Android uygulaması veri setinde kullanılmıştır. Statik analiz aşamasında Kuzgun adını verdiğimiz Java uygulaması geliştirilerek 9 kategoride öznitelik çıkarma işlemi yapılmıştır. Çalışma bu yönü itibariyle Au vd. (2012), Aafer vd. (2013), Yerima vd. (2014), Rosmansyah ve Dabarsyah (2015), Hou vd. (2016), Yuan vd. (2016), Hou vd. (2017), Feizollah vd. (2017), Idrees vd. (2017) gibi çalışmaların aksine sadece sadece izinler ve API çağrılarına odaklanmak yerine, Android uygulamalarından elde edilebilen amaçlar, sistem komutları gibi diğer özellikler de kullanılarak, ayrıntılı bir şekilde analiz yapılmış ve tespit mekanizmasında etkili olabilecek öznitelikleri göz önüne koymuştur. Öznitelik çıkarılması işleminden sonra Python kullanılarak boyut indirgeme yapılmış ve derin sinir ağları kullanılarak

zararlı yazılımlar tespit edilmiştir. Yapılan statik analiz sonucunda; eğitim setinde %99,74 doğruluk oranı, %99,73 F1 skoru, %99,69 kesinlik, %99,77 duyarlılık değerleri elde edilmiştir. Test veri setinde ise; %99,38 doğruluk oranı, %99,36 F1 skoru, %99,32 kesinlik, %99,39 duyarlılık değerleri elde edilmiştir.

Çalışma da dinamik analiz yöntemi gerçekleştirme amacıyla KuzgunDroid adında bir Android uygulaması geliştirilmiş ve PC ile Android emulatörü haberleştiren bir PC uygulaması geliştirilmiştir. Dinamik analiz yöntemi statik analiz yöntemine göre kullanılan araçlar ve zaman yönünden, zahmet ve zorluklar ihtiva etmektedir. Android uygulamasının belirli bir süre emülatörde çalıştırılmalı ve takip edilmelidir. Çalışmada 35.142 Android uygulaması dinamik analize tabi tutulmuştur. Her bir uygulamanın 1 dakika çalıştırıldığı düşünüldüğünde tamamlanma süresi yaklaşık 25 gün sürmektedir. Bu nedenle; incelediğimiz çalışmaların %70'i analizlerinde, veri setlerini 10.000 örneğin altında tuttuğu gözlemlenmiştir. Bu bakımdan çalışma; geniş bir zararlı yazılım ailesini barındırarak, çok çeşitte zararlı yazılım türlerini tespit etmeyi hedeflemiştir. Dinamik analiz ile elde edilen API çağrıları, statik özniteliklerle birleştirilerek hibrit analiz gerçekleştirilmiş, boyut indirgemeye tabi tutulmuş ve derin sinir ağlarıyla tespit mekanizması gerçekleştirilmiştir. Yapılan hibrit analiz sonucunda; eğitim setinde %99,43 doğruluk oranı, %99,41 F1 skoru, %99,42 kesinlik, %99,4 duyarlılık değerleri elde edilmiştir. Test veri setinde ise; %96,94 doğruluk oranı, %96,78 F1 skoru, %96,99 kesinlik, %96,59 duyarlılık değerleri elde edilmiştir.

.

2. KAYNAK ÖZETLERİ

Au vd. (2012), PScout adlı çalışmalarında Android izin sistemi ile ilgili ayrıntılı bir çalışma yapmışlardır. Android izinleri ile API çağrılarının ayrıntılı haritalamasını yapmışlardır. Özellikle Android tarafından dokümantasyonu yapılmamış API çağrıları ile bunların kullanılabilmesi için gerekli izinleri eşleştirmişlerdir. İzin gruplarından uygulamalarda kullanılmayan ve gerekli olmadığını düşündükleri izinleri çıkarmışlar hatta Android'in de bu izinleri kaldırdıklarını bildirmişlerdir. Aynı API setlerinin kullanımına izin veren izinler tespit etmişlerdir. Dolayısıyla Android izinleri ve API çağrıları ile ilgili ayrıntılı bir çalışma yapıp bu çalışmanın sonucunda elde ettikleri haritaları paylaşmışlardır.

Wu vd. (2012), çalışmalarında DroidMat adında bir sistem geliştirmişlerdir. İlk olarak DroidMat ile istenen izinler, amaç mesajları, aktiviteler, servisler, alıcılar, izinlerle ile ilgili API çağrılarını elde etmişlerdir. Sonunda K En Yakın Komşu algoritması ile sınıflandırma yapmışlardır. 238 zararlı uygulama ve 1500 zararsız uygulamadan oluşan bir veri seti oluşturmuşlar ve sınıflandırma sonucunda %97,87'lik bir tespit oranına erişmişlerdir.

Aafer vd. (2013), DroidAPIMiner adlı çalışmalarında tersine mühendislik yaparak uygulamaların kodlarından elde ettikleri API çağrılarını kullanarak statik analiz yapmışlardır. Yaklaşık 20.000 uygulamadan oluşan veri setleri 3.987 zararlı uygulama ve yaklaşık 16.000 zararsız uygulama ihtiva etmektedir. Yaptıkları testler sonucunda K En Yakın Komşu algoritması ile %99 doğruluk oranı elde ettiklerini belirtmişlerdir.

Yerima vd. (2014), Android zararlı yazılımlarını tespit etmek için veri madenciliği temelli üç yöntemi araştırmışlardır. Java tarafından geliştirilen özel bir uygulama kullanarak Android uygulama paketlerinden, otomatik tersine mühendislik teknikleri ile elde edilen verileri kullanarak Bayesian sınıflandırma yapmışlardır. Statik analiz yaptıkları modellerin ilkinde Manifest dosyalarından elde ettikleri standart Android izinlerini kullanmışlardır. İkinci modelde, kod özelliklerinden elde ettikleri potansiyel zararlıları işaret eden api çağrılarını kullanmışlardır.

Üçüncü modelde ise ilk iki modelde elde ettikleri Android izinleri ile API çağrılarını birlikte kullanarak sınıflandırma yapmışlardır. Modellerde, çok çeşitli kategorilerdeki 1.000 iyi huylu uygulama ile birlikte 49 Android zararlı yazılım ailesinden alınan 1.000 örnek kullanmışlardır. İzin tabanlı, kod özelliklerinin kullanıldığı API tabanlı ve karma özellikli modeller için sırasıyla 0,9, 0,92 ve 0,93 doğruluk değerlerine ulaşmışlardır.

Arp vd. (2014), zararlı uygulamaların doğrudan akıllı telefon üzerinden tespit edilmesini sağlayan, statik analiz tabanlı DREBIN adını verdikleri bir araç geliştirmişlerdir. 123.453 zararlı olmayan uygulama ve 5.560 tane zararlı uygulamadan oluşan bir veri seti kullanmışlardır. Bu örneklerden elde ettikleri öznitelikleri 8 grupta toplamışlardır. Bu öznitelikleri uygulamanın manifest dosyasından ve uygulamanın kaynak kodlarından elde etmişlerdir. Uygulamanın manifest dosyasından elde ettikleri öznitelikler; donanım özellikleri, istenen izinler, uygulama özellikleri, amaç filtreleridir. Uygulamanın kaynak kodlarından elde ettikleri öznitelikler ise; sınırlandırılmış API çağrıları, kullanılmış izinleri, şüpheli API çağrıları ve ağ adresleridir. Makine öğrenme yöntemlerinden Destek Vektör Makineleri kullanarak zararlı yazılımları tespit etmeye çalışmışlardır. Çalışmalarında %93,90 tespit oranı elde etmişlerdir.

Rosmansyah ve Dabarsyah (2015), statik analiz yöntemini kullandıkları çalışmada API çağrılarını kullanmışlardır. API çağrılarını, Manifest dosyasında <used-permission> etiketi altında tanımlanan izinin <call> etiketi ile çağırdığı sistem çağrılarını kullanarak elde etmişlerdir. Çalışmada 205 zararsız ve 207 zararlı uygulama kullanmışlardır. Rastgele Orman (Random Forest), J48, Destek Vektör Makineleri algoritmaları ile sınıflandırma yapmışlar ve ortalama %91,9'luk bir çıktı elde etmişlerdir.

Kabakuş vd. (2015), çalışmalarında literatürde yer alan çeşitli Android zararlı yazılım tespit ve/veya koruma stratejilerini; statik analiz yöntemi, dinamik analiz yöntemi, imza tabanlı analiz yöntemleri, kriptolanmış veri iletişimi ile koruma isimleri altında incelemişlerdir. İnceledikleri çalışmalarda kullanılan yöntemlerin manifest dosya analizi, API çağrıları, imza veri tabanı, güvenli veri

alışverişi ve makine öğrenmesi özelliklerini karşılaştırmışlardır. Çalışmalarının sonucunda literatürdeki çalışmaların güvenli veri alışverişi konusu üzerinde durmadıkları, daha çok zararlı yazılımların tespiti üzerine çalıştıkları ve araştırmacıların makine öğrenmesi yöntemlerini kullanarak tespit yöntemlerine başvurdukları belirtilmiştir. Ayrıca önerilen sistemlerin genelde uygulamanın sisteme yüklenmesinden sonra çalıştığını, kurulum öncesi tespit ve koruma mekanizmalarının kullanılmadığını belirtmişlerdir. Zararlı yazılımların doğurabileceği etkileri önlemek amacıyla uygulamaları sisteme yüklemeden önce zararlı yazılım tespitinde bulunarak kullanıcıları uyaracak daha kapsamlı zararlı yazılım tespit ve koruma sistemlerinin gerekli olduğu önerisinde bulunmuşlardır.

Fereidooni vd. (2016), ANASTASIA adını verdikleri çalışmada statik analiz yöntemi kullanmışlar ve makine öğrenme teknikleri kullanarak zararlı yazılım tespit sistemi oluşturmuşlardır. 18.677 adet zararlı uygulama ve 11.187 adet zararlı olmayan uygulamadan oluşan bir veri seti kullanmışlardır. Android uygulamalarından elde ettikleri amaçları, kullanılan izinleri, sistem komutlarını, şüpheli API çağrılarını, zararlı aktiviteleri öznitelik olarak kullanmışlardır. Aşırı Gradyan Güçlendirme (XGBoost), Rastgele Orman, Destek Vektör Makineleri gibi makine öğrenme teknikleri ile teste tabi tuttukları verilerinden en iyi sonucu %97'lik bir oranla XGBoost algoritması ile elde etmişlerdir.

Hou vd. (2016), Deep4MalDroid adlı çalışmalarında Android uygulamalarının, Linux çekirdek sistem çağrılarını çıkarmış ve öznitelik gösterimleri için ağırlıklı graf yöntemini kullanmışlardır. 3.000 uygulamanın kullanıldığı çalışmada Derin öğrenme metodu kullanılmıştır. Çalışmalarında %93,68 doğruluk değerine ulaşmışlardır.

Yuan vd. (2016), DroidDetector adını verdikleri çalışmalarında statik analiz ve dinamik analiz yaparak çıkardıkları öznitelikleri Derin Öğrenme yöntemi ile karakterize ederek zararlı uygulama tespiti yapmışlardır. Statik analizde Android izinleri ve hassas API çağrılarını elde etmişler, dinamik analizde ise kancalama (hooking) yöntemi ile uygulamaların davranışsal özelliklerine yönelik dosya giriş/çıkış, kısa mesaj servisleri, kriptolama ve ağ aksiyon özelliklerini elde

ederek 192 öznitelik başlığı çıkarmışlardır. Derin öğrenme metodu ile yaptıkları testlerde %96,76 doğruluk sonucu elde etmişlerdir.

Hou vd. (2017), çalışmalarında APK (Android Application Package-Android Uygulama Paketi)'leri APKTool kullanarak açmışlar ve dex dosyalarından smali kodlarını çıkarmışlardır. Smali kodlarından elde ettikleri API çağrılarını bloklara ayırarak Derin Sinir Ağları oluşturmuşlardır. Derin İnanç Ağı (Deep Belief Network), Yığınlı Oto Kodlayıcılar (Stacked AutoEncoders) derin öğrenme metotlarını kullanarak karşılaştırma yapmışlardır. Derin İnanç Ağları yöntemini, Yığınlı Oto Kodlayıcılar yönteminden daha başarılı bulmuşlardır. 5.000 uygulama kullandıkları çalışmada Derin İnanç Ağları yöntemiyle %96,66 doğruluk değerine ulaşmışlardır.

Feizollah vd. (2017), çalışmalarında Android amaç (intent)' larının zararlı uygulamaları tanımlamak için ayırt edici bir özellik olarak değerlendirmişlerdir. Amaçların, izinler gibi diğer iyi çalışılmış özelliklerle karşılaştırıldığında, zararlı yazılımların tanımlanmasında anlamsal olarak zengin özellikler barındırdığını ifade etmişlerdir. 1.846 zararsız ve 5.560 zararlı uygulama içeren 7.406 adetlik bir veri kümesi kullanılarak testler yapmışlardır. Bayes Ağları kullanarak yaptıkları testler sonucunda izinler kullanıldığında %83, amaçlar kullanıldığında %91 oranında tespit başarısı elde etmişlerdir. Her iki özelliğin birleştirilmesiyle yaptıkları test sonucunda ,%95,5'lik bir tespit oranına erişmişlerdir.

Idrees vd. (2017), PIndroid adını verdikleri çalışmalarında izin ve amaç tabanlı bir tespit yöntemi önermişlerdir. Android izinleri ve amaçlarının birbiriyle ilişkili olduğunu öne sürmüşler ve izinler ve amaçlardan oluşan öznitelikleri makine öğrenme teknikleri ile teste tabi tutmuşlardır. 1.745 adet uygulamadan oluşan bir uygulama veri seti ile yaptıkları denemelerde en iyi sonucu %99,8'lik bir doğruluk oranıyla Rastgele Orman algoritmasıyla elde etmişlerdir.

Milosevic vd. (2017), çalışmalarında 2 farklı makine öğrenme yöntemi kullanmışlardır. Statik analiz yöntemini kullandıkları çalışmada sınıflandırma ve kümeleme yapmışlardır. 200 zararlı 200 zararsız uygulama içeren M0Droid veri

seti kullanmışlardır. Sınıflandırma yöntemleri ile elde edilen sonuçların, en iyi kümeleme yönteminden %14 daha iyi performans gösterdiğini belirtmişlerdir. Uygulama izinlerini kullandıkları sınıflandırma yöntemlerinde SVM, Naive Bayes, C4.5, Jrip ve AdaBoost algoritmalarını kullanmışlar ve yapılan testler sonucunda %89 sınıflandırma başarısı elde edilmiştir. Tersine mühendislik teknikleri ile kaynak kodlarını elde etmişlerdir. Bu kodlardan da metin madenciliği yöntemleri kullanılarak sistem çağrıları, yayın alıcılar ve servisler tespit edilerek öznitelik çıkarmışlardır. Yaptıkları testlerde %95'in üzerinde doğruluk değeri elde ettiklerini belirtmişlerdir.

McLaughlin vd. (2017), manifest dosyasından ve dex dosyasından çıkardıkları opkod dizilerini kullanarak statik analiz yapmışlardır. 863 zararlı olmayan uygulama, 1260 zararlı uygulamadan oluşan veri setlerini konvolusyonel sinir ağları ile eğitmişlerdir. Çalışmanın sonunda %98 doğruluk oranı, %97 F-skoru değerine ulaşmışlardır.

Karbab vd. (2018), MalDozer isimli çalışmalarında API çağrılarını kullanarak statik analiz gerçekleştirmişlerdir. 37.627 zararlı olmayan uygulama, 20.089 zararlı uygulamadan oluşan bir veri seti kullanarak derin öğrenme yöntemi ile tespit mekanizması oluşturmuşlardır. Yaptıkları test sonucunda %96,29 F-skoru değeri elde etmişlerdir.

Alshahrani vd. (2018), DDefender adlı çalışmalarında kullanıcının cihazından özellikler çıkarmak için statik ve dinamik analiz tekniklerini kullanmışlar ve ardından zararlı uygulamaları tespit etmek için derin öğrenme algoritması uygulamışlardır. İlk önce, sistem çağrılarını, sistem bilgilerini, ağ trafiklerini ve denetlenen bir uygulamadan istenen izinlerini çıkarmak için dinamik analiz yöntemi kullanmışlar, ardından denetlenen uygulamadan uygulamanın bileşenleri gibi önemli özellikleri çıkarmak için statik analiz yöntemini kullanmışlardır. 2104 zararsız uygulama ve 2104 zararlı uygulama olmak üzere toplam 4208 uygulama kullanmışlardır. 1007 özellikten oluşan özellik setini kullanarak oluşturdukları sinir ağı ile gerçekleştirdikleri test sonucunda %95 doğruluk değeri elde etmişlerdir.

Yang vd. (2018), çalışmalarında zararlı davranışları karakterize etmek, zararlı uygulamaları tespit etme oranını artırmak, en alakalı ve etkili özellikleri çıkarmak amacıyla DroidWard adlı bir dinamik analiz yöntemi önermişlerdir. Öznitelikleri 15 ayrı kategoride toplamışlardır 9 tanesini klasik yöntemlerden ve 6 tanesini de yeni olarak belirtmişlerdir. Dinamik özellikleri çıkarırken DroidBox kullanmışlardır. Ancak DroidBox tarafından izlenen verilerin sınırlı olduğunu belirtmişler ve DroidBox'un kaynak kodlarında değişiklik yaparak Monkeyrunner ile hassas API çağrılarını simüle etmişlerdir. 258 zararlı uygulama ve 408 zararsız uygulamadan oluşan bir veri seti ile Destek Vektör Makineleri, Karar Ağacı, Rastgele Orman makine öğrenme tekniklerini kullanmışlardır. Destek Vektör Makineleri yöntemi ile %98,54 doğru sınıflandırma sonucu elde etmişlerdir.

Sugunan vd. (2018), statik ve dinamik analiz kullanarak zararlı ve zararsız uygulamaların davranışları hakkında karşılaştırmalı bir çalışma yapmışlardır. Statik özellikleri APKtool kullanarak çıkarmışlar ve dinamik özellikleri ise Droidbox APIMonitor kullanarak çıkarmışlardır. Çalışmada 150 zararlı uygulama ve 200 zararsız uygulama kullanmışlardır. Statik ve dinamik analizdeki özelliklerin birlikte kullanımı ile makine öğrenme algoritmaları RF, SVM, J48 ve Naive Bayes gibi yöntemlerle yaptıkları testlerin sonuçlarının statik ve dinamik özelliklerden daha başarılı olduğunu belirtmişlerdir.

Peynirci (2018), çalışmasında izinler, API çağrılarını, katar (string) özelliklerini kullanarak statik analiz yapmıştır. Belge sıklığı tabanlı bir model ile sınıflandırma yapmıştır. İki farklı veri seti kullanarak testler yapmıştır. Bunlar MalGenome ve AndroZoo veri setidir. Birçok makine öğrenmesi yöntemi ile WEKA'da testler yapıp sonuçlarını paylaşmıştır.

Cordonsky vd. (2018), statik ve dinamik analiz yöntemlerini birlikte kullanan bir sistem önermişlerdir. Cuckoo kum havuzunda analiz ettikleri zararlı uygulamalardan elde ettikleri API çağrıları, ağ aktiviteleri, katar dizileri gibi özellikleri JSON formatında kaydederek Derin Sinir Ağlarına girdi olarak

kullanmışlardır. 9.922 adet zararlı uygulamanın 7.759 tanesini eğitim için ayırmışlar, 2.163 tanesini de test verisi olarak kullanmışlardır. Zararlı uygulamaların ait olduğu aileyi belirlemeyi amaçladıkları çalışma sonunda %97,7 doğruluk oranı elde etmişlerdir.

Türker (2019), zararlı yazılımların ailelerine göre sınıflandırılması ile ilgili bir çalışma yapmıştır. Android uygulamaların istedikleri izinler ile API çağrılarını öznitelik olarak kullanmışlar ve makine öğrenme teknikleri ile sınıflandırma yapmıştır. APKTool kullanarak öznitelik çıkarımında bulunmuştur. Python Scikit-Learn kütüphanesi kullanarak makine öğrenme algoritmaları ile sınıflandırma yapmıştır. Yaptığı testler sonucunda aldığı doğruluk değerleriyle makro ortalamalı duyarlılık, hassasiyet ve F1 skoru değerlerini ayrı ayrı değerlendirmiş ve yorumlamıştır.

Alzaylaee vd. (2020), dinamik analiz yöntemi ile zararlı Android uygulamalarını tespit eden DL-Droid adını verdikleri bir çalışma yapmışlardır. Çalışmada Derin Öğrenme metodunu kullanmışlardır. Alzaylaee vd. (2016) tarafından geliştirilen DynaLog isimli uygulamayı kullanarak dinamik öznitelikleri çıkarmışlardır. Gerçek cihazlarda 11.505 zararlı uygulama, 19.620 zararsız uygulama ile testler yapmışlardır. DL-Droid'in geleneksel makine öğrenme tekniklerini geride bırakan sırasıyla %97,8 (yalnızca dinamik özelliklere sahip) ve %99,6 tespit oranına (dinamik + statik özellikli) tespit oranına eriştiğini belirtmişlerdir.

3. MATERYAL VE YÖNTEM

3.1. Android Platformu

Android, mobil cihazlar için geliştirilen Linux tabanlı bir işletim sistemidir. Açık kaynak kodlu bir işletim sistemi olan Android, Google ve Open Handset Alliance tarafından geliştirilmektedir (Alzahrani, 2019; Türker, 2019). Android, Linux işletim sistemi ve anahtar uygulama içerir. Bu yazılım sistemi akıllı telefonlarda ve tabletlerde çalışacak şekilde geliştirilmiş, daha sonra saat ve TV gibi cihazlara destek vermiştir. Android platformu, farklı katmanlardan oluşmaktadır. Her katmanın amaçları ve sorumlulukları vardır. Yığının en üstünde, stok uygulamaları ve kullanıcının Google Play Store'dan yüklediği uygulamalar gibi kullanıcı alanı uygulamaları bulunur. Genel Android sistem mimarisi, Şekil 3.1'de gösterilmiştir. Android mimarisi beş ana katmana sahiptir. Bu katmanlar; uygulama katmanı, JAVA API çatı katmanı, çalışma zamanı ve yerel kütüphane katmanı, donanım soyutlama katmanı ve çekirdek katmanıdır (Kulkarni, 2018).

Şekil 3.1. Android mimarisi (Google, 2019b)

Android mimarisinin en üst katmanı uygulama katmanıdır. Android kurulduğunda, e-posta, SMS mesajları, takvimler, internet tarayıcısı, kişiler ve daha birçok temel uygulama ile birlikte gelir. Platformla birlikte gelen uygulamaların yanı sıra kullanıcının yüklediği diğer uygulamalar da bu katmanda yer almaktadır (Google, 2019c).

Java API katmanında, uygulamaların yürütüldüğü ve denetlendiği ortamı birlikte şekillendiren bir grup hizmet sunulmaktadır. Android işletim sisteminin tüm özellik seti Java dilinde yazılmış API'lar aracılığıyla kullanılabilmektedir. Bu API'lar, modüler sistem bileşenlerinin ve hizmetlerinin yeniden kullanımını basitleştirerek Android uygulamaları oluşturmak için ihtiyaç duyulan yapı taşlarını oluştururlar (Google Play, 2018; Kulkarni, 2018).

Java API çatısı aşağıdaki bileşenlerden oluşmaktadır;

- Aktivite yöneticisinin, uygulamaların yaşam döngüsünü yönetme görevi vardır. Bunun yanı sıra, yığından geri dönerek aktiviteler arası geçişlere olanak sağlamaktadır (Google, 2019c).

- Kaynak yöneticisi, yerelleştirilmiş dizeler, grafikler ve düzen dosyaları gibi kod dışı kaynaklara erişimi sağlar (Google, 2019c).

- Bildirim yöneticisi, tüm uygulamaların, durum çubuğunda özel uyarılar görüntülemesini sağlar (Google, 2019c).

- Konum yöneticisi, konum bilgileri ve konum güncellemeleri için destek sağlar (Dubey ve Misra, 2016).

- Paket yöneticisi, sisteme uygulama yüklemek, yüklü uygulamalar ve bileşenleri hakkında bilgi sağlamaktan sorumludur (Dubey ve Misra, 2016).

- Telefon yöneticisi, abone kimliği, sim seri numarası, telefon şebekesi türü gibi telefon hizmetleri hakkında bilgi sağlar (Google, 2019c).

- Pencere yöneticisi, pencerelerin ekrandaki düzenlerini yönetmekten sorumludur. Bir uygulamayı açarken, kapatırken veya ekranı döndürürken otomatik olarak pencere geçişlerini ve animasyonlarını gerçekleştirir.

- İçeri sağlayıcı, uygulamaların diğer uygulamalardaki verilere erişmesini veya kendi verilerini onlarla paylaşmasını sağlar (Dubey ve Misra, 2016)

- Görüntü sistemi, uygulamaların kullanıcı ara yüzleri, metin kutuları, listeler, butonlar gibi hizmetleri sağlar.(Google, 2019c).

Bir diğer katmanda yerel C/C++ kütüphaneleri ve Android çalışma zamanı öğeleri bulunmaktadır. Çalışma zamanı (ART: Android RunTime) ve donanım soyutlama katmanı (HAL) gibi birçok temel Android sistem bileşeni ve hizmeti, C ve C ++ ile yazılmış yerel kütüphaneler gerektiren yerel kodlardan oluşturulmuştur. Bunun yanı sıra internet tarayıcısının desteği için Webkit, uygulamalarda 2D 3D grafik çizimlerine destek işlemleri için OpenGL, veri tabanı destekleri için SQLite, ses ve video destekleri için Media Çatısı gibi yapılar bu katman içerisinde yer almaktadır (Kiraz, 2017; Kulkarni, 2018; Google, 2019c; Türker, 2019).

Bu katmanın diğer bir bileşeni olan Android çalışma zamanı bileşeni de ART ve çekirdek kütüphanelerini bulundurur. Bu katman Android sürümlerine göre farklı özellikler ihtiva etmektedir. Android 5.0 (Lollipop-API 21)'den önce Dalvik Sanal Makine (DVM) ve çekirdek kütüphanelerden oluşmaktadır. Uygulamalar genellikle sanal bir makinede çalışmaktadırlar. Diğer platformların aksine, Android geleneksel Java Sanal Makinesi'ni (JVM) kullanmamaktadır. Bunun yerine, ".dex" dosyalarının (Dalvik Yürütülebilir) yürütüldüğü DVM kullanmaktadır. ".dex" dosyası; java.class dosyalarının derlenmesinden sonra, dx aracı kullanılarak oluşturulmaktadır. Takip eden proses aşamasında ".apk"

dosyası elde edilmektedir. Android'in bu tekniği her uygulamayı aynı VM'de çalıştırması ek bellek ve CPU kapasitesine ihtiyaç duymaktadır. Google'ın mobil ortamların gereksinimlerine göre değiştirilen DVM'yi tercih etmesinin nedeni budur. DVM daha sadedir ve geleneksel JVM'ye kıyasla daha az bellek kullanım alanına sahiptir.

Google Android 4.4 (KitKat) sürümünde ART'yi tanıttı. Android 5.0 ile DVM'nin yerine ART'ye geçtiğini duyurdu ve başlangıçta kullanıcıya DVM ve ART arasında seçim yapma seçeneği vermişlerdir. DVM ve ART arasındaki fark uygulamanın Java bayt kodunun ".dex" dosyasında işlenmesi sırasında ortaya çıkmaktadır. DVM, uygulama her başlatıldığında Java bayt kodunu yerel makine koduna yeniden derlemektedir. Bu işleme Just-In-Time (JIT) derlemesi denilmektedir. ART'de ise bayt kodu dex2oat aracını kullanarak kurulum sırasında kalıcı olarak yerel makine koduna derlenir. Bu nedenle, bu noktadan sonra zaten derlenmiş yerel kod yürütülebilir ve bu işleme Zaman Öncesi (AOT) derleme denir. AOT, uygulama yüklemesi sırasında daha fazla zaman gerektirse de, bir kez kurulduktan sonra uygulamalar daha hızlı olmaktadır. Çünkü uygulama her başlatıldığında daha fazla derleme gerekmemektedir. ART kullanmanın dezavantajı olarak da uygulamanın kapladığı alanın artması söylenebilmektedir. Google'ın bildirdiğine göre diğer üzerinde durulması gereken ise yazılan bir uygulama ART üzerinde iyi çalışıyorsa, Dalvik üzerinde de çalışması gerekir, ancak bunun tersi doğru olmayabilmektedir (Google, 2019b).

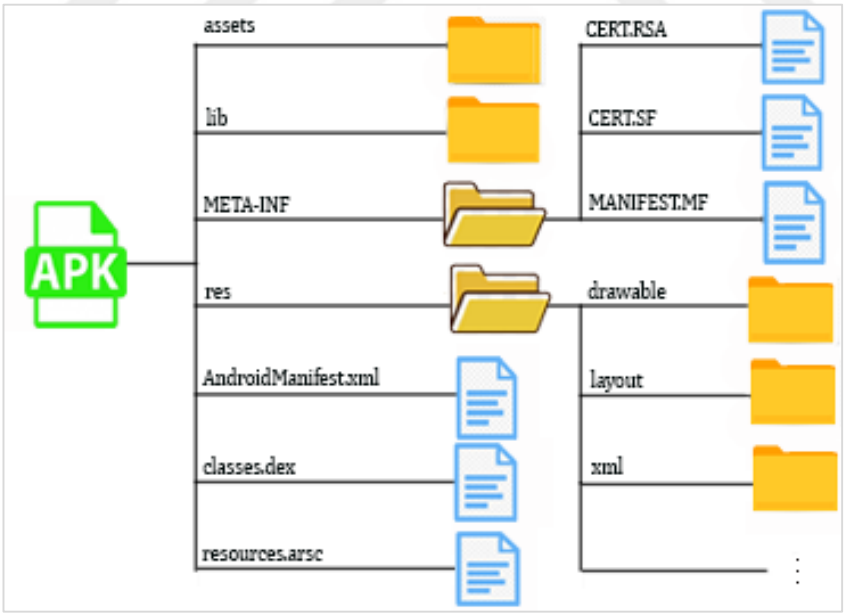
Android platformunun bir diğer katmanı Donanım Soyutlama katmanı (HAL), aygıt donanım özelliklerini, daha üst düzey Java API çatısının kullanabilmesi için arabirimler sağlamaktadır. HAL, ses, kamera ve bluetooth modülü gibi belirli bir donanım bileşeni için bir arabirim uygulayan birden çok kütüphane modülünden oluşur. Bir çerçeve API'si aygıt donanımına erişmek için bir çağrı yaptığında, Android sistemi söz konusu donanım bileşeni için kütüphane modülünü yükleyerek hizmet sunmaktadır.

Android platformunun en alt katmanı Linux çekirdek katmanıdır. Linux çekirdeği Android platformunun temelidir. Linux, Android'in, ürün üreticilerinin donanım

sürücüleri geliştirmesine olanak tanıyan önemli güvenlik özelliklerini kullanmasına izin verir. Android, işlem yönetimi, bellek yönetimi, ağ, güvenlik ayarları, dosya yönetimi gibi tüm temel sistem görevlerini yerine getirmek için Linux çekirdeğini kullanır. Bu katman, donanımla ilişkilidir ve tüm gerekli donanım sürücülerini kapsar.

3.2. Android Uygulamalarının Yapısı

Android platformunda geliştirilen uygulamalar çoğunlukla Java programlama dili kullanılarak geliştirilmektedir. Geliştirilen uygulamalar APK formatında kaydedilmektedir. APK olarak ifade edilen uygulamalar ".apk" dosya uzantısına sahiptirler. Bu ".apk" dosyası uygulama olarak kabul edilmekte ve Android sistemine yüklemek için kullanılmaktadır. Dosyalar, Dalvik yürütülebilir dosyaları, manifest dosyası ve ek olarak yürütülebilir dosyaya sağlanan bir dizi kaynaktan oluşan sıkıştırılmış zip arşivlerinden oluşmaktadır (Xu vd., 2013; Kulkarni, 2018). Şekil 3.2'de APK dosyasının yapısı gösterilmiştir.



Şekil 3.2. APK dosya formatı

APK'lar assets, classes.dex, res, lib, META-INF, Resources.arsc ve Androidmanifest.xml dosya ve klasörlerini içermektedir. Assets klasörü

derlenmemiş raw formatlı dosyalarınızı barındırmak için kullanılmaktadır. Lib klasörü kütüphane dosyalarını barındırmak için kullanılmaktadır. META-INF klasörü meta-data ve sertifika bilgilerini tutmak için kullanılmaktadır. Res klasörü uygulama içinde kullanılacak olan resimleri, uygulamanın ara yüzlerini temsil eden xml dosyalarını, uygulamada kullanılan menüleri temsil eden xml dosyalarını, metin, stil, renk gibi ayarların tanımlandığı xml dosyalarını bünyesinde barındırmaktadır. Resources.arsc dosyası önceden derlenmiş kaynak bilgilerini barındırır (Xu vd., 2013; Dubey ve Misra, 2016; Kiraz, 2017; Kulkarni, 2018; Türker, 2019).

AndroidManifest.xml dosyası uygulama için önemli bilgiler ihtiva etmektedir. Bu dosya içerisinde uygulama ile ilgili paket adı, versiyonu, çalışabileceği minimum sdk, maximum sdk ve hedef sdk numaraları, uygulamanın gereksinim duyduğu izinler, servisler, aktiviteler, yayın alıcılar, içerik sağlayıcılar gibi tanımlar yer almaktadır (Kiraz, 2017). Şekil 3.3'de örnek bir manifest dosyası verilmiştir.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="mobi.dinamik.dinamikanaliz">
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
      <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="DinamikAnaliz"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <meta-data
            android:name="xposedmodule"
            android:value="true" />
        <meta-data
            android:name="xposedminversion"
            android:value="30+" />
        <meta-data
            android:name="xposeddescription"
            android:value="Dinamik Analiz Aracı" />
        <activity android:name=".Arayuz.SplashActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name=".Arayuz.MainActivity"></activity>
    </application>
</manifest>
```

Şekil 3.3. Android manifest dosyası

Şekil 3.3'e bakıldığında uygulama paket adının tanımlandığı, uses-permission etiketi altında uygulamanın ihtiyaç duyduğu izinlerin tanımlandığı, application etiketi altında tema özellikleri, ara yüzleri içeren aktivitelerin tanımlandığı görülmektedir.

Classes.dex dosyası Android class dosyalarının derlenmesi sonucu oluşan Dalvik çalıştırılabilir kodlarını içermektedir.

3.3. Android İzinleri

Android izin mekanizmasının amacı, kullanıcısının gizliliğini korumaktır. Android uygulamaları, hassas sistem verilerine (kişiler ve SMS gibi) ve belirli sistem özelliklerine (kamera ve internet gibi) erişmek için izin istemek durumundadırlar. Gereksinim duyulan izinin özelliğine bağlı olarak, sistem otomatik olarak izin verebilir veya kullanıcıdan isteği onaylamasını isteyebilmektedir (Google, 2019a).

Android platformu izinleri, çeşitli koruma düzeylerine göre kategorize etmiştir. Buna göre izinler normal izinler, tehlikeli izinler, imza izinleri ve özel izinler olmak üzere 4 başlık altında kategorize edilmişlerdir (Davarcı, 2018)

Normal izinler, kullanıcının gizliliği veya diğer uygulamaların çalışmasını etkileyecek çok az risk bulunan izinleri kapsamaktadır. Örneğin, saat dilimini ayarlama izni normal izin kategorisinde bulunmaktadır (Idrees vd., 2017).

Tehlikeli izinler, kullanıcının özel bilgilerini içeren verilerini, kullanıcının depolanan verilerini veya diğer uygulamaların çalışmasını etkileyebilecek türdeki izinleri kapsamaktadır. Örneğin, kullanıcının kişilerini okuma izni tehlikeli bir izin kategorisine girmektedir. Bir uygulama tehlikeli bir izin kullanmış ise, kullanıcının istenen bu izine onay vermesi gerekmektedir. Kullanıcı izni onaylayana kadar, uygulamanız bu izne bağlı işlevleri kullanamamaktadır (Google, 2019a).

İmza izinler, uygulama yüklenmesi sırasında otomatik olarak verilir. Ancak yalnızca izni kullanmaya çalışan uygulama, izni tanımlayan uygulamayla aynı sertifika tarafından imzalandığında bu izni kullanabilmektedir (Idrees vd., 2017).

Özel izinler: Normal ve tehlikeli izinler gibi davranmayan birkaç izinden oluşmaktadır. SYSTEM_ALERT_WINDOW ve WRITE_SETTINGS özellikle hassastır, bu nedenle çoğu uygulama bunları kullanmamalıdır. Bir uygulama bu izinlerden birine ihtiyaç duyuyorsa, manifest dosyasında izni bildirmeli ve kullanıcının yetkisini isteyen bir niyet (Intent) göndermelidir. Sistem, kullanıcıya ayrıntılı bir yönetim ekranı göstererek niyetine yanıt vermektedir (Google, 2019a).

Android, izin isteme sisteminde Android 6.0 (Marshmallow) sürümü ile değişikliğe gitmiştir. Android 6.0 sürümüne kadar olan sürümlerde tehlikeli izinler kurulum esnasında kullanıcıdan talep ediliyor, kullanıcı onay verirse uygulama sisteme yüklenmekte onay vermezse kurulum gerçekleşmemekteydi. Kullanıcı uygulamayı kurabilmek için verdiği izinlerin verebileceği zararları göz ardı edebilmekteydi. Android 6.0 sürümü ve sonrasında ise tehlikeli olarak nitelendirilen izinler kurulum esnasında değil çalışma anında istenmeye başladı. Burada hedeflenen kullanıcı uygulamayı kurduktan sonra, bazı izinleri engelleyebilme ve kaldırabilme imkanını kullanıcıya sunulmasıdır. Bu noktada kullanıcıların da verdiği izinler hakkında bilgi sahibi olması gerekliliği doğmaktadır. Felt vd. yaptığı çalışmada kullanıcıların Android izinleri hakkında yeterli bilince sahip olmadıklarını vurgulamış kullanıcılardan sadece %17'sinin onayladığı izinlere dikkat ettiği, %42'sinin ise verdiği izinlere dikkat etmediği ortaya konmuştur (Felt vd., 2012; Aydın, 2019).

3.4. Android Uygulamalarının Bileşenleri

Android uygulama bileşenleri, Android uygulamalarının temelini oluşturan yapı taşlarıdır. Her bileşen, sistemin veya kullanıcının uygulamaya girebileceği bir giriş noktasıdır. Bazı bileşenler diğerlerine bağlıdır. Android uygulamalarında uygulama bileşenlerinin dört çeşidi bulunmaktadır. Bunlar aktiviteler, servisler,

yayın alıcıları, içerik sağlayıcıları olarak adlandırılmaktadır. Her bir bileşenin farklı bir görevi ve amacı vardır. Bu bileşenin nasıl oluşturulacağı ve nasıl yok edileceğini tanımlamaya yönelik bir yaşam döngüsü mevcuttur (Google, 2019c).

Aktiviteler (Activities): kullanıcıyla etkileşim için giriş noktası olarak tanımlanmaktadır. Aktiviteler kullanıcı ara yüzüne sahip bir ekranı temsil eder. E-posta uygulaması içeren bir yazılımda e-postaların listelendiği ekran bir aktivite, yeni e-posta hazırlamak için kullanılan ekran başka bir aktivite olabilmektedir. Bir uygulama birden fazla aktivite içerebilmektedir. Aktiviteler manifest dosyasında tanımlanmalıdır. Eğer uygulama birden fazla aktivite içermesi halinde, hangi aktivite uygulama ilk başlatıldığında açılacaksa bu aktivite ana aktivite (main activity) olarak manifest dosyasında belirtilmelidir (Alzahrani, 2019)

Servisler (Services): Bir uygulamanın arka planda çalışmasını sağlamak için kullanılan ve ara yüzü olmayan bileşenlerdir. Uzun sürebilecek işlemleri veya uzak bağlantı işlemlerinin arka planda yürütülmesi için geliştirilmişlerdir. Örnek olarak, geliştirilen bir müzik çalar uygulaması düşünüldüğünde seçilen bir müziğin arka planda dahi çalmaya devam etmesini sağlayan veya arka planda bir dosya indirme işleminin yerine getirilmesini sağlayan bileşen servis bileşenidir (Enck vd., 2009; Google, 2019c).

Yayın Alıcılar (Broadcast Receiver):, sistemin uygulamaya düzenli bir kullanıcı akışı dışında olayları iletmesini sağlayan ve uygulamanın sistem genelindeki yayın duyurularına yanıt vermesine olanak tanıyan bir bileşendir. Birçok yayın sistemden gelebilmektedir. Ekranın kapandığını, pilin zayıf olduğunu bildiren bir yayın bunlara örnek olarak verilebilmektedir. Ayrıca uygulamalar yayın başlatabilmektedir (Enck vd., 2009).

İçerik Sağlayıcılar (Content Providers): dosya sisteminde, SQLite veritabanında, web'de veya uygulamanızın erişebileceği diğer depolama konumlarında depolanan uygulama veri kümesini yöneten bileşendir. Uygulamalar arasında veri gönderimi ve paylaşımı için kullanılmaktadır. İçerik sağlayıcılar bileşeni

sayesinde gerekli yetkiye sahip olmak koşulu ile başka uygulamaların verileri sorgulanabilmekte ve bu verilere erişim sağlanabilmektedir (Liu ve Yu, 2011).

3.5. Android Zararlı Yazılımları

Android zararlı yazılımları (Malware), Android platformunu hedefleyen kötü amaçlı uygulamalar anlamına gelmektedir. Bu kötü amaçlı uygulamalar, kullanıcının farkında olmadan kullanıcının özel verilerini çalabilmekte, kullanıcılara finansal kayıp verdirebilmekte veya sistem ayrıcalıklarından yararlanarak yüksek ayrıcalık kazanabilmekte ve daha fazla kötü amaçlı işlem gerçekleştirebilmektedirler. Android zararlı yazılımlarının çeşitli türleri bulunmaktadır. Bunlardan bazıları: virüs (virus), solucan (worm), casus yazılımlar (spyware), reklam yazılımları (adware), Truva atı (Trojan), botnet, kök kullanıcı takımı (rootkit), arka kapılar (backdoor) olarak adlandırılmaktadırlar.

Virüsler ve solucanlar: virüsler bulaştığı cihazda yayılarak cihazdaki diğer dosyalara yayılan yazılımlar olarak, solucanlar ise kendi kendilerine çoğalarak farklı cihazlara ağ üzerinden yayılmayı amaçlayan yazılımlar olarak tanımlanmaktadırlar (Kaspersky, 2019).

Reklam yazılımı: istenmeyen reklamları indiren veya görüntüleyen yazılım olarak tanımlanır. Reklam yazılımları, arama isteklerini reklam web sitelerine yönlendirmek ve kişinin ziyaret ettiği web sayfalarının türleri hakkında veriler toplayarak özelleştirilmiş reklamları görüntüleme amacıyla oluşturulmuş yazılımlardır (Kaspersky, 2019).

Kök kullanıcı takımı ve arka kapı yazılımları, bulaştığı cihazlarda süper kullanıcı (root) yetkisi almaya çalışarak, cihaz üzerinde yönetici yetkisine sahip olmaktadırlar. Bu sayede anti virüs yazılımlarından saklanmaları kolaylaşmaktadır. Bu yazılımlar kullanıcının haberi olmadan uygulama durdurabilir, uygulama silebilir ve yeni yazılımlar indirebilirler. Aynı zamanda bu yazılımlar sayesinde saldırgan tarafından ele geçirilen cihaz uzaktan kontrol edilebilmektedir (Arshad vd., 2016; Davarcı, 2018).

Casus yazılımlar, bankacılık faaliyetleri, konumlar, kişiler ve kısa mesajlar gibi kişisel bilgileri gizlice toplayarak elde ettikleri bilgileri üçüncü taraflara gönderen yazılımlar olarak tanımlanmaktadırlar (Kaspersky, 2019).

Botnet, robot ve ağ manasına gelen İngilizce robot ve network kelimelerinin birlikte kullanılarak kısaltılmış hali olarak tanımlanmaktadır. Bot ise belirli işlemleri otomatik olarak yapmak için tasarlanarak geliştirilmiş programlardır. Birçok kullanışlı amaca hizmet etmek için kullanılmasının yanında zararlı yazılım üretenler tarafından kötü amaçlara hizmet edebilecek şekilde tasarlanabilmektedirler. Böyle bir zararlı yazılıma maruz kalmış cihaz artık bot olarak adlanmaktadırlar. Saldırganlar tarafından ele geçirilen cihazlar uzak bir sunucu tarafından kontrol edilebilmekte ve sunucudan gelen komutlarla saldırganın istediği eylem gerçekleştirilebilmektedir. Hassas verilerin çalınması, kullanıcının etkinliklerini izlemek, spam yaymak, DDos saldırıları başlatmak bu eylemlere örnek olarak verilebilmektedir (Aydın, 2019; Kaspersky, 2019).

Truva atı, zararsız bir yazılım görünümündeki kötü amaçlı yazılımdır. Genellikle gizli verileri elde etmek için geliştirilmişlerdir. Bankacılık ile ilgili işlemlerde kullanıcı hesabına ilişkin bilgileri elde etmek, kullanıcıdan gizli SMS göndermek gibi eylemleri gerçekleştirebilmektedirler (Aydın, 2019).

3.6. Android Yazılım Analizi

Android işletim sisteminin dünya genelinde kullanım oranının artması ve Android platformunu hedef alan saldırıların da artması geliştirilen Android yazılımlarının analiz edilerek, kullanıcı güvenliğinin ve işletim sisteminin tutarlılığının korunması açısından bir gereklilik haline gelmiştir. Literatürde çeşitli analiz yöntemleri mevcut olsa da son yıllarda analiz yöntemleri 3 kategoride incelenmektedir. Bu yöntemler; statik analiz, dinamik analiz, hibrit analiz yöntemi başlıkları altında kategorize edilmiştir (Bhilvare ve Manik, 2015; Idrees vd., 2017; Alshahrani vd., 2018).

3.6.1. Statik analiz yöntemi

Statik analiz yöntemi çeşitli tersine mühendislik teknikleri kullanılarak uygulamaların APK dosyasından alınan kaynak kodun analizini içerir. Statik analiz yönteminde uygulamalar emülatör ya da herhangi bir cihazda çalıştırılmadan analize tabi tutulmaktadırlar. APK dosyaları sıkıştırılmış dosya formatındadır. Winzip, Winrar gibi sıkıştırma programlarıyla açılabilir ve dex, manifest ve kaynak dosyalarına erişim sağlanabilmektedir. Bu dosyalara erişim sağlandıktan sonra okunabilir hale gelebilmeleri için çeşitli araçlar geliştirilmiştir. Java ile kodlanmış ve derlenmiş, DVM'de çalışabilecek hale gelmiş Classes.dex dosyasını, class dosya formatına çeviren dex2jar gibi araçlar kullanılarak kaynak kodları analiz edilebilmektedir. Buna benzer olarak apktool aracı kullanılarak kaynak koda çevirme işlemi yapılabilmektedir. Manifest dosyasının analizi için dosyanın okunabilir hale getirilmesinde ise AXML2jar, AXMLPrinter2.jar gibi uygulamalar kullanılmaktadır. Statik analiz yöntemi tersine mühendislik tekniklerine dayandığı yani kaynak kodların incelenmesi ilkesini benimsediği için herhangi bir gerçek cihaz ya da sanal cihazda çalıştırılmasına gerek duyulmamaktadır. Bu sayede zararlı bir uygulama incelemesi esnasında uygulamanın cihaza zarar verme durumu olmamaktadır. Bu statik analiz yönteminin bir avantajı olarak görülmektedir. Ancak uygulamada kodların anlaşılabilirliğini zorlaştırmak için kod gizleme (obfuscation) gibi yöntemler kullanılmış ise statik analiz yöntemi ile başarı elde edilemeyebilmektedir ki bu da statik analiz yönteminin dezavantajı olarak görülmektedir (Yerima vd., 2014; Bhilvare ve Manik, 2015; Türker, 2019).

3.6.2. Dinamik analiz yöntemi

Dinamik analiz yöntemi uygulamaların gerçek bir cihaz ya da bir sanal cihaz kullanılarak çalıştırılması ve uygulamanın çalışma zamanındaki davranışlarının takip edilmesi esasına dayanmaktadır. Uygulama çalıştırıldıktan sonra uygulamanın sistem üzerindeki etkileri, ağ üzerindeki davranışları, API çağrıları, batarya kullanımı gibi hareketleri takip edilerek geliştirilen yazılımın analizi mümkün olabilmektedir. Statik analiz yönteminde analizi zorlaştırmak için kullanılan kod gizleme yöntemi dinamik analiz yöntemi ile etkisini yitirmiş

olacak, analiz esnasında herhangi bir etkisi kalmayacaktır. Bu açıdan dinamik analiz yöntemi statik analiz yöntemine göre avantaj sağlamaktadır. Ancak dinamik analiz yönteminde uygulama çalıştırılarak analiz edildiğinden eğer uygulama zararlı bir uygulama ise gerçek cihazlarda çalıştırıldığında cihaz bundan etkilenecek, zarar görecektir. Bu nedenle uygulamalar sandbox (kum havuzu-sanal cihaz-emülator) adı verilen sistemlerde çalıştırılması daha uygun görülmemektedir. Sanal, izole edilmiş sistemlerin oluşturulması ise maliyet ve zahmet gerektiren bir yöntem olarak karşımıza çıkmaktadır. Aynı zamanda dinamik analiz yöntemi statik analize göre süre açısından uzun sürmektedir. Analiz edilecek sanal platformlarda performans isteniyorsa bu da belli bir maliyet gerektirmektedir. Şöyle ki Genymotion sanal cihaz oluşturup kullanmak için geliştirilen bir yazılımdır ve bu yazılım hız ve esneklik açısından faydalı bir yazılım olarak görülmektedir. Bu yazılım ücret mukabili edinilebilen bir yazılım olarak karşımıza çıkmaktadır (Bhilvare ve Manik, 2015; Kulkarni, 2018; Türker, 2019).

3.6.2.1. Android hata ayıklama köprüsü

Android hata ayıklama köprüsü (ADB: Android Debug Bridge), bilgisayar ile bu bilgisayara bağlı bir Android cihaz veya emülatör ile iletişimi sağlamak için geliştirilmiş bir komut satırı aracıdır. ADB aracı kullanılarak, Android aygıtlarında uygulama yükleme, uygulama kaldırma, cihaz bilgileri, log bilgileri, cihaza dosya gönderme, cihazdan dosya alma gibi birçok işlem komutlar vasıtasıyla gerçekleştirilebilmektedir. ADB aracının telefona erişebilmesi için Android cihaz ya da emülatörden hata ayıklama köprüsünün aktive edilmesi gerekmektedir.

3.6.2.2. Xposed uygulama çatısı ve kancalama

Xposed Android platformunda kullanılan rovo89 tarafından geliştirilen açık kaynak kodlu bir uygulama çatısıdır. Xposed yüklenmiş uygulamalarda doğrudan ya da dolaylı olarak değişiklik gerçekleştirebilen bir yazılımdır. Xposed uygulama çatısı, Android sisteminin çekirdek prosesi kanca Zygote (Hook Zygote) ile çalışan

süreci ve programın sonucunu değiştirme ilkesi ile çalışmaktadır. Kanca (Hook), kavramı mesajların işlenmesi için sistem çağrıları yoluyla sisteme bağlanan bir program segmenti olarak tanımlanabilmektedir. Herhangi bir mesaj gönderildiğinde, kanca programı hedef pencereye ulaşmadan önce mesajı yakalamakta ve kanca işlevi kontrolü ele geçirmektedir. Böyle bir durumda, kanca işlevi mesajı değiştirebilmekte, mesajı işlemeden iletmeye devam edebilmekte veya mesaj iletimini sonlandırmaya zorlayabilmektedir.

Android başlangıç mekanizması önce Linux çekirdeğin önyüklemesi ile başlamakta ve daha sonra Init prosesin çalıştırılması ile devam etmektedir. Init prosesi Zygote prosesi başlatmaktadır. Zygote ise tüm uygulamaların ve servislerin çalıştırılmasından sorumludur. Oluşturduğu bir soketten gelen çalıştırma ile ilgili istekleri dinleyen Zygote, uygulamalar tarafından kullanılacak olan sınıfları yüklemektedir (Türk ve Arslan, 2017).

Bu bağlamda Xposed uygulama çatısı, uygulama başlatma işlemleri Zygote prosesi tarafından başlatıldığından dolayı sistemdeki tüm uygulama işlemleri için Zygote prosesine ulaşarak kancalama yapma yoluna gitmektedir.

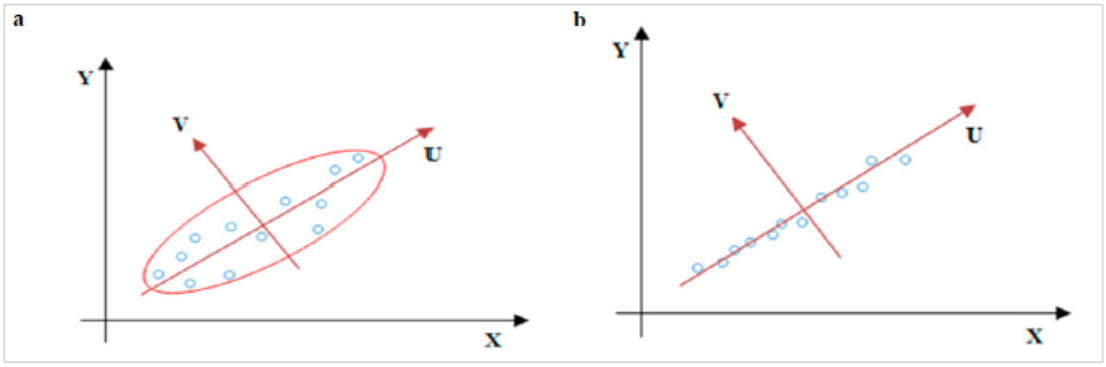
3.6.3. Hibrit analiz yöntemi

Hibrit analiz yöntemi, statik analiz yöntemi ve dinamik analiz yönteminin beraber kullanıldığı bir yöntemdir. Hibrit analiz yöntemi genel olarak iki aşamadan oluşmaktadır. Bu aşamalar, statik özelliklerin çıkarıldığı ilk aşama ve dinamik olarak çıkarılan özelliklerin elde edildiği ikinci aşama şeklindedir (Tong ve Yan, 2017). Literatürdeki çalışmalar ışığında örneklendirilecek olursa; izin ve API tabanlı bir çalışma, hibrit analiz yöntemi ile analiz yapacaksa; birinci aşamada manifest dosyasından izinleri statik olarak çıkarmakta ve ikinci aşamada emülatörde uygulama çalıştırılarak API çağrılarını dinamik olarak elde etmektedir. Hibrit analiz yöntemi ile hedeflenen ise statik analiz yöntemi ve dinamik analiz yönteminin dezavantajlarından kaçınarak doğru sonuçlara ulaşabilmektir.

3.7. Artırımlı Temel Bileşen Analizi

Temel bileşen analizi (PCA: Principal Component Analysis), veri kümesi içerisinde verileri temsil edebilecek temel özelliklere ulaşabilmek için kullanılan istatistiksel bir yöntemdir.

PCA, X-Y şeklindeki bir koordinat sistemi içinde çok değişkenli dağıtılmış veri seti ele alındığında veri setini en az veri kaybıyla ve en az değişkenle temsil edebilmeyi amaçlamaktadır. PCA başlangıçta orijinal veri kümesinin maksimum varyasyonunu bulur. Daha sonra X-Y koordinatındaki veri noktaları Şekil 3.4 de b ile gösterilen U-V koordinat sistemi ile ifade edilen farklı bir eksen üzerinde gösterilir. U ve V eksenindeki yönler temel bileşenler olarak adlandırılmaktadır. Verilerin değiştiği ana yön U ekseni, bu eksene dik olanı ise V-ekseni ile gösterilmektedir. Şekil 3.4'teki gibi V ekseninde gösterilen veri koordinatlarının sıfıra çok yaklaşması durumunda, V ekseni ile gösterilen veriler çıkarılabilir ve veri seti yalnızca bir U ekseni ile gösterilen değişkenler ile temsil edilebilmektedir (Ng, 2017).



Şekil 3.4. Temel bileşen analizi (Ng, 2017)

Artırımlı temel bileşen analizi (IPCA), veri setinin boyutunun çok büyük olması durumunda PCA yerine kullanılmaktadır. Veri setinin büyük olması algoritmayı koşacak sistem belleğinin yetersiz kalmasına neden olabilmektedir. IPCA bu bellek problemini aşabilmek amacıyla geliştirilmiş bir yöntemdir. İlk olarak Hall vd. tarafından sunulan ve daha sonraki çalışmalarla geliştirilen IPCA yönteminde

tüm eğitim örneklerini almak yerine eğitim örneklerini belirlenen sayıya ayırarak her seferde bu belirlenen sayı kadar örneğin işleme tabi tutulup iteratif olarak tüm eğitim örneklerinin işlenmesi esasına dayanmaktadır. Bu eğitim örneklerinin belli parçalara ayırarak işlenmesi, bellekteki yükü azaltmakta ve bu sayede bellek yetersizliği probleminin önüne geçilmesini hedeflemektedir (Hall vd., 1998; Ozawa vd., 2006).

3.8. Derin Öğrenme

Derin Öğrenme (Deep Learning:DL), yapay sinir ağları ile ilgili çalışmaların geliştirilmesi sonucu ortaya konmuş makine öğrenme tekniklerinden biri olarak karşımıza çıkmaktadır. Yapay sinir ağlarının temelleri, 1950'li yıllara dayanmaktadır. Yapay sinir ağlarının gelişim süreci perseptron algoritmasının geliştirilmesi ile başlatılmıştır. Kullanılan doğrusal aktivasyon fonksiyonunun karmaşık problemlerde başarısının düşük olması nedeniyle, 1970'li yıllarda aktivasyon fonksiyonunun doğrusal olmayan türü geliştirilmiştir. 1980'li yıllarda katmanların artırılması yoluna gidilerek perseptron çok katmanlı model yapısına büründürülmüştür. Daha sonra perseptronun eğitimi sırasında hatanın geri yayılması mantığına dayanan geri yayılım (backpropogation) algoritması geliştirilerek yapay sinir ağlarında başarım oranının artırılması hedeflenmiştir. Verilerin artması ve kurulan yapay sinir ağlarının boyutunun artması ile bu sinir ağlarının eğitimi için gerekli bilgisayar donanımının olmaması nedeniyle 2000'li yılların sonuna kadar popülaritesini kaybetmiştir. Hızla gelişen CPU ve GPU teknolojileri ile hesaplama gücünde meydana gelen artış ile yapay sinir ağları son yıllarda tekrar gözleri kendi üzerine çekebilmiştir. Günümüzde perseptron yapısının katmanlı ve içerdiği gizli katman sayısının birden fazla olması bu yapının derin ağlar olarak adlandırılmasına vesile olmuştur (Aygün, 2017; Ibrahım, 2019).

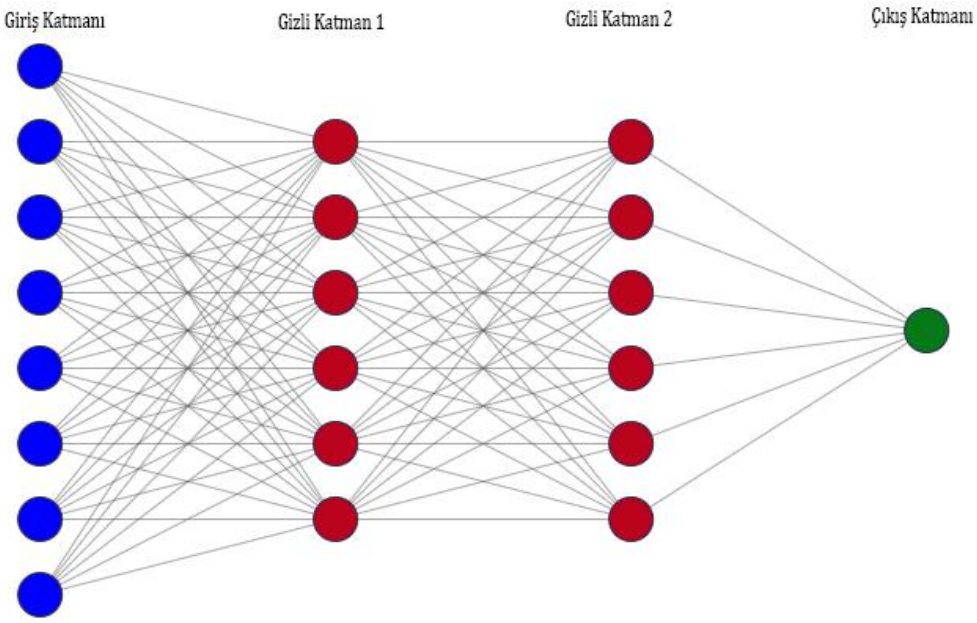
DL yöntemleri, büyük verilerin temsili ve özellikler ile ilgili olarak zengin veri kümeleri içinden karmaşık kalıpları tanımlamada başarılı oldukları için çözümü zor olarak nitelendirilen problemlere etkili çözümler sunabilmektedirler. DL, örnek bir kümeden veri setlerine ait temel özellikleri öğrenebilme yeteneğine

sahiptir. DL; doğal dil işleme, görüntü işleme, ses tanıma, zaman serisi analizi, zararlı yazılım tespiti gibi birçok alanda kullanılmaktadır (Kolosnjaji vd., 2016; Barros vd., 2017; Qiu vd., 2017; Ranjan vd., 2017; Zeyer vd., 2017; Cui vd., 2018; Mezgec vd., 2019; Tokmak ve Küçüksille, 2019).

DL içerisinde yapay sinir ağlarını temel alan çeşitli metotlar kullanılmaktadır. Bunlara örnek olarak; derin sinir ağı (DNN, Deep Neural Network), konvolüsyonel sinir ağları (CNN), öz-yinelemeli sinir ağı (RNN) verilebilmektedir.

3.8.1. Derin sinir ağı

İnsan beyni örnek alınarak, öğrenme aşamalarında nöronların kullanılması ilkesinin bilgisayar dünyasında simüle edilmesi yöntemi ile oluşturulan DL'nin kullandığı yöntemlerden biri DNN'dir. DNN olarak adlandırılan, ileri beslemeli sinir ağları veya birçok gizli katmana sahip çok katmanlı perseptronlar derin mimariye sahip modellerin iyi örnekleri arasında bulunmaktadırlar. DNN, giriş katmanı, çıkış katmanı ve birden çok gizli katmanı olan bir sinir ağı olarak ifade edilebilmektedir. DNN'nın her katmanında ele alınan problemle ilgili çıkarılan öznitelikler öğrenilmekte ve söz konusu katmanda öğrenilen öznitelikler, kendisinden sonra gelen katman için girdi değerlerini oluşturmaktadır. Bu sayede birinci katmandan başlayarak nihai katmana doğru özniteliklerin öğrenilerek gidildiği bir ağ yapısı oluşturulmuş olmaktadır (Tokmak ve Küçüksille, 2019). Bir giriş katmanı, 2 adet gizli katman ve bir çıkış katmanına sahip bir DNN Şekil 3.5'te gösterilmiştir.



Şekil 3.5. Derin sinir ağı yapısı

4. ARAŞTIRMA BULGULARI

Bu tez çalışmasında Android zararlı yazılımlarının tespiti için literatür ışığında statik analiz yöntemi ve hibrit analiz yöntemi uygulanmıştır. Bu çerçevede yapılan çalışma statik analiz ve hibrit analiz başlıkları altında açıklanmıştır.

4.1. Statik Analiz Uygulaması

Makine öğrenme teknikleri kullanılarak Android uygulamalarının statik analiz yöntemiyle analizi için özniteliklerin çıkarılması ve çıkarılan bu özniteliklerin makine öğrenme yöntemine girdi olarak verilmesi gerekmektedir. Bu amaçla öznitelik vektörü oluşturulması aşamasında Java programlama dilinde bir uygulama geliştirilmiş, oluşturulan öznitelik vektörü için boyut indirgeme ve DNN yöntemi uygulama aşamasında Python programlama dilinde bir uygulama geliştirilmiştir.

Android uygulamalarının analizi amacıyla Java programlama dilinde Kuzgun adında bir konsol uygulaması geliştirilmiştir. APK uygulama dosyalarını ayrıştırmak (parsing) için açık kaynak kodlu apkfile kütüphanesi geliştirilerek kullanılmıştır (Fenton, 2018). Bu uygulama ile klasör yolu belirtilen klasördeki tüm APK dosyalarının Manifest.xml, classes.dex dosyalarından analiz için gerekli öznitelikler çıkarılmış ve MySQL veri tabanına kaydedilmiştir. Tüm APK dosyalarından elde edilen öznitelikler veri tabanına kaydedildikten sonra, veri tabanından okunarak öznitelik vektörü csv dosya formatında kaydedilmiştir. Kaydedilen öznitelik dosyası Python modülü ile IPCA boyut indirgemeye tabi tutulmuş ve DNN Derin Öğrenme yöntemi ile sonuçlar alınmıştır.

4.1.1. Veri seti

Çalışmada kullanılan veri seti sayısal ve çeşitlilik açısından büyük ölçekli bir yapıya sahiptir. Veri seti içindeki zararlı Android uygulamaları AMD (Android Malware Dataset)'den edinilmiş, zararsız uygulamalar ise Hong Kong Bilim ve Teknoloji Üniversitesinden bir grup araştırmacının topladığı ticari

uygulamalardan edinilmiştir (Liu vd., 2016; Wei vd., 2017; AMD, 2018; Android Wake Lock Research, 2018).

AMD veri seti 24.553 örnek içermekte ve diskte kapladığı alan yaklaşık 60 GB boyutundadır. AMD veri seti 2010-2016 yılları arasında toplanmış ve 71 farklı zararlı yazılım ailesine dahil zararlı yazılımlardan oluşmaktadır. Zararlı yazılım veri setinin dahil olduğu aile, örnek sayısı, oluşturulma tarihi ve tespit tarihi Çizelge 4.1'de gösterilmiştir. Kuzgun yazılımı ile zararlı uygulamaların ayrıştırılması esnasında karşılaşılan sertifika hataları, imza hataları, kaynak tablo hataları gibi hatalardan dolayı çalışmada 24.503 tane zararlı yazılım uygulaması kullanılmıştır.

Çizelge 4.1. Zararlı yazılım aileleri

Zararlı Yazılım Ailesi	Örnek Sayısı	Zararlı Tipi	Oluşturulma Tarihi	Tespit Tarihi
Airpush	7843	Adware	2011-12	2014-03
AndroRAT	46	Backdoor	2013-03	2013-07
Andup	45	Adware	2013-06	2014-11
Aples	21	Ransom	2014-07	2014-07
BankBot	648	Trojan-Banker	2014-02	2015-03
Bankun	70	Trojan-Banker	2013-08	2013-07
Boqx	215	Trojan-Dropper	2012-11	2013-07
Boxer	44	Trojan-SMS	2012-09	2011-09
Cova	17	Trojan-SMS	2013-10	2014-06
Dowgin	3385	Adware	2013-04	2015-02
DroidKungFu	546	Backdoor	2011-06	2011-05
Erop	46	Trojan-SMS	2013-10	2014-08
FakeAngry	10	Backdoor	2012-11	2012-02
FakeAV	5	Trojan	2013-09	2014-04
FakeDoc	21	Trojan	2012-01	2012-05
FakeInst	2172	Trojan-SMS	2011-10	2012-05
FakePlayer	21	Trojan-SMS	2010-10	2010-08
FakeTimer	12	Trojan	2012-02	2012-01
FakeUpdates	5	Trojan	2012-02	2013-08
Finspy	9	Trojan-Spy	2014-08	2014-08
Fjcon	16	Backdoor	2012-08	2011-11
Fobus	4	Backdoor	2014-01	2015-03
Fusob	1277	Ransom	2015-01	2015-10
GingerMaster	128	Backdoor	2011-09	2011-08

GoldDream	53	Backdoor	2011-07	2011-07
Gorpo	36	Trojan-Dropper	2014-09	2015-08
Gumen	145	Trojan-SMS	2013-09	2013-10
Jisut	560	Ransom	2014-05	2014-06
Kemoge	15	Trojan-Dropper	2014-10	2015-10
Koler	69	Ransom	2014-10	2014-05
Ksapp	36	Trojan	2012-07	2013-01
Kuguo	1199	Adware	2012-03	2015-02
Kyview	175	Adware	2013-03	2013-04
Leech	128	Trojan-SMS	2015-01	2015-09
Lnk	5	Trojan	2013-10	2010-07
Lotoor	329	HackerTool	2010-08	2012-09
Mecor	1820	Trojan-Spy	2013-09	2015-07
Minimob	203	Adware	2013-07	2013-09
Mmarketpay	14	Trojan	2012-10	2012-07
MobileTX	17	Trojan	2012-02	2012-05
Mseg	235	Trojan	2013-08	2013-08
Mtk	67	Trojan	2013-06	2013-02
Nandrobox	76	Trojan	2011-11	2012-07
Obad	9	Backdoor	2013-05	2013-06
Ogel	6	Trojan-SMS	2013-09	2015-06
Opfake	10	Trojan-SMS	2013-05	2012-01
Penetho	18	HackerTool	2011-03	2012-10
Ramnit	8	Trojan-Dropper	2014-10	2014-11
Roop	48	Ransom	2015-04	2015-05
RuMMS	402	Trojan-SMS	2016-01	2016-04
SimpleLocker	173	Ransom	2014-05	2014-06
SlemBunk	174	Trojan-Banker	2014-04	2015-12
SmsKey	165	Trojan-SMS	2012-12	2013-04
SmsZombie	9	Trojan-Spy	2012-08	2012-08
Spambot	15	Backdoor	2013-09	2013-12
SpyBubble	10	Trojan-SMS	2012-08	2011-11
Stealer	25	Trojan-SMS	2013-10	2013-07
Steek	12	Trojan-Clicker	2012-01	2012-01
Svpeng	13	Trojan-Banker	2013-09	2013-11
Tesbo	5	Trojan-SMS	2013-08	2013-09
Triada	210	Backdoor	2015-02	2016-03
Univert	10	Backdoor	2013-11	2014-07
UpdtKiller	24	Trojan	2012-09	2012-07
Utchi	12	Adware	2012-11	2014-02
Vidro	23	Trojan-SMS	2013-07	2012-08
VikingHorde	7	Trojan-Dropper	2016-01	2016-05
Vmvol	13	Trojan-Spy	2013-03	2013-06
Winge	19	Trojan-Clicker	2013-03	2013-01
Youmi	1301	Adware	2011-09	2015-02

Zitmo	24	Trojan-Banker	2012-04	2011-10
Ztorg	20	Trojan-Dropper	2015-05	2015-08
Toplam	**24553**			

Zararlı olmayan yazılımlar, araştırmacı grup tarafından 224 bölümlük bloklar halinde paylaşılmıştır. Veri kümesi 44.736 adet zararsız uygulamadan oluşmaktadır. İndirme esnasında verilen linklerdeki problemler ve geliştirilen Kuzgun yazılımı ile uygulamaların ayrıştırılması esnasında karşılaşılan sertifika hataları, imza hataları, kaynak tablo hataları gibi hatalardan dolayı çalışmada bu örneklerden 38.044 tanesi kullanılmıştır ve bu uygulamaların diskte kapladığı alan yaklaşık 350 GB boyutundadır. Oluşturulan veri seti toplamda 62.547 adet uygulama içermektedir.

4.1.2. Statik analiz uygulaması öznitelikleri

Statik analiz yöntemi ile analiz gerçekleştirilirken Android zararlı yazılımlarını tespit edebilmek için AndroidManifest.xml dosyasından ve classes.dex dosyasından elde edilen veriler kullanılmıştır. Yapılan çalışmalar incelendiğinde; tespit mekanizmalarında çoğunlukla Android izinlerine dayanan çalışmaların yoğunlukta olduğu ancak tek başına izinlerin yeterli olmadığı görüşü ile API çağrıları, amaç, aktivite vb. gibi özelliklerin de ele alınması gerekliliği ortaya çıkmıştır. Bu amaçla çalışmada geniş bir öznitelik yelpazesi kullanılmıştır. Kullanılan öznitelikler Ö etiketi ile ifade edilmiştir.

$Ö_1$ İstenen izinler: Android güvenlik mekanizmasında önemli bir yer tutmaktadır. Google tarafından da Android 6.0 ile değişikliğe gidilmiş, önceki Android sürümlerinde kurulum esnasında tüm izinlerin kabulü yerine, kurulumdan sonra çalışma anında izin isteme stratejisine geçilmiştir. Zararlı yazılımlar aldıkları izinler vasıtası ile cihaza ve kişisel bilgilere erişim sağlayabilmekte ve çeşitli zararlar doğurabilmektedir. CAMERA izni alan bir yazılım kameraya erişim sağlayabilmekte ve kullanıcıdan habersiz fotoğraf ve video çekebilmektedir. SEND_SMS izni alan bir yazılım SMS mesajı gönderebilmekte, RECEIVE_SMS izni alan bir yazılım SMS mesajlarını alabilmektedir bu izinler sayesinde kullanıcıdan

habersiz gönderilen ve alınan mesajlar yüksek faturalara ve istenmeyen aboneliklere sebep olabilmektedir.

$Ö_2$ Amaçlar: Android uygulama bileşenleri arasında bilgi alışverişini ve veri aktarımını sağlamaktadırlar. Bir aktivitenin başlatılması, aktiviteler arası geçiş sağlanması veya bir servisin başlatılması Amaçlar sayesinde yapılmaktadır. Kötü amaçlı yazılımla ilgili Amaç mesajının tipik bir örneği BOOT COMPLETED, akıllı telefonu yeniden başlattıktan hemen sonra kötü amaçlı etkinliği tetiklemek için kullanılmaktadır.

$Ö_3$ Android uygulama bileşenleri: 4 farklı kategoride temsil edilmektedirler. Bunlar aktiviteler, servisler, içerik sağlayıcılar, yayın alıcılardır. Her uygulama bu bileşenleri manifest dosyasında deklare etmektedir. Bu bileşenlerin adları da çok bilinen zararlı yazılımları tanımlamaya yardımcı olabilmektedir. DroidKungFu zararlısının ikinci varyantı DroidKungFu2 com.eguan.state.Receiver yayın alıcı ismi ve com.eguan.state.StateService servis ismi ile tanımlanabilmektedir.

$Ö_4$ Kısıtlı API çağrıları: Android Güvenlik Mimarisine göre, bir Android uygulamasının manifest dosyasında REBOOT, DELETE_PACKAGE gibi kritik API çağrılarına erişmek için gerekli izinleri tanımlaması gerekmektedir. Uygulamanın manifest dosyasında istemeden bu API çağrılarını kullanıyor olması mümkündür. Kısıtlı API çağrılarının izin istenmeden kullanılması durumu zararlı davranışların ortaya çıkarılmasında takip edilmesi gereken bir olay olarak karşımıza çıkmaktadır. Bu, zararlı yazılımın Android platformunun getirdiği sınırlamaları aşmak için root exploit kullandığını gösterebilmektedir (Arp vd., 2014; Bhandari vd., 2015; Fereidooni vd., 2016).

$Ö_5$ Şüpheli API çağrıları: API çağrılarının bazıları hassas kaynaklara veya akıllı telefonun bilgilerine erişebilmektedir. Bu tür API çağrıları genellikle zararlı yazılım örneklerinde görülmekte ve kötü amaçlı davranışlara neden olabilmektedir. Hassas verilere erişimde kullanılan getDeviceId() ve getSubscriberId(),ağ üzerinden iletişim kurmak için kullanılan setWifiEnabled() ve execHttpRequest(), SMS almak ve göndermek için sendTextMessage(), kod

karmaşıklaştırma için kullanılan Cipher.getInstance() gibi API çağrıları şüpheli olarak görülmektedir (Arp vd., 2014; Seo vd., 2014).

$Ö_6$ Kullanılan izinler: API çağrıları takip edilerek hem talep edilen hem de fiilen kullanılan izinler olarak öznitelik vektöründe tanımlanmıştır. Kullanılan izinlerin analizi de uygulama davranışları hakkında belirleyici olduğundan dolayı belirleyici bir faktör olmaktadır. Kullanılan izinlerin tespiti için API çağrılarının gerektirdiği izinlerin haritalamasını yapan PScout adlı çalışma esas alınmıştır.

$Ö_7$ Kullanılmayan izinler: geliştirilen uygulamalarda manifest dosyasında tanımlanmış ama bu izin geliştirilen uygulama tarafından kullanılmayan izinler olarak tanımlanmıştır. Zararlı yazılım incelenmesi esnasında zararlının davranışlarının anlaşılmasını zorlaştırmak amacıyla bazı geliştiriciler tarafından fazladan izin tanımlanarak, zararlının davranış analizlerinin zorlaştırılması hedeflenebilmektedir. Bu amaçla kullanılmayan izinlerin takibi önem arz etmektedir. Kullanılmayan izinler PScout adlı çalışmada paylaşılan API izin haritası kullanılarak çıkarılmıştır.

$Ö_8$ Sistem Komutları: Android platformunda Linux işletim sisteminde olduğu gibi bir takım komut seti kullanılabilmektedir. Bu sistem komutları sayesinde zararlı uygulamalar root yetkisi alarak exploit kodu çalıştırabilmekte, indirme yapabilmekte ve çalıştırılabilir dosya kurabilmektedir (Fereidooni vd., 2016). (Seo vd. (2014)) çalışmasında en sık kullanılan sistem komutlarını listelemiştir. Bu listedeki komutlar öznitelik çıkarımında esas alınmıştır.

$Ö_9$ Ağ adresleri: Zararlı bir yazılım, aygıttan toplanan komutları almak veya verileri dışarı göndermek için düzenli olarak ağ bağlantıları kurabilmektedirler. Bu nedenle, uygulamanın kod bloğunda bulunan tüm IP adresleri, ana bilgisayar adları ve URL'ler öznitelik grubuna dahil edilmiştir.

Zararlı yazılımları tespit edebilmek için 9 grupta kategorize edilen öznitelikler DNN ile analizi esnasında vektörel olarak ifade edilmiştir. Bu amaçla 9 özellik seti içindeki string değerler Ö seti içinde tanımlanmıştır. Ö özellik seti yaklaşık

750.000 farklı özellikten oluşmaktadır. Denklem 4.1'de olduğu gibi Ö 9 özellik grubunun birleşimi olarak ifade edilebilmektedir.

$$Ö \coloneqq Ö_1 \cup Ö_2 \cup \ldots \cup Ö_9 \tag{4.1}$$

Ö seti kullanılarak, her boyutun 0 veya 1 olduğu |Ö| boyutlu vektör uzayı tanımlanmıştır. Bir *x* uygulaması , $\boldsymbol{\varphi}(\boldsymbol{x})$ vektörü ile eşleştirilir böylece x uygulamasından çıkarılan her özellik için ilgili boyut 1 diğer boyutlar 0 olarak haritalanmaktadır. Bu haritalama X uygulaması için Denklem 4.2' de gösterilmiştir.

$$\varphi: X \to \{0,1\}^{|Ö|}, \varphi(x) \mapsto \big(I(x, ö)\big)_{ö \epsilon Ö} \tag{4.2}$$

$\boldsymbol{I}(\boldsymbol{x}, ö)$ fonksiyonu basit olarak Denklem 4.3'te gösterilmiştir.

$$I(x,ö)=\begin{cases}1 & \text{x uygulaması ö özelliğini içeriyorsa}\\ 0 & \text{değilse}\end{cases} \tag{4.3}$$

$\boldsymbol{\varphi}(\boldsymbol{x})$ öznitelik vektörü Denklem 4.4'te gösterilmiştir.

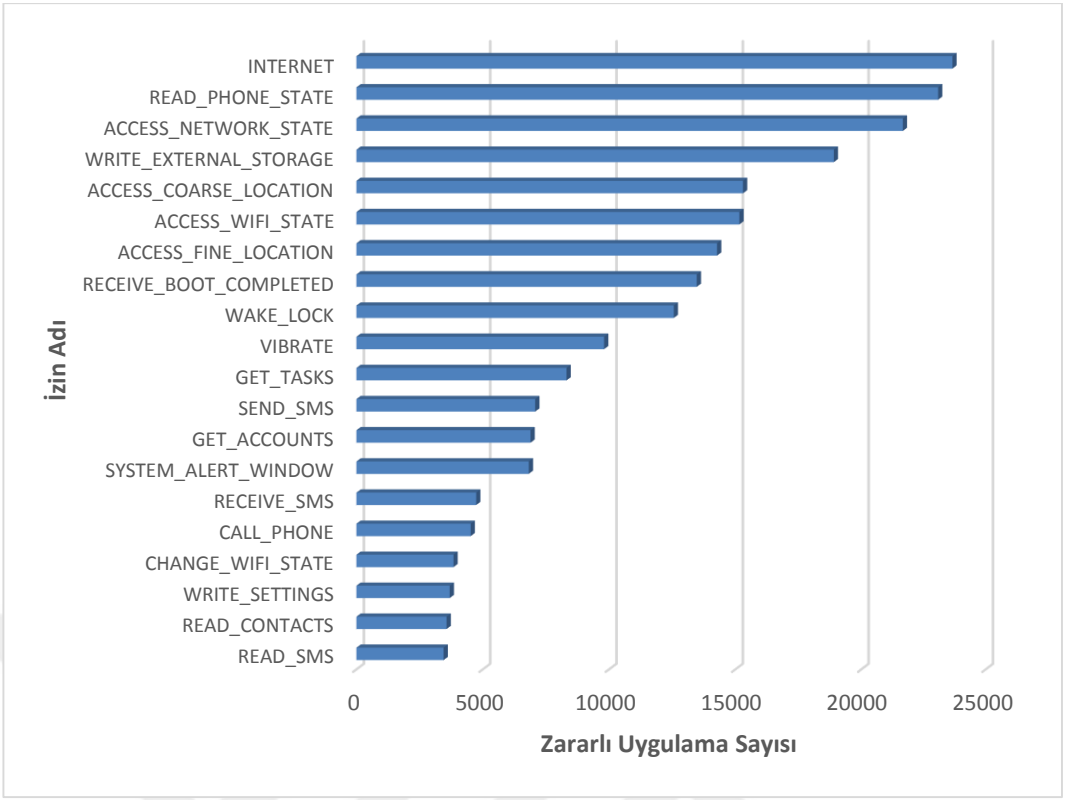
$$\varphi(x)\mapsto \begin{pmatrix} \ldots \\ 1 \\ 0 \\ \ldots \\ 1 \\ 0 \\ \ldots \end{pmatrix} \begin{matrix} \ldots \\ \text{android.permission.ACCESS\_NETWORK\_STATE} \\ \text{android.permission.READ\_PHONE\_NUMBERS} \\ \ldots \\ \text{getDeviceId} \\ \text{setWifiEnabled} \\ \ldots \end{matrix} \tag{4.4}$$

Çalışmanın bu kısmında zararlı ve zararlı olmayan uygulamalardan statik analiz sonucunda elde edilen veriler paylaşılmıştır. Şekil 4.1'de. zararlı uygulamalarda istenen izinler, Şekil 4.2'de zararlı olmayan uygulamalarda istenen izinler, Şekil 4.3'te zararlı uygulamalarda kullanılan amaçlar, Şekil 4.4'te.zararlı olmayan uygulamalarda kullanılan amaçlar, Şekil 4.5'de. zararlı uygulamalarda kullanılan Android bileşenleri, Şekil 4.6'da zararlı uygulamalarda kullanılan Android bileşenleri, Şekil 4.7'de zararlı uygulamalarda kullanılan kısıtlanmış API çağrıları, Şekil 4.8'de zararlı olmayan uygulamalarda kullanılan kısıtlanmış API çağrıları,

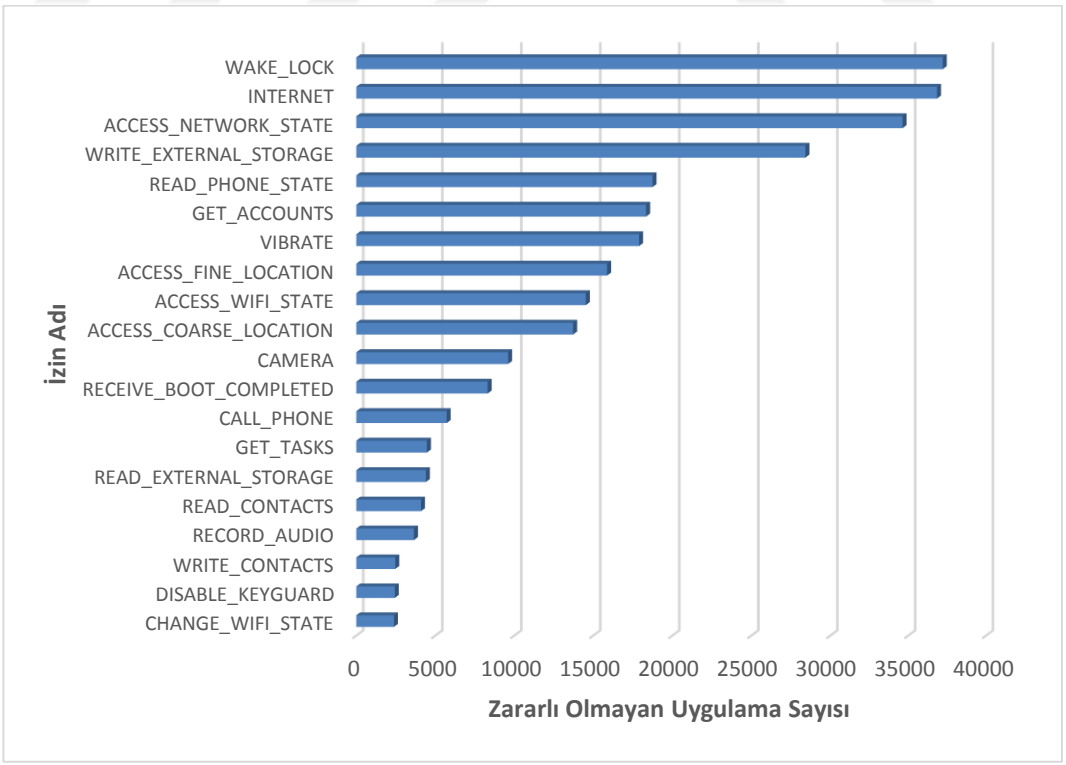
Şekil 4.9'da zararlı uygulamalarda kullanılan şüpheli api çağrıları, Şekil 4.10'da zararlı olmayan uygulamalarda kullanılan şüpheli api çağrıları, Şekil 4.11'de zararlı uygulamalarda istenip kullanılan izinler, Şekil 4.12'de zararlı olmayan uygulamalarda istenip kullanılan izinler**,** Şekil 4.13'de zararlı uygulamalarda istenip kullanılmayan izinler, Şekil 4.14'te zararlı olmayan uygulamalarda istenip kullanılmayan izinler, Şekil 4.15'de zararlı uygulamalarda kullanılan sistem komutları**,** Şekil 4.16'da zararlı olmayan uygulamalarda kullanılan sistem komutları, Şekil 4.17'de zararlı uygulamalarda kullanılan url adresleri, Şekil 4.18'de zararlı olmayan uygulamalarda kullanılan url adresleri gösterilmiştir.

Şekil 4.1'de görüldüğü gibi zararlı yazılımlarda en çok istenen izinler; Internet bağlantısı yapabilme, WiFi ayarlarını öğrenme ve değiştirme, SMS yazma ve okuma, konum bilgilerine ulaşma, telefon rehberine ulaşma, telefon çağrısı yapabilme ve ayarları değiştirme gibi izinlerdir.

Şekil 4.2'de ise zararlı olmayan yazılımlarda istenen izinler gösterilmiştir. Zararlı olmayan uygulamalarda da internet bağlantısı yapabilme, WiFi durumunu öğrenme ve değiştirme, SMS yazma ve okuma, konum bilgilerine ulaşma, telefon rehberine ulaşma, telefon çağrısı yapabilme ve ayarları değiştirme gibi izinler istenmiştir. Zararlı uygulamalar ile zararlı olmayan uygulamaların istedikleri izin sayılarına oransal olarak bakıldığında farklar görülebilmektedir. Örneğin zararlı uygulamalarda istenilen izinlerde READ.PHONE.STATE izni %90'ın üstünde istenme oranına sahipken, zararlı olmayan uygulamalarda bu oran %40 civarında görülmektedir.

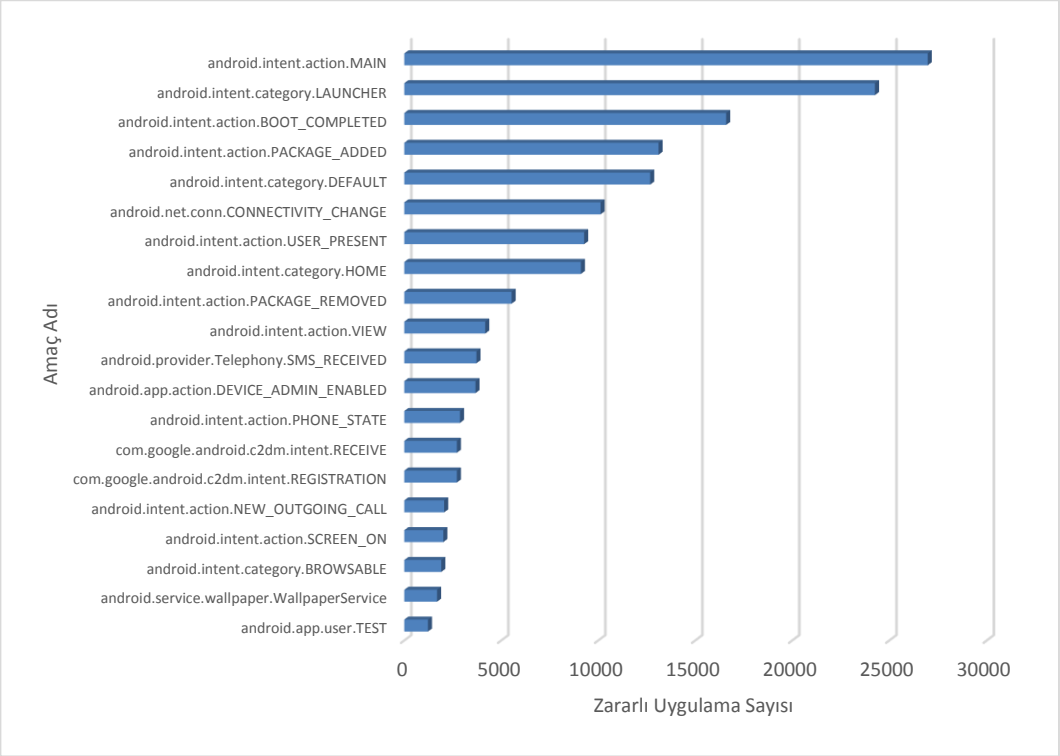


Şekil 4.1. Zararlı uygulamalarda istenen izinler

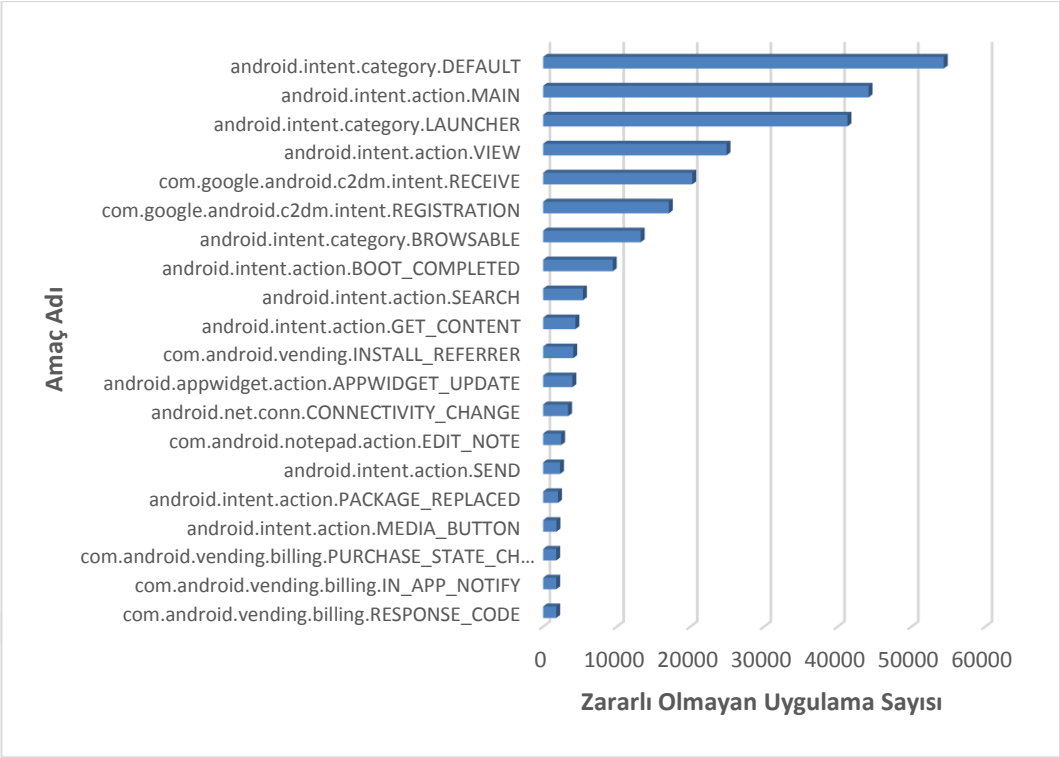


Şekil 4.2. Zararlı olmayan uygulamalarda istenen izinler

Şekil 4.3 ve Şekil 4.4'te Android amaçlarından elde edilen veriler gösterilmiştir. Zararlı uygulamalarda ve zararlı olmayan uygulamalarda ana aktivite tanımlaması için gerekli intent.action.MAIN ve category.LAUNCHER amaçların grafikte en üstte yer aldığı görülmektedir. Zararlı uygulamalardan elde edilen amaçlarda, cihazın yeniden başlatılması ile olayların çalışmasını sağlayan intent.action.BOOT_COMPLETED amacının sık kullanıldığı görülmektedir.

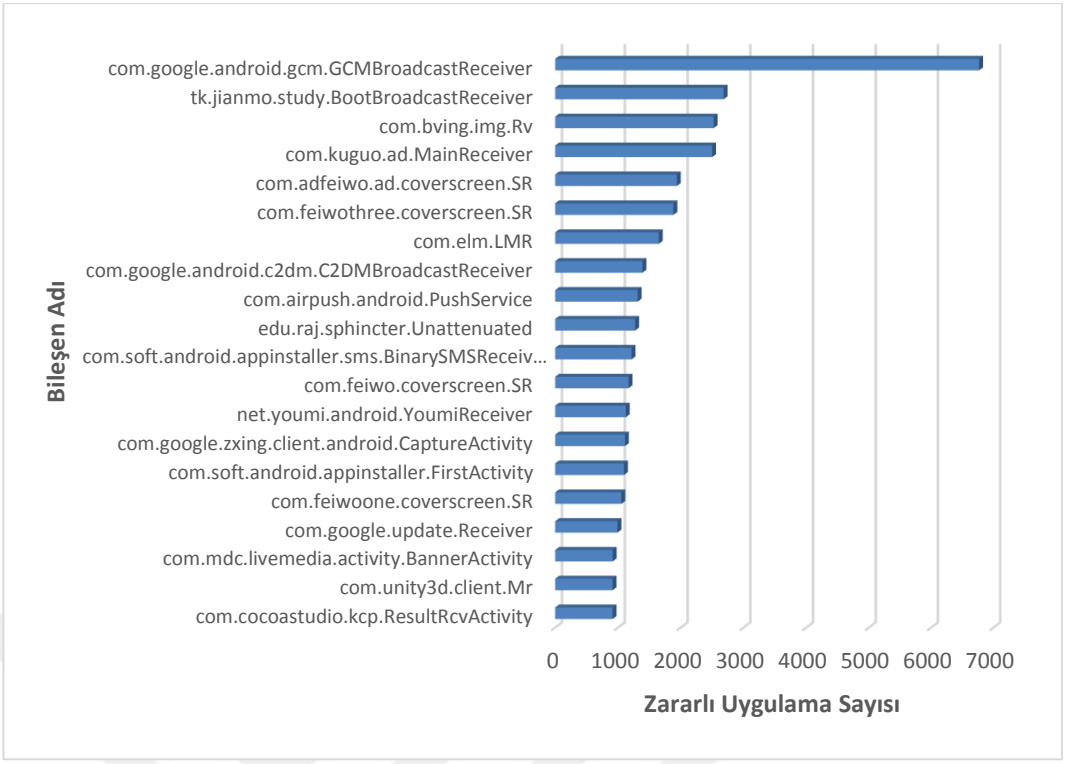


Şekil 4.3. Zararlı uygulamalarda kullanılan amaçlar

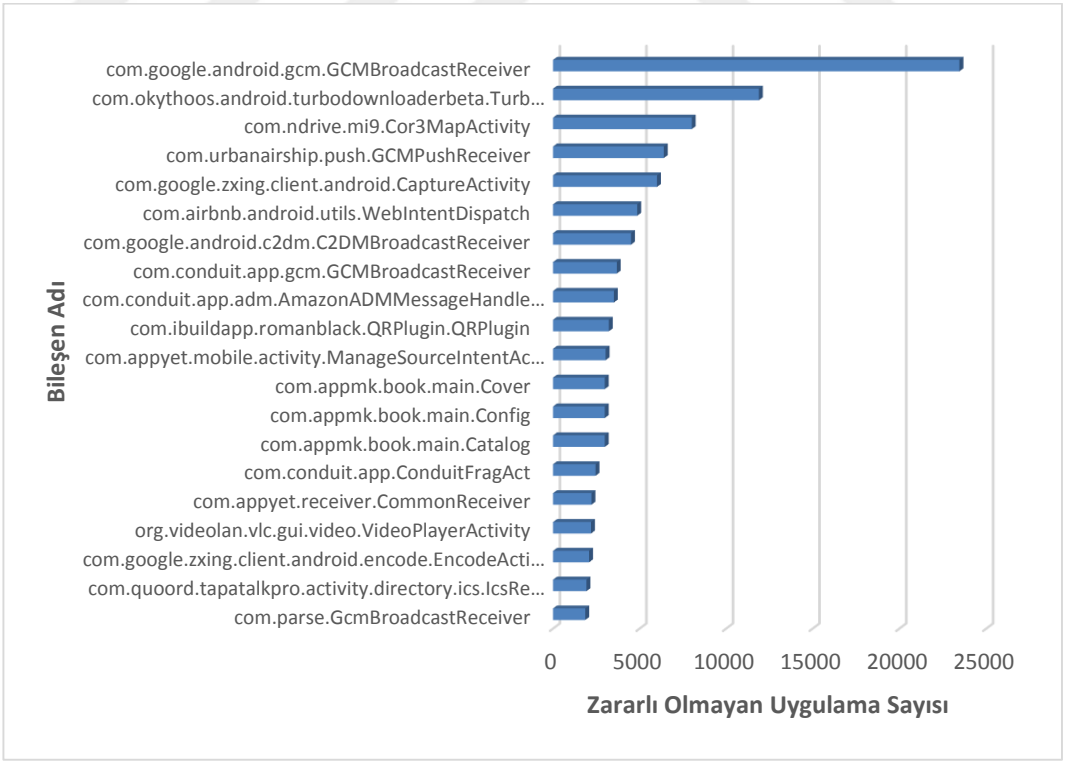


Şekil 4.4. Zararlı olmayan uygulamalarda kullanılan amaçlar

Şekil 4.5 ve Şekil 4.6'da ki grafiklerde zararlı ve zararlı olmayan uygulamalardan elde edilen Android bileşenleri gösterilmiştir. Grafiklere bakıldığında; uygulamalarda tanımlanan Android bileşen isimlerinin, uygulamaların zararlı ya da zararlı olmadığı hakkında karar vermeye yardımcı olduğu görülmektedir. Örneğin tk.jianmo.study.BootBroadcastReceiver bileşeninin bir zararlı uygulamaya karşılık geldiği, com.ndrive.mi9.Cor3MapActivity bileşeninin zararlı olmayan bir uygulamaya karşılık geldiği değerlendirilerek, uygulamanın zararlı ya da zararlı olmadığı hakkında karar vermede yardımcı olacak bir argüman olarak karşımıza çıkmaktadır.

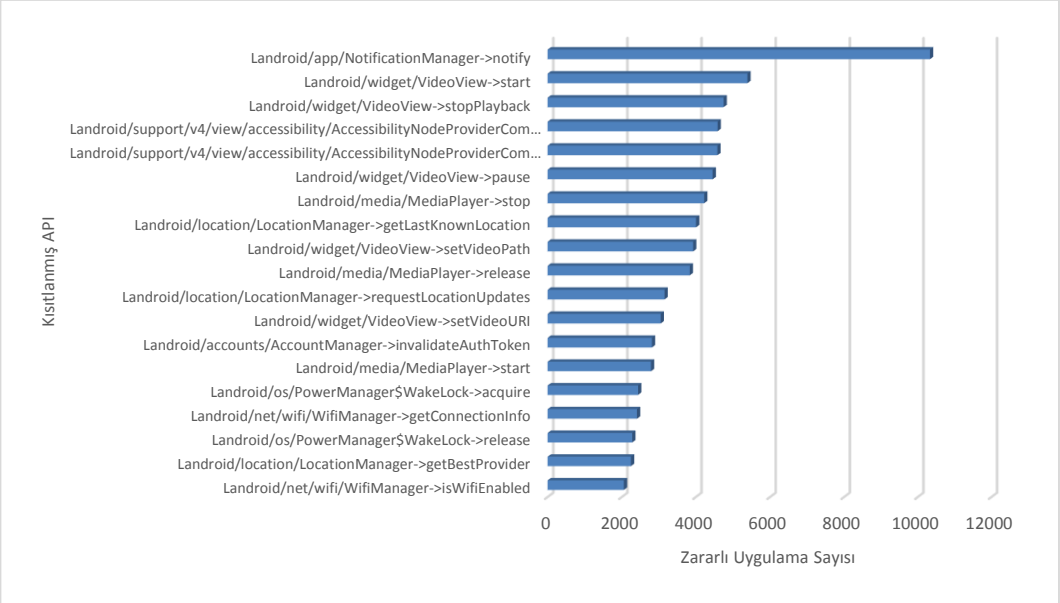


Şekil 4.5. Zararlı uygulamalarda kullanılan Android bileşenleri

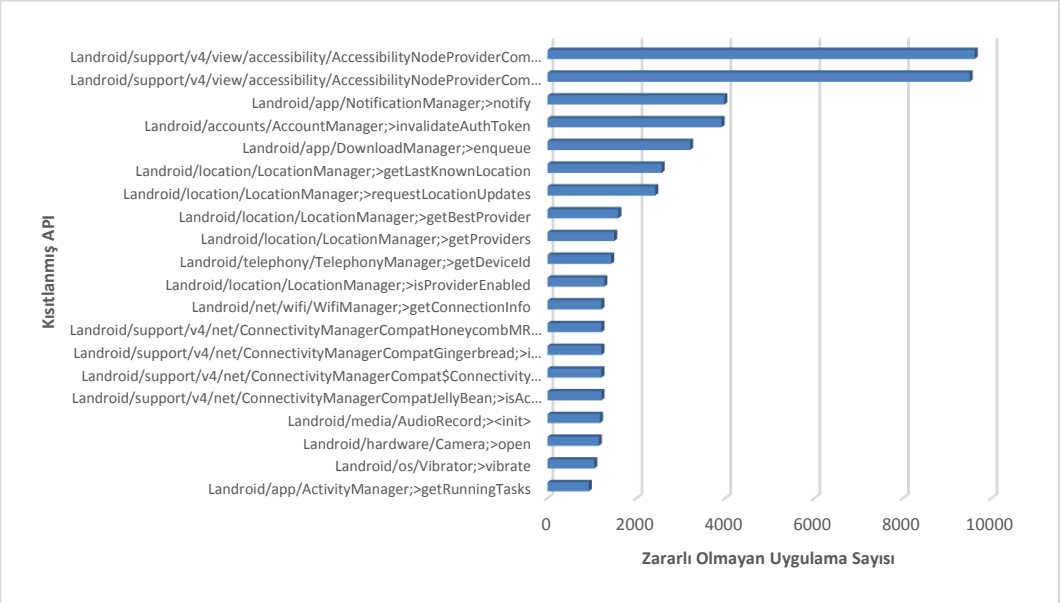


Şekil 4.6. Zararlı olmayan uygulamalarda kullanılan Android bileşenleri

Şekil 4.7 ve Şekil 4.8'de ki grafiklerde zararlı ve zararlı olmayan uygulamalardan elde edilen kısıtlanmış API çağrıları gösterilmiştir. Grafiklere bakıldığında, kısıtlanmış API çağrılarının zararlı olmayan uygulamalarda ki kullanım oranlarının zararlı olan uygulamalarda ki kullanım oranına göre düşük olduğu görülmektedir.

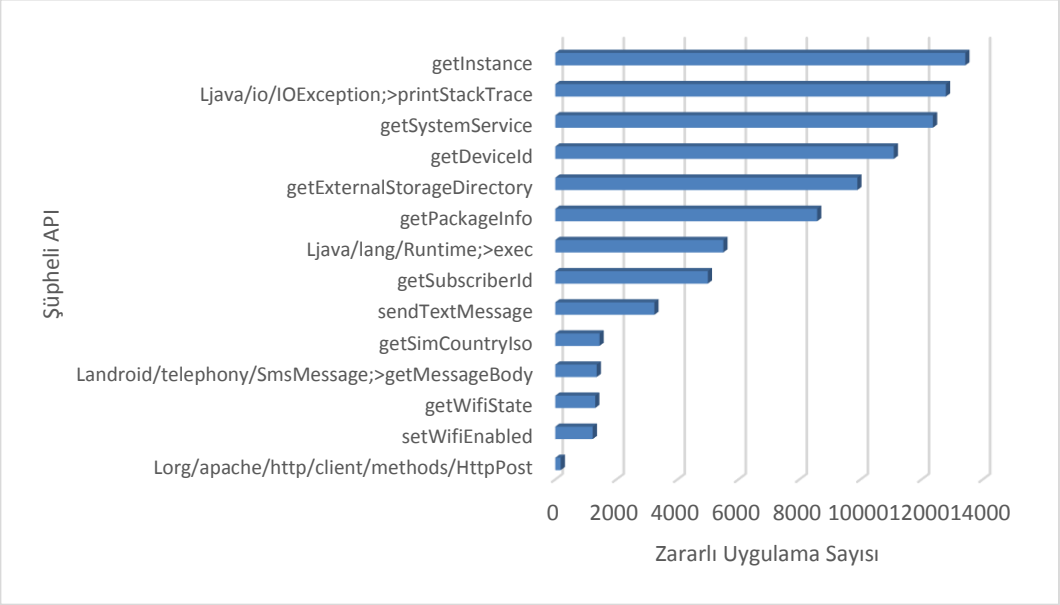


Şekil 4.7. Zararlı uygulamalarda kullanılan kısıtlanmış API çağrıları

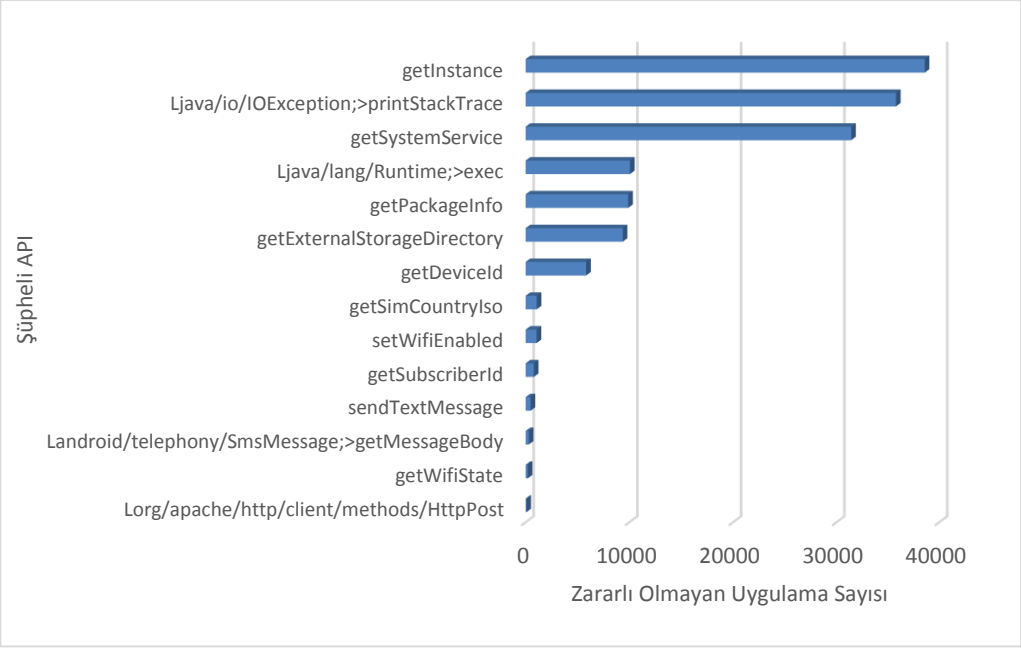


Şekil 4.8. Zararlı olmayan uygulamalarda kullanılan kısıtlanmış API çağrıları

Şekil 4.9 ve Şekil 4.10'da ki grafiklerde zararlı ve zararlı olmayan uygulamalardan elde edilen şüpheli API çağrıları gösterilmiştir. Grafiklere bakıldığında, şüpheli API çağrılarının zararlı olmayan uygulamalarda ki kullanım oranları ile zararlı olan uygulamalarda ki kullanım oranlarında farklar olduğu gözlemlenmektedir.

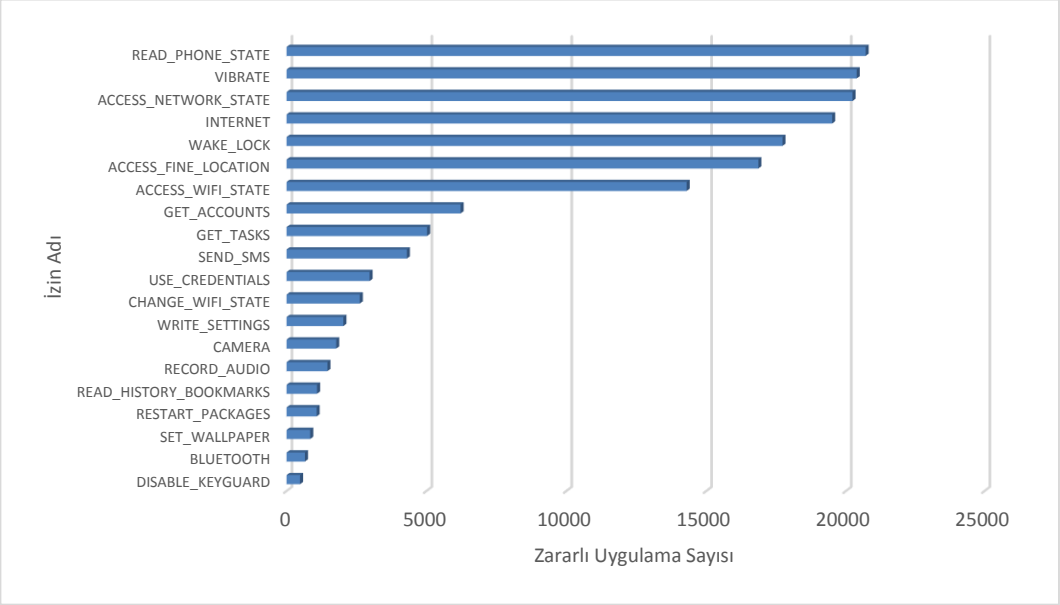


Şekil 4.9. Zararlı uygulamalarda kullanılan şüpheli API çağrıları

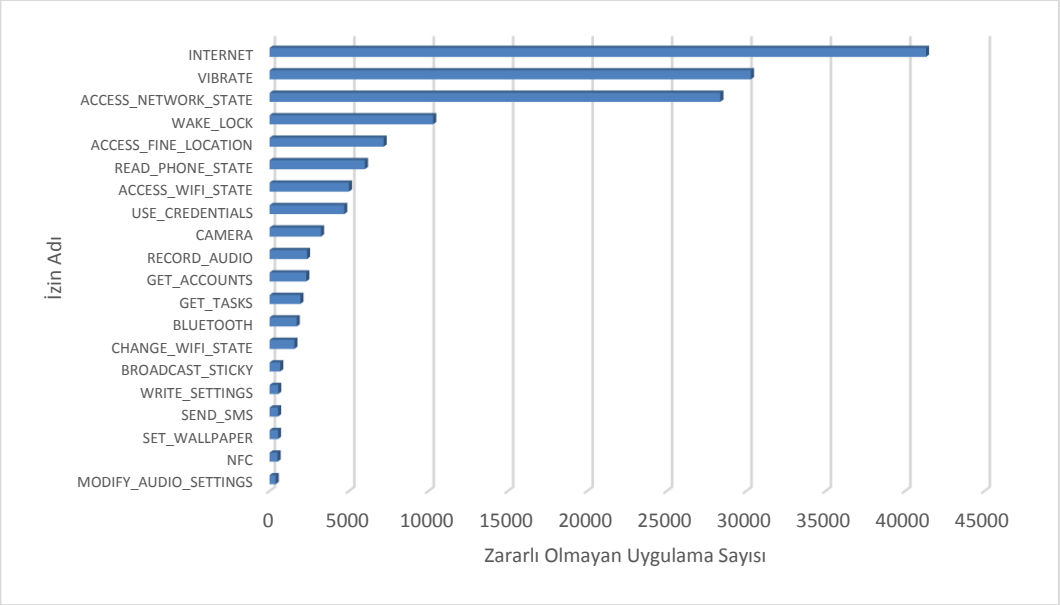


Şekil 4.10. Zararlı olmayan uygulamalarda kullanılan şüpheli API çağrıları

Şekil 4.11 ve Şekil 4.12'de ki grafiklerde zararlı ve zararlı olmayan uygulamalardan elde edilen uygulama tarafından istenen ve kullanılan izinler gösterilmiştir. Elde edilen verilerin istenen izinlerdeki özellikler ile paralellik gösterdiği görülmektedir.

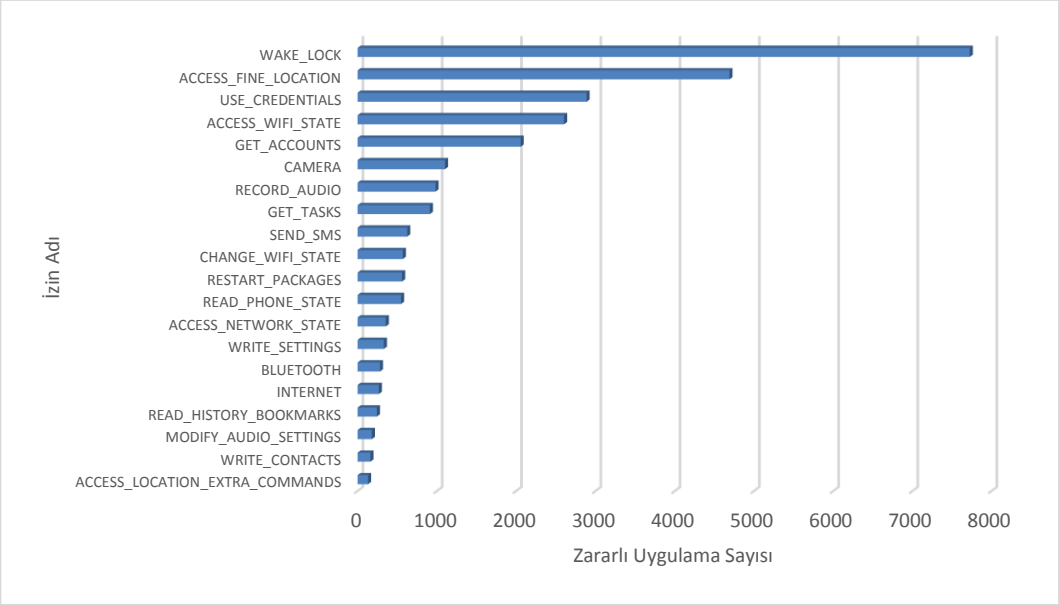


Şekil 4.11. Zararlı uygulamalarda istenip kullanılan izinler

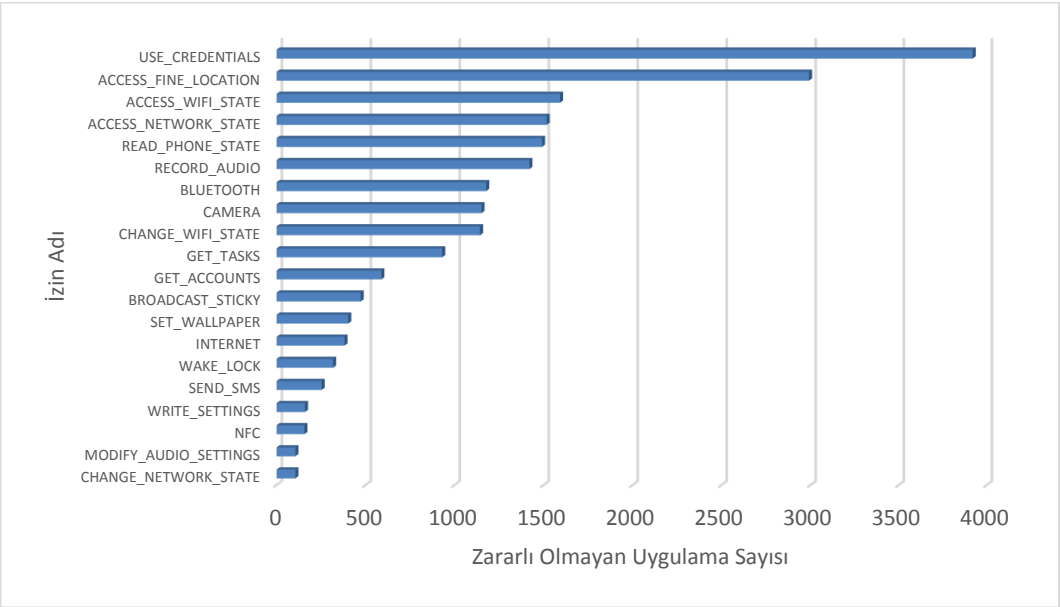


Şekil 4.12. Zararlı olmayan uygulamalarda istenip kullanılan izinler

Şekil 4.13 ve Şekil 4.14'te ki grafiklerde zararlı ve zararlı olmayan uygulamalardan elde edilen uygulama tarafından istenen ama kullanılmayan izinler gösterilmiştir. Elde edilen verilere bakıldığında, zararlı uygulamalarda istenen ama kullanılmayan izinlerin, zararlı olmayan uygulamalardakine göre oranının daha fazla olduğu görülmektedir.

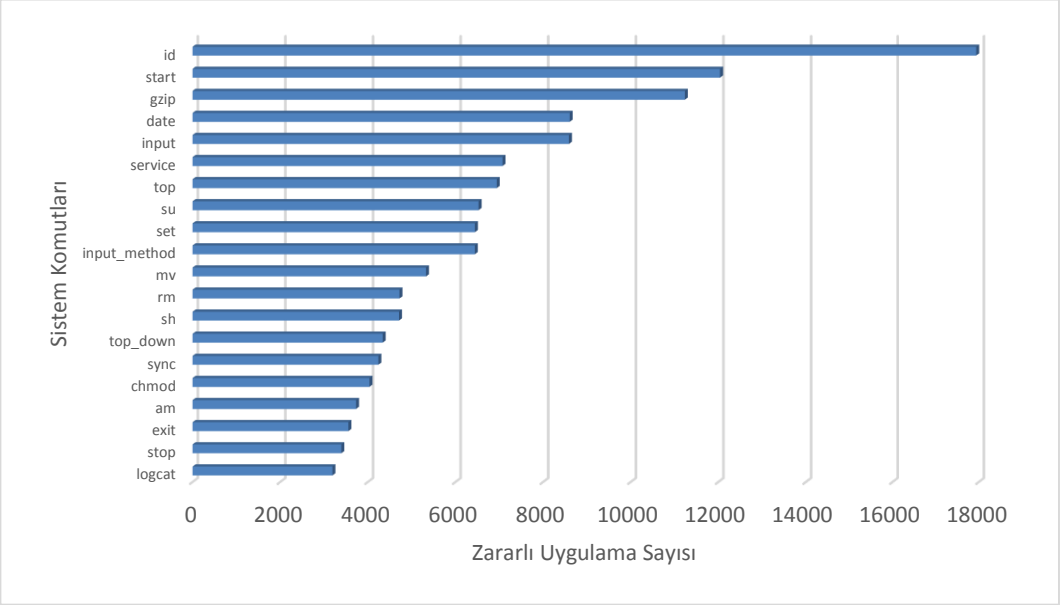


Şekil 4.13. Zararlı uygulamalarda istenip kullanılmayan izinler

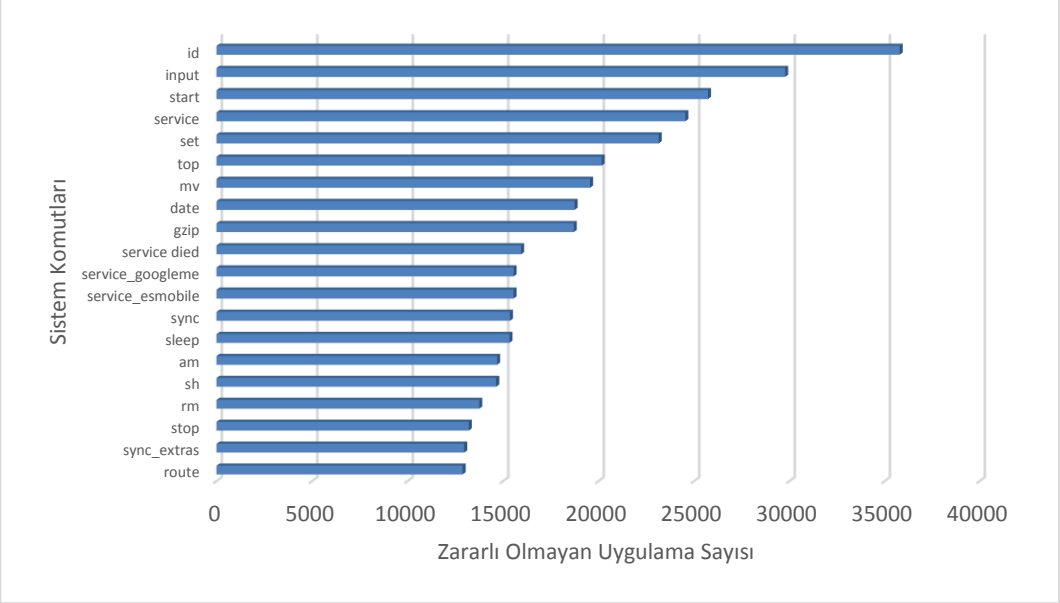


Şekil 4.14. Zararlı olmayan uygulamalarda istenip kullanılmayan izinler

Şekil 4.15 ve Şekil 4.16'da ki grafiklerde zararlı ve zararlı olmayan uygulamalardan elde edilen sistem komutları gösterilmiştir. Grafiklere bakıldığında özellikle "super user" yetkisi almak için kullanılan "su" komutunun zararlı uygulamalarda kullanımının çok olduğu görülmektedir.

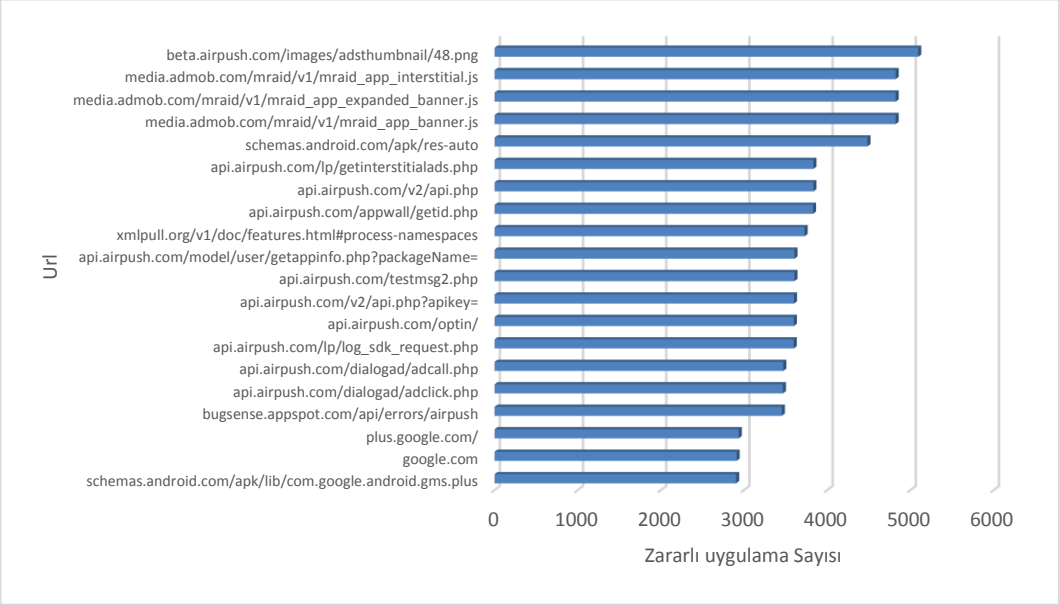


Şekil 4.15. Zararlı uygulamalarda kullanılan sistem komutları

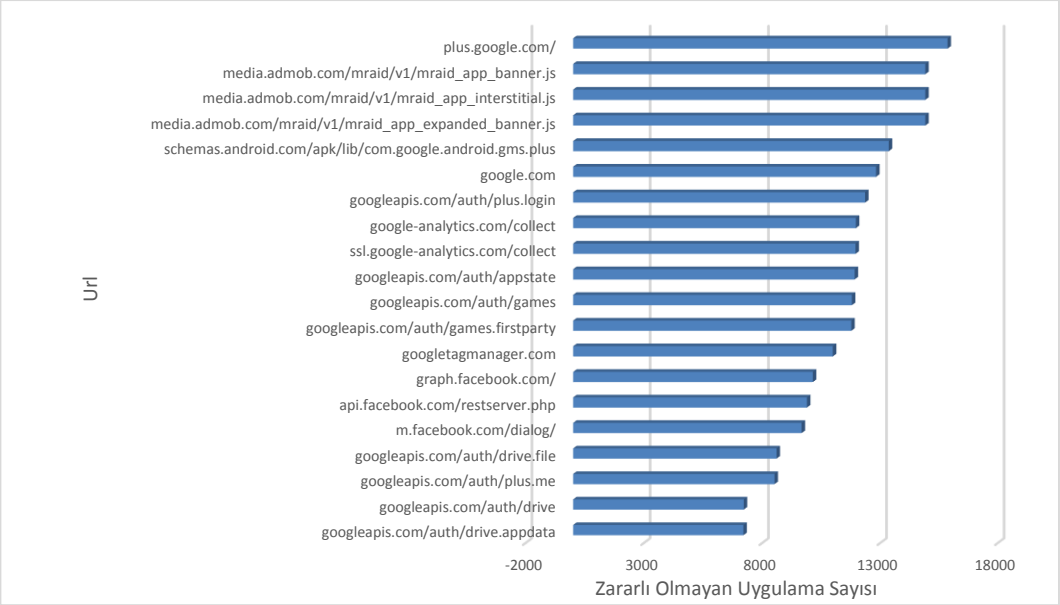


Şekil 4.16. Zararlı olmayan uygulamalarda kullanılan sistem komutları

Şekil 4.17 ve Şekil 4.18'de zararlı ve zararlı olmayan uygulamalardan elde edilen url adresleri gösterilmiştir. Grafiklere bakıldığında; uygulamalarda kullanılan url adreslerinin, uygulamaların zararlı ya da zararlı olmadığı hakkında karar vermeye yardımcı olduğu görülmektedir.



Şekil 4.17. Zararlı uygulamalarda kullanılan url adresleri



Şekil 4.18. Zararlı olmayan uygulamalarda kullanılan url adresleri

4.1.3. Statik analiz artırımlı PCA

Statik analiz işlemleri sonucunda 62.547 satır, 750.000 sütunluk bir veri matrisi elde edilmiştir. Böyle büyük bir veri seti ile analiz yapmak çok zor olacağı için öncelikle PCA yapılmaya çalışılmıştır. Ancak veri setinin büyüklüğü nedeniyle (120 GB) var olan donanımlarla bu işlem gerçekleştirilememiştir. Bu nedenle Google Cloud sunucularından 32 Core, 400 GB RAM belleğe sahip bir makine kiralama yoluna gidilmiştir. Bu makinaya Ubuntu işletim sistemi kurularak, analiz yapmak için gerekli Python kütüphaneleri kurulmuştur. Yine tek seferde PCA işleminde bellek sorunları ile karşılaşılmış ve IPCA yapılmasına karar verilmiştir. IPCA işlemi esnasında Python sklearn, h5py ve numpy kütüphaneleri kullanılmıştır. Çeşitli denemeler sonucunda veri 900 öznitelikle ifade edilebilmiştir. Bu 900 özniteliğin toplam veriyi açıklama oranı %95,72 olarak tespit edilmiştir.

4.1.4. Statik analiz derin sinir ağı

Elde edilen 900 öznitelik kurulan bir DNN ile modellenmiştir. DNN modeli Python kullanılarak oluşturulmuş ve sklearn, numpy, keras pandas kütüphaneleri kullanılmıştır. Modelleme sırasında verilerin rastgele %80'i eğitim, %20'si test için ayrılmıştır. Kurulan DNN modeli bir adet giriş katmanı, 2 adet gizli katman ve 1 adet çıkış katmanı içermektedir. Gizli katmanlar 6 adet düğümden oluşmaktadır. Modelin giriş katmanında ve gizli katmanlarında Rectifier Linear Units (RELU) aktivasyon fonksiyonu, çıkış katmanında ise sigmoid aktivasyon fonksiyonu kullanılmıştır. Kernel_initializer değeri uniform olarak tanımlanarak, başlangıç ağırlıkları uniform dağılım olarak belirlenmiştir. Modelin eğitiminden önce öğrenmenin yapılandırılması compile adı verilen fonksiyon yardımı ile yapılmıştır. Bu fonksiyonun öğrenme hızının ayarlanması için gerekli fonksiyonun belirlenmesi amacıyla optimizer değeri olarak adam, loss değeri model ikili sınıflandırmaya sahip olduğundan dolayı binary_crossentropy, model performansı değerlendirme amacı ile kullanılacak fonksiyonu belirlemek amacı ile metrics değeri accuracy olarak ayarlanmıştır. Son aşamada modelin eğitimi için fit fonksiyonu kullanılmıştır. Model eğitimi için fit fonksiyonuna modelin kaç defa çalışması gerektiğini belirten epoch değeri 10 olarak tanımlanmış ve

verilerin kaçarlı gruplar halinde alınacağını ayarlayan batch_size parametresi 20 olarak verilmiştir. Batch_size ve epoch değerleri modelin çalıştırılması sırasında yapılan denemeler sonucunda belirlenmiştir. Belirlenen epoch değeri ve batch_size değerinin üstündeki rakamlarda, doğruluk değerinin artmadığı gözlemlendiğinden dolayı epoch değeri 10, batch_size değeri 20 olarak tanımlanmıştır.

Kurulan model sonucunda; eğitim seti doğruluk oranı 0,9974, test seti doğruluk oranı 0,9938 olarak elde edilmiştir. Modelin eğitim seti için elde edilen başarı kriterleri Çizelge 4.2'de, test seti için elde edilen başarı kriterleri Çizelge 4.3'te gösterilmiştir.

Çizelge 4.2 Statik analiz eğitim seti, başarım sonuçları

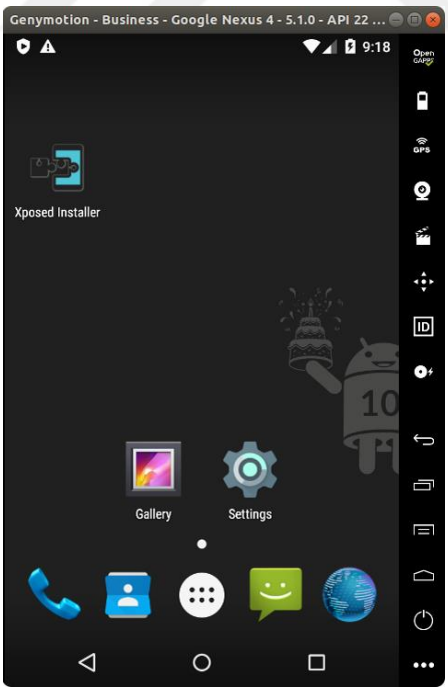
	Kesinlik (Precision)	Duyarlılık (Recall)	F1-skoru (F1-score)	Destek (Support)
Zararsız uygulama	0,9992	0,9965	0,9979	30.505
Zararlı uygulama	0,9946	0,9988	0,9967	19.532
Makro ortalama	0,9969	0,9977	0,9973	50.037
Ağırlıklı ortalama	0,9974	0,9974	0,9974	50.037
ROC AUC				**0,9976**
Doğruluk				**0,9974**

Çizelge 4.3 Statik analiz test seti, başarım sonuçları

	Kesinlik (Precision)	Duyarlılık (Recall)	F1-skoru (F1-score)	Destek (Support)
Zararsız uygulama	0,9963	0,9935	0,9949	7539
Zararlı uygulama	0,9902	0,9944	0,9923	4971
Makro ortalama	0,9932	0,9939	0,9936	12510
Ağırlıklı ortalama	0,9939	0,9938	0,9938	12510
ROC AUC				**0,9939**
Doğruluk				**0,9938**

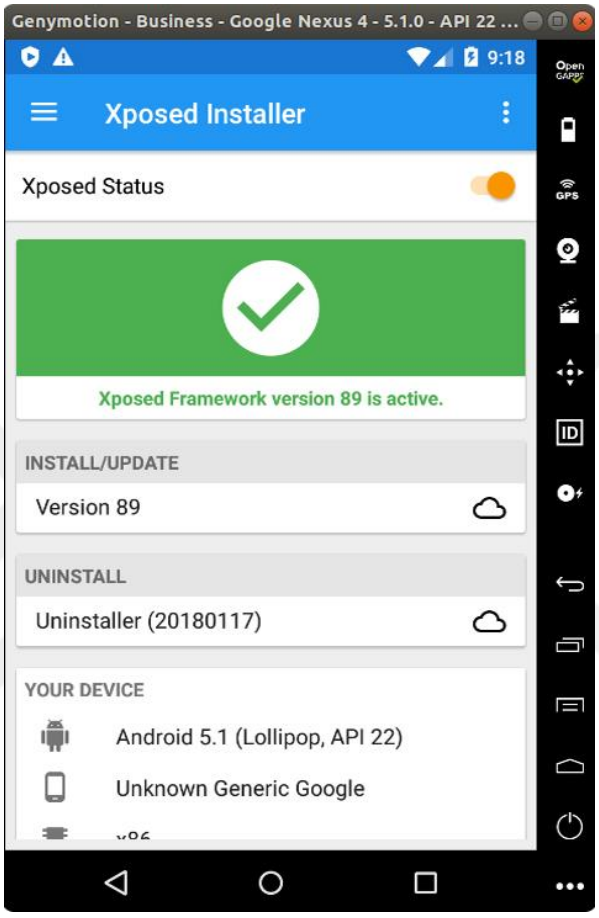
4.2. Hibrit Analiz Uygulaması

Hibrit analiz yöntemi; Android uygulamalarının statik analiz yöntemi ve dinamik analiz yönteminin birlikte uygulanması sonucunda gerçekleştirilen bir yöntemdir. Hibrit analiz yönteminin dinamik analiz aşaması Android uygulamalarının gerçek bir cihazda ya da sanal bir cihazda çalıştırılarak gerekli bilgilerin alınması ve uygulamanın davranışlarının belirlenmesi esasına dayanmaktadır. Zararlı uygulamaların gerçek cihazda verebileceği zararlar göz önüne alındığında, dinamik analizin gerçek ortamdan izole edilmiş sanal bir ortamda gerçekleştirilmesi daha uygun görünmektedir. Bu amaçla bu tez çalışmanın dinamik analiz kısmının gerçekleştirilmesi için Genymotion tarafından geliştirilen sanal Android emülatörü kullanılmıştır. Veri seti içindeki uygulamaların analizi sonucunda minimum sdk, maximum sdk, target sdk özellikleri dikkate alınarak uygulamaların çoğunun çalıştırılması hedeflenerek Android 5.1 Lollipop API 22 versiyonu kullanılmıştır. Kullanılan emülator Şekil 4.19'da gösterilmiştir.



Şekil 4.19. Android emülatör

Android uygulamaların çalışması esnasında kancalanacak metotların yakalanabilmesi için Xposed uygulama çatısı kullanılmış ve Şekil 4.20‘ de gösterilmiştir. Xposed installer 3.1.5 versiyonu ile birlikte Xposed Framework 89 versiyonu kullanılmıştır.

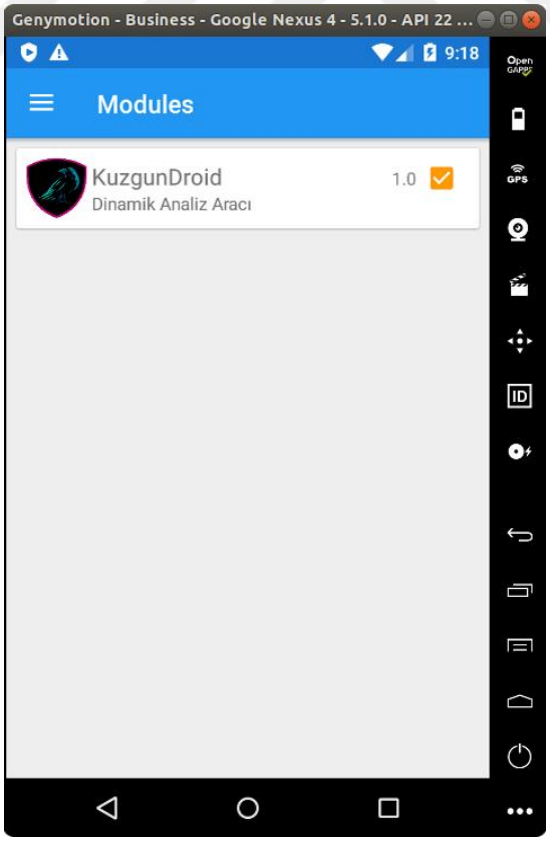


Şekil 4.20. Android emülatör ve xposed uygulama çatısı

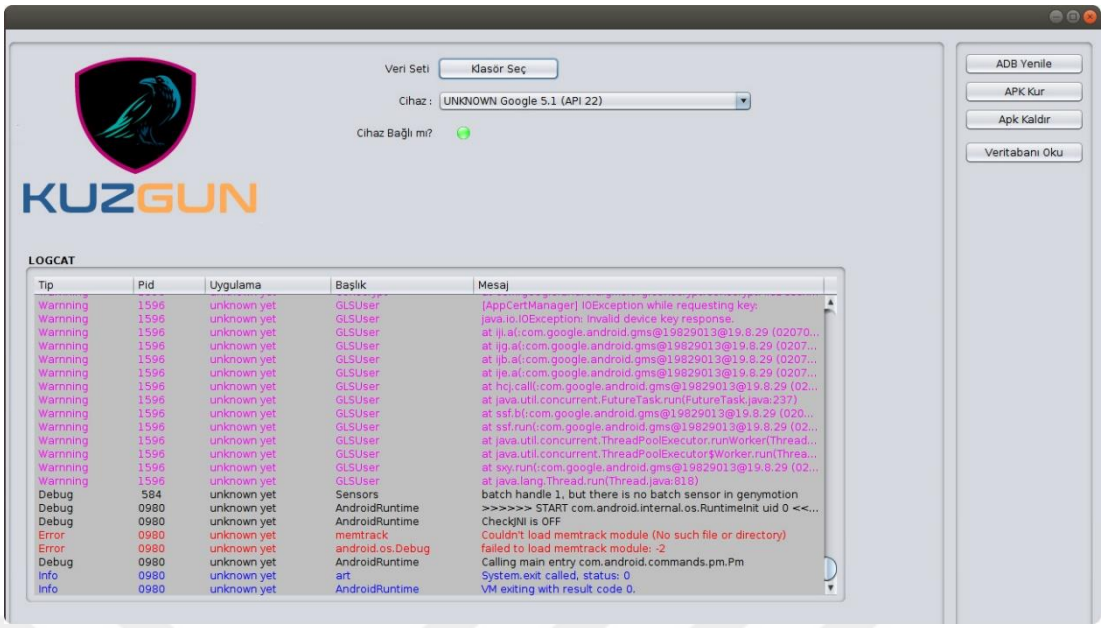
Xposed uygulama çatısı kullanabilmek için Android uygulaması geliştirilirken manifest dosyasında Xposed için gerekli tanımlamalar yapılmalıdır. Şekil 4.21‘de olduğu gibi manifest dosyasında gereken tanımlamalar uygulamamızda; “xposedmodule” true olarak yapılmış, “xposedminversion” 42 ve üstü olarak tanımlanmış, “xposeddescription” “Dinamik Analiz Aracı” olarak etiketlenmiştir. Uygulama çalıştırıldığında bu tanımlamalara bağlı olarak Xposed uygulama çatısına entegre olarak Şekil 4.22‘ de olduğu gibi modül olarak yüklenecek ve geliştirilen uygulamada istenen metotları ve fonksiyonları kancalayabilecek duruma gelecektir.

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.dinamikanaliz.kuzgundroid"
    android:versionCode="1"
    android:versionName="1.0" >

    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS"/>


    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_iauncher"
        android:label="KuzgunDroid"
        android:theme="@style/AppTheme" >
        <meta-data
            android:name="xposedmodule"
            android:value="true" />
        <meta-data
            android:name="xposedminversion"
            android:value="42+" />
        <meta-data
            android:name="xposeddescription"
            android:value="Dinamik Analiz Aracı" />
    </application>

</manifest>
```

Şekil 4.21. KuzgunDroid manifest dosyası



Şekil 4.22. KuzgunDroid modülü



Şekil 4.23. Kuzgun PC uygulaması

Şekil 4.23'te gösterilen Kuzgun PC uygulaması ile zararlı ve zararlı olmayan uygulamaları içeren klasör yolu belirtilerek uygulama çalıştırıldığında statik olarak uygulamanın izinleri, uygulama bileşenleri ve dinamik olarak kancalanan API isimleri alınarak veri tabanına kayıt işlemleri otomatik olarak yapılmaktadır. Emülatör ile PC arasındaki iletişim ADB komutları ile yapılmaktadır. Adb "install" komutu ile uygulama emülatörde kurulduktan sonra "am start" komutu ile çalıştırılıp, sistemden uygulamanın çalıştığına dair log mesajı alındıktan sonra 1 dakika boyunca uygulamanın çalışması takip edilmektedir. Uygulamaların emülatörde çalışma süresi olan 1 dakika belirlenirken; veri seti içinden seçilen bazı uygulamalar ile testler yapılmıştır. Yapılan denemeler sonucunda kullanılan kanca listesi için 1 dakikanın yeterli olduğu sonucuna varılmıştır. Bu 1 dakika içinde "logcat" komutu ile log kayıtları okunarak kurulan uygulama hakkında API çağrıları alınarak veri tabanına kayıt edilmektedir. Kurulum esnasında veya çalıştırılma esnasında hata ile karşılaşılmışsa bu uygulama ile ilgili kayıt gerçekleştirilmemiştir. Bu hatalar genel olarak Android sürümlerinde getirilen değişikliklerden veya uygulamada yapılan tanımlardan kaynaklanmaktadır. Örneğin Android 4 sürümünden sonra ART kavramını getirmiş ve daha önceki sürümlerde yazılan uygulamalarda çalışma garantisi vermemiştir. Bunun yanı sıra bazı API çağrılarının kaldırmış veya izin düzeylerinin değiştirilmiş

olmasından dolayı bazı uygulamalar kullandığımız emülatörde çalışmamaktadır. Hata alınmadan 1 dakika boyunca çalıştıktan sonra uygulama “shell am force-stop” komutu ile durdurulup “pm uninstall” komutu kullanılarak kaldırılmış ve "pm clear" komutu ile uygulama bilgileri kaldırılmıştır.

4.2.1. Hibrit analiz veri seti

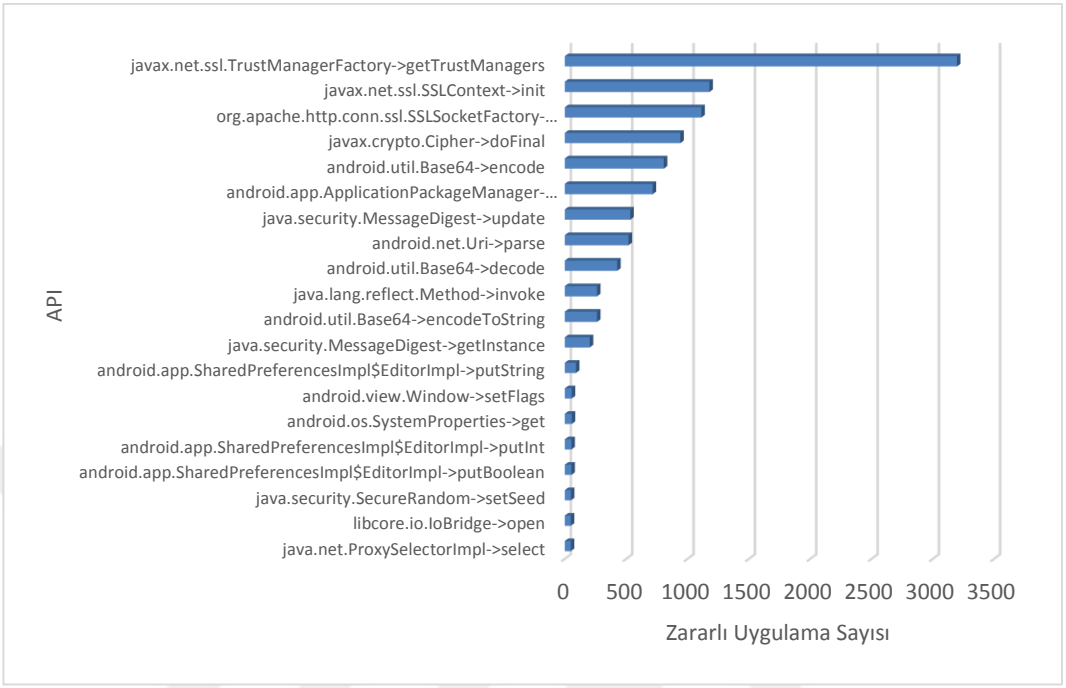
Çalışmanın hibrit analiz safhasında statik analiz aşamasında kullanılan veri seti kullanılmıştır. Veri setinde kullanılan uygulamalar dinamik analize tabi tutulurken Android sürümlerinde getirilen yeniliklerden dolayı bazı uygulamalarda çalışma hatası alınmıştır. Analizlerin tutarlılığının sağlanması açısından hata alınan uygulamalar veri setinden çıkarılmıştır. Toplamda 35.142 Android uygulaması hata alınmadan analiz edilmiş bunlardan 14.205 adedi zararlı uygulama, 20.937 zararlı olmayan uygulamadan oluşmaktadır.

4.2.2. Hibrit analiz öznitelikleri

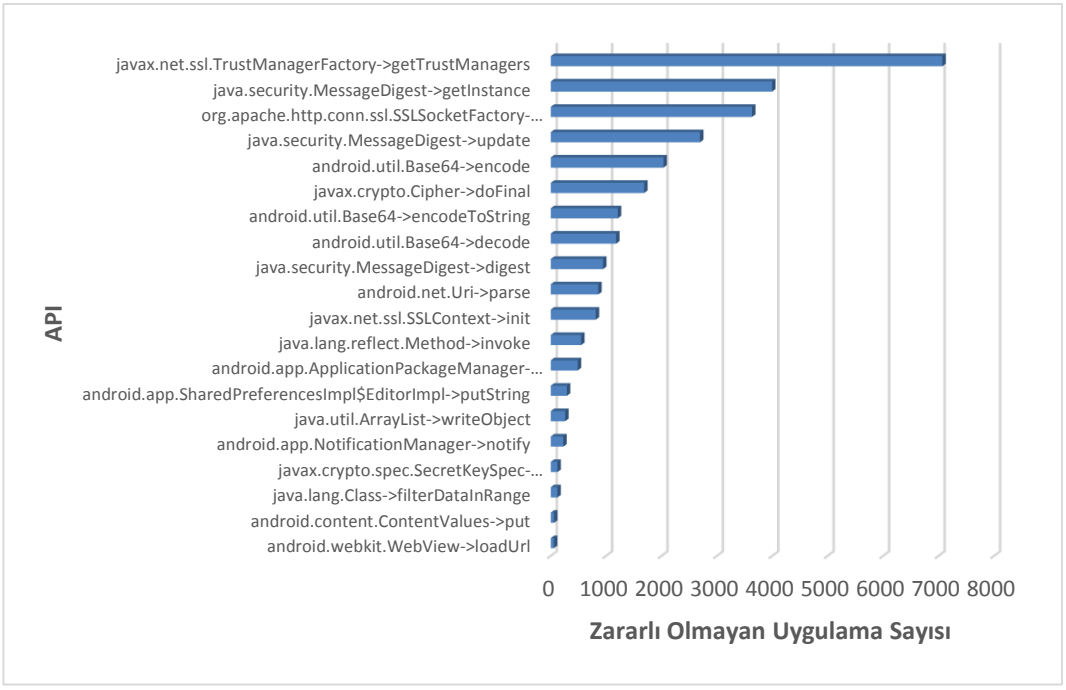
Hibrit analizde, dinamik olarak Android uygulamalarının çalıştırılması sonucu kancalanan API çağrıları kullanılmıştır. Kancalama için seçilen API listesi çalışmanın ekler kısmında verilmiştir. Bunun yanında statik olarak elde edilen izinler, Android bileşenleri dinamik analiz yönteminde öznitelik olarak kullanılmıştır. Şekil 4.24’te zararlı uygulamalardan kancalanan API çağrıları, Şekil 4.25’te zararlı olmayan uygulamalardan kancalanan API çağrıları, Şekil 4.26’te zararlı uygulamalarda istenen izinler, Şekil 4.27’te zararlı olmayan uygulamalarda istenen izinler, Şekil 4.28’de zararlı uygulamalarda kullanılan Android bileşenleri, Şekil 4.29’da zararlı olmayan uygulamalarda kullanılan Android bileşenleri gösterilmiştir.

Şekil 4.24 ve Şekil 4.25’te dinamik olarak elde edilen API çağrıları grafik olarak gösterilmiştir. Zararlı uygulamalarda şifreleme metotlarının zararlı olmayan uygulamalara göre daha sıklıkla kullanıldığı gözlemlenmektedir.

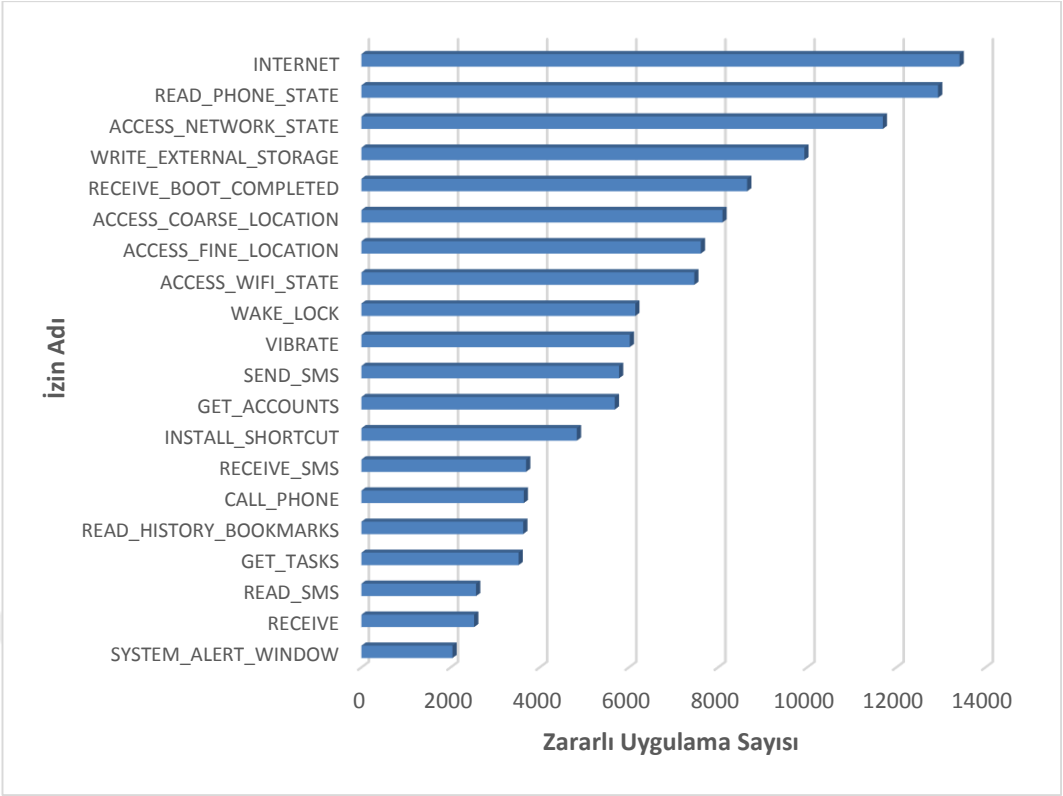
Statik olarak elde edilen verilerin paylaşıldığı takip eden grafikler çalışmanın statik analiz kısmında elde edilen veriler ile paralellik göstermektedir.



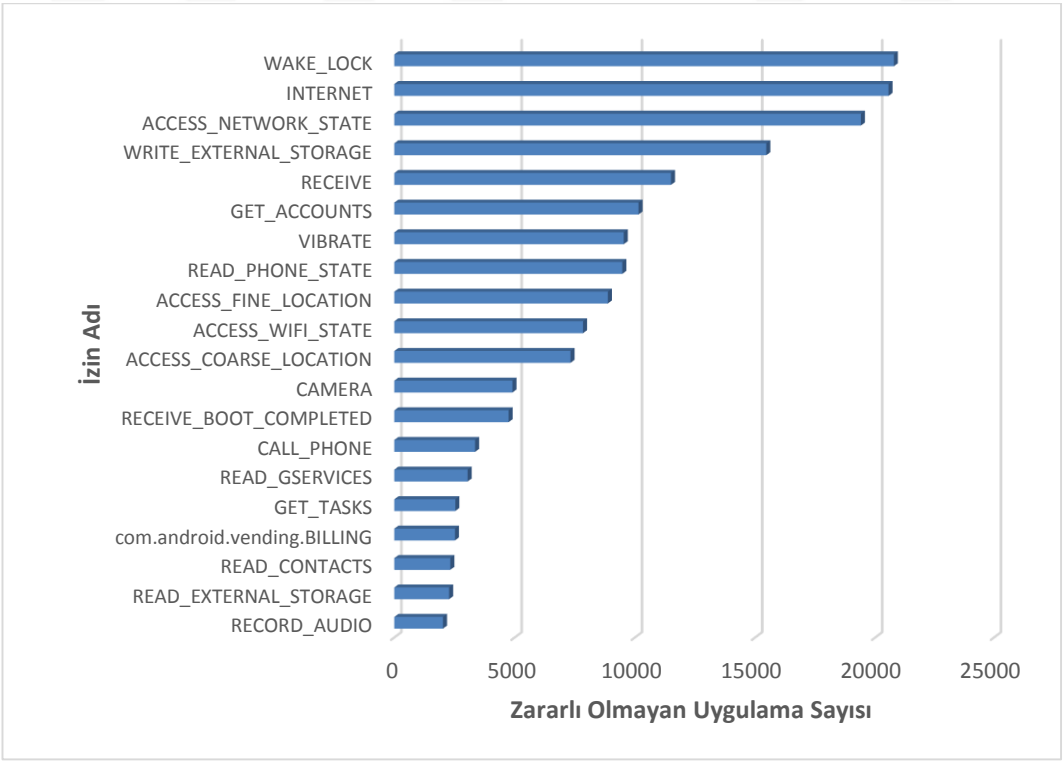
Şekil 4.24. Zararlı uygulamalardan kancalanan API çağrıları



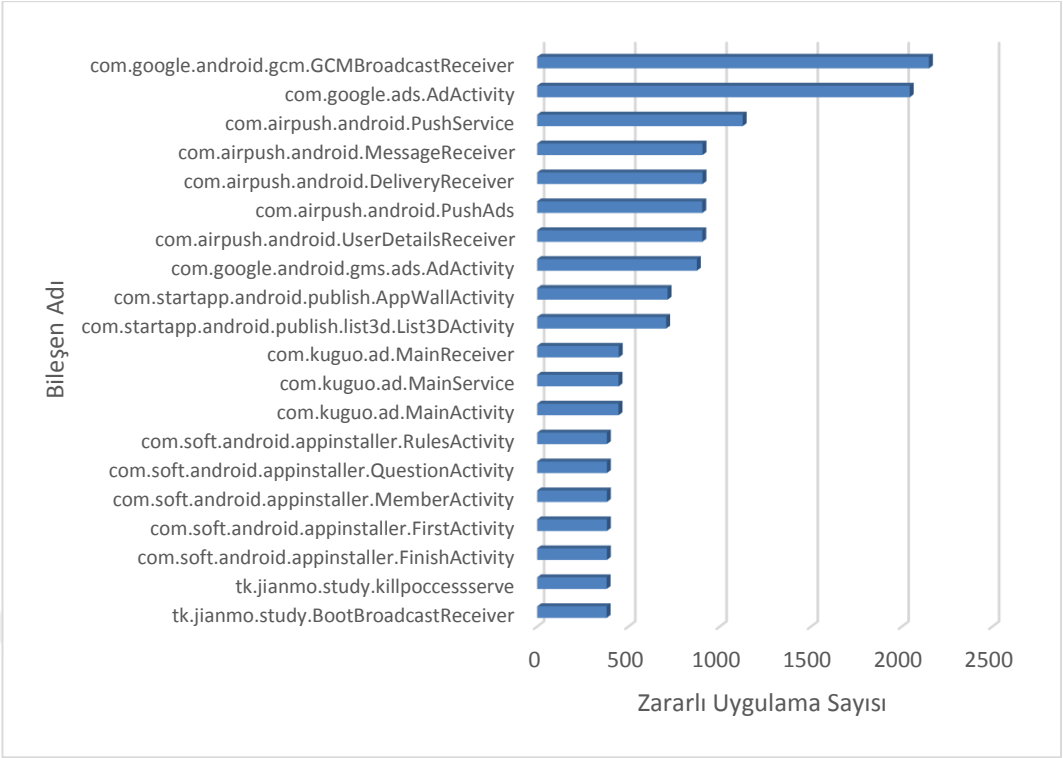
Şekil 4.25. Zararlı olmayan uygulamalardan kancalanan API çağrıları



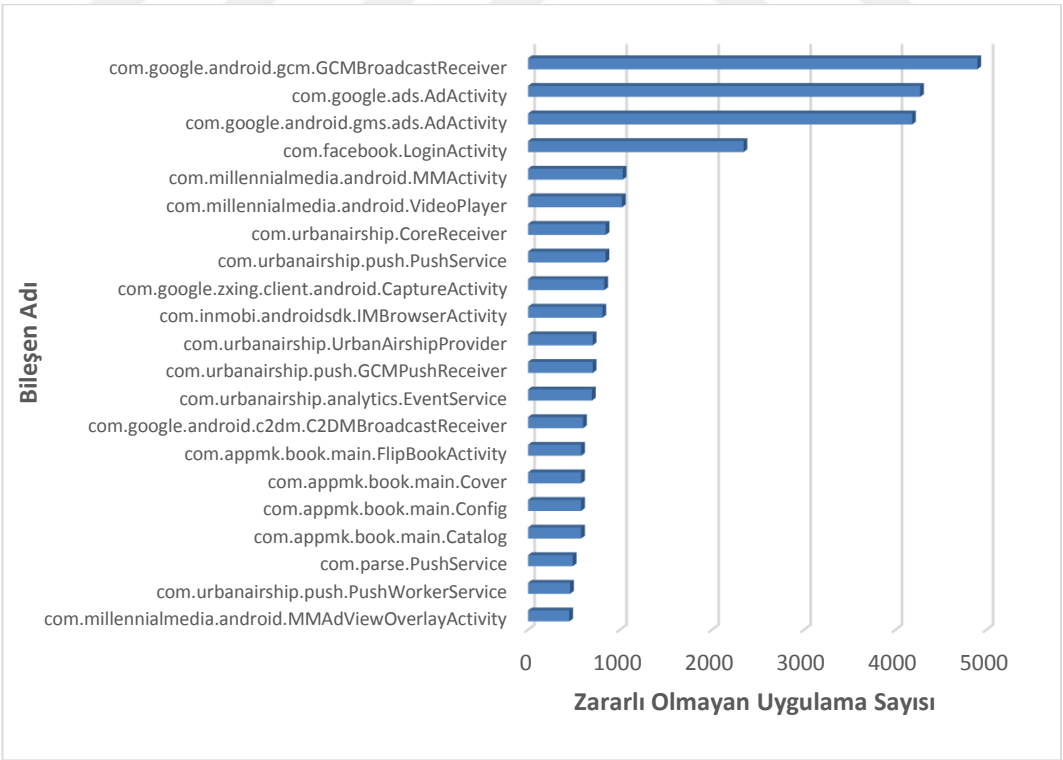
Şekil 4.26. Zararlı uygulamalarda istenen izinler



Şekil 4.27. Zararlı olmayan uygulamalarda istenen izinler



Şekil 4.28. Zararlı uygulamalardan elde edilen Android bileşenleri



Şekil 4.29. Zararlı olmayan uygulamalarda elde edilen Android bileşenleri

4.2.3. Hibrit analiz artırımlı PCA

Hibrit analiz işlemleri sonucunda 35.142 satır, 375.000 sütunluk bir veri matrisi elde edilmiştir. Böyle büyük bir veri seti ile analiz yapmak çok zor olacağı için öncelikle PCA yapılmaya çalışılmıştır ancak veri setinin büyüklüğü nedeniyle (40 GB) var olan donanımlarla bu işlem gerçekleştirilememiştir. Bu nedenle Google Cloud sunucularından 32 Core, 240 GB RAM belleğe sahip bir makine kiralama yoluna gidilmiştir. Bu makinaya Ubuntu işletim sistemi kurularak, analiz yapmak için gerekli Python kütüphaneleri kurulmuştur. Yine tek seferde PCA işleminde bellek sorunları ile karşılaşılmış ve IPCA yapılmasına karar verilmiştir. IPCA işlemi esnasında Python sklearn, h5py ve numpy kütüphaneleri kullanılmıştır. Çeşitli denemeler sonucunda veri 5.000 öznitelikle ifade edilebilmiştir. Bu 5.000 özniteliğin toplam veriyi açıklama oranı %92 olarak tespit edilmiştir.

4.2.4. Hibrit analiz derin sinir ağı

Elde edilen 5.000 öznitelik kurulan bir DNN ile modellenmiştir. Modelleme sırasında verilerin rastgele %80'i eğitim, %20'si test için ayrılmıştır. Kurulan DNN modeli bir adet giriş katmanı, 2 adet gizli katman ve 1 adet çıkış katmanı içermektedir. Gizli katmanlar 6 adet düğümden oluşmaktadır. Modelin giriş katmanında ve gizli katmanlarında Rectifier Linear Units (RELU) aktivasyon fonksiyonu, çıkış katmanında ise sigmoid aktivasyon fonksiyonu kullanılmıştır. Kernel_initializer değeri uniform olarak tanımlanarak, başlangıç ağırlıkları uniform dağılım olarak belirlenmiştir. Modelin eğitiminden önce öğrenmenin yapılandırılması compile adı verilen fonksiyon yardımı ile yapılmıştır. Bu fonksiyonun öğrenme hızının ayarlanması için gerekli fonksiyonun belirlenmesi amacıyla optimizer değeri olarak adam, loss değeri model ikili sınıflandırmaya sahip olduğundan dolayı binary_crossentropy, model performansı değerlendirme amacı ile kullanılacak fonksiyonu belirlemek amacı ile metrics değeri accuracy olarak ayarlanmıştır. Son aşamada modelin eğitimi için fit fonksiyonu kullanılmıştır. Model eğitimi için fit fonksiyonuna modelin kaç defa çalışması gerektiğini belirten epoch değeri 10 olarak tanımlanmış ve verilerin kaçarlı gruplar halinde alınacağını ayarlayan batch_size parametresi 20 olarak verilmiştir. Batch_size ve epoch değerleri modelin çalıştırılması sırasında yapılan

denemeler sonucunda belirlenmiştir. Belirlenen epoch değeri ve batch_size değerinin üstündeki rakamlarda, doğruluk değerinin artmadığı gözlemlendiğinden dolayı epoch değeri 10, batch_size değeri 20 olarak tanımlanmıştır.

Kurulan model sonucunda; eğitim seti doğruluk oranı 0,9943, test seti doğruluk oranı 0,9694 olarak elde edilmiştir. Modelin eğitim seti için elde edilen başarı kriterleri Çizelge 4.4'te, test seti için elde edilen başarı kriterleri Çizelge 4.5'te gösterilmiştir.

Çizelge 4.4 Hibrit analiz eğitim seti, başarım sonuçları

	Kesinlik (Precision)	Duyarlılık (Recall)	F1-skoru (F1-score)	Destek (Support)
Zararsız uygulama	0,9949	0,9957	0,9953	16.901
Zararlı uygulama	0,9936	0,9922	0,9929	11.212
Makro ortalama	0,9942	0,994	0,9941	28.113
Ağırlıklı ortalama	0,9943	0,9943	0,9943	28.113
ROC AUC				**0,9939**
Doğruluk				**0,9943**

Çizelge 4.5 Hibrit analiz test seti, başarım sonuçları

	Kesinlik (Precision)	Duyarlılık (Recall)	F1-skoru (F1-score)	Destek (Support)
Zararsız uygulama	0,968	0,9822	0,9751	4278
Zararlı uygulama	0,9717	0,9495	0,9605	2751
Makro ortalama	0,9699	0,9659	0,9678	7029
Ağırlıklı ortalama	0,9694	0,9694	0,9693	7029
ROC AUC				**0,9658**
Doğruluk				**0,9694**

5. TARTIŞMA VE SONUÇLAR

Android zararlı yazılımlarının kullanıcıları tehdit ettiği günümüz teknoloji dünyasında güvenlik tedbirlerinin alınması, zararlı yazılımların tespit edilerek verebilecekleri zararların önlenmesi önemli bir çalışma sahası olarak karşımıza çıkmaktadır. Bu önemli konu çerçevesinde yapılan tez çalışmasında; Android işletim sistemini hedef alan zararlı uygulamaların tespiti hedeflenmiştir.

Android zararlı yazılımların tespiti için statik analiz yöntemi, statik analiz ve dinamik analiz yönteminin birlikte kullanıldığı hibrit analiz yöntemi uygulanmıştır. Literatürdeki diğer uygulamalara bakıldığında nitelik ve nicelik açısından büyük veri olarak adlandırılabilecek bir veri seti kullanılmıştır. Bu veri seti üzerinde ayrıntılı bir şekilde diğer çalışmalarda olmayan öznitelik vektörü oluşturulmuştur.

Statik analiz yönteminin uygulanmasında Java programlama dili kullanılarak Kuzgun adında geliştirilen yazılımla uygulamalar çalıştırılmadan, 9 ayrı kategoride belirlenen çok geniş bir yelpazede öznitelik çıkarımı yapılmıştır. 9 ayrı kategoride çıkarılan özniteliklerin biri veya birkaç tanesi diğer çalışmalarda kullanılmış olsa da bu özniteliklerin tümünün, bir arada kullanılması açısından yapılan çalışma, literatürde ilk olma özelliği taşımaktadır. Çalışmanın veri seti bölümlerinde verildiği üzere elde edilen Android uygulamalarının çok çeşitli olması ve daha fazla zararlı uygulamanın tespiti hedeflendiği için öznitelik kategori sayısı zararlı varyantlarının davranışlarına yönelik olarak geniş bir açıdan incelenmiştir. Böylesine geniş öznitelik incelemesi, beraberinde analiz zorlukları da getirmiştir. Şöyle ki çıkarılan öznitelik kategorisi sayısal olarak fazla olduğundan doğal olarak elde edilen toplam öznitelik sayısı da fazla olmuştur. Toplamda özniteliklerin hafızada kapladığı alan da boyut olarak mevcut kullanılan donanımla analiz yapılmasına engel teşkil etmiştir. Bu amaçla Google Cloud sunucusu kiralama yöntemine başvurularak, öncelikle öznitelikler üzerinde IPCA yöntemi ile boyut indirgeme yöntemi kullanılmıştır. Yaklaşık olarak 750.000 öznitelikten oluşan veri seti IPCA yöntemi ile 900 öznitelikle temsil edilebilmiştir. IPCA yöntemi ile indirgenen bu 900 öznitelik %95,72

oranında temsil oranına sahip olmuştur. IPCA yönteminde elde edilen öznitelikler kurulan DNN modelinin girdilerini oluşturmuştur. Kurulan DNN modeli 900 düğümden oluşan giriş katmanı 6 adet düğümden oluşan birinci gizli katman, 6 düğümden oluşan ikinci gizli katman ve 1 adet çıkış katmanını barındırmaktadır. Kurulan model sonucu gerçekleştirilen eğitimle, eğitim setinde %99,74 doğruluk oranı, %99,73 F1 skoru, %99,69 kesinlik, %99,77 duyarlılık değerleri elde edilmiştir. Test veri setinde ise; %99,38 doğruluk oranı, %99,36 F1 skoru, %99,32 kesinlik, %99,39 duyarlılık değerleri elde edilmiştir.

Çalışmanın hibrit analiz kısmında statik analizden farklı olarak uygulamalar çalıştırılarak uygulamaların davranışlarının takibi esas alınmış ve çalıştırılma esnasında kancalanan API çağrıları, statik analizle elde edilen özelliklere eklenmiştir. Dinamik analiz olarak adlandırılan, Android uygulamalarının sanal bir cihazda veya gerçek bir cihazda çalıştırılması esasına dayanan yöntemin uygulanması adına KuzgunDroid adını verdiğimiz bir Android uygulaması geliştirilmiş ve stratejik öneme sahip API çağrıları kancalama metodu ile elde edilmiştir. Aynı zamanda Android cihaz ile iletişimi sağlayan, uygulamaları emülatöre gönderen, çalıştıran, sonuçları takip eden ve emülatöre kurulan uygulamayı kaldıran yine Kuzgun adında bir Java uygulaması geliştirilmiştir. Kuzgun PC uygulaması, Android emülatörden elde edilen API çağrıları ile statik olarak elde edilen öznitelikleri birleştirerek Mysql veri tabanına kaydını sağlama özelliklerinin yanı sıra bu öznitelikleri veri tabanından okuyarak DNN modeli için gerekli formatta kayıt edebilme özelliğine de sahip olarak geliştirilmiştir. API çağrılarının elde edilmesi literatürde ki çalışmalarda ifade edildiği gibi zorlu ve zahmetli bir süreç ihtiva etmektedir. Çünkü her bir Android uygulaması emülatöre kurulduktan sonra belirli bir süre çalıştırılmalıdır ki uygulamanın davranışları takip edilebilsin. Bu tez çalışmasında olduğu gibi veri sayısının çok olduğu bir çalışmada bu durum hem zaman hem de çalışılan platform açısından büyük öz veri gerektirmektedir. Mevcut Android sanal cihazlarının dezavantajlı olması nedeniyle Genymotion Android cihazı lisansı olarak alınarak çalışmada kullanılmıştır. Bu sayede geliştirdiğimiz Kuzgun uygulamalarının derlenmesi ve denenmesi esnasında kolaylık sağlanmıştır. Dinamik analiz yapılırken karşılaşılan diğer dezavantaj ise toplamda analizin gerçekleştirilmesi için

harcanan süre olarak karşımıza çıkmaktadır. Literatür ışığında her uygulama 1 dakika emülatörde çalıştırılmıştır. Veri seti sayısı dikkate alındığında analiz için harcanan süre yaklaşık 25 gündür. Bu zorluklar dikkate alındığında diğer çalışmalar veri setlerini genel olarak sayısal anlamda kısıtlı tutmuşlardır. Ancak bu çalışmada zaman ve diğer zorluk kısıtları dikkate alınmadan daha fazla zararlı uygulama ailesinin tespiti için bu zorluklar göz ardı edilmiştir. Hibrit analiz yöntemi sonucunda 35.142 uygulamadan elde edilen 375.000 öznitelik vektörü elde edilmiştir. Statik analizde olduğu gibi mevcut donanımlarımız bu veriyi analiz etmede yetersiz kalmıştır. Bu amaçla Google Cloud sunucusu kiralama yöntemine başvurularak, öncelikle öznitelikler üzerinde IPCA yöntemi ile boyut indirgeme yöntemi kullanılmıştır. Yaklaşık olarak 375.000 öznitelikten oluşan veri seti IPCA yöntemi ile 5.000 öznitelikle temsil edilebilmiştir. IPCA yöntemi ile indirgenen bu 5.000 öznitelik %92 oranında temsil oranına sahip olmuştur. IPCA yönteminde elde edilen öznitelikler kurulan DNN modelinin girdilerini oluşturmuştur. Kurulan DNN modeli 5.000 düğümden oluşan giriş katmanı 6 adet düğümden oluşan birinci gizli katman, 6 düğümden oluşan ikinci gizli katman ve 1 adet çıkış katmanını barındırmaktadır. Kurulan model sonucu gerçekleştirilen eğitimle, eğitim setinde %99,43 doğruluk oranı, %99,41 F1 skoru, %99,42 kesinlik, %99,4 duyarlılık değerleri elde edilmiştir. Test veri setinde ise; %96,94 doğruluk oranı, %96,78 F1 skoru, %96,99 kesinlik, %96,59 duyarlılık değerleri elde edilmiştir.

Çizelge 5.1. Literatürdeki bazı DL çalışmalarının elde ettiği sonuçlar (Alzaylaee vd., 2020)

Çalışmalar	**Analiz Yöntemi**	**Zararsız Uyg. Sayısı**	**Zararlı Uyg. Sayısı**	**Doğruluk**	**Kesinlik**	**Duyarlılık**	**F1-skoru**
DroidDetector Yuan vd. (2016)	Statik	880	880	89,03	90,39	89,04	89,76
DroidDetector Yuan vd. (2016)	Dinamik	880	880	71,25	72,59	71,25	71,92
DroidDetector Yuan vd. (2016)	Statik & Dinamik	880	880	96,76	96,78	96,76	96,76
CNN McLaughlin vd. (2017)	Statik	863	1.260	98	99	95	97
MalDozer Karbab vd. (2018)	Statik	37.627	20.089	-	96,29	96,29	96,29

Deep4MalDroid Hou vd. (2016)	Dinamik	1.500	1.500	93,68	93,96	93,36	93,68
AutoDroid Hou vd. (2017)	Statik	2.500	2.500	96,66	96,55	96,76	96,66
Ddefender Alshahrani vd. (2018)	Statik & Dinamik	2.104	2.104	95,13	-	-	95,45
DL-Droid Alzaylaee vd. (2020)	Dinamik	19.620	11.505	94,95	94,08	97,78	95,89
DL-Droid Alzaylaee vd. (2020)	Statik & Dinamik	19.620	11.505	95,42	95,31	97,19	96,24

Çizelge 5.2. Kuzgun ile elde edilen sonuçlar

Tez Çalışması	**Analiz Yöntemi**	**Zararsız Uyg. Sayısı**	**Zararlı Uyg. Sayısı**	**Doğruluk**	**Kesinlik**	**Duyarlılık**	**F1-skoru**
KuzgunDroid	Statik & Dinamik	20.937	14.205	96,94	96,99	96,59	96,93
Kuzgun	Statik	38.044	24.503	99,38	99,32	99,39	99,36

Literatürdeki bazı DL çalışmalarının elde ettiği sonuçlar Çizelge 5.1'de verilmiştir. Kuzgun ile yapılan statik analiz ve hibrit analiz sonucu elde edilen başarı kriterleri Çizelge 5.2'de verilmiştir. Bu yüzde oranlara bakıldığında çalışmada seçilen özniteliklerin ve kancalanan API çağrılarının isabetli olduğu kanaatine varılmıştır. Ancak dinamik analiz için araştırılan kancalama metot ve fonksiyonlarının daha ayrıntılı çalışılarak buradan elde edilecek tespit başarımının daha yukarıya çekilebilmesinin hedeflenebileceği sonucuna ulaşılmıştır.

KAYNAKLAR

Aafer, Y., Du, W., Yin, H., 2013. Droidapiminer: Mining Api-Level Features For Robust Malware Detection In Android. International Conference on Security and Privacy in Communication Systems, 25-28 September, Sydney, NSW, Australia, 86-103.

Alshahrani, H., Mansourt, H., Thorn, S., Alshehri, A., Alzahrani, A., Fu, H., 2018. Ddefender: Android Application Threat Detection Using Static and Dynamic Analysis. 2018 IEEE International Conference on Consumer Electronics (ICCE), 12-14 January, Las Vegas, USA, 1-6.

Alzahrani, A.D.A., 2019. Intelligent Behavior-based Ransomware Detection System for Android Platform. Oakland University, 174s.

Alzaylaee, M.K., Yerima, S.Y., Sezer, S., 2016. Dynalog: An Automated Dynamic Analysis Framework for Characterizing Android Applications. 2016 International Conference on Cyber Security and Protection Of Digital Services (Cyber Security), 13-14 June, London, United Kingdom, 1-8.

Alzaylaee, M.K., Yerima, S.Y., Sezer, S., 2020. DL-Droid: Deep Learning Based Android Malware Detection Using Real Devices. Computers & Security, 89, 101663.

AMD. 2018. Android Malware Dataset. Erişim Tarihi: 10.05.2018. http://amd.arguslab.org/.

Anagnostopoulos, M., Kambourakis, G., Gritzalis, S., 2016. New Facets of Mobile Botnet: Architecture and Evaluation. International Journal of Information Security, 15(5), 455-473.

Android Wake Lock Research. 2018. Erişim Tarihi: 06.05.2018. http://sccpu2.cse.ust.hk/elite/downloadApks.html.

Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K., Siemens, C., 2014. Drebin: Effective and Explainable Detection of Android Malware in Your Pocket. Ndss, 23-26.

Arshad, S., Shah, M.A., Khan, A., Ahmed, M., 2016. Android Malware Detection & Protection: A Survey. International Journal of Advanced Computer Science and Applications, 7(2), 463-475.

Au, K.W.Y., Zhou, Y.F., Huang, Z., Lie, D., 2012. Pscout: Analyzing the Android Permission Specification. Proceedings of the 2012 ACM Conference on Computer and Communications Security, 16-18 October Raleigh North Carolina, USA, 217-228.

Aydın, A., 2019. Hibrit Analiz Yöntemlerini Kullanarak Makine Öğrenmesi Yardımıyla Android Kötücül Yazılımların Tespit Edilmesi. Gazi

Üniversitesi, Fen Bilimleri Enstitüsü, Bilgisayar Mühendisliği Yüksek Lisans Tezi, 93s, Ankara.

Aygün, R.C., 2017. Derin Öğrenme Yöntemleri İle Bilgisayar Ağlarında Güvenliğe Yönelik Anormallik Tespiti. Yıldız Teknik Üniversitesi, Fen Bilimleri Enstitüsü, Bilgisayar Mühendisliği Anabilim Dalı, Yüksek Lisans Tezi, 83s, İstanbul.

Barros, P., Parisi, G.I., Weber, C., Wermter, S., 2017. Emotion-Modulated Attention Improves Expression Recognition: A Deep Learning Model. Neurocomputing, 253, 104-114.

Bhandari, S., Gupta, R., Laxmi, V., Gaur, M.S., Zemmari, A., Anikeev, M., 2015. DRACO: DRoid Analyst Combo an Android Malware Analysis Framework. Proceedings of The 8th International Conference on Security of Information and Networks, 8-10 September, Sochi, Russia, 283-289.

Bhilvare, A., Manik, T., 2015. An Overview of Different Malware Analysis Techniques in Android. IJSRD - International Journal for Scientific Research & Development, 3(01).

Cordonsky, I., Rosenberg, I., Sicard, G., David, E.O., 2018. DeepOrigin: End-To-End Deep Learning for Detection of New Malware Families. 2018 International Joint Conference on Neural Networks (IJCNN), 8-13 July, Rio de Jenario, Brazil, 1-7.

Cui, Z., Xue, F., Cai, X., Cao, Y., Wang, G.-g., Chen, J., 2018. Detection of Malicious Code Variants Based on Deep Learning. IEEE Transactions on Industrial Informatics, 14(7), 3187-3196.

Davarcı, M.E., 2018. Hibrit Sınıflandırıcılar ile Android İşletim Sistemi İçin Zararlı Yazılım Tespiti. Gazi Üniversitesi, Fen Bilimleri Enstitüsü, Bilgisayar Mühendisliği, Yüksek Lisans Tezi, 81s, Ankara. .

Dubey, A., Misra, A., 2016. Android Security: Attacks and Defenses. CRC Press, 249s, New York.

Enck, W., Ongtang, M., McDaniel, P., 2009. Understanding Android Security. IEEE security & privacy, 7(1), 50-57.

Feizollah, A., Anuar, N.B., Salleh, R., Suarez-Tangil, G., Furnell, S., 2017. Androdialysis: Analysis of Android Intent Effectiveness in Malware Detection. Computers & Security, 65, 121-134.

Felt, A.P., Ha, E., Egelman, S., Haney, A., Chin, E., Wagner, D., 2012. Android Permissions: User Attention, Comprehension, and Behavior. Proceedings of the Eighth Symposium on Usable Privacy and Security, 11-13 July, Washington, D.C, USA, 3.

Fenton, C., 2018. Erişim Tarihi: 06.07.2018. https://github.com/CalebFenton/apkfile.

Fereidooni, H., Conti, M., Yao, D., Sperduti, A., 2016. ANASTASIA: Android Malware Detection Using Static Analysis of Applications. 2016 8th IFIP International Conference on New Technologies, Mobility and Security (NTMS), 21-23 November, Larnaca, Cyprus, 1-5.

Google. 2019a. Erişim Tarihi: 28.12.2019. https://developer.android.com/guide-/topics/permissions/overview.

Google. 2019b. Erişim Tarihi: 11.21.2019. https://developer.android.com/guide-/platform.

Google. 2019c. Erişim Tarihi: 02.12.2019. https://developer.android.com/-guide/topics/.

Google Play. 2018. Erişim Tarihi: 11.12.2018. https://www.android.com/play-protect/.

Hall, P.M., Marshall, A.D., Martin, R.R., 1998. Incremental Eigenanalysis for Classification. BMVC, 286-295.

Hou, S., Saas, A., Chen, L., Ye, Y., 2016. Deep4maldroid: A Deep Learning Framework for Android Malware Detection Based on Linux Kernel System Call Graphs. 2016 IEEE/WIC/ACM International Conference on Web Intelligence Workshops (WIW), 13-16 October, Omaha, NE, USA, 104-111.

Hou, S., Saas, A., Chen, L., Ye, Y., Bourlai, T., 2017. Deep Neural Networks for Automatic Android Malware Detection. Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2017, 31 July - 03 August, Sydney, Australia, 803-810.

Ibrahım, M.R., 2019. Derin Öğrenme Yöntemiyle Ağ Saldırı Tespit Sistemi. Selçuk Üniversitesi, Fen Bilimleri Enstitüsü, Bilgisayar Mühendisliği Anabilim Dalı, Yüksek Lisans Tezi, 74s, Konya.

IDC. 2019. International Data Corporation. Erişim Tarihi: 02.11.2019. https://www.idc.com/promo/smartphone-market-share/os.

Idrees, F., Rajarajan, M., Conti, M., Chen, T.M., Rahulamathavan, Y., 2017. PIndroid: A novel Android Malware Detection System Using Ensemble Learning Methods. Computers & Security, 68, 36-46.

Kabakuş, A.T., Doğru, İ.A., Çetin, A., 2015. Android Kötücül Yazılım Tespit Ve Koruma Sistemleri. Erciyes Üniversitesi Fen Bilimleri Enstitüsü Fen Bilimleri Dergisi, 31(1), 9-16.

Karbab, E.B., Debbabi, M., Derhab, A., Mouheb, D., 2018. MalDozer: Automatic framework for android malware detection using deep learning. Digital Investigation, 24, S48-S59.

Kaspersky. 2019. Erişim Tarihi: 04.12.2019. https://www.kaspersky.com.tr/resource-center/threats/computer-viruses-and-malware-facts-and-faqs.

Kiraz, Ö., 2017. Web Tabanlı Android Kötücül Yazılım Tespit Sistemi. Gazi Üniversitesi, Fen Bilimleri Enstitüsü, Bilgisayar Mühendisliği, Yüksek Lisans Tezi, 86s, Ankara.

Kolosnjaji, B., Zarras, A., Webster, G., Eckert, C., 2016. Deep Learning for Classification of Malware System Call Sequences. Australasian Joint Conference on Artificial Intelligence, 5-8 December, Hobart, Australia, 137-149.

Kulkarni, K., 2018. Android Malware Detection through Permission and App Component Analysis using Machine Learning Algorithms. University of Toledo, Computer Science and Engineering, M.Sc. Thesis, Toledo.

Liu, J., Yu, J., 2011. Research on Development of Android Applications. 2011 4th International Conference on Intelligent Networks and Intelligent Systems, 1-3 November, Kuming, China, 69-72.

Liu, Y., Xu, C., Cheung, S.-C., Terragni, V., 2016. Understanding and Detecting Wake Lock Misuses for Android Applications. Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 13-18 November, Seattle, WA, USA, 396-409.

McAfee. 2018. Erişim Tarihi: https://www.mcafee.com/enterprise/en-us/assets/reports/rp-mobile-threat-report-2018.pdf.

McLaughlin, N., Martinez del Rincon, J., Kang, B., Yerima, S., Miller, P., Sezer, S., Safaei, Y., Trickel, E., Zhao, Z., Doupé, A., 2017. Deep android malware detection. Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy, Scottsdale, Arizona, USA, 301-308.

Mezgec, S., Eftimov, T., Bucher, T., Seljak, B.K., 2019. Mixed Deep Learning and Natural Language Processing Method for Fake-Food Image Recognition and Standardization to Help Automated Dietary Assessment. Public health nutrition, 22(7), 1193-1202.

Milosevic, N., Dehghantanha, A., Choo, K.-K.R., 2017. Machine Learning Aided Android Malware Classification. Computers & Electrical Engineering, 61, 266-274.

Ng, S., 2017. Principal Component Analysis to Reduce Dimension on Digital Image. Procedia computer science, 111, 113-119.

Ozawa, S., Pang, S., Kasabov, N., 2006. An Incremental Principal Component Analysis for Chunk Data. 2006 IEEE International Conference on Fuzzy Systems, 16-21 July, Vancouver, BC, Canada, 2278-2285.

Peynirci, G., 2018. Malware Detection For The Android Platform Using Machine Learning Techniques. Yaşar University, Computer Engineering, Phd Thesis, 166s, İzmir.

Qiu, X., Ren, Y., Suganthan, P.N., Amaratunga, G.A., 2017. Empirical Mode Decomposition Based Ensemble Deep Learning for Load Demand Time Series Forecasting. Applied Soft Computing, 54, 246-255.

Ranjan, R., Patel, V.M., Chellappa, R., 2017. Hyperface: A Deep Multi-Task Learning Framework for Face Detection, Landmark Localization, Pose Estimation, and Gender Recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence, 41(1), 121-135.

Rosmansyah, Y., Dabarsyah, B., 2015. Malware Detection on Android Smartphones Using API Class and Machine Learning. 2015 International Conference on Electrical Engineering and Informatics (ICEEI), 10-11 August, Denpasar, Indonesia, 294-297.

Seo, S.-H., Gupta, A., Sallam, A.M., Bertino, E., Yim, K., 2014. Detecting Mobile Malware Threats to Homeland Security Through Static Analysis. Journal of Network and Computer Applications, 38, 43-53.

Statista. 2019. Erişim Tarihi: 02.11.2019. https://www.statista.com/statistics/-266210/number-of-available-applications-in-the-google-play-store/.

Sugunan, K., Kumar, T.G., Dhanya, K. 2018. Static and Dynamic Analysis for Android Malware Detection. In: Advances in Big Data and Cloud Computing. Springer, 147-155s.

Tokmak, M., Küçüksille, E.U., 2019. Kötü Amaçlı Windows Çalıştırılabilir Dosyalarının Derin Öğrenme İle Tespiti. Bilge International Journal of Science and Technology Research, 3(1), 67-76.

Tong, F., Yan, Z., 2017. A Hybrid Approach of Mobile Malware Detection in Android. Journal of Parallel and Distributed computing, 103, 22-31.

Türk, E., Arslan, S., 2017. Boot Time Optimization for Android-Based Embedded Systems. 2017 International Conference on Computer Science and Engineering (UBMK), 5-8 October, Antalya, Turkey, 1004-1008.

Türker, S., 2019. Zararlı Android Yazılımlarının Makine Öğrenmesi ile Ailelerine Göre Sınıflandırılması. Hacettepe Üniversitesi, Fen Bilimleri Enstitüsü, Bilgisayar Mühendisliği, Yüksek Lisans Tezi, 66s, Ankara.

Wei, F., Li, Y., Roy, S., Ou, X., Zhou, W., 2017. Deep Ground Truth Analysis of Current Android Malware. International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, 6-7 July, Bonn, Germany, 252-276.

Wu, D.-J., Mao, C.-H., Wei, T.-E., Lee, H.-M., Wu, K.-P., 2012. Droidmat: Android Malware Detection Through Manifest and Api Calls Tracing. 2012 Seventh Asia Joint Conference on Information Security, 9-10 August, Tokyo, Japan, 62-69.

Xu, L., Wu, S.F., Chen, H., 2013. Techniques and Tools for Analyzing and Understanding Android Applications. Citeseer.

Yang, Y., Wei, Z., Xu, Y., He, H., Wang, W., 2018. Droidward: an Effective Dynamic Analysis Method for Vetting Android Applications. Cluster Computing, 21(1), 265-275.

Yerima, S.Y., Sezer, S., McWilliams, G., 2014. Analysis of Bayesian Classification-Based Approaches for Android Malware Detection. IET Information Security, 8(1), 25-36.

Yuan, Z., Lu, Y., Xue, Y., 2016. Droiddetector: Android Malware Characterization and Detection Using Deep Learning. Tsinghua Science and Technology, 21(1), 114-123.

Zeyer, A., Doetsch, P., Voigtlaender, P., Schlüter, R., Ney, H., 2017. A Comprehensive Study of Deep Bidirectional LSTM RNNs for Acoustic Modeling in Speech Recognition. 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 5-9 March, New Orleans, LA, USA, 2462-2466.

EKLER


```
"Kancalama": [
      {
        "class_name": "android.telephony.TelephonyManager",
        "method": "getDeviceId",
      },
      {
        "class_name": "android.telephony.TelephonyManager",
        "method": "getSubscriberId",
      },
      {
        "class_name": "android.telephony.TelephonyManager",
        "method": "getLine1Number",
      },
      {
        "class_name": "android.telephony.TelephonyManager",
        "method": "getNetworkOperator",
      },
      {
        "class_name": "android.telephony.TelephonyManager",
        "method": "getNetworkOperatorName",
      },
      {
        "class_name": "android.telephony.TelephonyManager",
        "method": "getSimOperatorName",
      },
      {
        "class_name": "android.net.wifi.WifiInfo",
        "method": "getMacAddress",
      },
      {
        "class_name": "android.telephony.TelephonyManager",
        "method": "getSimCountryIso",


      },
      {
        "class_name": "android.telephony.TelephonyManager",
        "method": "getSimSerialNumber",


      },
      {
        "class_name": "android.telephony.TelephonyManager",
        "method": "getNetworkCountryIso",
       },
      {
        "class_name": "android.telephony.TelephonyManager",
```

```
   "method": "getDeviceSoftwareVersion",

},
{
   "class_name": "android.os.Debug",
   "method": "isDebuggerConnected",
     },
{
   "class_name": "android.app.SharedPreferencesImpl$EditorImpl",
   "method": "putString",

},
{
   "class_name": "android.app.SharedPreferencesImpl$EditorImpl",
   "method": "putBoolean",
},
{
   "class_name": "android.app.SharedPreferencesImpl$EditorImpl",
   "method": "putInt",
},
{
   "class_name": "android.app.SharedPreferencesImpl$EditorImpl",
   "method": "putLong",

},
{
   "class_name": "android.app.SharedPreferencesImpl$EditorImpl",
   "method": "putFloat",
},
{
   "class_name": "android.content.ContentValues",
   "method": "put",
},
{
   "class_name": "java.net.URL",
   "method": "openConnection",
},
{
   "class_name": "org.apache.http.impl.client.AbstractHttpClient",
   "method": "execute",
    },
{
   "class_name": "android.app.ContextImpl",
   "method": "registerReceiver",
},
{
   "class_name": "android.app.ActivityThread",
   "method": "handleReceiver",
},
```

```
    {
      "class_name": "android.app.Activity",
      "method": "startActivity",
       },
    {
      "class_name": "dalvik.system.BaseDexClassLoader",
      "method": "findResource",
    },
    {
      "class_name": "dalvik.system.BaseDexClassLoader",
      "method": "findLibrary",
    },
    {
      "class_name": "dalvik.system.DexFile",
      "method": "loadDex",
    },
    {
      "class_name": "dalvik.system.DexClassLoader",
      "method": null,
    },
    {
      "class_name": "dalvik.system.BaseDexClassLoader",
      "method": "findResources",
    },
    {
      "class_name": "dalvik.system.DexFile",
      "method": "loadClass",
    },
    {
      "class_name": "dalvik.system.DexFile",
      "method": null,
    },
    {
      "class_name": "dalvik.system.PathClassLoader",
      "method": null,
    },
    {
      "class_name": "java.lang.reflect.Method",
      "method": "invoke",
    },
    {
      "class_name": "javax.crypto.spec.SecretKeySpec",
      "method": null,
    },
{
      "class_name": "javax.crypto.spec.IvParameterSpec ",
      "method": null,
    },
{
```

```
        "class_name": "javax.crypto.spec.PBEKeySpec",
        "method": null,
      },
      {
        "class_name": "javax.crypto.Cipher",
        "method": "doFinal",
      },
 {
        "class_name": "javax.crypto.Cipher",
        "method": "getIV",
      },
{
        "class_name": "java.security.SecureRandom",
        "method": "setSeed",
      },
{
        "class_name": "javax.crypto.Cipher",
        "method": "getInstance",
      },
      {
        "class_name": "javax.crypto.Mac",
        "method": "doFinal",
      },
      {
        "class_name": "android.app.ApplicationPackageManager",
        "method": "setComponentEnabledSetting",
      },
      {
        "class_name": "android.app.NotificationManager",
        "method": "notify",
      },
      {
        "class_name": "android.util.Base64",
        "method": "decode",
      },
      {
        "class_name": "android.telephony.TelephonyManager",
        "method": "listen",
      },
      {
        "class_name": "android.util.Base64",
        "method": "encode",
      },
      {
        "class_name": "android.util.Base64",
        "method": "encodeToString",


      },
```

```
{
   "class_name": "android.net.ConnectivityManager",
   "method": "setMobileDataEnabled",

},
{
   "class_name": "android.content.BroadcastReceiver",
   "method": "abortBroadcast",
},
{
   "class_name": "android.telephony.SmsManager",
   "method": "sendTextMessage",
},
{
   "class_name": "android.telephony.SmsManager",
   "method": "sendMultipartTextMessage",
},
{
   "class_name": "java.lang.Runtime",
   "method": "exec",
},
{
   "class_name": "java.lang.ProcessBuilder",
   "method": "start",
},
{
   "class_name": "java.io.FileOutputStream",
   "method": "write",
},
    {
   "class_name": "java.io.File",
   "method": "null",
},
{
   "class_name": "java.io.FileInputStream",
   "method": "read",


},
{
   "class_name": "android.app.ActivityManager",
   "method": "killBackgroundProcesses",
},
{
   "class_name": "android.os.Process",
   "method": "killProcess",
},
{
   "class_name": "android.content.ContentResolver",
```

```
    "method": "query",
  },
  {
    "class_name": "android.content.ContentResolver",
    "method": "registerContentObserver",
  },
  {
    "class_name": "android.content.ContentResolver",
    "method": "insert",
  },
  {
    "class_name": "android.location.Location",
    "method": "getLatitude",
  },
  {
    "class_name": "android.location.Location",
    "method": "getLongitude",
  },
   {
    "class_name": "android.location.LocationManager",
    "method": "getBestProvider",
  },
   {
    "class_name": "android.accounts.AccountManager",
    "method": "getAccounts",
  },
   {
    "class_name": "android.accounts.AccountManager",
    "method": "getAccountsByType",
  },
   {
    "class_name": "android.accounts.AccountManager",
    "method": "getAccountsByType",
  },
{
    "class_name": "android.content.ClipboardManager ",
    "method": "setPrimaryClip",
  },
  {
    "class_name": "android.content.ContentResolver",
    "method": "delete",
  },
  {
    "class_name": "android.media.AudioRecord",
    "method": "startRecording",
  },

  {
    "class_name": "android.media.MediaRecorder",
```

```
        "method": "start",
    },
    {
        "class_name": "android.os.SystemProperties",
        "method": "get",
    },
    {
        "class_name": "android.app.ApplicationPackageManager",
        "method": "getInstalledPackages",
    },
{
        "class_name": "android.content.Context ",
        "method": "openFileOutput",
    },
{
        "class_name": "android.view.Window",
        "method": "setFlags",
    },
{
        "class_name": "android.view.SurfaceView",
        "method": "setSecure",
    },
        {
        "class_name": "libcore.io.IoBridge",
        "method": "open",

        "type": "file"
    },
{
        "class_name": "java.security.MessageDigest",
        "method": "update",
    },
{
        "class_name": "java.security.MessageDigest",
        "method": "digest",
    },
{
        "class_name": "android.webkit.WebView",
        "method": "loadUrl",
    },
{
        "class_name": "java.net.URL",
        "method": "null",
    },
{
        "class_name": "android.content.Context",
        "method": "startActivities",
```

```
    },
{
    "class_name": "android.content.Context ",
    "method": "sendBroadcast",
    },
{
    "class_name": "android.media.MediaRecorder",
    "method": "stop",


    },
{
    "class_name": "android.net.Uri",
    "method": "parse",
    },

{
    "class_name": "java.security.MessageDigest",
    "method": "getInstance",
    },
{
    "class_name": "java.net.ProxySelectorImpl",
    "method": "select",
    },
{
    "class_name": "org.apache.http.impl.client.DefaultHttpClient",
    "method": "null",
    },
{
    "class_name": "sun.net.spi.DefaultProxySelector",
    "method": "select",
    },
{
    "class_name": "java.io.FileInputStream",
    "method": "null",
    },
{
    "class_name": "java.io.ObjectInputStream",
    "method": "readObject",
    },
{
    "class_name": "android.content.Context",
    "method": "getSharedPreferences",
    },
{
    "class_name": "android.app.SharedPreferencesImpl$EditorImpl",
    "method": "contains",
    },
{
```

```
        "class_name": "android.app.SharedPreferencesImpl$EditorImpl",
        "method": "getLong",
    },
{
        "class_name": "android.app.SharedPreferencesImpl$EditorImpl",
        "method": "getFloat",
    },
{
        "class_name": "android.app.SharedPreferencesImpl$EditorImpl",
        "method": "getBoolean",
    },
{
        "class_name": "android.app.SharedPreferencesImpl$EditorImpl",
        "method": "getInt",
    },
{
        "class_name": "android.app.SharedPreferencesImpl$EditorImpl",
        "method": "getStringSet",
    },
{
        "class_name": "android.app.SharedPreferencesImpl$EditorImpl",
        "method": "apply",
    },{
        "class_name": "android.app.SharedPreferencesImpl$EditorImpl",
        "method": "getStringSet",
    },{
        "class_name": "android.app.SharedPreferencesImpl$EditorImpl",
        "method": "commit",
    },{
        "class_name": "android.app.SharedPreferencesImpl$EditorImpl",
        "method": "null",
    },
{
        "class_name": "android.telephony.SmsManager",
        "method": "sendDataMessage",
    },
{
        "class_name": "android.database.sqlite.SQLiteDatabase ",
        "method": "execSQL",
    },
{
        "class_name": "android.database.sqlite.SQLiteDatabase",
        "method": "update",
    },
{
        "class_name": "android.database.sqlite.SQLiteDatabase",
        "method": "update",
    },
{
```

```
        "class_name": "android.database.sqlite.SQLiteDatabase",
        "method": "insert",
    },
{
        "class_name": "android.database.sqlite.SQLiteDatabase",
        "method": "query",
    },
{
        "class_name": "net.sqlcipher.database.SQLiteDatabase",
        "method": "execSQL",


    },
{
        "class_name": "net.sqlcipher.database.SQLiteDatabase",
        "method": "openOrCreateDatabase",
    },
{
        "class_name": "net.sqlcipher.database.SQLiteDatabase",
        "method": "rawQuery",
    },

{
        "class_name": "android.app.Activity",
        "method": "managedQuery",
    },
{
        "class_name": "android.content.Context",
        "method": "getDatabasePath",
    },
{
        "class_name": "android.webkit.WebView",
        "method": "addJavascriptInterface",
    },
{
        "class_name": "android.webkit.WebView",
        "method": "loadData",
    },

{
        "class_name": "android.webkit.WebView",
        "method": "setWebChromeClient",
    },
{
        "class_name": "android.webkit.WebView",
        "method": "setWebViewClient",
    },
{
        "class_name": "android.webkit.WebView",
```

```
        "method": "setWebContentsDebuggingEnabled",


    },
 {
        "class_name": "javax.net.ssl.TrustManagerFactory",
        "method": "getTrustManagers",
    },
{
        "class_name": "javax.net.ssl.SSLContext",
        "method": "init",
    },
{
        "class_name": "javax.net.ssl.HttpsURLConnection",
        "method": "setSSLSocketFactory",


    },
{
        "class_name": "org.apache.http.conn.ssl.HttpsURLConnection",
        "method": "setDefaultHostnameVerifier",
    },
{
        "class_name": "org.apache.http.conn.ssl.SSLSocketFactory",
        "method": "getSocketFactory",
    }
{
        "class_name": "org.apache.http.conn.ssl.SSLSocketFactory",
        "method": "createKeyManagers",

    },
{
        "class_name": "org.apache.http.conn.ssl.SSLSocketFactory",
        "method": "isSecure",

    },


{
        "class_name": "okhttp3.CertificatePinner",
        "method": "findMatchingPins",

    },
{
        "class_name": "android.os.Process",
        "method": "start",

    }
  ],
  "trace": false}
```

ÖZGEÇMİŞ

Adı Soyadı : Mahmut TOKMAK

Doğum Yeri ve Yılı : Yalvaç, 1978

Medeni Hali : Evli

Yabancı Dili : İngilizce

E-posta : mahmuttokmak@isparta.edu.tr

Eğitim Durumu

Lisans :.Selçuk Üniversitesi, Mühendislik Mimarlık Fakültesi, Bilgisayar Mühendisliği

Yüksek Lisans : Süleyman Demirel Üniversitesi, Teknik Eğitim Fakültesi, Elektronik-Bilgisayar Eğitimi

Mesleki Deneyim

Süleyman Demirel Üniversitesi Gelendost MYO 2003-2018

Isparta Uygulamalı Bilimler Üniversitesi Gelendost MYO 2018-

Yayınlar

Küçüksille, E.U.,Tokmak, M., 2011. Yapay Arı Kolonisi Algoritması Kullanarak Otomatik Ders Çizelgeleme. Süleyman Demirel Üniversitesi Fen Bilimleri Enstitüsü Dergisi 15-3,203-210.

Tokmak, M., Küçüksille, E.U., 2018. Makine Öğrenme Yöntemleri ile Windows Api Tabanlı Kötü Amaçlı Yazılım Tespiti. International Conference on Science and Technology (ICONST 2018), pp.151-159, Prizren-KOSOVO, 5-9 September 2018.

Tokmak, M., Küçüksille, E.U., 2019. Kötü Amaçlı Windows Çalıştırılabilir Dosyalarının Derin Öğrenme İle Tespiti. Bilge International Journal of Science and Technology Research, 3 (1), 67-76.