
GAZİ ÜNİVERSİTESİ
Gazi
1926

İZİN TABANLI STATİK ANALİZ YÖNTEMİ İLE ANDROİD UYGULAMALARIN SINIFLANDIRILMASI

Gülsüm KAYABAŞI

YÜKSEK LİSANS TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

GAZİ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

ARALIK 2016

Gülsüm KAYABAŞI tarafından hazırlanan "İZİN TABANLI STATİK ANALİZ YÖNTEMİ İLE ANDROİD UYGULAMALARIN SINIFLANDIRILMASI" adlı tez çalışması aşağıdaki jüri tarafından OY BİRLİĞİ ile Gazi Üniversitesi Bilgisayar Mühendisliği Anabilim Dalında YÜKSEK LİSANS TEZİ olarak kabul edilmiştir.

Danışman: Yrd. Doç. Dr. İbrahim Alper DOĞRU

Bilgisayar Mühendisliği, Gazi Üniversitesi

Bu tezin, kapsam ve kalite olarak Yüksek Lisans Tezi olduğunu onaylıyorum ...……

Başkan : Doç. Dr. Aydın ÇETİN

Bilgisayar Mühendisliği, Gazi Üniversitesi

Bu tezin, kapsam ve kalite olarak Yüksek Lisans Tezi olduğunu onaylıyorum ……...

Üye : Yrd. Doç. Dr. Mehmet ŞİMŞEK

Bilgisayar Mühendisliği, Düzce Üniversitesi

Bu tezin, kapsam ve kalite olarak Yüksek Lisans Tezi olduğunu onaylıyorum ……...

Tez Savunma Tarihi: 19/12/2016

Jüri tarafından kabul edilen bu tezin Yüksek Lisans Tezi olması için gerekli şartları yerine getirdiğini onaylıyorum.

…….…….

Prof. Dr. Hadi GÖKÇEN

Fen Bilimleri Enstitüsü Müdürü

ETİK BEYAN

Gazi Üniversitesi Fen Bilimleri Enstitüsü Tez Yazım Kurallarına uygun olarak hazırladığım bu tez çalışmasında;

- Tez içinde sunduğum verileri, bilgileri ve dokümanları akademik ve etik kurallar çerçevesinde elde ettiğimi,
- Tüm bilgi, belge, değerlendirme ve sonuçları bilimsel etik ve ahlak kurallarına uygun olarak sunduğumu,
- Tez çalışmasında yararlandığım eserlerin tümüne uygun atıfta bulunarak kaynak gösterdiğimi,
- Kullanılan verilerde herhangi bir değişiklik yapmadığımı,
- Bu tezde sunduğum çalışmanın özgün olduğunu,

bildirir, aksi bir durumda aleyhime doğabilecek tüm hak kayıplarını kabullendiğimi beyan ederim.

Gülsüm KAYABAŞI

19/12/2016

İZİN TABANLI STATİK ANALİZ YÖNTEMİ İLE ANDROİD UYGULAMALARIN SINIFLANDIRILMASI

(Yüksek Lisans Tezi)

Gülsüm KAYABAŞI

GAZİ ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

Aralık 2016

ÖZET

Günümüz teknolojisinde kullanılan akıllı cihazlar mobil işletim sistemlerine sahiptir. İstatistiksel verilere göre açık kaynak kodlu ve Linux tabanlı bir mobil işletim sistemi olan Android halen en popüler işletim sistemi olduğu görülmektedir. Android'in popüler olması, açık kaynak kod yapısı ve sınırlı sayıda koruyucu yazılımların olması bu işletim sistemini kötücül yazılım geliştiricilerin ve saldırganların başlıca hedefi haline getirmiştir. Android marketler üzerinden indirilen uygulamaların kötücül amaçlı mobil yazılımların yayılımında önemli bir yer tuttuğu düşünülmektedir. Android uygulamaların sayısı ve çeşitliliği göz önünde bulundurulduğunda, kötücül uygulamaların tespiti ve önlenmesi oldukça zordur. Bu tezde, mobil cihazlarda kötücül uygulama tespiti yapmak amacıyla izin ve api çağrı tabanlı olarak uygulamaları inceleyen bir yapı oluşturulmuştur. İncelemede Drebin veri seti üzerinde statik analiz yöntemi kullanılmış, inceleme neticesinde de farklı sınıflandırma algoritmaları kullanılarak uygulamalar sınıflandırılmış ve sınıflandırma işlemini en performanslı şekilde yapan algoritma tespiti gerçekleştirilmiştir.

Bilim Kodu : 92429

Anahtar Kelimeler : Mobil işletim sistemi, Android, kötücül yazılım, mobil uygulama güvenliği, statik analiz metotları, sınıflandırma

Sayfa Adedi : 66

Danışman : Yrd. Doç. Dr. İbrahim Alper DOĞRU

CLASSIFICATION OF ANDROID APPLICATIONS WITH PERMISSION BASED STATIC ANALYSIS METHODS

(M. Sc. Thesis)

Gülsüm KAYABAŞI

GAZİ UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

December 2016

ABSTRACT

The smart devices used in today's technology have mobile operating systems. Statistically, Android, which is an open source and Linux based mobile operating system is still the most popular operating system in the market. The popularity of Android, its open source code structure and the availability of a limited number of preventive software have made this operating system the primary target of malware developers and attackers. Applications downloaded from the Android market are thought to have an important place in the spread of malicious mobile software. When considering the number and variety of Android applications, detection and prevention of malicious applications is very difficult. In this thesis, a structure that permits the applications and APIcall-based has been formed in order to detect malicious applications on mobile devices. This thesis, static analysis method was used on the Drebin dataset and applications were classified by using different classification algorithms and the detection of algorithm which performed the classification process in the most efficient way were carried out

Science Code : 92429

Key Words : Mobile operating system, Android, malware software, mobile application security, static analysis methods, classification

Page Number : 66

Supervisor : Assist. Prof. Dr. İbrahim Alper DOĞRU

TEŞEKKÜR

Yüksek lisans çalışmalarımın her aşamasında kıymetli katkıları ve eleştirileriyle yol gösteren, beni her zaman çalışmalarımın ilerlemesi yönünde teşvik eden ve güven veren değerli danışmanım Yrd.Doç.Dr. İbrahim Alper DOĞRU'ya teşekkür ederim.

Ayrıca, okul hayatım boyunca her türlü zorluğa rağmen desteklerini ve sevgilerini benden esirgemeyen, bana gönül bağıyla bağlı olan annem, babam, kardeşimle birlikte bana olan sevgisi ve sabrı hiç bitmeyen müstakbel eşime tüm kalbimle sonsuz teşekkür ediyorum.

İÇİNDEKİLER

Sayfa

Sayfa

ÇİZELGELERİN LİSTESİ

Çizelge **Sayfa**

ŞEKİLLERİN LİSTESİ

Şekil **Sayfa**

Şekil **Sayfa**

SİMGELER VE KISALTMALAR

Bu çalışmada kullanılmış kısaltmalar, açıklamaları ile birlikte aşağıda sunulmuştur.

Kısaltmalar	Açıklamalar
API	Uygulama Programlama Arayüzü
APN	Access Point Name
AUC	Area Under The Curve
CHEX	Component Hijacking Examiner)
CPU	Central Processing Unit
CS-SNB	Cost Sensitive-Selective Naive Bayes
GPS	Global Positioning System
IOS	Iphone Operating System
IPC	Inter Process Communication
JNI	Java Native Interface
KNN	K Nearest Neighbour
MADAM	Multi-level Anomaly Detector for Android Malware
MMS	Multimedya Mesaj
RAM	Random Access Memory
SHA	Secure Hash Algorithm
SVD	Singular Value Decomposition
SVM	Support Vector Machine
USB	Universal Serial Bus
VPN	Virtual Private Network

1. GİRİŞ

İstemediğimiz kötü niyetli yazılımlardan olan virüsler, solucanlar, Truva atları, kullanıcıların bilgilerini toplamaya çalışan reklam içerikli programlar (casus) şahsi bilgisayarlarımız ve kurum içinde kullandığımız bilgisayarlar için tehlike meydana getirmişlerdir [1]. Akıllı telefon kullanıcı miktarında artış oldukça, söz konusu telefonların kurulduğu platformlarda da bazı tehditlerin oluşması diye bir gerçek bulunmaktadır [2]. Bu tehditlerin arasında kişisel bilgisayarlarımızda yaşadığımıza benzer şekilde, akıllı diye nitelendirdiğimiz cep telefonlarımız ve çok gelişmiş küçük bilgisayarlar da bu tehditlerin etki alanına girmektedir [3,4].

Geleneksel olarak adlandırdığımız bilgisayar sistemlerinde olduğu gibi, akıllı platformlarda zararlı yazılım tespiti için çeşitli çözümler bulunmaktadır. Bu çözümlerin geneli imza tabanlı yaklaşımlara dayanmaktadır [1]. Verimli ve kolay olduğu bilinen imza tabanlı yaklaşımlarda ikili dosya eşleşimi yapılmaktadır. Ancak bu gibi çözümler zararlı yazılımı tespit etmek için yeterli değildir. Bunun sebebi kullanılan imzaların kalite ve kapsamının çeşitlilik göstermesidir. Diğer bir husus ise, imzaların oluşumu statik ve dinamik analiz yöntemlerine ihtiyaç olduğunu göstermiştir. Bu yöntemler zaman alır ve süreçlerin interaktif olabilmesi için konunun uzmanlar tarafından gerçekleştirilmesi gerekmektedir [5].

Günümüz piyasasında mevcut olan akıllı cihazları kötü niyetli uygulamalardan korumak için alınan tedbirlerin pek çoğu imzalama bakış açısıyla geliştirildiklerinden zayıftırlar. Zayıf bulunmasının sebebi ise imzanın kullanılabilir hale geldiği süreç içerisinde kullanıcıların farklı ve daha önce kötü niyetli bir uygulama ile karşılaşmalarıdır. Bu konuda geliştirilen çalışmalara bakıldığında; Bulygin [6] 700.000'i geçen sayıda akıllı cihaz kullanarak bir MMS solucanının rastgele hedeflediği bir telefon rehberindeki numaralara yaklaşık 3 saat içerisinde bulaşabildiğini, Oberhide ve arkadaşları [7], bir imza tabanlı anti virüs motoru için yeni tehditleri tespit etmesi için gerekli ortalama sürenin 48 gün olduğunu göstermiştir. Akıllı cihazlardaki kötü niyetli uygulamalar için virüslü bir cihazda herhangi bir güvenlik önlemi yoksa eğer saniyeler içerisinde cihaza önemli zararlar verebilmektedir. Hal böyleyken; akıllı cihaz mobil işletim sistemlerinden biri olan Android, uygulama geliştiren ekipler için dikkat çekici bir sistem olmuştur. Açık kaynak kurabilmek maksadıyla, güvenlik araçları çekirdek seviyesinde geliştirme yapabilir. Bu da Android

işletim sistemi kurulu olan bir mobil cihaz için kaynaklarının kısıtlı sayıda ve sahip olduğu güvenlik mekanizmasının kapsamının dar olmasına sebebiyet verir [7].

Söz konusu cihazların sahip olduğu kısıtlı kaynaklar sebebiyle, Android makinalarda kötü niyetli yazılımların var olduğunu tespit edebilmek için statik mekanizmalar odak haline gelmiştir. Kötücül uygulamanın varlığını saptayabilmek için kullanılan statik analiz bakış açısı, az kaynak tüketen ve taşınabilir cihazların gereksinim duyduğu basit denilebilecek durumdaki sınıflandırıcıların bu duruma uyumlu olmasını sağlamaktadır. Mobil cihazın hesap yükünün azaltılmasına yardımcı olmak maksadıyla genellikle dış sunucular kullanılmaktadır. Bu da dış sunucuya bağımlı olmak anlamına gelmektedir. Dış sunuculara bağlı olmak çeşitli entegrasyonlar istediğinden maliyeti yüksektir [8,9].

İçinde bulunduğumuz çağın gereklerine uygun olan bilgisayar ve iletişim platformları saldırganların kötü niyetli yaklaşımlarına karşı oldukça tedbirlidir. Bu saldırganların amacı solucan, virüs, Truva atı vb. kötü amaçlı yazılımın yayılmasını artırmaktır. Söz konusu yazılımların yaygınlaşması özel şirketlere, şahsi kullanıcılara ve ulusa önemli zararlar verebilir. Bu yazılımların yaygın olmasının en önemli sebeplerinden biri internet kullanımının her geçen gün daha da artmasıdır. Bu yoğunluk yayılmayı olumlu yönden etkilemektedir. Özellikle internet bağlantı hızlarının yükselmesiyle, yeni zararlı yazılımların geliştirilmesinde artış olmuştur. Bu artış zararlı yazılımlardan korunma mekanizmaları oluşturulmasına ve zararlı yazılımı kurulum esnasında veya kurulumdan sonra fark etmeyi sağlayıcı ihtiyaçlar doğurmuştur. Böylelikle zararlı yazılım tespit etmek için çeşitli metotlar geliştirilmeye başlanmıştır [10]. Kötü niyetli uygulama tespiti yapabilmek maksadıyla çeşitli analiz yöntemleri bulunmaktadır. Bu yöntemler statik, dinamik ve imza tabanlı analiz olarak bilinmektedir [11].

Bu tezde analiz metotlarından birisi olan statik analiz yöntemi kullanılarak izin tabanlı yaklaşım ile Android uygulamalarının en performanslı algoritma ile sınıflandırılması yapılmıştır. Tez, 5 bölümden oluşmakta olup, 2.bölümünde tez çalışmasında kullanılan yöntem ve araçlardan bahsedilmiş, 3.bölümde tez çalışmasının sistem tasarımı ve mimarisi detaylı bir şekilde anlatılmıştır. 4.bölümde tez çalışması sonucunda elde edilen bulgulara dayalı tartışmadan bahsedilmiştir. 5.bölümde sonuçlar karşılaştırılmış yapılan tez çalışması ile ilgili son olarak bir değerlendirme yapılmıştır.

2. YÖNTEM VE ARAÇLAR

Mobil cihazlarda kötü niyetli uygulama tespiti yapabilmek maksadıyla bazı analiz yöntemleri bulunmaktadır [10]. Tez çalışmasında statik analiz yöntemi kullanarak Android işletim sistemi uygulamaları için kötücül yazılım analizi yapan bir model oluşturulmuştur. Bu bölümde oluşturulan modelden, literatürde kullanılmış statik analiz yöntemi kullanan araçlardan ve literatürdeki diğer analiz yöntemlerinden bahsedilecektir.

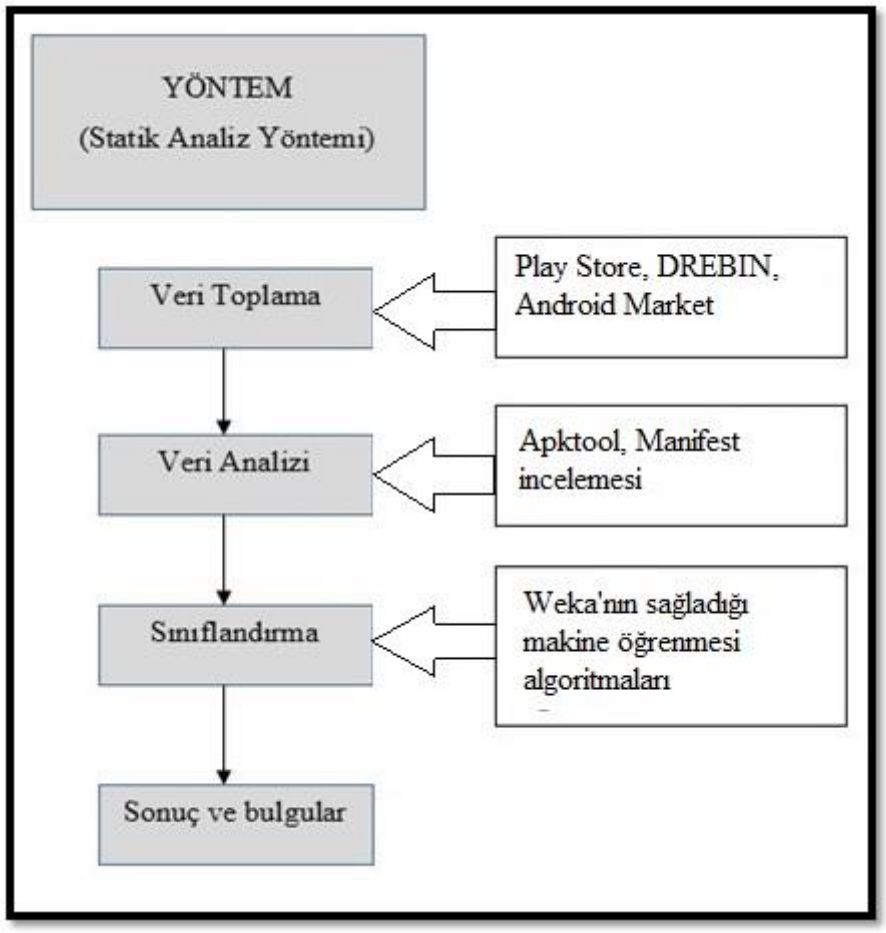
2.1. Statik Analiz Yöntemi

Android işletim sistemi gelişimiyle birlikte kötü niyetli yazılım çeşitleri ve sayısı artmıştır. Kötücül yazılımların artmasıyla, bunlardan korunma metotları paralel olarak artış göstermiştir. Statik analiz yöntemleri Android işletim sistemine sahip mobil cihaz için uygulamaların kurulum aşamalarından önce uygulamanın verilerinin kötücül yazılım tespitinde bulunulmasına dayanır. Kötücül yazılımın cihaza kurulmadan önce tespit edilmesi statik analiz yöntemi ile kötü niyetli yazılımın belirlenmesi ve bu yazılımlardan korunmanın en büyük avantajıdır. Bu sayede, mobil cihaz kötü niyetli yazılımın vereceği etkilere maruz kalmamaktadır [5].

Bu yöntemde; uygulama ile ilgili bilgi veya uygulamanın binary (ikili) veya kaynak kodunda çok net veya net olmayan incelemelerin bulunduğu davranışların olduğu düşünülmektedir. Söz konusu yöntemin teknikleri çok olmamasına karşın etkin ve hızlıdır. Kullanılan tekniklerde bulanıklaştırma arttıkça; az sayıda bulunan çeşitli zararlı uygulamalarla baş etmek beceri anlamında kolaylaşmış olur [5].

2.1.1. Tez çalışmasında kullanılan statik analiz yöntemi

Bu tez çalışmasında kullanılan statik analiz yöntemi için yeni bir model oluşturulmuştur. Bu modelin basamakları veri toplama, veri analizi, sınıflandırma, sonuç ve bulgulara ulaşmadır. Oluşturulan modelin şeması Şekil 2.1'de görülmektedir.



Şekil 2.1. Statik analiz model şeması

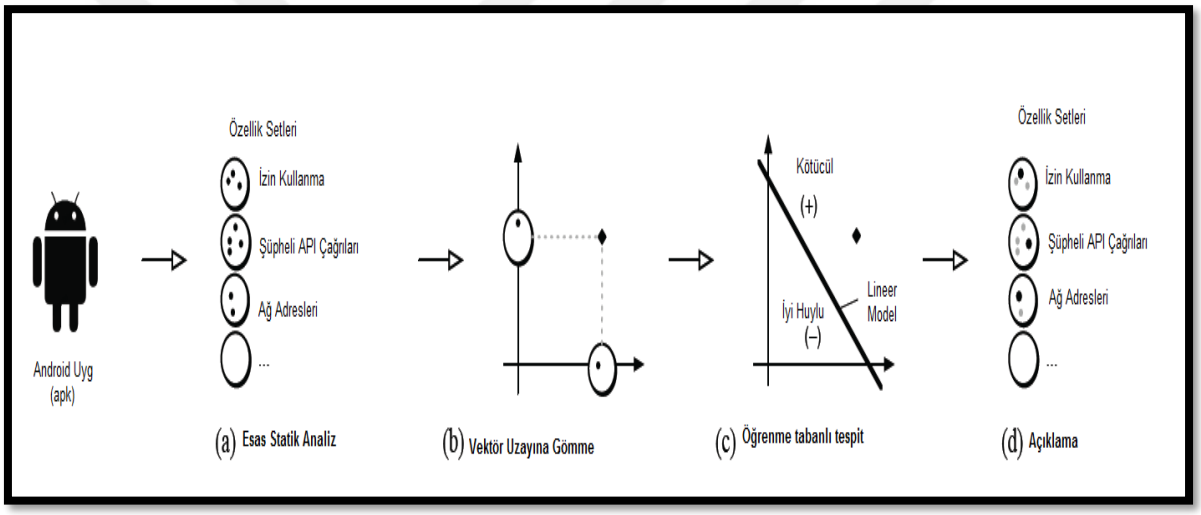
Tez çalışmasındaki veri toplama işlemi için, GooglePlay'den indirilen iyi niyetli uygulamalar Android Market kullanılarak indirilmiştir. Bu kısımda kötü niyetli uygulamalar için ise Drebin veri seti kullanılmıştır. Veri analizi basamağında, oluşturulan veri setinin Android Manifest incelemesi yapılmış ve apktool analiz aracı kullanılarak uygulamaların smali dosyalarında erişilmiştir. Smali dosyalarının içerisinde tez çalışmasında sınıflandırma işlemini gerçekleştirebilmek için kullanılan API çağrıları ve izinler bulunmaktadır. Sınıflandırma ise Weka uygulamasının sunduğu maline öğrenmesi algoritmaları kullanılarak yapılmıştır. Tez çalışmasının Weka kullanımı kısmında çıkan bulgular sonucunda kıyaslama yapılmıştır.

2.1.2. Literatürde geçen statik analiz yöntemleri

Statik analiz yöntemi kullanılarak kötü niyetli uygulama tespiti yapan çeşitli araçlar bulunmaktadır. Literatürde geçen diğer bazı statik analiz yöntemleri devam eden başlıklarda bahsedilmektedir.

Drebin

Drebin[12], statik analiz yöntemine makine öğrenmesi bakış açısı da katarak Android'de kötücül uygulama tespitinde kullanılan bir araç haline gelmiştir. Bu araç, uygulamanın kaynak kodlarını ve AndroidManifest dosyasını kullanıp uygulamanın kullandığı izinleri, uygulama programlama ara yüzü çağrılarını ve ağ adresleri gibi çeşitli özellikleri de toplar. Bu özelliklerin hepsi toplanarak bir vektör uzayına gömülmüştür ve çıkan desenler yardımıyla kötü niyetli uygulama tespiti yapılmıştır. Drebin kötü niyetli uygulama tespit aracı tarafından yapılan statik analiz adımlarını gösteren şematik yapı Şekil 2.2'de gösterilmiştir.



Şekil 2.2. Drebin aracının statik analiz adımları [12]

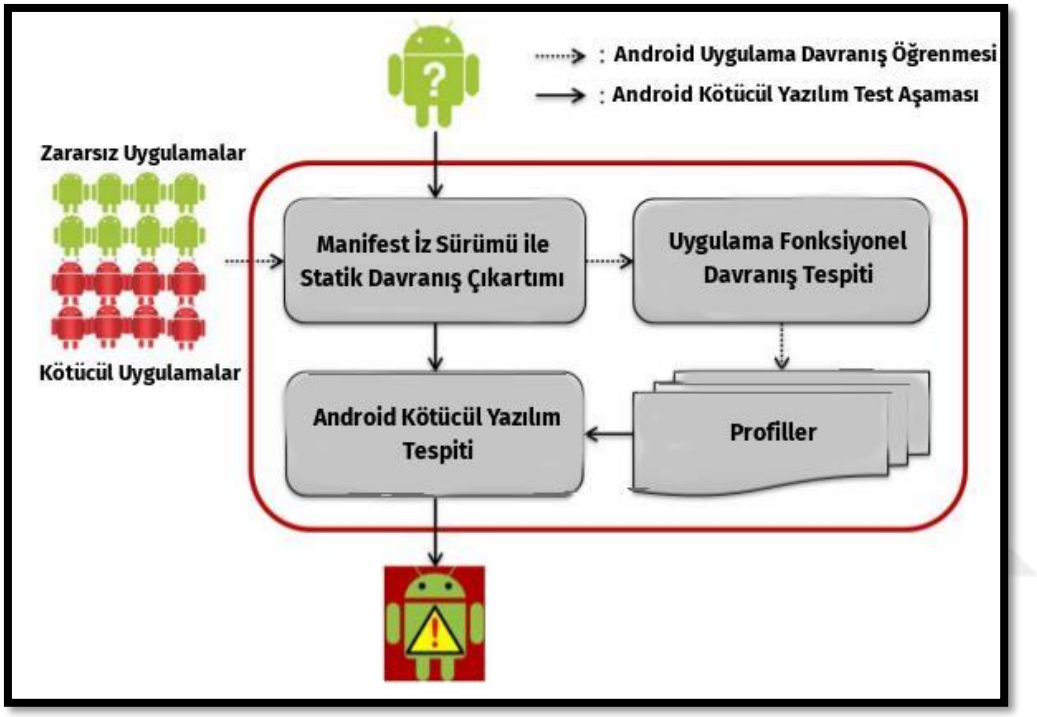
Şekil 2.2'de uygulamada kullanılan özellik setleri oluşturulmuş ve bu özellikler bir vektör uzayına gömülmüştür. Bu işlem lineer olan bir öğrenme modeli ile birleştirilerek iyi ve kötü niyetli uygulamaları birbirinden ayırabilmeye yardımcı olmuştur.

DroidMat

Öznitelik(feature) tabanlı mekanizma kullanan statik analiz araçlarından biridir. Bu statik analiz paradigmasında Android kötücül yazılım tespiti için tespitin verimli olması ve yetenekli analizler yapması dikkate alınmıştır. Bu mekanizma statik bilgileri dikkate alır. Bunların içinde; izinler, bileşenlerin dağılımı, geçen mesajların niyeti ve API çağrıları Android uygulamaların davranışlarını karakterize etmek için kullanılır. Android

zararlılarının farklı niyetlerini tanımak amacıyla farklı kümeleme algoritmaları kötücül yazılım modelleme yeteneğini iyileştirmek için uygulanmıştır.

DroidMat[13] aracı ise, AndroidManifest dosyasını ve uygulamanın kullandığı izinlere yönelik uygulama programlama ara yüzü (API) çağrıları üzerinden kötü niyetli uygulama tespiti yapar. DroidMat aracının mimari yapısı Şekil 2.3'deki gibidir.

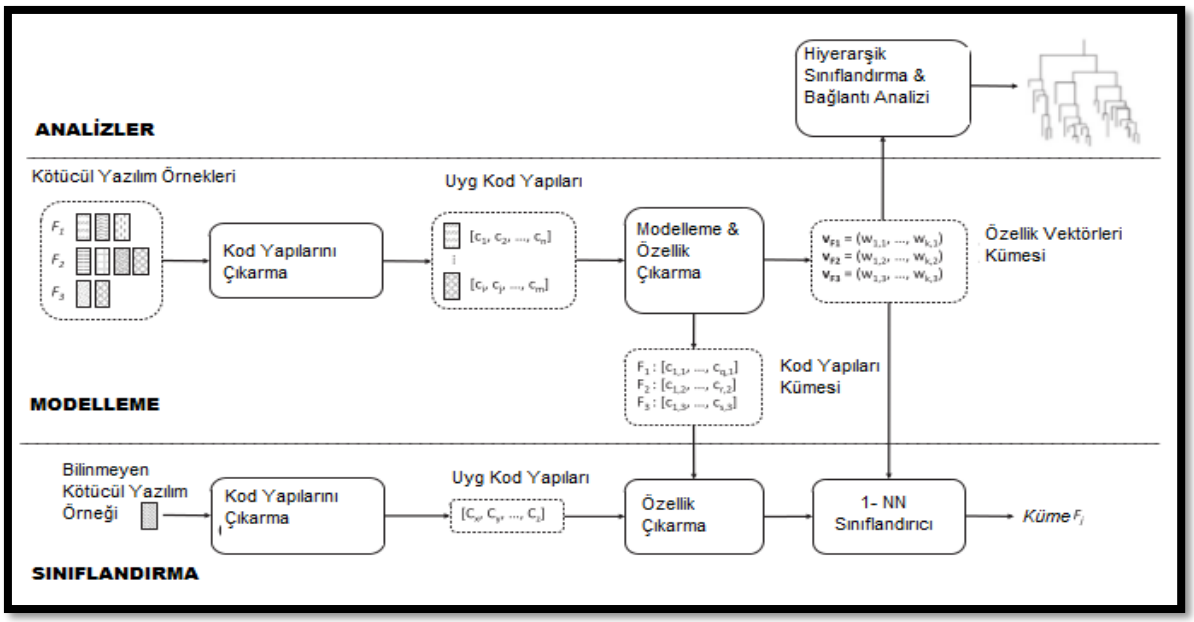


Şekil 2.3. DroidMat mimari yapısı [14]

Şekil 2.3'de mimari yapıya bakacak olursak, AndroidManifest dosyası kullanılarak statik analiz gerçekleştirilmiştir. Analiz sonuçları alındıktan sonra fonksiyonel davranış tespiti aşamasına geçilmiştir. Daha sonra K-means algoritmasıyla kötü niyetli uygulama modelleri oluşturulmuştur. Bu algoritmanın istediği küme sayısı tekil değer ayrışımı (SVD-Singular Value Decomposition) algoritması ile kullanılarak belirlenmiştir. Daha sonra ise k=1 olduğu durumda kNN (k Nearest Neighbours) sınıflandırma algoritması kullanılarak uygulamanın kötü niyetli olup olmadığının tespiti yapılmıştır. [13].

DENDROİD

DENDROİD, statik analizi iki yolla gerçekleştiren bir analiz aracıdır. Birincisi, mobil platformda bulunan yeni kötü niyetli uygulama ailelerinin karakteristik özelliklerini belirleme kısmında kod yapılarını çok iyi kullanmaktadır. Geliştirilmiş kod metotlarının iç yapısına odaklanmak komutların özel dizilerine göre odaklanmaktan daha avantajlı olup bulanıklaştırmaya da engel olmuştur. Ayrıca bunun gibi yapılar, akıllı cihazlardaki kötü niyetli uygulamanın kodlarının yeniden kullanılarak daha hızlı kod geliştirimi sağladığı ispatlanmıştır. DENDROİD aracının mimarisi Şekil 2.4'te gösterilmiştir [15].



Şekil 2.4. DENDROID mimari yapısı [15]

Şekil 2.4'te de görüldüğü gibi, eldeki örnek sete sınıflandırma işlemi yapmak, kod parçalarını incelemek veya kötü niyetli uygulama sınıflarının değişim sürecindeki bağlantıları üzerinde yapılan çalışmaları otomatik halen getirmek için ilk defa bu araç metin madenciliği denilen bir teknik kullanımını öne sürmüştür. Buna ek olarak, metin madenciliği adlı teknikte veri miktarının boyutunun büyük olması problem teşkil etmemiş, çalışma ona göre kodlanmış ve özellik çıkarımı etkin olarak gerçekleştirilmiştir [15].

RiskRanker

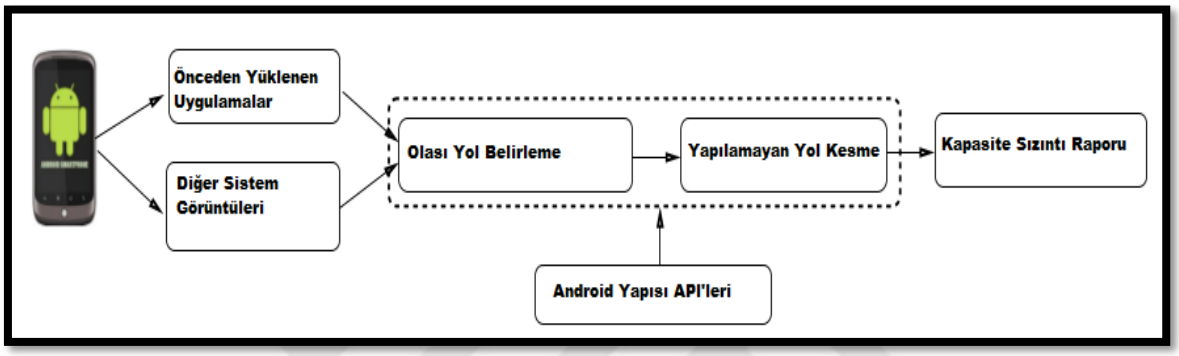
RiskRanker, Android market içinde bulunan sıfırıncı gün kötü niyetli uygulamalarını bulmak ve etkisiz hale getirmek için geliştirilmiş bir araçtır. RiskRanker statik analizinin amacı, güvenilir olmayan uygulamalar arasından yüksek ihtimalle zararlı olanları bulup değerlendirmek ve ölçülebilir olma, verimlilik ve doğruluk gibi gereksinimleri üst seviyeye çıkararak bunların tümünün olduğu yönetimi kolay yapılabilen bir çevre (alan) oluşturmaktır. Kapsamlı bir şekilde anlatmak gerekirse, bin ve üzerindeki sayıdaki uygulamaları ele alabilmek için kaynağın verimiyle doğru orantılı olarak ölçülebilir olma durumuna bakılması gerekmektedir. Buna ek olarak kötü niyetli uygulamanın tespitini yapabilmek ve bunun doğruluğunu kanıtlamak için girdileri ayırt etmek gereksinimi doğmaktadır. Bu analiz aracında risk unsurları üç seviyeye ayrılmaktadır. Bu seviyeler: düşük, orta ve yüksek risktir. Düşük risk seviyesinden yüksek risk seviyesine doğru kullanılan uygulamalara erişmek ne kadar zorlaşırsa risk seviyesi o kadar azalmaktadır. Yani uygulamalara erişmek ile risk seviyeleri arasında ters orantı olduğu görülmektedir. RiskRanker'da risk analizi iki aşamada yapılmaktadır. İlk aşamada kullanılan uygulamaların risk taşıma durumlarına bakılmaktadır. Orta ve yüksek risk seviyesine sahip olanlar için tehlike unsurunun kapsamı gözden geçirilmiştir. 118,318 adet uygulama kullanılarak bu uygulamaların testi yapılmış ve bunların 3281 tanesinin risk unsuru taşıdığı tespit edilmiştir. 3281 tane uygulamanın da 322 tanesinin sıfırıncı gün kötü niyetli uygulaması olduğu kanısına varılmıştır [16,17].

AppProfiler

Statik analiz araçlarından biri olan AppProfiler'ın yapmak istediği, kötü niyetli uygulama tespiti yapmaktan ziyade uygulamaların yasallığı hususunda uygulama kullanıcılarına ayrıntılı bilgi vermektir. Basit kodlar geliştirerek uygulamaları analiz edebilmek oldukça zordur. Asıl olarak yapılmak istenen, Android API'sinin kullanımının nasıl olduğu veya bu ara yüzün dışarıdan olup olmadığını bilebilmektir. Gizlilik ihlalinin kabul edilmediği durumları tespit etmek gibi bir uğraşı yoktur. Örnek verecek olursak, mobil cihaz çalındığından casus yazılım sayesinde bu fark edilebilir. Diğerleri kullanımdan kullanıma farklılık göstermektedir. Örneğin, bazı kullanıcılar konum bilgisi olan yazılımları bilinçli olarak kullanmamaktadır. AppProfiler'ın esas amacı cihaza kurulan uygulamaların kullanıcılarına uygulamalar hakkında bilgi vermektir [18].

Woodpecker

Kapasite sızıntılarını meydana çıkarmayı basitleştirmek için geliştirilmiş Woodpecker adında bir sistem geliştirilmiştir. Bu araç, daha önce cihaza kurulan uygulamalar üzerindeki veri akışını kontrol edip analizini yaparak, sistemsel açıdan açık veya savunması yapılmamış kullanıcı ara yüzlerinden tehlike içeren bir izin erişimini tespit etmek için kullanılmaktadır. Woodpecker mimarisi Şekil 2.5'te gösterildiği gibidir [19].



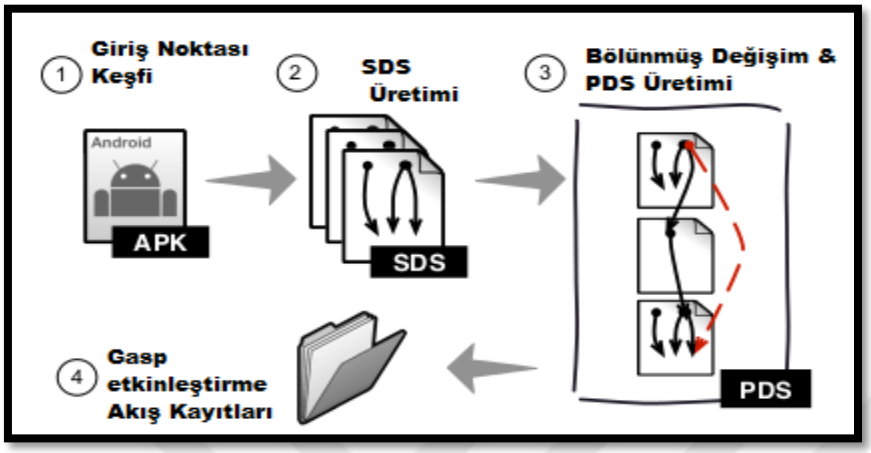
Şekil 2.5. Woodpecker mimarisi [19]

Kapasite sızıntılarının oluştuğu durumları incelemede en başarılı olan birbirinden farklı iki kapasite sızıntısı vardır. İzin istemeyen servislerin bulunduğu veya erişimi bulunan uygulama ara yüzlerinin kesin izinlere erişimini sağlayan kapasite sızıntıları açıktır. Bunun tam tersi olarak Kapalı kapasite sızıntıları ise bu açık ara yüz ve servisler haricindekilere izin vermektedir. Sonuç itibariyle açık kapasite sızıntıları güvenlik anlamında ciddi hatalara neden olur. Kapalı kapasite sızıntıları ise, Android işletim sisteminde izin tabanlı güvenlik anlayışını ortadan kaldırır ve yazılımın sahip olduğu becerileri yanlış algılamasına neden olmaktadır.

CHEX

Android işletim isteminde kurulu olan uygulamalardaki uygulamayı oluşturan parçaları (bileşen) kaçırma açıklıklarını bulmak için hataları otomatik olarak ayıklayabilen CHEX (Component Hijacking Examiner) mimarisi statik analiz yöntemlerinden biridir. CHEX, söz konusu açıklıkların modellenmesini öncelikle bir veri akış analizi yaparak Android uygulamalarının analiz edip sistemin bağımsız olan grafikleri dışındakilere erişebilen testler yapmaktadır. Android'deki farklı yazılım geliştirme metotları sayesinde

kullanıcılara analizi problemleriyle uğraştırmamak için birden fazla giriş noktası modellenir ve bu noktalar bütünlüğü koruyan ve bileşenlerin keşfini kolaylaştıran yeni bir teknik meydana getirmiştir. CHEX'in iş akışı Şekil 2.6'da görüldüğü gibidir [20].



Şekil 2.6. CHEX iş akış diyagramı [20]

CHEX aracında, Dalysis (Dalvik Bytecode Analysis) mimarisi örnek alınmıştır. Söz konusu çerçeve Android uygulamalarının bayt kodlarındaki analizleri kolaylaştıran için birçok tipi desteklemektedir. CHEX'in çalışma metoduna göre, Android uygulama öncelikle veri özetlerine bölünür ve bu bölünen bileşenler birbirlerine bitişik olarak bağlanmaktadır. CHEX'de , 5486 adet Android uygulaması incelenerek, 254 tanesinde potansiyel bileşen kaçırma açıklığı bulunmuştur. CHEX oldukça hızlı bir mimari sunmakta olup bir uygulamada ortalama çalışma süresi 37.02 saniyedir. Bu yeterince hızlıdır. Hızlı olması sayesinde test senaryoları ve uygulama incelemeleri kolaylıkla yapılmaktadır [20].

2.1.3. Literatürdeki diğer analiz yöntemleri

Android İşletim Sisteminde kötü niyetli uygulama tespiti için kullanılan statik analiz yöntemlerinin dışında dinamik ve imza tabanlı analiz dediğimiz iki yöntem daha bulunmaktadır. Bu bölümde literatürde dinamik veya imza tabanlı olarak kötücül yazılım tespiti yapan analiz yöntemleri ve her bir analiz yöntemini kullanan araçlar açıklama ve şekillerle sunulmuştur [11].

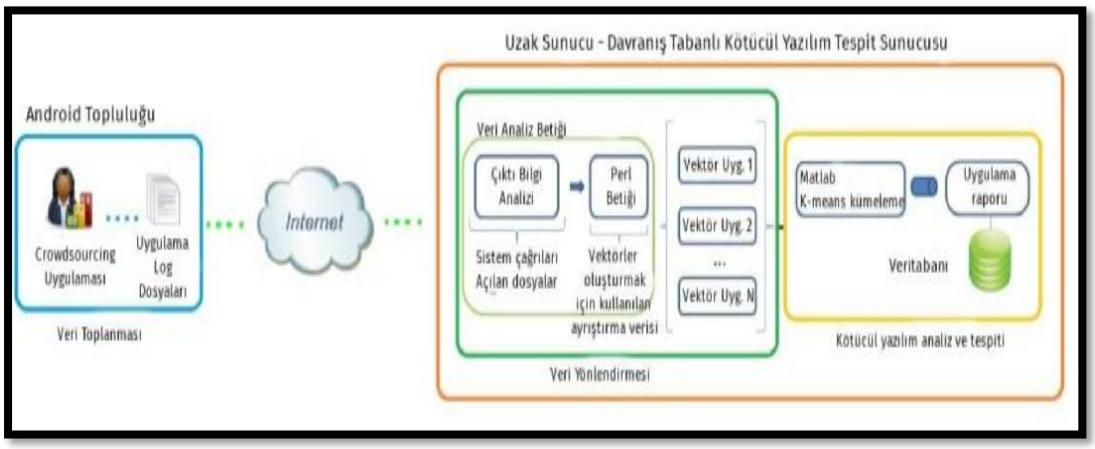
Dinamik analiz yöntemi

Dinamik analiz ve statik analiz yöntemleri arasındaki fark, yapılan analizlerin uygulamanın çalışma anında (runtime) yapılmasıdır. Dinamik analiz yönteminin en önemli avantajlarından biri, statik analiz yöntemini kullanan araçlarla tespit edilemeyecek kadar zor olan eksiklik veya açıkların ortaya çıkarılmasını sağlamasıdır [14].

Dinamik analiz yaklaşımında, uygulamanın gerçek davranışı ekrana yansıtıldığı için bulanıklaştırma yöntemleri nedeniyle bazı problemler yaşanmaktadır. Bununla birlikte bu yaklaşımın diğer dezavantajları da bulunmaktadır. İlk olarak, uygulamanın kötü niyetli fonksiyonları aktif olduğunda (kötü niyetli uygulamanın zarar verdiği zafiyetli uygulamalar) simülasyon zorluğunun bulunmasıdır. İkincisi, her kötü niyetli uygulamada yapılan zararlı faaliyetlerin tespit edilme süresinin ne kadar olduğunun belirsizliğidir [10].

CrowDroid

Crowdroid, sistem çağrılarını toplamak için Strace adı verilen sistem çağrılarının izlenmesine yardımcı olan Linux komutu kullanılarak çalışmaktadır. Sistem çağrılarını yakalama amacı, Android uygulamalar tarafından oluşturulan bütün olaylarla bir çıkış dosyası (output file) oluşturmaktır. Bu dosya uygulama tarafından gerçekleştirilen her sistem çağrısının sayısını, zaman damgalarının yürütme zamanını, açılmış ve erişilebilir dosyalar gibi kullanışlı bilgilere erişmeye yardımcı olur. Kullanılan bu son özellik her Android uygulamanın yürütülme davranışını temsil etmektedir. Croiwdroid mimarisi Şekil 2.7'de gösterilmiştir [21].

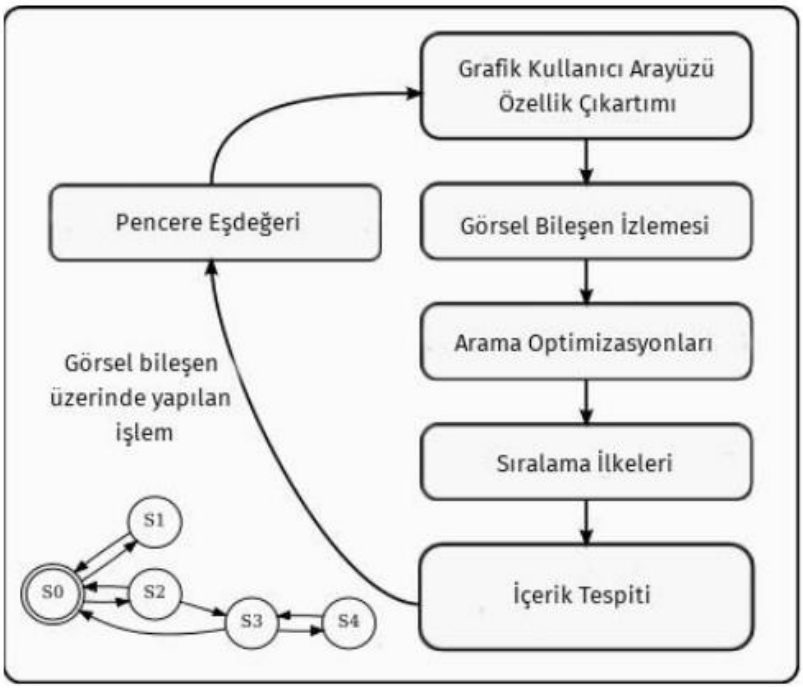


Şekil 2.7. Crowdroid mimarisi [21]

Genel olarak Crowdroid, veri (cihazın bilgileri, kurulu uygulama listesi ve izleme kayıtları) toplanması, veri işleme, kötü niyetli uygulama analizi ve tespiti basamaklarından oluşmaktadır. Şekil 2.7'de de görüldüğü üzere, veri toplama kısmında uygulama ve uygulamanın log dosyaları, kötü niyetli uygulama tespiti yapmak için kullanılan uzak sunucuda ise veri işleme ile birlikte ve kötü niyetli uygulama analizi ve tespiti yapan algoritmaların tutulduğu veri tabanı bulunmaktadır [13,22].

AppsPlayground

Rastogi ve arkadaşları [23] tarafından geliştirilen AppsPlayground isimli analiz aracı ile akıllı telefonlardaki uygulamaların dinamik analizi otomatik olarak yapılmaktadır. AppsPlayground aracının çekirdek katmanı izlemesi, API çağrı takibi, olay tetikleme, tetkik ve maskeleme tekniklerini kullanması verimli bir analiz aracı olduğunu göstermektedir. Şekil 2.8'de AppsPlayground akılcı çalışma prensibi gösterilmiştir [14,22].

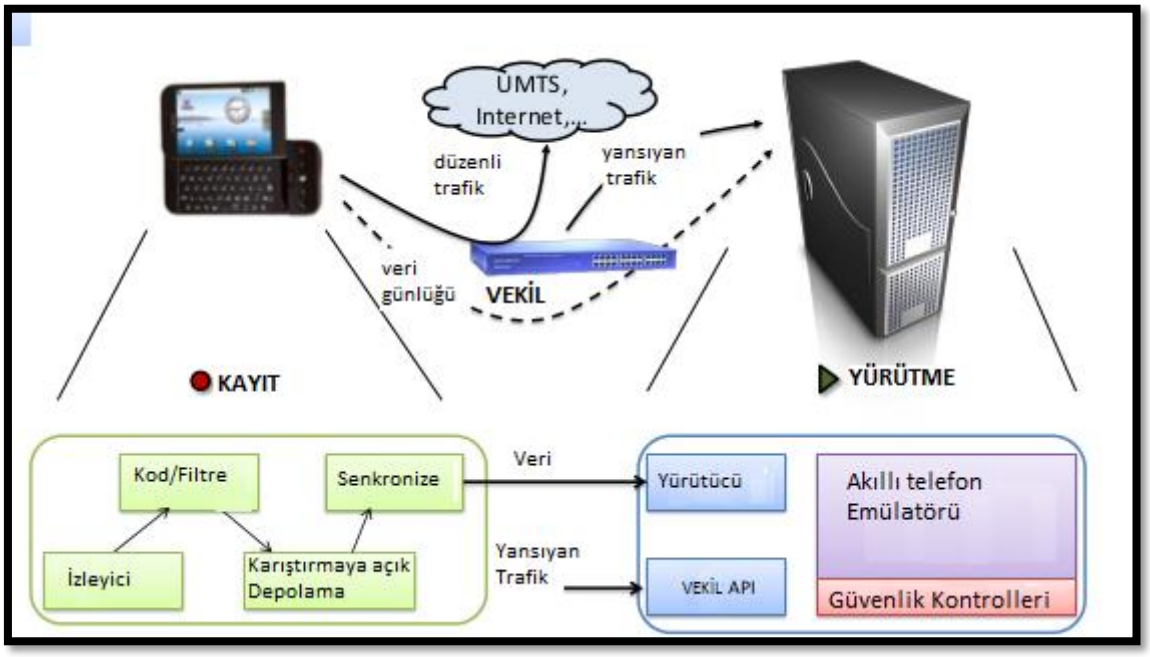


Şekil 2.8. AppsPlayground ön izlemesi [23]

Şekil 2.8'de de görüldüğü üzere, bu analiz aracından otomatik olarak gerçekleştirilen tespit işlemi görsel bileşenin grafik kullanıcı ara yüzündeki özellik çıkarımı ile başlayıp, arama optimizasyonları ve kullanılan sıralama ilkeleri sonucunda içeriğin kötü veya iyi niyetli olduğuna karar verilmektedir.

Paranoid Android

Android işletim sistemine kurulan programlara karşı yapılan güvenlik taramaları akıllı cihazların sanal alemde orijinal kopyalarına uygulanmaktadır. Birden fazla tespit tekniğinin senkron şekilde kullanabilmesi için sunucular kullanılmaktadır. Mobil cihazlar yerine sunucuları kullanan bu araç uygulamalara karşı yaptığı virüs taramalarında yazılımsal olarak düşük maliyet sağlamaktadır [14]. Paranoid Android mimarisinin ön izlemesi Şekil 2.9'de gösterilmiştir [24].

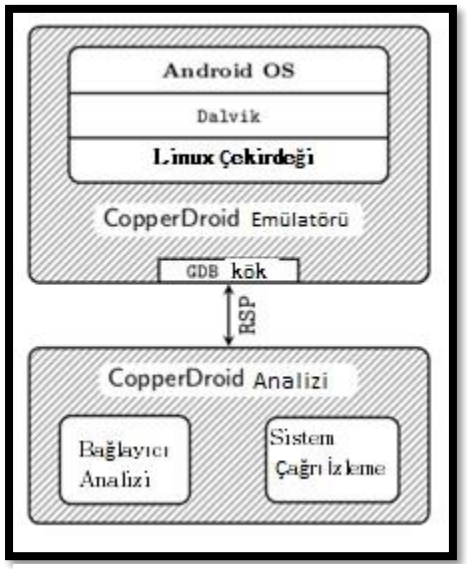


Şekil 2.9. Paranoid Android mimarisinin ön izlemesi [24]

Kullanılan sunucularda bir tane akıllı telefon emülatörü ve güvenlik kontrollerini sağlayan bir yapı bulunmaktadır. Şekil 2.9'daki ön izlemeden de anlaşılacağı üzere, izleme, kodlama ve depolama işlemleri senkronize bir şekilde yapılmaktadır [24].

CopperDroid

CopperDroid, sadece Android işletim sistemine has uygulama özelliklerini QEMU adlı bir emülatör üstünde karakterize eder ve dinamik analiz yöntemini uygulamış olur. Bu uygulama özelliklerinin analizi sistemin çağrıları sayesinde gözlemlenebilmektedir. Sistem çağrılarının talepleri üzerinden gözlemleme yapılıyor da denilebilir. CopperDroid mimarisi Şekil 2.10'de gösterilmiştir [25].

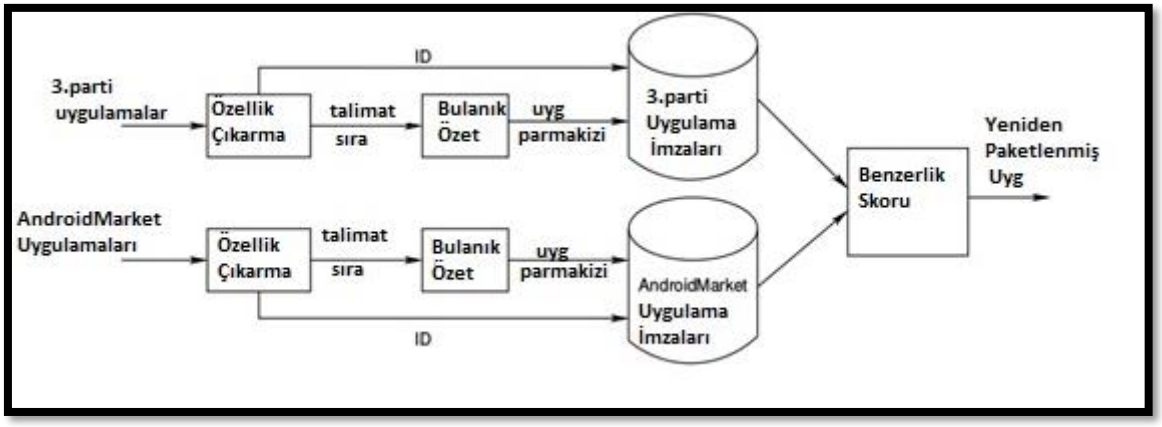


Şekil 2.10. CopperDroid mimarisi [25]

Mimaride de görüldüğü gibi CopperDroid, Java uygulamalarından, Java Native Interface (JNI) üzerinden veya doğal (native) kod geliştirmeleri yaparak özellik(davranış) çıkarımı yapmaktadır [14].

DroidMOSS

DroidMOSS, temel olarak 3 adıma sahiptir. İlk adımda, her uygulamadan uygulamanın talimatlarının ve yazar bilgilerinin bulunduğu iki ana özellik çıkarılmaktadır. Bu iki özellik her uygulamayı benzersiz tanımlamak için kullanılır. İkinci adımda ise, her uygulama için parmak izi üretimi gerçekleştirilmektedir. Bunun sebebi her uygulamanın yüz binlerce talimat içerebiliyor olmasıdır. Buna benzerlik ölçümü yapabilmek için parmak izi gibi daha kısa bir dizi içine yoğunlaşabilmek maksadıyla ihtiyaç duyulmaktadır. Son olarak parmak izine dayalı uygulamalarda; üçüncü adım ise uygulamaların kaynağını anlamaktır. DriodMOSS mimarisinin ön izlemesi Şekil 2.11'de gösterilmiştir [26].



Şekil 2.11. DroidMOSS mimarisi ön izlemesi [26]

DroidMOSS mimarisine bakılarak söz konusu adımlar görülebilmektedir. Özellik çıkarımından sonra çeşitlik talimatlar kullanılarak elimizde bulunan bulanık özet sayesinde uygulama için parmak izi üretimi yapılmaktadır. Bu işlemlerin sonucunda üçüncü parti uygulamaları ve Android yasal marketindeki uygulamaların benzerlikleri kontrol edilmektedir.

İmza tabanlı analiz yöntemi

Kötü niyetli uygulama tespit yöntemlerinden olan statik analiz ve dinamik analiz metotları, sezgisel ve imza tabanlı olarak iki şekilde uygulanmaktadır. Anti virüs üreticileri imza tabanlı yöntemi diğer yöntemlerden daha çok kullanmaktadır. Bu yöntemde, kötü niyetli uygulama benzeri olmayan bir imzayla tanımlanmaya çalışılmaktadır. Zayıflık gösterdiği zamanlarda, imza tabanlı yöntemler bilinenin dışında bir kötü niyetli uygulama kodu için kullanışlı olmamaktadır. Sezgisel tabanlı yöntemde ise, kötü ve iyi niyetli uygulamalar konunun uzmanları veya makine öğrenmesi tekniklerine dayanarak çeşitli kurallarla yapılmaktadır.

Günümüzde, sezgisel tabanlı yöntemleri geliştirmek ve otomatik hale getirmek için sınıflandırma algoritmaları kullanılmaktadır. Bu yöntemler, kullanılan dosyanın ikili kodunu veya uygulamanın yürütme anındaki davranışını yansıtmaktadır. Uygulamaların iyi veya kötü niyetli olduğunu sınıflandırırken çeşitli sınıflandırıcılar kullanılmaktadır [10]. Bu sınıflandırma algoritmalarından ileriki bölümlerde bahsedilmiştir.

Söz konusu yöntem kullanılarak analizi yapılan uygulamalar, imza veri tabanında tutularak

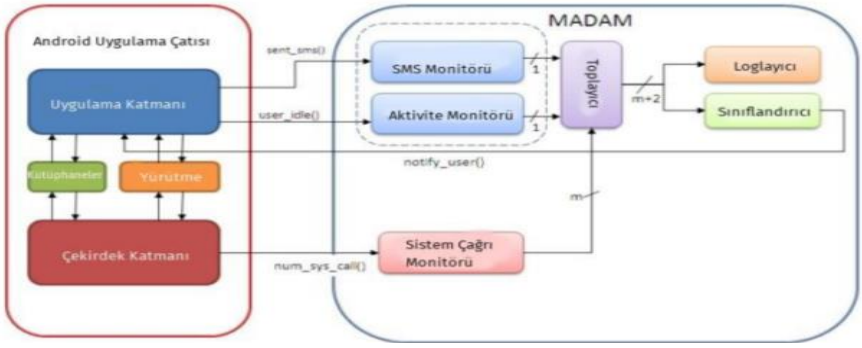
devamlı öğrenmeye yönelik bir eğilim göstermektedir. Bu analiz için merkezi bir sunucuya ve imzaların tutulacağı bir veri tabanına ihtiyaç duyulmaktadır. Bu sunucu analiz aşamalarını gerçekleştirirken, veri tabanı ise analiz sonuçlarının saklanmasını ve sonraki analizler için yeniden kullanabilmemizi sağlar [14].

Tractor Beam

Mobil cihazda çalışabilecek ve merkezi bir sunucuya kablosuz bağlantı alanından bilgi aktarımı yapmak için Tractor Beam [27] adında bir servis geliştirilmiştir. Bu araç sayesinde zaman aralıklarıyla bit değişimleri ve ofset değerleri kablosuz bağlantı üzerinden merkezi sunucuya gönderilir. Tractor Beam, bit değişimlerini gösteren Secure Hash Algorithm-256 (SHA-256) algoritması kullanarak özet değerleri sakladığından disk boyutu konusunda problem teşkil etmemektedir. Merkezi sunucu veri tabanında, Tractor Beam aracılığıyla gönderilmiş bit değişimleri ve ofset değerleri ilişkisel olarak tutulmaktadır. Merkezi sunucu, sistem imajları oluşturmakta ve bu imajları analiz uygulama çatısı aracılığıyla kayıt altına almaktadır. Bu işlemi yaparken uygulama çatısında detektörler ve log kayıtçılar bulunmaktadır. Detektörler, kötü niyetli uygulama tespit tekniklerini barındırarak log kayıtçılar kötü niyetli olduğunu düşündükleri bütün olayları kayıt altına almaktadır [14].

MADAM

MADAM [28] Android'deki kötü niyetli uygulamalar için birden fazla seviyesi olan aykırılık tespit aracıdır. MADAM, kernel (çekirdek) ve uygulama seviyelerinde kötü niyetli uygulama tespiti yapmaktadır. Şekil 2.12'de MADAM fonksiyonel blokları sunulmuştur.



Şekil 2. 12. MADAM fonsiyonel blokları [28]

Bu araçta, kernel seviyesinde kullanıcı aktivitelerini, dosyaya ve belleğe erişimleri, giden ve gelen veri trafiğini, enerji tüketimini ve sensor durumunun izlenmesini sağlayan sistem çağrılarından faydalanılmaktadır. Uygulama seviyesi tespitlerinde çıkarılan özellikler kullanıcının boşta olup olmadığını, gelen-giden SMS sayısını ve Bluetooth, Wi-Fi analizlerini belirlemekte kullanılmaktadır. MADAM aracının seviye özellikleri Çizelge 2.1'de gösterilmiştir.

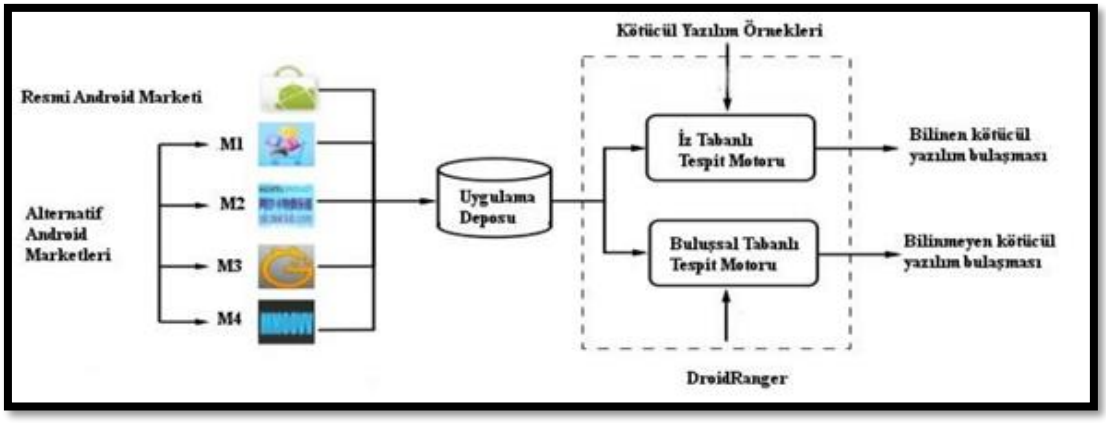
Çizelge 2.1. MADAM seviye özellikleri

Seviye	Özellik
Çekirdek	Sistem çağrıları Çalışma Esnasındaki İşlemler Hafızası dolu olmayan RAM CPU kullanımı
Kullanıcı/Uygulamalar	pasif/aktif kullanım Klavyeyle yapılan işlemler Aranan numaralar Gelen/Giden SMS'ler Bluetooth/Wi-Fi analizleri

MADAM, Çekirdek seviyesindeki özellikleri: sistem çağrıları, çalışma esnasındaki işlemler, hafızası tam kapasite ile dolu olmayan RAM ve CPU kullanımıdır. Kullanıcı/Uygulamalar seviyesindeki özellikler: pasif/aktif kullanım, klavyeyle yapılan işlemler, aranan numaralar, gelen/giden SMS'ler ve bluetooth/Wi-Fi analizleridir. Çekirdek ve Kullanıcı / Uygulama seviyesi tarafındaki özellikleri toplu bir şekilde Çizelge 2.1'de gösterilmiştir.

DroidRanger

Kötü niyetli uygulamaların çeşitli sınıfları bulunmaktadır. Bu sınıflar arasında en yeni olanları tespit edebilmek için DroidRanger [29] aracı geliştirilmiştir. Bu araç, daha önce bilinmeyen kötü niyetli uygulama sınıflarının keşfedilmesinden sonra bunlara filtreleme şemaları uygulanmaktadır. DroidRanger yazılım mimarisi Şekil 2.13'deki gibidir [14].



Şekil 2. 13. DroidRanger yazılım mimarisi [29]

Şekil 2.13'teki mimariye göre resmi Android marketindeki uygulamalar 4 sınıfa ayrılmış ve bunlar uygulama deposuna alınarak iz tabanlı ve buluşsal tabanlı denilen tespit motorlarında kötü niyetli olup olmadıklarının tespiti yapılmıştır.

Kötü niyetli uygulama tespitine yönelik yapılan çalışmaların özellik karşılaştırmalar Çizelge 2.2'de gösterilmiştir.

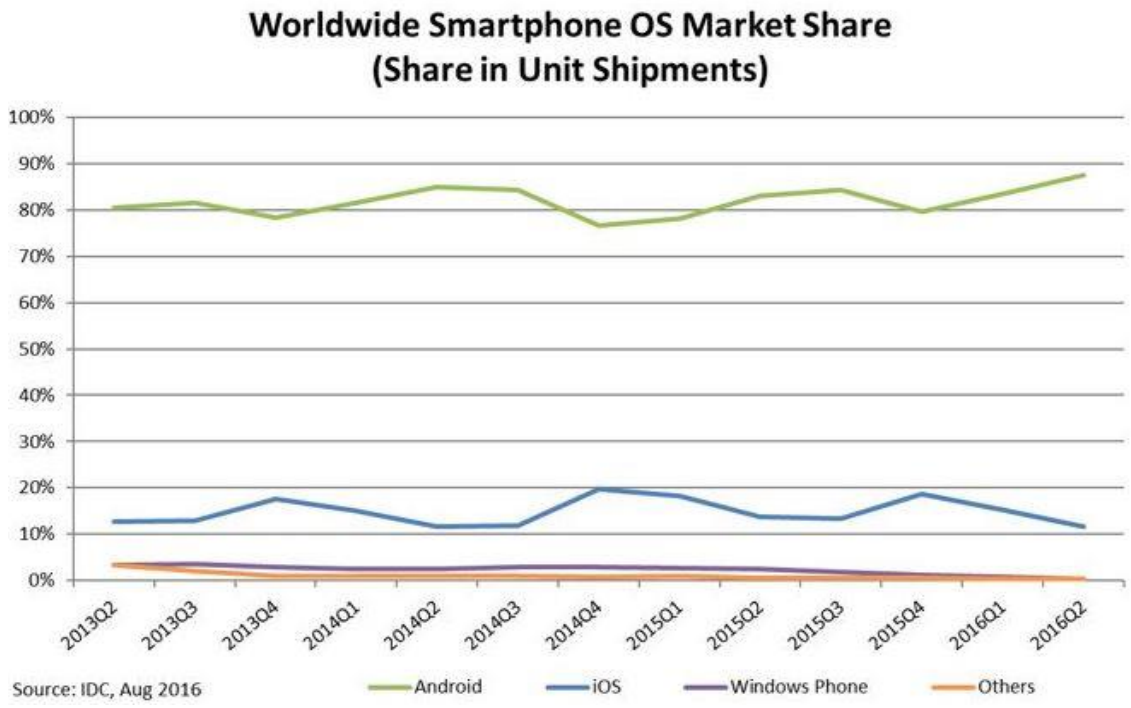
Çizelge 2.2. Kötü niyetli uygulama tespitinde kullanılan araçlar

Kötücül yazılım tespit aracı	Makine öğrenmesi	Manifest incelemesi	API analizi	İzin analizi
DroidBox	X	X	✓	✓
ComDroid	X	✓	✓	✓
Crowdroid	✓	X	✓	✓
DroidRanger	✓	✓	✓	✓
DroidMat	✓	✓	✓	✓
MADAM	✓	✓	✓	✓
Andromaly	✓	X	✓	✓
Dendroid	✓	✓	X	X
DREBIN	✓	✓	✓	✓

Çizelge 2.2'de makine öğrenmesi algoritmasını kullanmayan tespit araçları DroidBox ve ComDroid'tir. Bu çalışmanın kapsamı makine öğrenmesi tabanlı bir sınıflandırma olduğu için bu çalışmalara yer verilmemiştir. Manifest incelemesi yapmayan Crowdroid API analizi ve izin analizi tekniklerini kullandığı için çalışmada bahsedilmiştir. Dendroid aracı API analizi ve izin analizi yapmamaktadır ancak farklı bir statik analiz tekniği kullandığı için çalışmada yer verilmiştir [18].

2.2. Android İşletim Sistemi

Mobil işletim sistemleri, kullanıcıların taleplerine göre farklılık göstermektedir. Açık kaynak kodlu ve Linux tabanlı bir mobil işletim sistemi olan Android işletim sisteminin yapılan çeşitli araştırmalar sonucunda en popüler işletim sistemi olduğu görülmüştür. Şekil 2.14'de sunulduğu üzere, IDC tarafından yapılan araştırma sonucunda Android'in %80'i aşan kullanıcı oranıyla en çok kullanılan mobil işletim sistemi olduğu ortaya çıkmıştır [32].



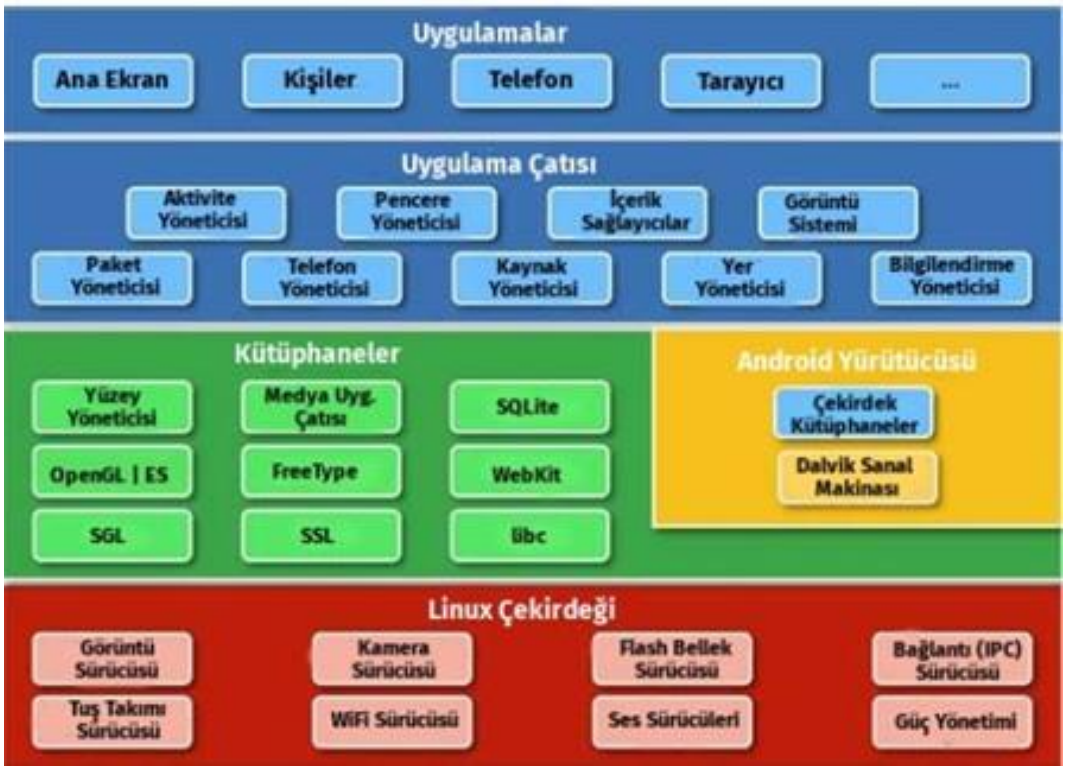
Şekil 2.14. İşletim sistemi kullanım grafiği

Bu grafikte de görüldüğü üzere 2016 yılının ikinci yarısına kadar Android işletim sistemi kullanımının diğer işletim sistemlerine göre daha yaygın olduğu görülmektedir. Android'in popüler olması ve açık kaynak kod yapısı, kötücül yazılımların hedefi olmasına sebep olmuştur. Kötü niyetli saldırıların hedefi olan bir işletim sisteminde tez çalışması

yapmanın toplumsal olarak da faydalı olacağı düşünülerek bu tezde Android işletim sistemi kullanılmıştır. Bu başlık altında Android işletim sistemi genel olarak anlatıldıktan sonra, güvenliğinden, söz konusu işletim sistemini oluşturan bileşenlerden ve tezde en çok kullanılan kaynak kod kullanımıyla ile ilgili gerekli bilgilere yer verilmiştir.

2.2.1. Android işletim sistemine genel bakış

Mobil cihazlarda kullanılan açık kaynaklı ve Linux çekirdeğine yazılmış işletim sistemini Android işletim sistemi olarak tanımlayabiliriz. Sistem donanımsal olarak günümüzde kullanılan tüm işlemciler üzerinde çalışacak biçimde geliştirilmiştir. Android işletim sistemi Java bayt kodlara benzeyen dalvik bayt kodları çalıştırabildiği gibi akıllı telefonlar, tabletler, sanal gerçeklik gözlükleri gibi birçok mobil cihazda çalıştırılabilir. Uygulama geliştirmeyi kolaylaştıran kullanışlı kütüphaneler sağlamaktadır [33]. Mimari olarak 5 uygulama parçası ve 4 katmandan oluşmakta olan Android işletim sisteminin mimarisi ise Şekil 2.15'de görülmektedir [14].



Şekil 2.15. Android işletim sistemi mimarisi

Android işletim sistemi mimarisinde görüldüğü üzere farklı katmanlardan oluşmaktadır. İlk katman olan Linux, işlem, Wi-Fi ve bluetooth, bellek ve USB gibi mobil cihazların yönetim fonksiyonlarını elde ederek cihaz kaynaklarına ulaşılmaktadır. İkinci katmanda kütüphaneler ve Android yürütme zamanı olmak üzere iki uygulama parçası bulunmaktadır. Kütüphaneler, uygulamaların kullandığı Android'in kullanıcıya sunduğu dış kütüphaneleri ifade etmektedir. Android yürütme zamanı ise Android uygulamalarının çalıştığı Dalvik sanal makinesi ile temel Android kütüphanelerini sağlamaktadır. Dalvik sanal makinesi Java sanal makinesine (JSM) benzeyen bir bileşendir. Her uygulamanın kendine has bir dalvik sanal makinesi bulunmaktadır. Android böylece her bir uygulama için ayrı ayrı kum havuzu oluşturmuş olur. Üçüncü katmanda uygulama çatısı bulunmaktadır. Bu katmanda uygulamaların kullanabileceği servisler sağlamaktadır. Dördüncü katman olarak adlandıracağımız en son katmanda ise mobil cihazlar üzerinden telefon görüşmesi yapma, kısa mesaj gönderme, anlık mesajlaşma, internette dolaşma ve elektronik posta (e-mail) gönderme gibi amaçlar için kullanılan Android uygulamaları bulunmaktadır [33].

Java sanal makinesi ile dalvik sanal makinesi birbirleriyle benzerlik gösterir. İkisi arasındaki farklar aşağıda verilmiştir [34]:

- Her ikisinde de java kodları çalışmaktadır. Ancak yürütme esnasında farklılıklar gösterirler. Java uygulamalarında JSM yürütme esnasında çok sayıda sınıf dosyalarından meydana gediği için fonksiyon çağırma yapıldığında hangi sınıf dosyasından çağırma yapılıyorsa o yüklenmektedir. Dalvik sanal makinesinde ise bu durum sınıf dosyalarının bir arada bulunduğu bütün dosyaları kapsayan tek bir dex dosyasından oluşmaktadır.

- JSM yığın (stack) tabanlıdır ancak Dalvik sanal makinesi yazmaç (register) tabanlıdır. Bu yüzden JSM'ye göre daha hızlı işlem yapmaktadır, ama bellek içerisinde tuttuğu yer daha büyüktür.

- En önemli farklarından bir diğeri de komut setleridir. JSM'de 200, Dalvik’de 218 tane işlem kodu mevcuttur. Bu uzunluğun sebebi Dalvik’deki komutların kaynak ve hedef yazmacı (register) içermeleridir. Böylelikle Java bayt koduna sahip programlara göre Dalvik bayt koduna sahip programların komut sayısı daha azdır. Java byte koduna sahip

programlar Dalvik bayt koduna sahip programlara göre %30 daha çok komut içerir ve buna karşın kod boyutu da %35 daha azdır [35].

- Dalvik bayt kodlarındaki veri tipleri kesinlikle ayırt edilemez. Ancak Java'daki bayt kodlarında veri tipleri birbirinden ayırt edilebilmektedir. Bu ayırt edilememe durumu sadece int ve float gibi veri tiplerini kapsamaz. Bu durum farklı veri tiplerinin tanımlandığı dizilerde de geçerli olmaktadır. Örneğin, Java'da bulunan null tipi Dalvik'de yoktur. Bu durumlar için Dalvik'de sabit olarak sıfır değeri kullanılmaktadır.
- Java bayt kodunda nesnelerin kıyası yapılırken nesne referansı ve null kıyaslaması yapılmaktadır Dalvik bayt kodda ise nesneler önce int veri tipine dönüştürülmekte ve int ve sıfır kıyaslaması yapılmaktadır.
- Dalvik bayt kodları kaynak kodlarına kesinlikle benzemezken Java bayt kodları yapısal anlamda çok benzemektedir.
- Java programlarında her sınıfın kendine özgü sabitler havuzu bulunmaktayken, Dalvik programlarında sınıflar ortak sabitler havuzuna sahiptir.

Android işletim sistemi uygulamaları, sisteminin ulaştığı kaynakları son kullanıcıya sunmaktadır. Android uygulamaların işletim sisteminde ulaşım hakkı sadece kendi dizininde bulunan dosyalara verilmektedir ve önceliği düşük olacak şekilde çalışmaktadır. Her uygulama için bir kum havuzu bulunur ve her uygulama kendi havuzunda çalışmaktadır. Böylelikle uygulamaların diğer uygulamalarla etkileşimi engellenmiş olmaktadır. Android uygulamalarını mobil cihazlara yüklemeden önce kullanıcıya liste halinde uygulamaların kullanmak istediği izinleri gösterilmektedir. Böylelikle kötü niyetli uygulamaların diğer uygulama ve dosyalara ulaşımlarını engellenmektedir. Ancak gerekli müsaadeler kullanıcı tarafından verilir ise uygulamalar birbiriyle etkileşebilmektedir. Android platformuna uygulamaların yasal bir şekilde indirilebilmesi için Play Store [36] adı verilen market kullanılmaktadır. Ancak bu market dışında da çeşitli marketlerden uygulama indirmek ve kurmak da Android'de izne tabii olmaktadır.

Android platformuna resmi market veya resmi market dışından uygulamalar indirilebildiği için bu uygulamaların ne kadar güvenli oldukları önem arz etmektedir. Android işletim sistemi varsayılan olarak bu özelliği kapatmış, kullanıcı isterse bu özelliği aktif hale getirmektedir. Android, uygulamaların indirilmesinde resmi olmayan marketlerden indirilmesini tavsiye etmemekle birlikte bunu kullanıcının inisiyatifine bırakmaktadır [37].

Android'in resmi marketi dışında bir marketten uygulamak indirilmek istendiğinde mobil cihazımızda bilinmeyen kaynaklardan uygulama yüklemeye izin ver seçeneğini aktif bırakmamız gerekmektedir.

2.2.2. Android işletim sistemi güvenliği

Android işletim sistemini kullanırken bazı kötü niyetli uygulamaların da engellenmesi gerekmektedir. Bu tür engellemeleri yapabilmek için söz konusun platformda iki tane esas alınacak güvenlik mekanizması vardır. Bu mekanizmalar aşağıdaki şekildedir:

Android market güvenliği

Market güvenliği, uygulamaların yüklendiği market tarafından mobil cihazımıza indirilmeden önce yapılan analiz ve tespitleri içermektedir. Bunu yapabilmesi için iki farklı metot bulunmaktadır. İlk metot da uygulamaları statik ve dinamik olarak adlandırılan analiz yöntemleriyle kötü niyetli olup olmadığı belirlenmektedir. Google Bouncer denilen servis tarafından Android'de söz konusu işlem yapılabilmektedir [37]. Bu servis yasal kabul ettiğimiz Android market içerisinde var olan ve yeni eklenen programlar arasında yüklenen programların iyi niyetli olup olmadığına kara vermektedir. İlk olarak uygulamanın bilinen zararlı yazılımlara benzeyip benzemediğini araştırmaktadır. Buna ek olarak uygulamanın hareketine göre önceden analiz edilmiş uygulamalar ile karşılaştırarak muhtemel riskleri çıkarmaktadır. Böylelikle uygulamalar hem statik hem de dinamik olarak analiz edilmiş olur. Ancak Google Bouncer servisi varlığına karşın resmi Android marketinin zararlı yazılımlar üzerinde güvenlik düzeyinin yüksek kesin olduğundan bahsedilemez. Bunun sebebi ise Android uygulama geliştiricilere kaynak kodlarının bütününü açtığı için uygulamaların, güvenlik mekanizmalarını geçmesi daha da kolaylaşır. Ayrıca Android haricinde IOS gibi platformlar, yasal marketleri dışındaki marketlerden uygulama indirmeye izin vermeyip kendi yasal marketlerine ek güvenlik sağlamış olurlar. Ancak Android'de yasal olmayan marketlerin kullanımı konusunda bir kısıtlama bulunmamakta ve bu sayede üçüncü parti denilen uygulamalar indirilebilmektedir. Fakat bu işlem de mobil cihazımızı karşılaşılabilecek saldırılara karşı zayıflatmaktadır [38].

Başka bir yöntemse; programları geliştiren kişiye göre özel olarak imzalamaktır. Bu sayede program kim tarafından geliştirilmiş, kim tarafında yayına girmiş belli olmakta ve

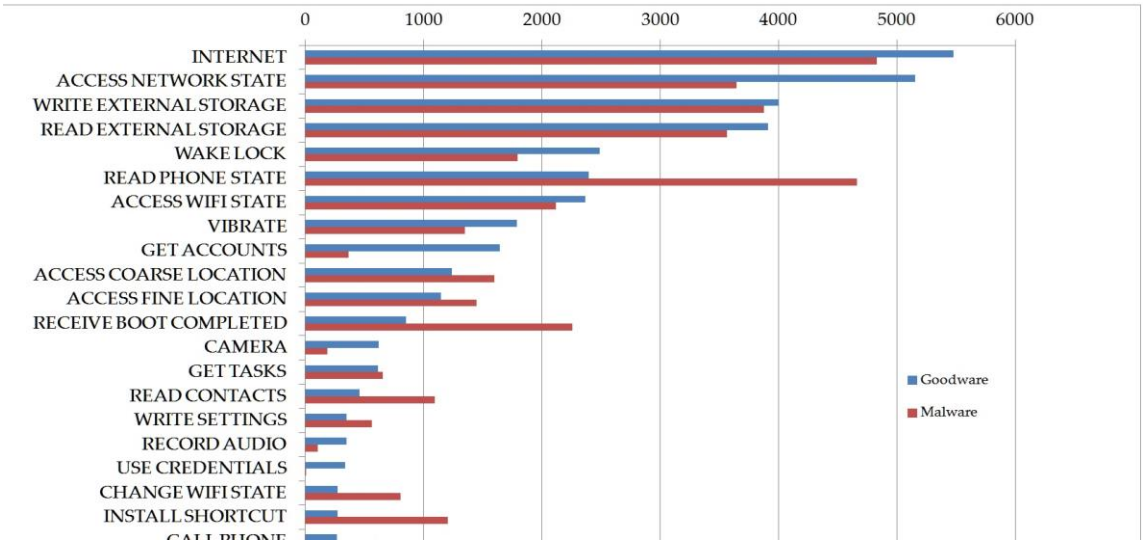
programın bütünlüğü de kolay denetlenebilmektedir. Ancak literatürdeki bazı çalışmalarda imzalamanın kolaylıkla zafiyetlerinin bulunabileceği görülmektedir [39].

Android platformu güvenliği

Android'deki uygulamaların, cihaza kurulması sırasında ve kurulum tamamlandıktan sonra çeşitli güvenlik mekanizmaları Android platformu tarafından uygulanmaktadır. Bu güvenlik mekanizmaları aşağıdaki gibidir:

İzinler

İzinler, uygulamaların kullanılan işletim sisteminde hangi kaynaklardan faydalandıklarını belirlemektedir. İndirmek istediğimiz uygulama, mobil cihazımıza kurulmadan önce indirilmek istendiğinde uygulamanın istediği izinler listelenmekte ve kullanıcıya uygulamaların kullanacağı izinler gösterilmektedir. Bu sayede, uygulama mobil cihaza indirilmeden önce kullanıcı hem istek dışı bir işlem yapmamış olmakta hem de kurulan uygulama cihazımızın hangi kaynaklarına erişiyor onun bilgisine sahip olunmaktadır. Android KitKat 4.4. versiyonunun ve API seviye 19'da tanımlı sistemin kaynaklarına erişmesi için gerekli 145 tane izin bulunmaktadır. Drebin veri setinde bulunan kötü niyetli uygulamaların en çok kullandığı 20 izin Şekil 2.16'da gösterilmektedir [40].



Şekil 2.16. Drebin veri setinde bulunan zararlı yazılımların en çok kullandığı 20 izin

Kum havuzunda çalıştırma (Sandboxing)

Yapılan bir çalışmada, uygulamaların, yazılım geliştirme safhasında gerçekleştirilen yanlışlardan ve kötü doküman haline getirmekten ötürü çok fazla sayıda izin talep ettikleri ortaya çıkmıştır. Ancak bu durum tamamıyla kullanıcıya bağlıdır ve kullanıcı uygulamanın erişmek istediği kaynakları bize sunan belgeyi kabul ettiği sürece herhangi bir güvenlik sağlanmamış olmaktadır. Yani bu durum tek başına yeterli değildir [41].

Android'deki her uygulamanın kendine ait kullanıcı tanımlama numarası bulunmaktadır ve her uygulama farklı işlevlerde çalıştırılmaktadır [33]. Bu durum uygulamaların çekirdek seviyesinde kum havuzunda çalışmasını sağlamaktadır. Söz konusu havuz, uygulamaların diğer uygulamalara ait bilgilere erişimini engellemekte ve izinsiz herhangi bir işlem yapılmasını önlemektedir.

Uygulamaların birbirleriyle etkileşimleri

Java dili sayesinde, Android'deki uygulamalar arasında kaynak paylaşma yapabilmek için bazı yapılar kullanılmaktadır. Bu yapılar iş parçacıkları, soketler ve paylaşımı olan belleklerdir. Ancak bu durum hataya sebep olmakta ve yönetimsel anlamda zorluklar yaşamaktadır. Bu nedenler söz konusu platform için karşılıklı iletişimi sağlamak için bir mekanizma yapılmıştır. Buna ek olarak, bu platform birbirinden farklı uygulamaların aynı amacı gerçekleştirmesi durumlarına engel olacak şekilde geliştirilmiştir. İşlemler arası haberleşme (IPC) dediğimiz bu yapı sayesinde uygulamalar birbirlerinin çalıştırdıkları işleri çalıştırabilmektedir [33].

Çekirdek güvenliği

Çok sayıda kullanıcı tarafından kullanılan çekirdeklerde hatalar ve açıklıklar pek çok insan tarafından araştırılıp, bu hatalar ve açıklıkları düzeltmek için geliştirmeler yapılmaktadır. Linux çekirdeği de bu çekirdeklerden biridir. Bu çekirdeği kullanan kişilerin birbirlerinden ayrı tutulması en temel özelliklerinden biridir. Örneğin bir kullanıcı tarafından kullanılan kaynaklar ve çeşitli dosyalar başka bir kullanıcı tarafından kesinlikle kullanılmamaktadır. Android de bu çekirdeğe sahip olduğu için aynı güvenlik özelliklerine sahiptir. Söz konusu

çekirdeğin Android işletim sistemine olan etkileri şu şekilde sıralanabilir: kullanıcı bazlı izin modeli, işlemlerin birbirinden ayrı tutulması ve gelişmiş IPC mekanizması [33].

2.2.3. Android uygulamalarını oluşturan parçalar

Android'deki uygulamalar, Java ile geliştirilir ve uygulamaların kodları class dosyalarında tutulur. Uygulamaların ara yüzleri xml dosyaları ile hazırlanmaktadır. Manifest dosyası adı verilen dosyada uygulamadaki servis ve aktivitelerin ne tür izinler kullanmak istediği ve uygulamanın başlama noktasının hangi aktiviteyle olacağı bilgileri bulunmaktadır. Tüm bu dosyalar (resim ve kaynak dosyaları dahil) *dex* dosyasında tutulmaktadır. Dosyaların tümünün derlenmesiyle *apk* dosyası oluşturulmaktadır. Android'in çalışan dosyasına verilen isim apk'dır. Android uygulamaları, yürütme zamanında bir main fonksiyonu veya başlangıç noktası içermemektedir. Manifest dosyasında ana bileşen olarak belirtilen bileşen başlangıç noktası olur. Bu dosyada uygulamanın önemli bilgilerini içermektedir. Manifest dosyası, uygulama ile ilgili önemli bilgileri barındıran bir dosyadır. Android, yukarıdaki cümlelerde de anlatıldığı üzere bu dosyanın aldığı bilgilere göre uygulamaları çalıştırmaktadır. Android uygulamaları 4 temel bileşenden oluşmaktadır: yayın alıcılar, içerik sağlayıcılar, servisler ve aktiviteler [41].

Yayın alıcılar

Yayın alıcılar (broadcast receiver), sistem veya uygulama olayına bağlanmayı sağlamaktadır. Olay etkinleştiğinde Android olaya bağlı olan uygulamayı bilgilendirmektedir. Böylelikle uygulamalar telefona mesaj veya arama geldiği anda, cihaza USB ve kulaklık takıldığında vb. olaylardan haberdar olmaktadır. Örneğin; ACTION_UMS_CONNECTED durumu etkinleştiğinde Android bu olaya yayın alıcılar ile bağlanan uygulamaları bilgilendirmekte ve bu uygulamalar da ona göre işlevlerini yerine getirmektedirler. Yayın alıcı olan nesnesin hangisi olduğu da Manifest dosyasında belirtilmektedir.

İçerik sağlayıcılar

İçerik sağlayıcılar (content provider) uygulamaların veri tabanındaki verilere erişimini sağlamak için kullanılmaktadır. Uygulamalar arası veri tabanı paylaşımı için de

kullanılmaktadır. Bu sayede bir içerikten birden fazla uygulama faydalanmış ve dağıtım yapılmış olur. Her içerik sağlayıcının içindeki bilgiye göre bir kimliği vardır ve bu kimlikle içerik sağlayıcıya bağlanılmaktadır. Manifest dosyasında içerik sağlayıcı bilgileri de belirtilmektedir.

Servisler

Herhangi bir kullanıcı ara yüzüne sahip değildirler ve arka planda uzun süre çalışabilmektedirler [42]. Bir uygulama parçasında çalışan işlem, bu parça kapanınca servisler sayesinde işlemlerini yürütmeye devam eder. Bu sayede uzun soluklu işlemlerin çalışması durumunda uygulamada takılmalar olmaz ve çalışmaya devam eder. Diğer uygulama parçaları uzak prosedür çağrısı ara yüzü aracılığıyla servisleri kullanmaktadır. Böylece diğer uygulama parçaları tarafından çağrılabilmektedirler.

Aktiviteler

Sistem kaynaklarını kullanabilmek maksadıyla kullanıcı ara yüzlerini tanımlamayı sağlar. Sistem kaynaklarına örnek olarak: arama yapmak, mesaj çekmek, fotoğraf çekmek vb. verilebilir [43]. Bir uygulama birden fazla aktiviteden meydana gelir ve bir anda bir aktivite aktif olabilmektedir. Bir aktivite çalıştığında diğeri durdurulur ve durdurulan aktivite yığına atılır. Böylece o aktiviteye dönüş yapılınca uygulama kaldığı yerden devam edebilmektedir.

2.2.4. Kaynak kod kullanımı

Android işletim sisteminde kötücül yazılımları diğerlerinden ayırt edebilmek için mevcut olan kötü niyetli uygulamaların kaynak kodlarına bakmak gerekir. Bu nedenle burada tersine mühendislik yaklaşımlarından bahsedilecektir [44]. Bu iki yaklaşım aşağıdaki gibidir:

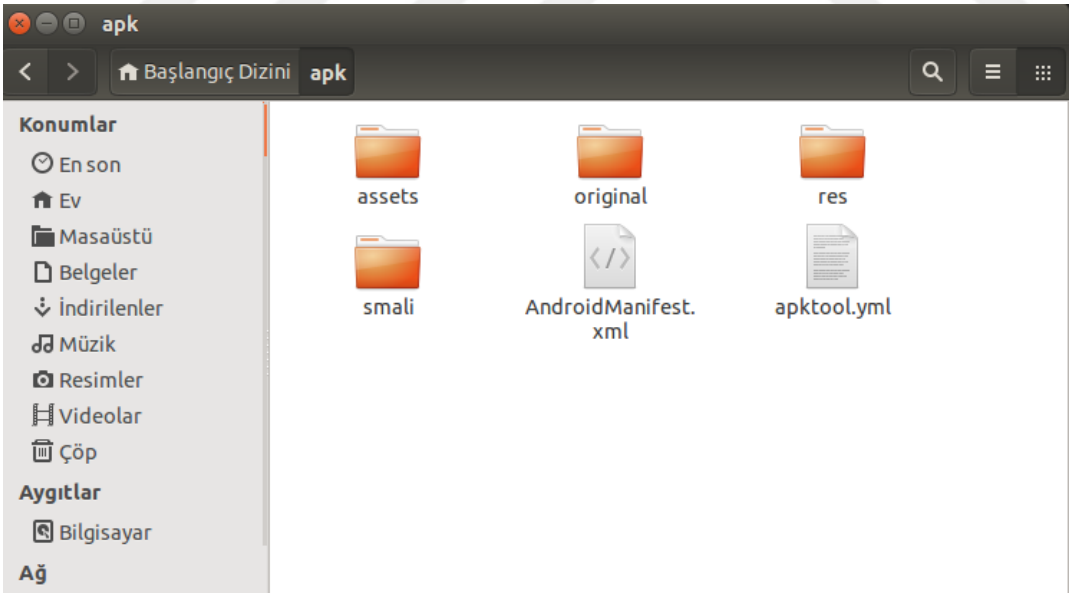
Kaynak koda dönüş

Kaynak koda dönüş (decompiling), Apk dosyalarından java dosyalarına erişmek demektir. Bu işlem için öncelikle apk dosyası dex2jar [45] aracı kullanarak dönüşüm yapılmaktadır.

Dönüştürülen jar dosyasından class dosyalarına erişilmektedir. Son olarak da Java için geliştirilmiş tersine derleme aracı [46] olarak ne kullanılıyorsa class dosyaları java dosyalarına dönüştürülmektedir.

Tersine derleme

Tersine derleme (disassmbling), dalvik bayt kodlarının bulunduğu dex dosyalarından smali dosyalarını çıkarmak demektir. Bu işlemi gerçekleştirmek için *apktool* [47] aracı kullanılmaktadır. Jad [46] ve Jeb [48] gibi tersine derleme yapan bazı araçlar kullanarak Android uygulamalarından Java kaynak kodlarına erişilebilmektedir ancak paketleme esnasında sorunlar yaşandığı için apk dosyasına tekrardan dönmek imkansızlaşmaktadır. Dalvik bayt kodlarına apk dosyası içinde yer alan dex dosyasından erişilebilmektedir. Dalvik bayt kodlar bir araya gelerek smali dosyalarını oluşturmaktadır ve apktool [47] sayesinde bu smali dosyalarına erişilebilmektedir. smali dosyalarında değişiklik yapıp yeniden apk dosyasına dönülebilmektedir. Tersine derleme görmüş bir Android uygulamasındaki dosyalar Şekil 2.17'de görülmektedir.



Şekil 2.17. Bir android uygulamasının tersine derlenmiş hali

Şekil 2.17'de görüldüğü gibi bir Android uygulaması genel anlamda smali kodlarından (smali), resim, video ve xml dosyaları gibi kaynak dosyalardan (res), üçüncü parti kütüphanelerden (lib), manifest dosyasından (AndroidManifest.xml) ve diğer dosyalardan (assets) oluşmaktadır.

İlk yaklaşımda java dosyalarından apk dosyalarını üretmek problemli olduğu için bu çalışmada ikinci yaklaşım olan tersine derleme yaklaşımı kullanılmış ve smali dosyaları kullanılarak kötü ve iyi niyetli uygulamalar sınıflandırılmıştır.

2.3. Makine Öğrenmesi ve Sınıflandırma

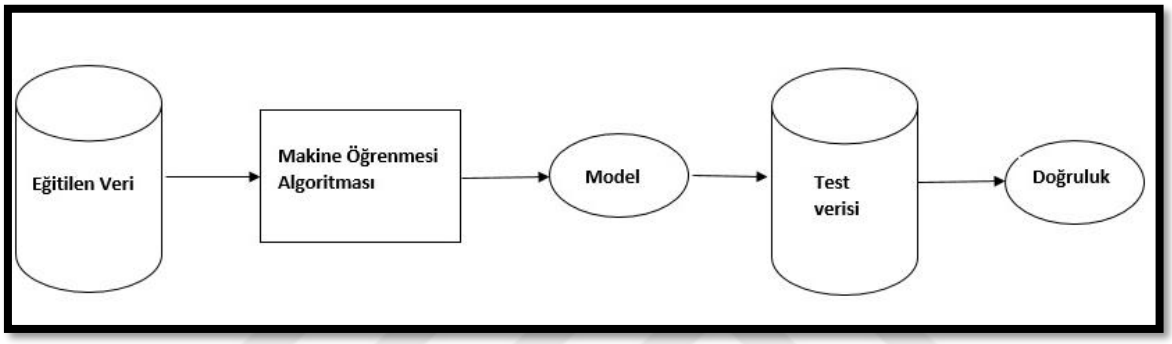
Makine öğrenmesi sınıflandırmayı çeşitli görevlere ayırmaktadır. Bu görevler şunlardır: binary, çoklu sınıf, çoklu etiketleme ve hiyerarşik görevlerdir. Aşağıda, sınıflandırmanın bu sahasında kullanılan 24 performans ölçümünün sistematik analizinden ve değerlendirme kriterlerinden bahsedilmektedir. Odaklanılan nokta hesaplama maliyeti ve zamandan bağımsız olarak daha iyi sınıflandırma tespitidir. Verinin karakteristik özellikleriyle bağlantılı olan karmaşıklık matrisindeki değişiklikler göz önünde bulundurularak, ölçümün değerini etkilemeyecek değişimlerin tipleri analiz edilmiştir. Sonuç olarak, sınıflandırma problemindeki ilişkili tüm etiket dağılım değişiklikleriyle ilgili ölçü sabit değer niteliği (*measure invariance*) oluşturulmuştur. Metin sınıflandırma işlemi konuyu birkaç örnek incelemesiyle desteklemiş ve çalışma esnasında makine öğrenmesine yönelik güncel gelişmeler dikkate alınmıştır. Çalışma ile, bazı durumlarda, yeni sabit nicelik özelliklerinin durumu, tekdüzelik özellikleri ekleyerek ve çoklu sınıflandırma, çoklu etiket ve hiyerarşik ölçümleri göz önünde bulundurarak genişletilmiştir [49].

Makine öğrenmesi algoritmaları teorik ve deney tabanlı veya her ikisi açısından değerlendirilebiliyor olsa da, deneysel değerlendirmeler, algoritma değerlendirilmesinde en fazla kullanılan yöntem konumundadır. Çalışmada değerlendirme için, algoritma mukayesesi ve belirli domaindeki algoritmaların uygulanabilirliği olmak üzere iki amacın ayrımını yapmaktadır. Deneysel mukayeseler, daha çok algoritmaların farklı verisetlerine uygulanması ve algoritmaların uygulandığı sınıflandırmaların performanslarının değerlendirilmesiyle yapılmaktadır. En çok kullanılan sınıflandırma ölçütü doğruluktur. Öğrenme setlerinin bazılarında pozitif örneklerin doğru tanımlanması diğerlerinde

olmadığı kadar önemli olabilirken; negatif örneklerin doğru tanımlanması ya da veri ve sınıflandırma etiketleri arasındaki tutarsızlıklar daha çok dikkat çekici olabilmektedir [49].

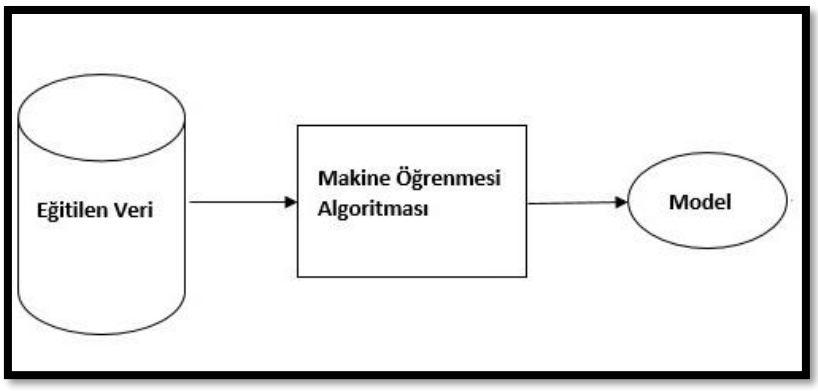
2.3.1. Sınıflandırma için kullanılan görevler

Tez çalışmasında sınıflandırma yapılacağı için danışmanlı bir öğrenme yaklaşımı kullanılmıştır. Danışmanlı makine öğrenmesi, etiketli gözlemleri içermektedir. Tezde kullanılan danışmanlı makine öğrenmesi modelini ayrıntılı olarak açıklaması Şekil 2.18'de gösterilmiştir.



Şekil 2.18. Danışmanlı makine öğrenmesi modeli

Tezde Şekilde de görüldüğü gibi statik analiz metodu kullanarak oluşturduğumuz veri setinin eğitilmiş hali bulunmaktadır. Eğitilen veri makine öğrenmesi algoritmaları kullanılarak danışmanlı öğrenme modelimiz oluşturulmuştur. Daha sonra test olarak kullandığımız veriler aracılığıyla algoritmaların doğruluk tespitleri yapılmıştır. Doğruluk ile ilgili kavramlara daha sonraki başlıklarda yer verilecektir. Danışmansız makine öğrenmesi modeli kullanılmış olsaydı test verileri kullanılamayacak ve bir doğruluk değeri hesaplanmayacaktı. Danışmansız makine öğrenmesi de Şekil 2.19'daki gibidir.



Şekil 2.19. Danışmansız makine öğrenmesi modeli

İki şekilden de anlaşıldığı gibi tez çalışmasındaki ihtiyacı karşılayan model denetimli makine öğrenmesi modelidir. Bu modelde, algoritmanın eğitim ve test sürecinde veri etiketlerine erişim izni verilmektedir. x_1, x_2 , $x_{3....}$, x_n şeklindeki veri setlerinin, önceden tanımlanmış C_1, $C_{2....}$, C_n sınıflarına atandığını düşünürsek sınıflandırma aşağıda belirtilen görevlerden birine verilir:

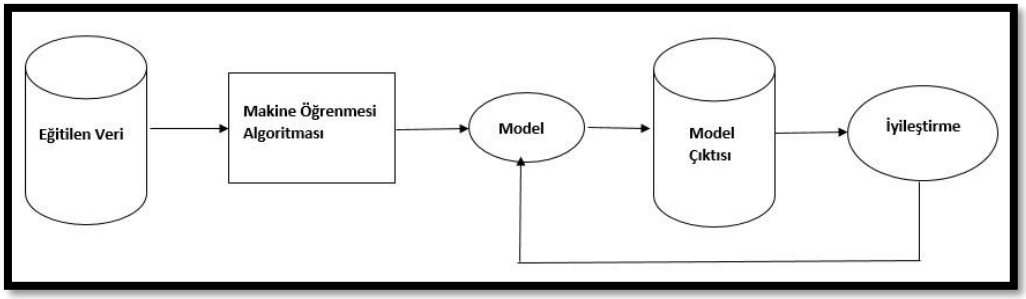
Binary: En sık kullanılan sınıflandırma görevidir ve girdi nesnesi, örtüşen iki sınıftan sadece birine atanır. Sınıflar, iyi tanımlanmış (*oy etiketleri gibi*), belirsiz (*görüş içerikli etiketler gibi*) ya da her ikisi şeklinde (*medikal, metin gibi*) olabilmektedir.

Çoklu-Sınıf: Girdi nesnesinin atanabilmesi için sınıflar arasında örtüşme görülmemesi gerekmektedir ve bu sınıflardan sadece birine atanmaktadır. Çoklu sınıf sorunları, desen tanımada popüler olan 3-sınıf veri setinde iris tiplerinin tespit edilmesi sorununu içermektedir [50]. Binary durumuna gelince çoklu sınıf kategorize etme öznel veya nesnel, iyi tanımlanmış ya da belirsiz olabilmektedir.

Çok Etiketlik: Girdi nesnesi birbiriyle örtüşmeyen birkaç sınıfa atanabilir. Örnekler, maya genlerinin fonksiyonelliğinin sınıflandırılmasını [51], büyük verilerden olay yeri tespiti elde edilmesini [52] veya metin veri tabanı hizalanmasını [53] ve makine tercümesinde kelime sıralamasını içermektedir. Binary, çoklu sınıf ve çok etiketli olma problemleri, kategorilerin izole edildiği ve birbirleri arasındaki ilişkilerin öneminin yok sayıldığı düz sınıflandırmalara şekil vermektedir.

Hiyerarşik: Girdi nesnesi sınıfların alt sınıflara ya da gruplara bölündüğü sınıflardan sadece birine atanır. Hiyerarşi belirlenir ve sınıflandırma süresince değiştirilemez. Hiyerarşik sınıflandırma, düz sınıflandırmalara dönüştürülebilir. Örneğin reuter koleksiyon verisi çoklu sınıf [54], çok etiketli [55] ya da hiyerarşik [56] olabilmektedir [49].

Danışmanlı ve danışmansız makine öğrenmesi modellerinin dışında destekli makine öğrenmesi modeli de bulunmaktadır. Destekli makine öğrenmesi modeli de Şekil 2.20'de gösterilmiştir.



Şekil 2.20. Destekli makine öğrenmesi modeli

Şekilde görülen destekli öğrenme modelinde eğitilen verilerin makine öğrenmesi algoritmalarıyla ortaya çıkan model çıktılarında iyileştirme yapılır ve bu iyileştirme model her kullanıldığında bütün model çıktılarına yapılmaktadır.

2.3.2. Sınıflandırma için performans ölçümleri

Bir sınıflandırmanın doğruluğu, doğru negatif sınıflara dahil olmamış, doğru tanımlanmış sınıf örnekleri (*doğru pozitif*) ve yanlış bir şekilde yanlış pozitif sınıflara atanmamış veya yanlış negatif olarak belirlenmemiş örneklerin doğru tanımlanma sayısının hesaplanması ile değerlendirilebilmektedir. Bu dört sayı (*tp*, *tn*, *fp*, *fn*) binary sınıflandırma için Çizelge 2.3'de gösterilen karışıklık matrisini oluşturur.

Çizelge 2. 3. Karışıklık matrisi

Veri Sınıfı	Pozitif Sınıflandırma	Negatif Sınıflandırma
Pozitif	Doğru Pozitif (*tp*)	Yanlış Negatif (*fn*)
Negatif	Yanlış Pozitif (*fp*)	Doğru Negatif (*tn*)

Karışıklık matrisindeki değerlere bağlı olarak binary sınıflandırma için en çok kullanılan ölçümler doğruluk (accuracy), kesinlik (precision), duyarlılık (recall), f-ölçütü (fscore), belirlilik (specificity) ve AUC (Area Under The Curve)' dur. Bu ölçüm değerlerine ilişkin tablo ölçüm ve değerlendirme kriterleriyle birlikte Çizelge 2.4'de gösterilmiştir.

Çizelge 2. 4. Binary Sınıflandırma için kullanılan ölçüm ve değerlendirme kriterleri

Ölçüm	Değerlendirme Kriteri
Doğruluk	Sınıflandırmanın toplam geçerliliği
Kesinlik	Sınıflandırma tarafından verilen pozitif etiketlerle veri etiketlerinin uyumu
Duyarlılık	Pozitif etiketlerin tanılanması için sınıflandırmanın geçerliliği
F-Ölçütü	Verinin pozitif etiketlileriyle sınıflandırma tarafından verilenlerin arasındaki ilişki
Belirlilik	Negatif etiketleri tanımlayan sınıflandırmanın nasıl geçerli olduğu
AUC	Yanlış sınıflandırmadan kaçınmak için sınıflandırmanın yeteneğin

Çoklu sınıflandırma için kullanılan ölçümler ve değerlendirme kriterleri Çizelge 2.5'de sunulmuştur. Tüm sınıflandırmanın kalitesi genellikle, C_1, C_2 ... sınıfları için aynı ölçümlerin hesaplamalarının ortalaması veya kümülatif doğru pozitif, doğru negatif, yanlış pozitif, yanlış negatif değerlerini sağlamak için sayıların toplamı ve performansın ölçümlerinin hesaplanması şeklinde iki yolla değerlendirilmektedir.

Çizelge 2. 5. Çoklu sınıflandırma için kullanılan ölçüm ve değerlendirme kriterleri

Ölçüm	Değerlendirme Kriteri
Average Accuracy	Sınıflandırmanın her bir sınıf için ortalama geçerliliği
Error Rate	Sınıf başı ortalama hata oranı
Precisionμ	Her bir metnin kararlarının ortalaması hesaplanmışsa, sınıflandırmaya ait sınıf etiketlerinin uygunluğu
Recallμ	Her bir metnin kararlarının ortalaması hesaplanmışsa, sınıf etiketlerinin tanımlanması için sınıflandırmanın geçerliliği
Fscoreμ	Her bir metnin kararlarının ortalamasına bağlı olarak verinin pozitif etiketlileriyle sınıflandırma tarafından etiketlenen veriler arasındaki ilişki
PrecisionM	Sınıflandırmaya ait sınıf etiket değerlerinin her bir sınıf için ortalama değeri
RecallM	Sınıf etiketlerinin tanımlanması için her bir sınıfın ortalama geçerliliği
FscoreM	Her bir sınıfın ortalamasına bağlı olarak verinin pozitif etiketlileriyle sınıflandırma tarafından etiketlenen veriler arasındaki ilişki

Çok konulu sınıflandırmanın kalitesi, kısmi ya da bütün halde sınıf etiketi karşılaştırılması üzerinden değerlendirilmektedir [57]. Başlangıçta tüm sınıf ve etiketlerinin eşit olduğu varsayılır. Çoklu sınıflandırma ölçümleri, önem değerinden ya da sıralarından bağımsız

olarak doğru ve yanlış sınıf tanımlamalarının sayısını tutar. Çok konulu sınıflandırma için kullanılan ölçümler ve değerlendirme kriterleri Çizelge 2.6'da gösterilmiştir.

Çizelge 2. 6. Çoklu sınıflandırma için kullanılan ölçümler ve değerlendirme kriterleri

Ölçüm	Değerlendirme Kriteri
Exact Match Ratio	Tam olarak sınıflandırılmış her bir metnin ortalaması
Labelling Fscore	Kısmi karşılaştırılmalı her bir metnin ortalama sınıflandırılması
Retrieval Fscore	Kısmi karşılaştırmalı her bir sınıfın ortalama sınıflandırılması
Hamming Loss	Sınıf başı toplam hatalı her bir örneğin ortalaması.

Hiyerarşik sınıflandırma örneklerinin ölçüm değerleri ve değerlendirme kriterleri Çizelge 2.7'de gösterilmiştir. Bu ölçümlerin hiyerarşik problemleri kapsadığı düşünülmektedir ve hem soy hem de ata performansını değerlendirebildiği belirlenmiştir [56].

Çizelge 2. 7. Hiyerarşik Sınıflandırmada kullanılan ölçüm ve değerlendirme kriterleri

Ölçüm	Değerlendirme Kriteri
Precision↓↑	Sınıflandırmanın verdiği alt sınıf etiketleriyle ilgili alt sınıf etiketleri üzerindeki pozitif anlaşma
Recall↓	Veri tarafından verilen alt sınıf etiketleriyle ilgili alt sınıf etiketleri üzerindeki pozitif anlaşma
Fscore↓	Sınıflandırma tarafından verilen ve verinin pozitif alt sınıf etiketleri arasındaki ilişki
Precision↑	Sınıflandırma tarafından verilen süper sınıf etiketleriyle ilgili olarak süper sınıflar üzerindeki pozitif anlaşma
Recall↑	Veri tarafından verilen süer sınıf etiketleriyle ilgili olarak süper sınıflar üzerindeki pozitif anlaşma
Fscore↑	Sınıflandırma tarafından verilen ve verinin pozitif süper sınıf etiketleri arasındaki ilişki

Bütün sınıflandırma türlerine göre ölçüm ve değerlendirme kriterlerinden bahsedilmiş ve bu çalışmada binary sınıflandırma için kullanılan ölçüm ve değerlendirme kriterleri kullanılmıştır.

2.3.3. Makine öğrenmesi algoritmaları

Tez çalışmasında sınıflandırma yapmak için kullanılan makine öğrenmesi algoritmaları bu başlık altında toplanmıştır. Bu algoritmalar aşağıda verilmiştir:

K-NN algoritması

K-NN (K-Nearest Neighbour) tüm durumları içeren ve benzer ölçüm temelli yeni durumları sınıflandıran basit bir algoritmadır. K-NN algoritması 1970'den beri istatiksel tahmin ve motif tanıma gibi birçok uygulamadan kullanılmaktadır. Parametrik olmayan sınıflandırma yöntemidir ve genelde daha az yapılanmış NN (structured less NN) ve yapı tabanlı NN (structure based NN) teknikleri olarak iki ayrı tipte sınıflandırılırlar [58]. Structured less NN tekniğinde tüm veri eğitim ve test örnek verisi olarak sınıflandırılır. Eğitim noktasından örnek noktasına kadar mesafe hesaplanır/değerlendirilir ve en düşük mesafe değeri en yakın komşu olarak adlandırılır. Structure based NN tekniği verinin yapılarıyla ilgilenmektedir. En yakın komşu sınıflandırması daha çok tüm nitelikler devam olduğunda kullanılır [59].

K-NN algoritması kısaca iki adımda anlatılabilir:

- Bilinmeyen örneğe en yakın K eğitim örnekleri bulunur,
- Daha çok bu K örnekleri için oldukça yaygın sınıflandırma toplanır.

Tez çalışmasında kullanılan weka uygulamasının sınıflandırıcılarından biri olan K-NN algoritmasında sınıflandırma weka.classifiers.lazy.Ibk şeması seçilerek yapılmaktadır.

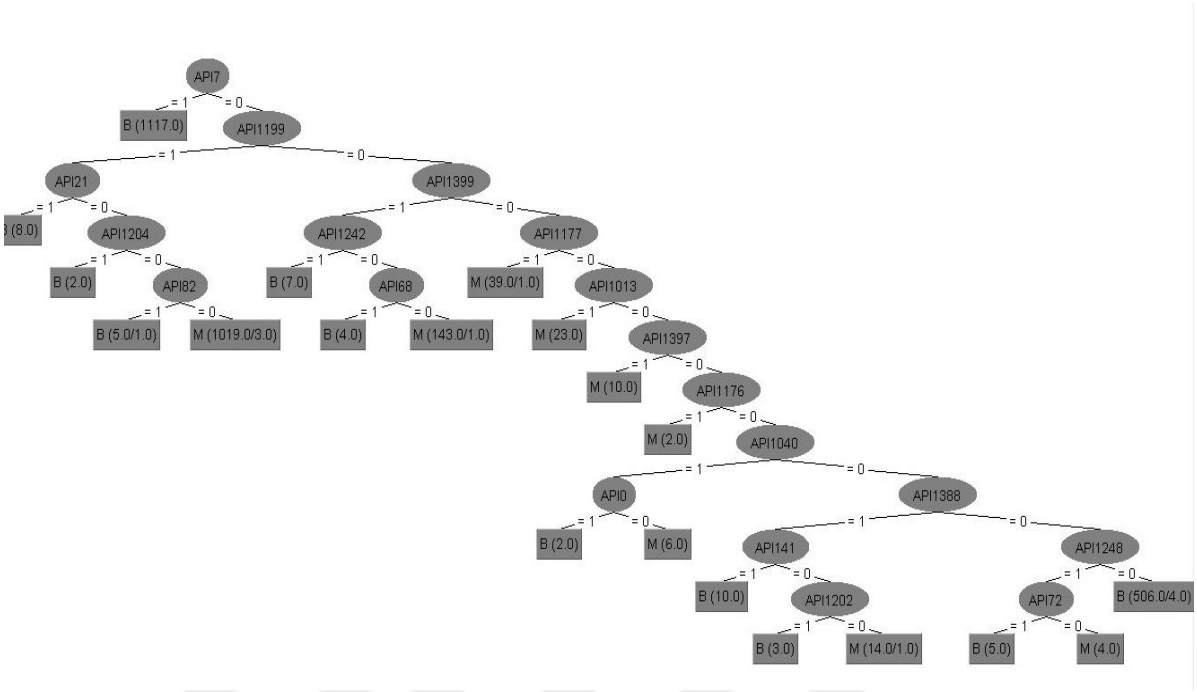
Karar ağaçları

Murthy 1998 yılında karar ağaçlarına yönelik bir inceleme sunmuştur. Karar ağaçları özelik değerlerine göre sıralanarak oluşturulan ağaçlardır. Bir özelliğin bir değerine göre veri kümesi iki alt kümeye bölünür. Bölmede kullanılan özellikler ve değerler karar düğümüne yerleştirilir. Tanımlamayı ve tahmin etmeyi kolaylaştıran bir model olan ağaç görünümündeki karar ağacı modeli, karar alıcıya karar alırken hangi etkenlerin göz önünde bulundurulması ve her bir faktörün kararın farklı boyutları ile eskiden nasıl ilişkili olduğunun belirlenmesi hususunda yardımcı olurlar. Karar ağacı kategorize edilmiş özellikler üzerinde uygulandığı zaman iyi bir çözüm yolu olabilir. En önemli

özelliklerinden biri de kolay anlaşılmasıdır [60].

C4.5 popüler bir karar ağacı algoritmasıdır ve ID3 karar ağaç algoritmasının güncellenmiş hali olup Quinlan tarafından geliştirilmiştir. C4.4 algoritması, ağacın kökünü test etmek için gerekli özelliğin seçimi ile başlar. Her bir özellik eğitim örnekleriyle kıyaslanarak uygunluk açısından istatiksel olarak hesaplanır. En uygun olan özellik, kök boğum noktasını test etmek için kullanılır. Yavru boğum noktaları ise, eğitim örneklerine bağlı kalınarak her bir potansiyel özellik değerlerinin araştırılmasından sonra oluşturulur. Bu işlem her bir yavru boğum noktasına uygun özellik değeri seçilene kadar devam eder. Bu algoritmanın önemi, test edilecek boğum noktaları için toplanan bilgiler ışığında doğru özelliğin seçiliyor olmasıdır [61].

Diğer popüler karar ağacı algoritmalarından biri de J48 karar ağacı algoritmasıdır. Verinin kullanımını en etkin şekilde gerçekleştiren algoritmalardan biridir. Çalışmada kullanılan verilerin özelliklerini çıkarırken seçme işlemini otomatik olarak gerçekleştirmektedir. En kazançlı olduğu durumda eldeki örnekleri bölebilen yinelemeli algoritmalardan biridir. J48 karar ağacının yapısı, denenecek verileri bölerken ağaçtaki en iyi değişkeni seçme süresince yukarıdan aşağıya bütün düğümler için yapacak şekildedir. Yani zayıf gördüğü dallar budanmaktadır. Bunun anlamı, bir karar ağacı algoritmasının amacının veri keşfine çıkmak değil de veriler için en basit olacak şekilde sınıflandırma modeli oluşturabilmektir [62]. Tez çalışmasında kullanılan J48 karar ağacının ağaç yapısı Şekil 2.21’deki gibidir.



Şekil 2.21. Tezdeki J 48 karar ağacı yapısı

Tez çalışmasında kullanılan Weka uygulamasının sınıflandırıcılarından biri olan j.48 algoritmasında sınıflandırma weka.classifiers.trees.j48 şeması seçilerek yapılmaktadır.

Naive Bayes algoritması

Naive Bayes sınıflandırıcı, gözetimli sınıflandırma için en basit model olup, makine öğrenmesi ve veri madenciliği için etkili öğrenme algoritmalarından biridir. Naive Bayes sınıflandırıcı, sınıf değişkeni C ve ona ait tahmin edici özellikler $X=\{X_1, X_2, \ldots X_n\}$ ile bir yapı tanımlamaktadır. Tahmin edici özellikler farklı özelikler seti ve devamlı özellikler seti olarak ikiye ayrılmaktadır. Seçici naive bayes ise tahmin yapmak için sadece o sınıfa ait özellikleri kullanan halidir [63].

Maliyet hassasiyetli metotların amacı, 0 (isabet) ve 1(kayıp)'den farklı hatalı/yanlış sınıflandırma maliyetlerini hesaba katmalarıdır. Bu metotlar, sınıflandırma doğrulamaları ve sınıflandırma maliyetleriyle ilgilenirler. CS-SNB-Accuracy (Cost Sensitive-Selective Naive Bayes) sınıflandırma doğrulama değerini maksimum seviyeye, CS-SNB-Cost (Cost Sensitive-Selective Naive Bayes) ise sınıflandırma maliyetlerini minimum seviyeye sahip oldukları değişkenlerin yaptığı katkı ile sağlarlar. Kısaca ilk yaklaşımda (CS-SNB-Accuracy), sınıflandırma doğrulama oranını artıran değişken modele eklenir, ikinci

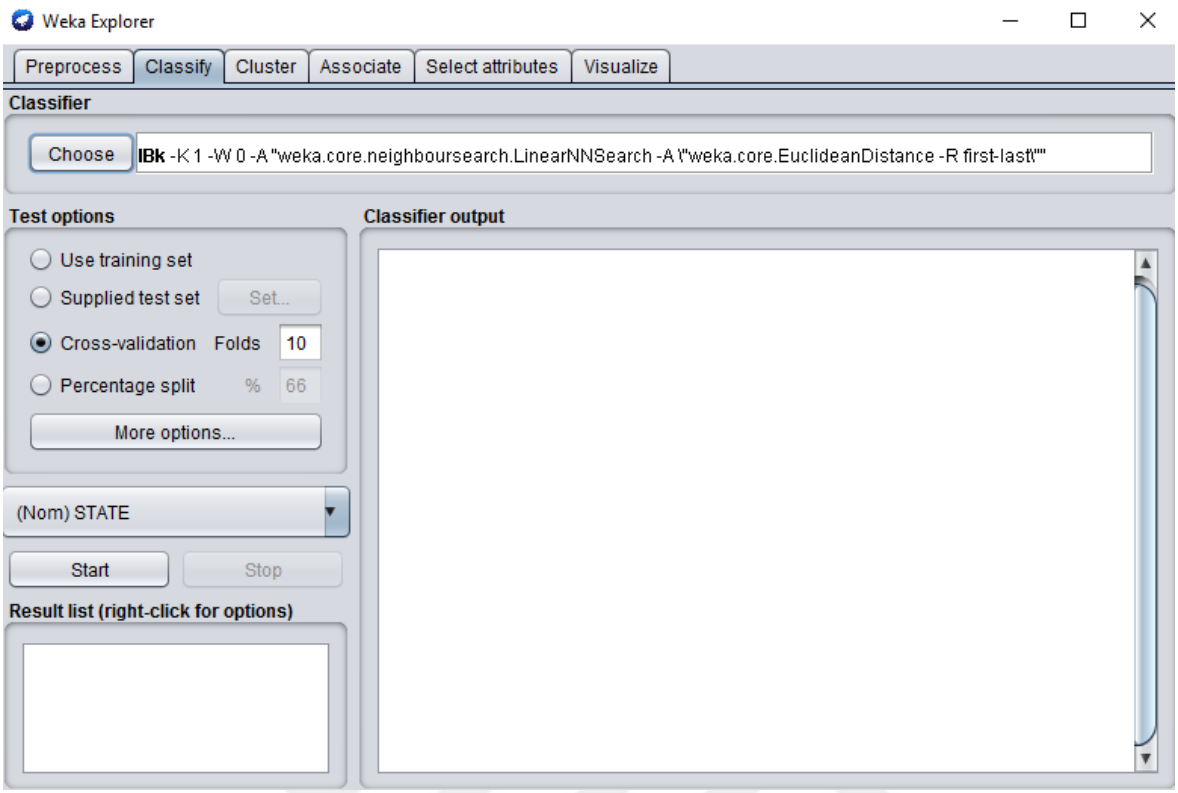
yaklaşımda ise, asıl sınıf ve tahmin edilen sınıf arasındaki hatalı/yanlış sınıflandırma maliyetini düşüren değişken modele eklenir [63].

Burada modellerin maliyeti ve doğrulama değerini hesaplamak için kullanılmaktadır ve yöntemin aşamaları şu şekildedir:

Başlangıçta modelde herhangi bir tahmin edici değerler bulunmamaktadır. Algoritma, ardışık karşılaştırmalar için sonuçlanan modellerin hatalı sınıflandırma maliyetlerini saklar. Doğrulama eşik değeri hesaplanır. Her bir adımda algoritma özel değerlerin modele ait olup olmadığını kontrol eder. Kullanılmamış değişkenlerin modele eklenip eklenmemesi, eğitim veri küme setlerindeki ortalama maliyet değerlerine bakılarak karar verilir. İlk olarak tahmin edilebilir sınıf hesaplanır, sonra minimum maliyet değerli sınıfı seçmek için her bir sınıfın eşik değer olasılıkları yeniden düzenlenir. En sonunda yeniden düzenlenmiş sınıf ve asıl sınıf, her bir adımda seçili kullanılmamış değişkenleri hesaba katarak modellerin maliyetlerini hesaplamak için kullanılır. Tüm modellerin maliyetleri hesaplandıktan sonra maliyet açısından en düşük değere sahip değişken modele eklenmek için seçilir. Eğer yeni modelin maliyeti mevcut modelin maliyetinden düşükse ilgili değişken modele kalıcı olarak eklenir [63].

Tez çalışmasında kullanılan weka uygulamasının sınıflandırıcılarından biri olan Naive Bayes algoritmasında sınıflandırma weka.classifiers.bayes.NaiveBayes şeması seçilerek yapılmaktadır.

Tez çalışmasında tercih edilen denetimli makine öğrenmesi modeli için kullanılan algoritmalarda k-katlamalı çapraz doğrulama (k-fold cross validation) adı verilen test seçeneği seçilmiştir. Weka'da bu seçim algoritma tercihi yapıldıktan sonra yapılmaktadır. Weka uygulamasındaki test seçenekleri Şekil 2.22'de gösterilmiştir.



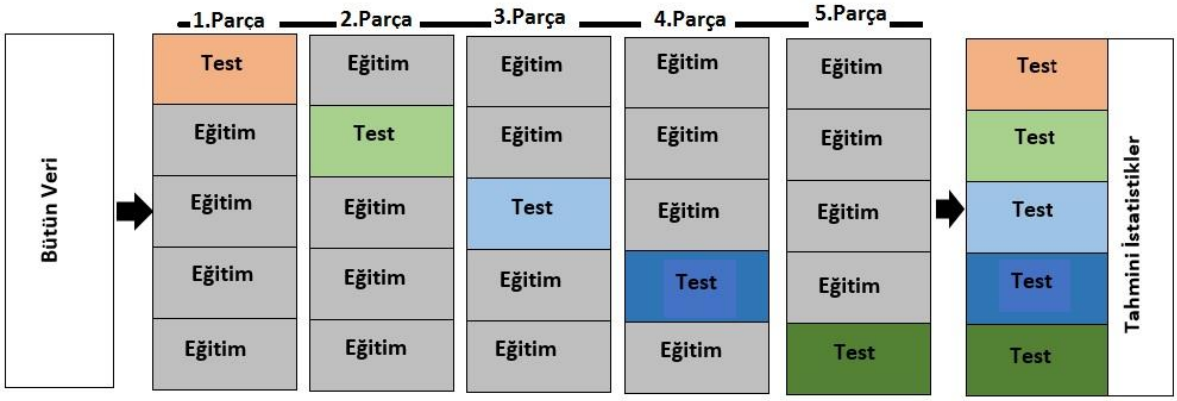
Şekil 2.22. Weka uygulamasındaki test seçenekleri ekranı

Tezde katlama değeri 10 olarak varsayılmış ve bütün makine öğrenmesi algoritmaları için bu değer kullanılmıştır. 10 katlamaları çapraz doğrulama işleminde veri kümesi 10'a bölünmektedir. Bölünen parçalardan %33'ü test için kullanılmakta ve geri kalanı eğitim için kullanılmaktadır. En sonunda toplama işlemi yapıldığından hangi parçadan başladığımız önemli değildir [80].

Sistemi çalıştırdıktan sonra, sınıflandırma (makine öğrenmesi) algoritması çalıştırılarak sonuçlar elde edilir. Bu işlem her parça için yapılmaktadır ve elimizde 10 sonuç olmaktadır. Genel başarı veya genel hata elde edilen 10 sonucun ortalaması olmaktadır [80].

10 katlamalı çapraz doğrulama (10-fold cross validation) için 10 kere model çalıştırılır. Her adımda veri kümesinin 1/3'ü kadar daha önce test için kullanılmamış parçası, test için kullanılırken geri kalan kısmı eğitim için kullanılmaktadır [82].

Tezde kullanılan katlama değeri varsayılan (default) 10 olarak seçilmiştir. Konuyu daha basit haliyle özetlemek gerekirse 5 katlamalı çapraz doğrulama için çizilen Şekil 2.23 aşağıdaki gibidir.

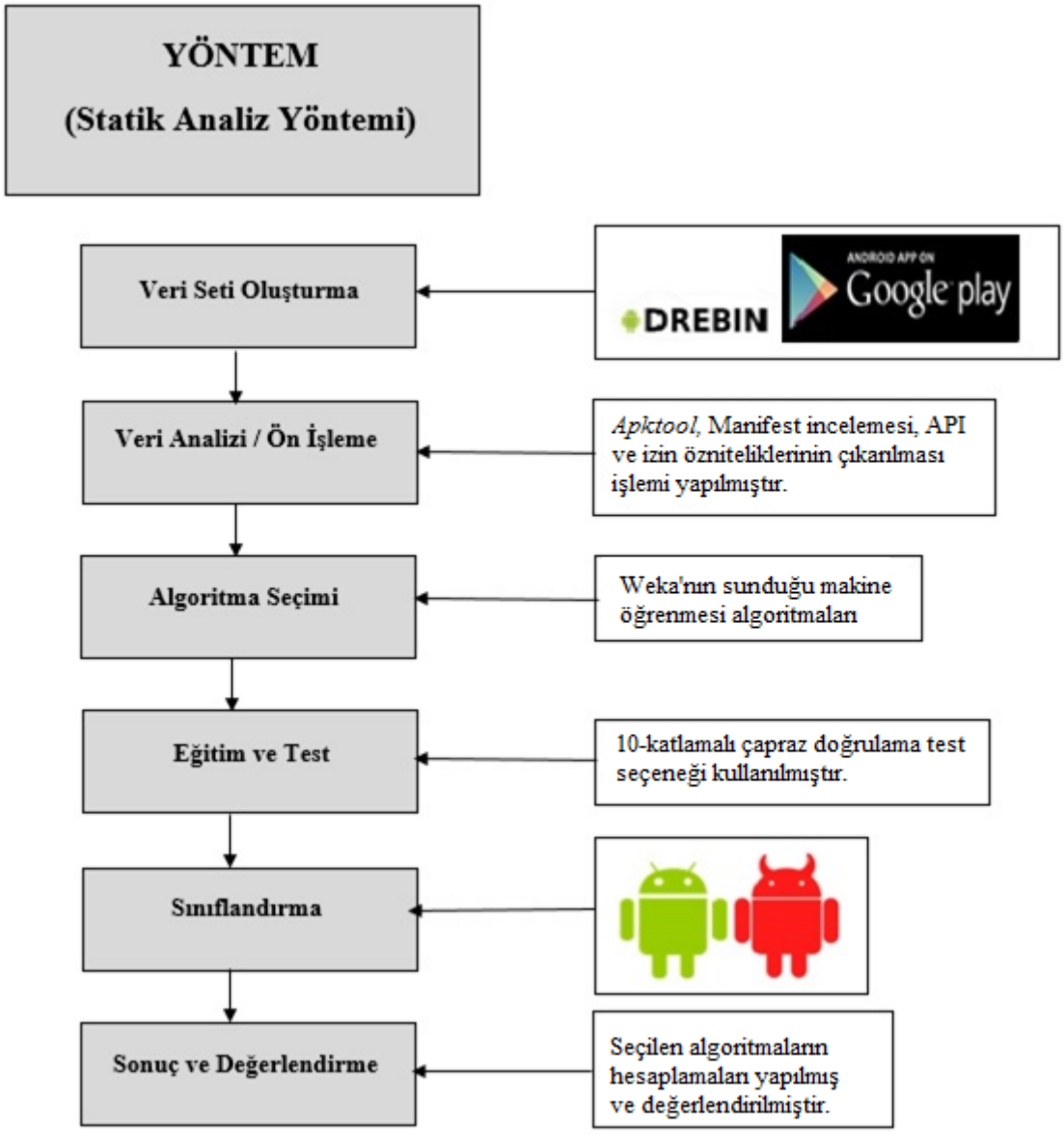


Şekil 2.23. 5-katlamalı çapraz doğrulama modeli

Şekildeki gibi bütün veri kümesini 5 katlamalı çapraz doğrulama yapmak için 5'e böldük ve bir parçası test için geri kalan kısmı eğitim için ayrıldı. Kullanılmayan her parça kalan parçalarda test verisi olarak kullanılmaktadır. İşlem sonunda elde edilen test sayısı 5 kere bu işlem yapıldığı için çıkan sonucun ortalaması alınmaktadır. Bunun sonucunda çıkan sonuç sınıflandırma algoritmasının doğruluk değerini vermektedir [82].

3. SİSTEM MİMARİSİ

Bu bölümde Android işletim sistemi için geliştirilmiş uygulamaların statik analiz yöntemiyle sınıflandırılması sisteminin mimarisi ayrıntılı bir şekilde anlatılmıştır. Sistem mimarisi Şekil 3.1'deki gibidir.



Şekil 3. 1 Tez çalışmasının sistem mimarisi

Şekil 3.1'deki sistem mimarisinde yapılan tez özetlenmiştir. Bu mimariye göre tez çalışmasının basamakları güvenli ve zararlı uygulamalardan oluşan veri setinin oluşturulması, veri analizi/ön işlemesi yapmak maksadıyla uygulamalarına tersine

derlenmesi, izin ve API özniteliklerinin çıkarılması, sistemin makine öğrenmesi algoritmaları kullanılarak eğitimi ve sınıflandırmadır.

3.1. Güvenli ve Zararlı Uygulama Veri Setinin Oluşturulması

Tez için kullanılan Android versiyonu Android 1.1'dir. Güvenli olan uygulamalar için veri seti oluşturmak maksadıyla; Android'in yasal olarak kullanılan marketi Google Play'den faydalanılmıştır [64]. Bu marketten otomatik olarak en çok indirilmiş olan uygulamaların güvenilir olduğu düşünülerek Android Market kullanılarak bir program kodlanmıştır [65]. Android Market kullanılarak kodlanan program için uygulamaları indireceği mobil cihaza verilen bir Android ID kullanılmaktadır. Bu ID ile sisteme giriş yapılmış ve uygulamalar indirilmiştir.

Güvenli uygulama veri seti için; Play Store üzerinden kodlanan bir program aracılığıyla otomatik olarak indirilme sayısı 3.000.000 ve üzerinde olan 1400 adet uygulama indirilmiştir.

Zararlı yazılımlar için oluşturulan veri seti için Android kötücül yazılımlarında yapılan araştırmaları teşvik etmek ve farklı tespit yaklaşımlarını kıyaslayabilmek için halka açık olarak kullanılabilen Drebin veri seti kullanılmıştır. Bu veri seti 179 farklı kötücül yazılım ailesinden 5560 tane uygulama içermektedir. Bu örnekler Ağustos 2010 ile Ekim 2012 yılları arasındaki bir periyotta toplanmıştır [66].

Statik analiz yöntemiyle kötü niyetli uygulama tespitine yönelik literatürde pek çok çalışma yapıldığı görülmüştür. Söz konusu çalışmalarda kullanılan statik analiz araçları ile test edilen uygulama sayıları çeşitlilik göstermektedir. Tezde kullanılan yöntem ve araçların literatürdeki diğer statik analiz yöntemi kullanılan araçlarla kıyaslaması yapılmıştır. Yapılan çalışmaların literatürde hazırlandıkları yıl ve kullandıkları veri setlerindeki uygulama sayılarının kıyaslamaları Çizelge 3.1'de gösterilmiştir. Tablonun en sonuna tez çalışması için test edilen uygulama sayısı da eklenmiştir [30,31].

Çizelge 3. 1. Statik analiz modeliyle çalışmalarda kullanılan uygulama sayıları

S.No.	Kullanılan Araç	Yılı	Test Edilen Uygulama Sayısı
1	LeakMiner [67]	2012	1750
2	Androsimilar [68]	2013	6779
3	Dendroid [13]	2014	1231
4	Appprofiler [18]	2013	2782
5	Riskranker [16]	2012	118318
6	CHEX [20]	2012	5486
7	Drebin [12]	2014	129013
8	Droidlegacy [69]	2014	1100
9	Mama [70]	2013	666
10	Droidmat [13]	2012	1738
11	Asdroid [71]	2012	125249
12	Droidapiminer [72]	2013	20
13	Droidanalytics [73]	2013	150368
14	Droidbarrier [74]	2014	405
15	Tez	2016	6960

Çizelge 3.1'de statik analiz yöntemiyle kötü niyetli uygulama tespitine yönelik yapılan çalışmalarda kullanılan veri setlerinin içerisinde bulunan uygulama sayılarına yer verilmiştir. Bu kıyaslama en eski 2012 yılı göz önüne alınarak yapılmıştır. Test edilen uygulama sayısının yoğun olduğu RiskRanker, Drebin, Asdroid ve Droidanalytics sadece risk seviyelerini göz önünde bulundurarak yapılandırıldıkları için çok sayıda uygulamayı kullanabilmişlerdir. Ayrıca bu tez çalışmasında kullanılan uygulama sayısına en son 2012 yılında CHEX aracıyla 5486 uygulama ve 2013 yılında Androsimilar aracıyla 6779 uygulama test edilerek yaklaşılmıştır. Diğer çalışmalar yıl bazında daha eski ve test edilen uygulama sayıları çok daha azdır. Ancak yapılan çalışmalarda tezin ilerleyen kısımlarında da görüleceği gibi test edilen uygulama sayısının artması kullanılan statik analiz aracının başarısını etkilemesiyle, tam tersi olarak test edilen uygulama sayısının azalması kullanılan statik analiz aracının daha performanslı olacağı anlamına gelmesi durumu göz önünde bulundurularak, bu ikilinin birbirleriyle doğru orantılı olduğu gibi bir çıkarım yapılamaz. Çünkü her çalışmada kullanılan özellikler ve kullanılan algoritmalar farklılık göstermekte ve teknolojinin ilerlemesi birçok tespit aracının performansını etkilemektedir.

3.2. Uygulamaların Tersine Derlenmesi

Android için geliştirilmiş olan uygulamalar Java programlama dilinde yazılmıştır. Uygulamalar bu programlama dilinde yazıldığı için uygulamaların kaynak kodları ilk olarak Java bayt koduna, daha sonra dex bayt koduna çevrilmektedir. Yapılan bu işleme uygulamaların tersine derlenmesi denilmektedir. Kaynak kodlar da çevrilme işleri yapılarak muhafaza edilmektedir. Bu işlemlerden sonra bir araya toplanıp dex uzantılı dosyası içinde derli bir şekilde saklanmaktadır. Statik analiz yönteminin kullanılması için apk uzantısı olan bu uygulamaların smali koduna çevrilmesini yapmak yani tersine derlemek için apktool kullanılmıştır. Hazır olarak *apktool* ile tersine derleme yapan web siteleri bulunmaktadır [83]. Ancak tez çalışması için *apktool* kullanarak tersine derleme işlemi yapılabilen bir program kodlanmıştır [75]. Tez çalışması için *apktool 1.4.3* versiyonu kullanılmıştır. Tek bir uygulama için apktool kullanımı aşağıdaki komutla yapılabilmektedir. Bu işlem sonrasında ilgili apk dosyasının içeriği aynı isimle bir klasöre çıkartılmaktadır.

apktool -d "uygulama ismi".apk

Bunu kötü ve iyi niyetli uygulamaların bulunduğu veri setine teker teker uyguladıktan sonra uygulamaların analizi yapılmıştır. Veri setinde bulunan uygulamalarda olmazsa olmaz denilen AndroidManifest.xml dosyası da bu analiz ile oluşmaktadır. Bu dosya xml uzantılı olarak oluşturulmaktadır. Bunun sebebi hem makinenin (tez çalışması için mobil cihazın) hem de insanların okumasını kolaylaştırmaktır. Örnek AndroidManifest.xml dosyası içeriği aşağıda görülmektedir.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest android:versionCode="2" android:versionName="1.1"
package="jp.tjkapp.droid1lwp"
xmlns:android="http://schemas.android.com/apk/res/android">
<uses-sdk android:minSdkVersion="7" />
<application android:label="@string/app_name" android:icon="@drawable/icon">
<service android:label="@string/app_name" android:icon="@drawable/icon"
android:name=".lpwp" android:permission="android.permission.BIND_WALLPAPER"
android:enabled="true">
```

```
<intent-filter>
<action android:name="android.service.wallpaper.WallpaperService" />
</intent-filter>
<meta-data android:name="android.service.wallpaper" android:resource="@xml/lpwp" />
</service>
<activity android:theme="@android:style/Theme.Light.WallpaperSettings"
android:label="@string/settings" android:name=".lpwpSettings" android:exported="true"
/>
<receiver android:name="com.xxx.yyy.MyBoolService">
<intent-filter>
<action android:name="android.intent.action.BOOT_COMPLETED" />
</intent-filter>
</receiver>
<receiver android:name="com.xxx.yyy.MyAlarmReceiver">
<intent-filter>
<action android:name="com.lz.myservicestart" />
</intent-filter>
</receiver>
<service android:name="com.xxx.yyy.MyService" android:enabled="true" />
<receiver android:name="com.xxx.yyy.NetWorkReceiver">
<intent-filter>
<action android:name="android.net.conn.CONNECTIVITY_CHANGE" />
</intent-filter>
</receiver>
<receiver android:name="com.xxx.yyy.CustomBroadcastReceiver">
<intent-filter>
<action android:name="android.intent.action.PHONE_STATE" />
</intent-filter>
</receiver>
</application>
<uses-permission android:name="android.permission.WRITE_APN_SETTINGS" />
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"
/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

<uses-permission android:name="android.permission.READ_PHONE_STATE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.MODIFY_PHONE_STATE" />
</manifest>

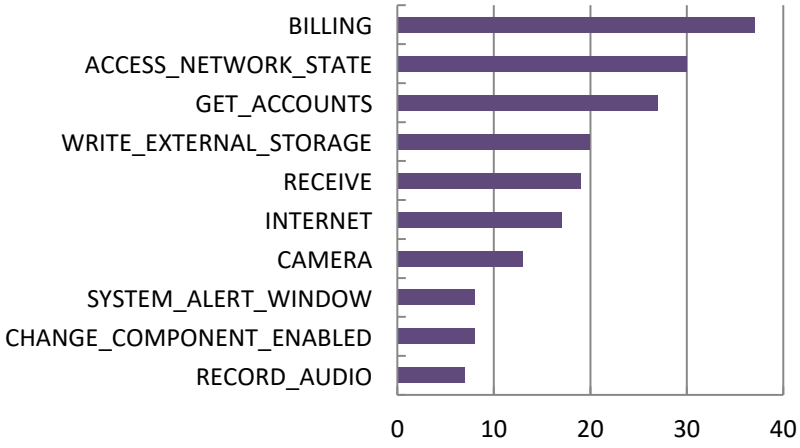
Bu dosya içerisinde pek çok etiket bulunmaktadır. Bu etiketlerin başlıcaları sırasıyla manifest (paket ismi, versiyon numarası gibi özelliklerin bulunduğu), application (uygulamanın temel değişkenlerinin bulunduğu), service ve receiver etiketleridir [80].

3.3. İzin ve API Çağrılarının Bulunması

Tez çalışmasında kullanmak maksadıyla indirilen güvenilir uygulamalar ve DREBIN veri setinden elde ettiğimiz kötü niyetli uygulamaları net bir şekilde sınıflandırabilmek maksadıyla izinler ve API çağrıları kullanılarak başarılı sonuçlar elde edilmiştir. Çünkü zararlı yazılımların geliştirilme amaçları düşünüldüğünde hedefe ulaşmak için API çağrıları ve izinlerine ihtiyaç duyulmaktadır. Bu sebeple oluşturduğumuz iyi ve kötü niyetli uygulama veri setinde en çok kullanılan izinlerin ve bir programcının kodladığı uygulamanın, kodun çalışma şekillerine erişmesini sağlayan fonksiyon olan API çağrılarının tespiti yapılmıştır.

İzin ve API çağrılarının bulunması için iyi ve kötü niyetli uygulama veri setimizdeki tüm API çağrıları ve izinlerin çıkarılması için bir program kodlanmıştır. Daha sonra çıkarılan izin ve API çağrılarına göre bir kullanma ve kullanmama durumu için geliştirilen program bu bilgileri bir txt dosyasına kullanım durumu olanlara 1 olmayanlara 0 verecek şekilde eklemektedir. Daha sonra buradaki işleme göre izin ve API çağrılarının kullanım sıklığına bakılmıştır. Böylece güvenli veri seti ile zararlı veri setinin birbirinden ne kadar ayrıldığı görülmüştür. Söz konusu programın oluşturduğu txt dosyasındaki bilgiler grafiklere yansıtılmıştır.

İyi niyetli uygulamaların bulunduğu güvenli veri setinde en çok kullanılan ilk 10 izin aşağıdaki Şekil 3.2'de gösterilmiştir.

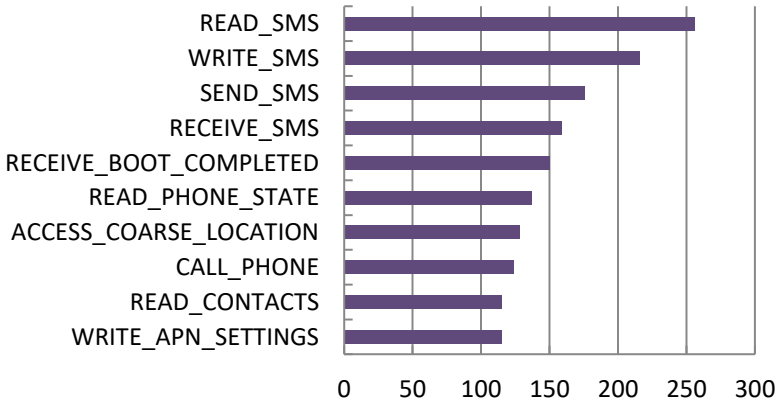


Şekil 3. 2. Güvenli veri setinde kullanılan ilk 10 izin

Şekilde görüldüğü gibi en çok kullanılan izin BILLING iznidir. Bu izin tez çalışmasında yasal market olarak kullandığımız Play Store'un uygulamalar içinde ürünlerin satılmasına izin verdiği bir satın alma servisine erişim iznidir.

İlk on izin içinde en az kullanılan izin RECORD_AUDIO iznidir. Bu izin de uygulamanın mikrofon kullanma isteğinin kabul edildiği izindir.

Kötü niyetli uygulamaların bulunduğu zararlı veri setinde en çok kullanılan ilk 10 izin aşağıdaki Şekil 3.3'te gösterilmiştir.

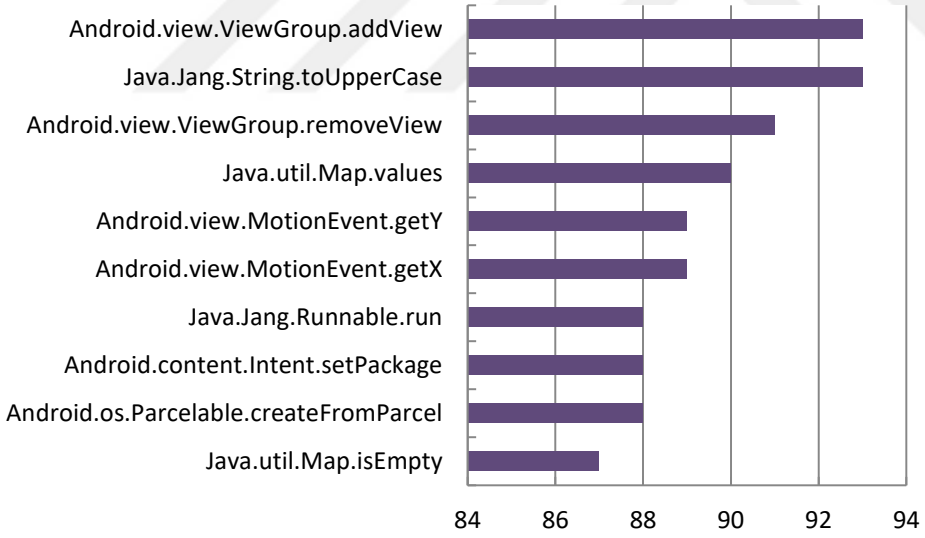


Şekil 3. 3. Zararlı veri setinde kullanılan ilk 10 izin

Şekilde görüldüğü gibi en çok kullanılan izin READ_SMS iznidir. Uygulamanın indirilme aşamasında bu izin kabul edilirse uygulamanın bütün mobil cihazda bulunan kısa mesajları okuyacağı anlamına gelmektedir.

Zararlı veri setinde kullanılan ilk on izin içinde en az kullanılan izinler READ_CONTACTS ve WRITE_APN_SETTINGS izinleridir. READ_CONTACTS izninin uygulama indirilmeden önce kabul edilmesi kullanıcının telefon rehberine ulaşmak anlamına gelmektedir. APN (Access Point Name), mobil cihazda bulunan erişim noktası anlamına gelmektedir. WRITE_APN_SETTINGS iznine uygulamayı indirmeden izin verilirse mobil cihazdaki APN ayarlarını yazmaya izin vermek anlamına gelmektedir. VPN (Virtual Private Network)'in mobil cihazlardaki karşılığı olduğu için APN'deki bilgilere erişmek oldukça sakıncalıdır.

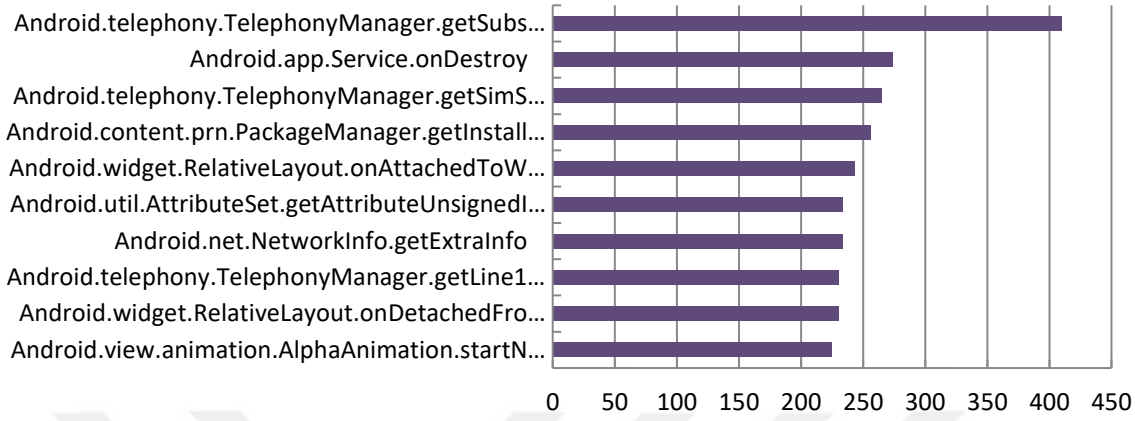
İyi niyetli uygulamaların bulunduğu güvenli veri setinde en çok kullanılan ilk 10 API çağrısı aşağıdaki Şekil 3.4'te gösterilmiştir.



Şekil 3. 4. Güvenilir veri setinde kullanılan ilk 10 API çağrısı

Şekilde görüldüğü gibi en çok kullanılan API çağrısı ViewGroup sınıfına ait olan bir fonksiyon olan addView fonksiyonudur. Bu grafikte de olduğu gibi API çağrıları özel işlemleri kolaylaştırmak için kullanılmaktadır. Tez çalışmasında güvenilir veri setinde kullanılan ilk on API çağrısı içinde en az kullanılan API çağrısı Java.util.Map.isEmpty çağrısıdır. API'de java kodlarının haritalama yaptığı yerde boş veya dolu olma durumunu kontrol eden fonksiyondur.

Kötü niyetli uygulamaların bulunduğu zararlı veri setinde en çok kullanılan ilk 10 API çağrısı Şekil 3.5'te gösterilmiştir.



Şekil 3. 5. Zararlı veri setinde kullanılan ilk 10 API çağrısı

Şekilde görüldüğü gibi Android.telephony.TelephonyManager.getSubscribID API çağrısı en çok kullanılan çağrıdır. Zararlı veri setinde kullanılan ilk on API çağrısı içinde en az kullanılan API çağrısı Android.view.animation.AlphaAnimation.startNow çağrısıdır.

3.4. Sistemin Eğitimiyle Sınıflandırma İşlemi

Güvenilir ve zararlı uygulama veri seti kullanılarak önce sistem eğitilmiş ve daha sonra sınıflandırma işlemi yapılmıştır. Sistemin eğitilme sürecinde bir veri madenciliği programı olan *Weka* uygulaması kullanılmıştır [76]. Weka, veri madenciliği konusu için makine öğrenmesi algoritmalarının bir arada bulunduğu bir programdır. Weka da toplu bir şekilde bulunan algoritmalar kendi oluşturduğumuz Java kodumuza veya herhangi bir verisetine direk uygulanabilecek şekilde yapılmıştır. Weka, veri ön işleme, sınıflandırma, kümeleme, birleştirme kuralları ve görselleştirme yapabilmek için araçlar içermektedir [77]. Weka kullanımına başlamadan önce weka uygulamasının kullanmak istediği *.arff* uzantılı dosyayı oluşturulması gerekmektedir. Bu sebeple çalışma için bir program kodlanmıştır. Bu program elimizdeki veri seti için bir out dosyası oluşturarak weka uygulamasının kullanılmasına imkan vermektedir. Sistemin eğitiminde kullanılan algoritmalar tez çalışmasının 4. bölümünde anlatılmıştır.

4. BULGULAR VE TARTIŞMA

Bu tez çalışmasında, Drebin [66] projesinin 5560 adet kötü niyetli uygulama veri seti ile Play Store'dan indirilmiş en popüler 1400 uygulama kullanılmıştır. Çalışmada 5 tane birbirinden farklı makine öğrenmesi algoritması kullanılmış ve sınıflandırma işlemini en performanslı şekilde uygulayabilen algoritmalar belirlenmiştir.

Çalışmanın başarılı olduğunu gösterebilmek için en yaygın olarak kullanılan ölçüm kriterlerinden biri olan doğruluk değeri kullanılmıştır. Doğruluk ölçütü, çalışmada doğru olarak sınıflandırılan veri seti sayısının tüm veri seti sayısına bölümü demektir.

$$Doğruluk=\frac{TP+TN}{TP+TN+FP+FN} \quad (4.1)$$

Hata oranı ise çalışmada hatalı olarak sınıflandırdığımız veri seti sayısının tüm veri seti sayısına bölümü demektir. Çalışmanın amacına ulaşabilmesi için doğruluk ölçütü değerinin yüksek ve hata oranı değerinin de düşük olması beklenmektedir.

$$Hata=\frac{FP+FN}{TP+TN+FP+FN} \quad (4.2)$$

Kesinlik hesabı ise;

$$Kesinlik=\frac{\mathrm{TP}}{\mathrm{TP+FP}} \quad (4.3)$$

Duyarlılık hesabı ise;

$$Duyarlılık=\frac{\mathrm{TP}}{\mathrm{TP+FN}} \quad (4.4)$$

Kesinlik ve duyarlılık ölçümlerinin harmonik ortalaması F-ölçütünün vermektedir.

$$F\text{-}Ölçütü=\frac{2*\mathrm{Kesinlik}*\mathrm{Duyarlılık}}{\mathrm{Kesinlik+Duyarlılık}} \quad (4.5)$$

Tezde kullanılan algoritmalardan J 48 algoritmasının detaylı doğruluk hesapları Çizelge 4.1'de gösterilmiştir.

Çizelge 4. 1. J 48 algoritması için ölçüm tablosu

	TP Oranı	FP Oranı	Kesinlik	Duyarlılık	F-Ölçütü	Sınıf
	0.987	0.011	0.992	0.987	0.989	B
	0.989	0.013	0.983	0.989	0.986	M
Ağırlıklı Ortalama	0.988	0.012	0.988	0.988	0.988	

Tezde kullanılan algoritmalardan ID3 algoritmasının detaylı doğruluk hesapları Çizelge 4.2'de gösterilmiştir.

Çizelge 4. 2. ID3 algoritması için ölçüm tablosu

	TP Oranı	FP Oranı	Kesinlik	Duyarlılık	F-Ölçütü	Sınıf
	0.983	0.014	0.99	0.983	0.986	B
	0.986	0.017	0.978	0.986	0.982	M
Ağırlıklı Ortalama	0.985	0.015	0.985	0.985	0.985	

Tezde kullanılan algoritmalardan Naive Bayes algoritmasının detaylı doğruluk hesapları Çizelge 4.3'de gösterilmiştir.

Çizelge 4. 3. Naive Bayes algoritması için ölçüm tablosu

	TP Oranı	FP Oranı	Kesinlik	Duyarlılık	F-Ölçütü	Sınıf
	0.786	0.001	0.999	0.786	0.88	B
	0.999	0.214	0.779	0.999	0.875	M
Ağırlıklı Ortalama	0.878	0.092	0.905	0.878	0.878	

Tezde kullanılan algoritmalardan KNN algoritmasının detaylı doğruluk hesapları Çizelge 4.4'de gösterilmiştir.

Çizelge 4. 4. KNN algoritması için ölçüm tablosu

	TP Oranı	FP Oranı	Kesinlik	Duyarlılık	F-Ölçütü	Sınıf
	0.989	0.015	0.989	0.989	0.989	B
	0.985	0.011	0.985	0.985	0.985	M
Ağırlıklı Ortalama	0.987	0.013	0.987	0.987	0.987	

Tezde kullanılan algoritmalardan FT Karar ağacı algoritmasının detaylı doğruluk hesapları Çizelge 4.5'de gösterilmiştir.

Çizelge 4. 5. FT Karar ağacı algoritması için ölçüm tablosu

	TP Rate	FP Rate	Kesinlik	Duyarlılık	F-Ölçütü	Sınıf
	0.991	0.008	0.991	0.991	0.991	B
	0.991	0.008	0.991	0.991	0.991	M
Ağırlıklı Ortalama	0.991	0.008	0.991	0.991	0.991	

Statik Analiz yöntemiyle kötü niyetli uygulama tespitine yönelik yapılan çalışmalar ve tezde kullanılan algoritmalar ile yapılan çalışmaların karşılaştırmaları Çizelge 4.6'da gösterilmiştir.

Çizelge 4.6. Statik analiz yöntemleri karşılaştırmaları

Yapılan Çalışma	Algoritma	Özellikler	Veriseti	Metrikler	Başarı Oranı
Shen ve ark. [78]	VF2	Android uygulama bileşenleri	Genome Veriseti	Tespit oranı	%86,36
Drebin	SVM	Android Manifest dosyası ve kaynak kodlar	Drebin Veriseti	Tespit oranı	%94
Droidmat	kNN	İzinler ve API çağrıları	Genome Veriseti	Precision, Recall, F-measure	%97,87
Sheen ve ark. [79]	Ensemble	Uygulama izinleri ve API çağrıları	Genome veriseti	Precision, Recall, F-measure	%88
Tez	Naive Bayes	Uygulama izinleri ve API çağrıları	Drebin Versieti	Accuracy, Error, Precision, Recall, F-measure	%78,6
Tez	ID3	Uygulama izinleri ve API çağrıları	Drebin Versieti	Accuracy, Error, Precision, Recall, F-measure	%98,3
Tez	J48	Uygulama izinleri ve API çağrıları	Drebin Versieti	Accuracy, Error, Precision, Recall, F-measure	%98,7
Tez	kNN	Uygulama izinleri ve API çağrıları	Drebin Versieti	Accuracy, Error, Precision, Recall, F-measure	%98,9
Tez	FT Karar Ağacı	Uygulama izinleri ve API çağrıları	Drebin Versieti	Accuracy, Error, Precision, Recall, F-measure	%99,2

Çizelge 4.6'da yapılan çalışmalar, çalışmaların kullandığı algoritmalar, araçların özellikleri, oluşturulan veri setleri, kullanılan metrikler ve başarı oranları verilmiştir. Veri setinde analiz edilen iyi ve kötü niyetli uygulamaların özellikleri çıkarıldığında uygulama izinleri ve API çağrılarının en başarılı özellik kullanımı olduğu görülmüştür. Diğer

çalışmalarda görülen özellikleri izinler ve API çağrıları fazlasıyla kapsamaktadır. Metrik anlamında en kapsamlı çalışmanın yapılan tez olduğu görülmüştür. Ayrıca çizelgedeki oranlara bakıldığında en iyi algoritmanın kNN algoritması olduğu görülmektedir. Ancak tez çalışmasında, bu algoritma %'0,3'lük bir fark ile FT karar ağacı algoritmasında daha az performanslıdır. Kullanılan veri seti sayısı ve metrikler göz önünde bulundurularak en performanslı algoritmanın FT karar ağacı olduğunu göstermektedir.

5. SONUÇ

Bu tezde, Android uygulamalar içerisinden iyi ve yasal olarak kullanılan Drebin veri setindeki kötü niyetli uygulamalar bir araya getirilerek uygulamalar mobil cihaza kurulmadan incelenmiştir. Bu inceleme izin ve API çağrısı tabanlı yapılmıştır. İncelenen uygulamaların davranış tespitleri yapıldıktan sonra sistem makine öğrenmesi algoritmalarıyla eğitilerek sınıflandırma işlemi gerçekleştirilmiştir. Sınıflandırma sonuçlarına göre de tezde kullanılan algoritmaların performans değerlerine bakılmıştır.

Tezde kullanılan algoritmaların performans değerleri geniş kapsamlı olarak tutulmuş ve sistem eğitimi için kullanılması gereken bütün değerlendirme kriterleri kullanılmıştır. Değerlendirme kriterlerinin fazla olması en doğru ve performansı en yüksek algoritmanın ortaya çıkması için yardımcı olmuştur.

Tez çalışması sonucunda yapılan algoritma hesaplarına göre kullandığımız 5 tane algoritma için en performanslı algoritmanın FT Karar Ağacı algoritması olduğu ve çizelgelerdeki oranlamalarla sınıflandırma yaptıkları görülmüştür. Diğer dört algoritmadan Naive Bayes dışındaki algoritmaların performansının birbirlerine yakın olduğu tespit edilmiştir. FT Karar Ağacı algoritması her ölçüm kriteri için başarı gösterdiği ve başarı oranlarının diğerlerinden yüksek olduğu tespiti yapılmıştır. Ayrıca bu çalışmanın diğer çalışmalarla kıyaslanmasına da yer verilmiştir.

KAYNAKLAR

1. Schmidt, A. D., Bye, R., Schmidt, H. G., Clausen, J., Kiraz, O., Yuksel, K. A., Albayrak, S. (2009). *Static analysis of executables for collaborative malware detection on android*. In 2009 IEEE International Conference on Communications (1-5). IEEE.

2. Seo, S., Lee, D., Yim, K. (2012). *Analysis on maliciousness for mobile applications,* The Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing, 126-129.

3. Shabtai, A., Fledel, Y., Elovici, Y. (2010). *Automated Static Code Analysis for Classifying Android Applications Using Machine Learning*, International Conference on Computational Intelligence and Security, 329-333.

4. Shabtai, A. (2010). *Malware Detection on Mobile Devices*, The Eleventh International Conference on Mobile Data Management (MDM), 289 – 290.

5. Kayabaşı, G., Doğru, İ. A. (2015). *Mobil Cihazlarda Zararlı Yazılım Tespitinde Kullanılan Statik Analiz Araçları*. 8. Uluslararası Bilgi Güvenliği ve Kriptoloji Konferansı (ISC Turkey 2015), Ankara.

6. Bulygin, Y. (2007). *Epidemics of Mobile Worms*. The 26th IEEE International Performance Computing and Communications Conference, April 11-13, 2007, New Orleans, Louisiana,USA. IEEE Computer Society, 475–478.

7. Oberheide, J. Cooke, E., Jahanian, F. (2008). *Cloudav: N-version antivirus in the network cloud* In Proceedings of the 17th USENIX Security Symposium (Security'08), San Jose, CA, July 2008. IEEE Criteria f or Class IE Electric Systems (Standards style), IEEE Standard 308, 1969.

8. Samfat, D., Molva, R. (1997). IDAMN: An Intrusion Detection Architecture for Mobile Networks. *IEEE Journal on Selected Areas in Communications*, 15(7), 1373–1380.

9. Bose, A., Hu, X., Shin, K. G., Park, T. (2008). *Behavioral detection of malware on mobile handsets.* In Proceeding of the 6th international conference on Mobile systems, applications, and services, 225–238.

10. Shabtai, A., Kanonov, U., Elovici, Y., Glezer, C., & Weiss, Y. (2012). Andromaly: a behavioral malware detection framework for android devices. *Journal of Intelligent Information Systems*, 38(1), 161-190.

11. He, D., Chan, S., Guizani, M. (2015). *Mobile application security: malware threats and defenses*. Wireless Communications, IEEE, 22 (1). 138-144.

12. Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., & Rieck, K. (2014, February). *DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket*. In NDSS.

13. Wu, D. J., Mao, C. H., Wei, T. E., Lee, H. M., & Wu, K. P. (2012, August*). Droidmat: Android malware detection through manifest and api calls tracing*. In Information Security (Asia JCIS), 2012 Seventh Asia Joint Conference on (62-69). IEEE.

14. Kabakuş, A. T., Doğru İ. A., Çetin, A. (2015). Android kötücül yazılım tespit ve koruma sistemleri. *in Erciyes Üniversitesi Fen Bilimleri Enstitüsü Dergisi*, 31(1), 9-16

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

16. Grace, M., Zhou, Y., Zhang, Q., Zou, S., & Jiang, X. (2012, June). *Riskranker: scalable and accurate zero-day android malware detection*. In Proceedings of the 10th international conference on Mobile systems, applications, and services (281-294).

17. Wang, Z.,Chenlong, L., Zhenlong, Y., Guan, Y.,Xue, Y. (2016). DroidChain: A novel Android malware detection method based on behavior chains. *Pervasive and Mobile Computing*, 32, 3-14.

18. Rosen, S., Qian, Z., Mao, Z. M. (2013). *Appprofiler: a flexible method of exposing privacy-related behavior in android applications to end users*. In Proceeding 3rd ACM conference on Data and application security and privacy, (221–232).

19. Grace, M., Zhou, Y., Wang, Z., Jiang,X. (2012). *Systematic Detection of Capability Leaks in Stock Android Smartphones.* In NDSS, 14, 19.

20. Lu, L., Li, Z., Wu, Z., Lee, W., Jiang, G. (2012). *Chex: statically vetting android apps for component hijacking vulnerabilities*. In Proceedings of the 2012 ACM conference on Computer and communications security (229-240).

21. Burguera, I., Zurutuza, U., & Nadjm-Tehrani, S. (2011, October). *Crowdroid: behavior-based malware detection system for android.* In Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices (pp. 15-26). ACM.

22. Kabakuş, A. T., Doğru, İ. A., Çetin, A. (2015). APK Auditor: Permission-based Android malware detection system. *Digital Investigation*, 13,1-14.

23. Rastogi, V., Chen, Y., Enck, W. (2013). *AppsPlayground :Automatic Security Analysis of Smartphone Applications.* In CODASPY '13 Proceedings of the third ACM conference on Data and application security and privacy. 209–220.

24. Portokalidis, G., Homburg, P., Anagnostakis, K., Bos, H. (2010). *Paranoid Android: Versatile Protection For Smartphones.* In Annual Computer Security Applications Conference (ACSAC), 347–356.

25. Reina, A., Fattori, A., Cavallaro, L. (2013). *A System Call-Centric Analysis and Stimulation Technique to Automatically Reconstruct Android Malware Behaviors*. In Proceedings of the 6th European Workshop on System Security

26. Zhou, W., Zhou, Y., Jiang, X., Ning, P. (2012). *Detecting repackaged smartphone applications in third-party android marketplaces.* In Proc. 2nd ACM conference on Data and Application Security and Privacy. 317–326.

27. Guido, M., Ondricek, J., Grover, J., Wilburn, D., Nguyen, T., & Hunt, A. (2013). Automated identification of installed malicious Android applications. *Digital Investigation*, 10, S96-S104.

28. Dini, G., Martinelli, F., Saracino, A., & Sgandurra, D. (2012, October). *MADAM: a multi-level anomaly detector for android malware*. In International Conference on Mathematical Methods, Models, and Architectures for Computer Network Security (240-253).

29. Barrera, P. C., Oorschot, V., Somayaji, A. (2010). *A Methodology for Empirical Analysis of Permission-Based Security Models and its Application to Android Categories and Subject Descriptors*, 17th ACM Conference on Computer and Communications Security, 73–84.

30. Utku, A., Doğru, İ. A. (2016). Mobil Kötücül Yazılımlar Ve Güvenlik Çözümleri Üzerine Bir İnceleme. *Gazi Üniversitesi Fen Bilimleri Dergisi Part C: Tasarım ve Teknoloji*, 4(2), 49-64.

31. Feizollah, A., Anuar, N. B., Salleh, R., AbdulWahab, A. W. (2015). A review on feature selection in mobile malware detection, *Digital Investigation*, 13, 22-37.

32. İnternet: Llamas, R. (2016). IDC: Smartphone OS Market Share 2016, 2015. URL:http://www.webcitation.org/query?url=http%3A%2F%2Fwww.idc.com%2Fprodserv%2Fsmartphone-os-market-share.jsp+&date=2016-12-14. Son Erişim Tarihi: 10 Kasım 2016.

33. İnternet: Android. Android security overview, URL: http://www.webcitation.org/query?url=https%3A%2F%2Fsource.android.com%2Fsecurity%2F&date=2017-01-09. Son Erişim Tarihi: 10 Kasım 2016.

34. Enck, W., Octeau, D., McDaniel, P., Chaudhuri, S. (2011). *A Study of Android Application Security*. The 20th USENIX Security Symposium. 2(1), 2.

35. İnternet: Burns, J. Developing secure mobile applications for android,. http://www.webcitation.org/query?url=https%3A%2F%2Fwww.isecpartners.com%2Fmedia%2F11991%2Fisec_securing_android_apps.pdf&date=2017-01-09. Son Erişim Tarihi: 10 Kasım 2016.

36. İnternet: Google. Google play. URL: http://www.webcitation.org/query?url=https%3A%2F%2Fplay.google.com%2FStore&date=2017-01-09. Son Erişim Tarihi: 10 Kasım 2016.

37. İnternet: Lockheimer, H. Android and security, URL: http://www.webcitation.org/query?url=http%3A%2F%2Fgooglemobile.blogspot.com.t

r%2F2012%2F02%2Fandroid-and-security.html&date=2017-01-09. Son Erişim Tarihi: 10 Kasım 2016

38. Schmidt, A. D. (2011). Detection of smartphone malware. *Berlin Institute of Technology*.

39. Felt, A. P., Chin, E., Hanna, S., Song, D., & Wagner, D. (2011). *Android permissions demystified*. In Proceedings of the 18th ACM conference on Computer and communications security (627-638).

40. de la Puerta, J. G., Sanz, B., Grueiro, I. S., & Bringas, P. G. (2015). *The Evolution of Permission as Feature for Android Malware Detection*. In International Joint Conference (389-400).

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

42. İnternet: Android. Services. URL: http://www.webcitation.org/query?url=http%3A%2F%2Fdeveloper.android.com%2Fguide%2Fcomponents%2Fservices.html.&date=2017-01-09. Son Erişim Tarihi: 10 Kasım 2016

43. İnternet: Android. Activities. URL: http://www.webcitation.org/query?url=http%3A%2F%2Fdeveloper.android.com%2Fguide%2Fcomponents%2Factivities.html&date=2017-01-09. Son Erişim Tarihi: 10 Kasım 2016

44. İnternet: Android.Building and running. URL: http://www.webcitation.org/query?url=http%3A%2F%2Fdeveloper.android.com%2Ftools%2Fbuilding%2Findex.html&date=2017-01-09. Son Erişim Tarihi: 10 Kasım 2016

45. İnternet: dex2jar. URL: http://www.webcitation.org/query?url=https%3A%2F%2Fcode.google.com%2Fp%2Fdex2jar%2F&date=2017-01-09. Son Erişim Tarihi: 10 Kasım 2016

46. İnternet: JAD. URL: http://www.webcitation.org/query?url=http%3A%2F%2Fvaraneckas.com%2Fjad%2F&date=2017-01-09. Son Erişim Tarihi: 10 Kasım 2016

47. İnternet: Apktool: A tool for reverse engineering Android apk files. URL: http://www.webcitation.org/query?url=https%3A%2F%2Fcode.google.com%2Fp%2Fandroid-apktool%2F&date=2017-01-09. Son Erişim Tarihi: 10 Kasım 2016

48. İnternet: JEB.The interactive android decompiler. URL: http://www.webcitation.org/query?url=http%3A%2F%2Fwww.android-decompiler.com&date=2017-01-09. Son Erişim Tarihi: 10 Kasım 2016

49. Sokolova, M., Lapalme, G. (2009). A systematic analysis of performance measures for classification tasks. *Information Processing & Management*, 45(4), 427-437.

50. Duda, R. O., & Hart, P. E. (1973). Pattern classification and scene analysis. *New York: Wiley*.

51. Mewes, H.-W., Albermann, K., Heumann, K., Lieb, S., & Pfeiffer, F. (1997). MIPS: A database for protein sequences, homology data and yeast genome information. *Nucleic Acids Research*, 25(1), 28–30.

52. Li, T., Zhang, C., Zhu, S. (2006). *Empirical studies on multi-label classification*. In Proceedings of the 18th IEEE international conference on tools with artificial intelligence (86–92).

53. Kiritchenko, S., Matwin, S., Nock, R., & Famili, A. F. (2006). *Learning and evaluation in the presence of class hierarchies: Application to text categorization*. In Proceedings of the 19th Canadian conference on AI, (395–406).

54. Bobicev, V., & Sokolova, M. (2008). *An effective and robust method for short text classification*. In Proceedings of the association for the advancement of artificial intelligence, (1444–1445).

55. Tikk, D., Biró, G. (2003*). Experiments with multi-label text classifier on the Reuters collection.* In Proceedings of the international conference on computational cybernetics, (pp. 33–38).

56. Sun, A., Lim, E.-P., & Ng, W.-K. (2003). Performance measurement framework for hierarchical text classification. *Journal of the Americal Society for Information Science and Technology*, 54(11), 1014–1028."

57. Kazawa, H., Izumitani, T., Taira, H., & Maeda, E. (2005*). Maximal margin labeling for multi-topic text categorization.* In Advances in neural information processing systems (NIPS'04), (17:649–656).

58. Bhatia, N. (2010). Survey of nearest neighbor techniques. *International Journal of Computer Science and Information Security*, 8(2), 302-305.

59. Deekshatulu, B. L., & Chandra, P. (2013). Classification of heart disease using k-nearest neighbor and genetic algorithm. *Procedia Technology*, 10, 85-94.

60. Kotsiantis, S. B., Zaharakis, I., & Pintelas, P. (2007). Supervised machine learning: A review of classification techniques. *Informatica*, 31, 249-268

61. Gokgoz, E., & Subasi, A. (2015). Comparison of decision tree algorithms for EMG signal classification using DWT. *Biomedical Signal Processing and Control*, 18, 138-144.

62. Alpaydın, E. (2000). *Zeki veri madenciliği: ham veriden altın bilgiye ulaşma yöntemleri.* Bilişim 2000 eğitim semineri.

63. Ibáñez, A., Bielza, C., & Larrañaga, P. (2014). Cost-sensitive selective naive Bayes classifiers for predicting the increase of the h-index for scientific journals. *Neurocomputing*, 135, 42-52.

64. İnternet: Google Play. URL: http://www.webcitation.org/query?url=https%3A%2F%2Fplay.google.com%2Fstore%2Fapps&date=2017-01-09. Son Erişim Tarihi: 10 Kasım 2016

65. İnternet: Android Market API. URL: http://www.webcitation.org/query?url=http%3A%2F%2Fcode.google.com%2Fp%2Fandroid-market-api&date=2017-01-09. Son Erişim Tarihi: 10 Kasım 2016

66. İnternet: The Drebin Dataset. URL: http://www.webcitation.org/query?url=http%3A%2F%2Fuser.informatik.uni-goettingen.de%2F%7Edarp%2Fdrebin%2F&date=2017-01-09. Son Erişim Tarihi: 10 Kasım 2016

67. Yang, Z., & Yang, M. (2012, November). *Leakminer: Detect information leakage on android with static taint analysis*. In Software Engineering (WCSE), 2012 Third World Congress, (101-104).

68. Faruki, P., Ganmoor, V., Laxmi, V., Gaur, M. S., & Bharmal, A. (2013, November). *AndroSimilar: robust statistical feature signature for Android malware detection*. In Proceedings of the 6th International Conference on Security of Information and Networks, (152-159).

69. Deshotels, L., Notani, V., & Lakhotia, A. (2014, January). *Droidlegacy: Automated familial classification of android malware.* In Proceedings of ACM SIGPLAN on Program Protection and Reverse Engineering Workshop.

70. Sanz, B., Santos, I., Laorden, C., Ugarte-Pedrero, X., Nieves, J., Bringas, P. G., & Álvarez Marañón, G. (2013). MAMA: manifest analysis for malware detection in android. *Cybernetics and Systems*, 44(6-7), 469-488.

71. Huang, J., Zhang, X., Tan, L., Wang, P., & Liang, B. (2014, May). *AsDroid: detecting stealthy behaviors in Android applications by user interface and program behavior contradiction*. In Proceedings of the 36th International Conference on Software Engineering, (1036-1046).

72. Aafer, Y., Du, W., & Yin, H. (2013, September). *DroidAPIMiner: Mining API-level features for robust malware detection in android.* In International Conference on Security and Privacy in Communication Systems, (86-103).

73. Zheng, M., Sun, M., & Lui, J. C. (2013, July). *Droid analytics: a signature based analytic system to collect, extract, analyze and associate android malware.* In 2013 12th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (163-171).

74. Almohri, H. M., Yao, D. D., & Kafura, D. (2014, March). *Droidbarrier: Know what is executing on your android.* In Proceedings of the 4th ACM conference on Data and application security and privacy (257-264). ACM.

75. İnternet: Android Apktool. URL: http://www.webcitation.org/query?url=https%3A%2F%2Fcode.google.com%2Fp%2Fandroid-apktool%2F&date=2017-01-09. Son Erişim Tarihi: 10 Kasım 2016

76. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. (2009). The WEKA data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1), 10-18.

77. İnternet: Weka 3: Data Mining Software in Java. URL: http://www.webcitation.org/query?url=http%3A%2F%2Fwww.cs.waikato.ac.nz%2Fml%2Fweka%2F&date=2017-01-09. Son Erişim Tarihi: 10 Kasım 2016

78. Shen, Y., Chien, R., & Hung, S. (2014). Toward Efficient Dynamic Analysis and Testing for Android Malware. *Journal of IT Convergence Practice*, 2(3), 14-23.

79. Sheen, S., Anitha, R., & Natarajan, V. (2015). Android based malware detection using a multifeature collaborative decision fusion approach. *Neurocomputing*, 151, 905-912.

80. Hall, P., Racine, J., & Li, Q. (2004). Cross-validation and the estimation of conditional probability densities. *Journal of the American Statistical Association*, 99(468), 1015-1026.

81. Rodriguez, J. D., Perez, A., Lozano, J. A. (2010). Sensitivity analysis of k-fold cross validation in prediction error estimation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *32*(3), 569-575.

82. İnternet: Turkcell || Geleceği Yazanlar. URL: http://www.webcitation.org/query?url=https%3A%2F%2Fgelecegiyazanlar.turkcell.com.tr%2Fkonu%2Fandroid%2Fegitim%2Fandroid-201%2Fandroidmanifest.xml&date=2017-01-09. Son Erişim Tarihi 10 Kasım 2016

83. İnternet:Apktool,decode apk. URL: http://www.webcitation.org/query?url=http%3A%2F%2Fwww.javadecompilers.com%2Fapktool&date=2017-01-09. Son Erişim Tarihi 10 Kasım 2016

ÖZGEÇMİŞ

Kişisel Bilgiler

Soyadı, adı : KAYABAŞI, Gülsüm
Uyruğu : T.C.
Doğum tarihi ve yeri : : 05.11.1987, Ankara
Medeni hali : Bekar
Telefon : 0 (312) 402 51 28
e-mail : gulsum.kayabasi@gazi.edu.tr

Eğitim

Derece	Eğitim Birimi	Mezuniyet tarihi
Yüksek lisans	Gazi Üniversitesi /Bilgisayar Mühendisliği	Devam Ediyor
Lisans	Osmangazi Üniversitesi /Bilgisayar	2010
Lise	Etimesgut Mehmetçik Anadolu Lisesi	2005

İş Deneyimi

Yıl	Yer	Görev
2010-Halen	Genelkurmay Başkanlığı	Sistem Yöneticisi

Yabancı Dil

İngilizce

Yayınlar

Kayabaşı, G., Doğru, İ. A. (2015). Mobil Cihazlarda Zararlı Yazılım Tespitinde Kullanılan Statik Analiz Araçları. *8. Uluslararası Bilgi Güvenliği ve Kriptoloji Konferansı* (ISC Turkey 2015)

KAYABAŞI, G., DOĞRU, İ. A. (2016). Mobil Uygulamaların Sınıflandırılmasında Kullanılan Makine Öğrenmesi Algoritmalarının Güvenirlilik Tespiti. *9. Uluslararası Bilgi Güvenliği ve Kriptoloji Konferansı* (ISC Turkey 2016), Ankara.

Hobiler

Kickbox, basketbol oynamak, kitap okumak

GAZİ GELECEKTİR...