T.C.
İSTANBUL ÜNİVERSİTESİ-CERRAHPAŞA
LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

DOKTORA TEZİ

MAKİNE ÖĞRENMESİ YAKLAŞIMI İLE ANDROİD UYGULAMALARIN GÜVENLİK ANALİZİ

Gökhan ÖZOĞUR

DANIŞMAN
Doç. Dr. Muhammed Ali AYDIN

II. DANIŞMAN
Dr. Öğr. Üyesi Mehmet Ali ERTÜRK

Bilgisayar Mühendisliği Anabilim Dalı

Bilgisayar Mühendisliği Programı

Mayıs, 2023

Eşim Hatice NİZAM ÖZOĞUR'a ithaf ediyorum…

BÜTÇE DESTEKLERİ

MAKİNE ÖĞRENMESİ YAKLAŞIMI İLE ANDROİD UYGULAMALARIN GÜVENLİK ANALİZİ

Bu tez çalışması için herhangi bir kurumdan bütçe desteği alınmamıştır.

TEŞEKKÜR

Doktora tez çalışmam süresince bilgi ve tecrübeleriyle bana yol gösteren ve yardımlarını esirgemeyen değerli tez danışmanlarım Doç.Dr. Muhammed Ali AYDIN'a ve Dr.Öğr.Üyesi Mehmet Ali ERTÜRK'e teşekkür ederim. Tez izleme komitemde yer alarak çalışmama fikirleri ile katkıda bulunan Prof.Dr. Berk CANBERK'e ve Dr.Öğr.Üyesi Özgür Can TURNA'ya saygı ve şükranlarımı sunarım. Siber Güvenlik/Kriptoloji alanında YÖK 100/2000 Doktora Bursu ile çalışmalarımı destekleyen Yükseköğretim Kurulu'na teşekkür ederim. İyi bir eğitim almam için maddi ve manevi desteklerini esirgemeyen sevgili aileme teşekkürlerimi bir borç bilirim. Hayatın her alanında olduğu gibi doktora çalışmalarım sırasında da yanımda olan sevgili eşime destekleri için teşekkür ederim.

Mayıs 2023 **Gökhan ÖZOĞUR**

İÇİNDEKİLER

Sayfa No

ŞEKİL LİSTESİ

Sayfa No

TABLO LİSTESİ

Sayfa No

KISALTMA LİSTESİ

Kısaltmalar		Açıklama
API	:	Uygulama Programı Arayüzü
APK	:	Android Uygulama Paketi
ART	:	Android Uygulama Çalıştırma Ortamı
CBoW	:	Sürekli Kelime Çantası
CNN	:	Evrişimli Sinir Ağları
CPU	:	Merkezi İşlem Birim
DDoS	:	Dağıtılmış Hizmet Reddi
DEX	:	Dalvik Çalıştırma Dosyası
DVM	:	Dalvik Sanal Makinesi
FN	:	Yanlış Negatif
FNR	:	Yanlış Negatif Oranı
FP	:	Yanlış Pozitif
FPR	:	Yanlış Pozitif Oranı
GHz	:	Gigahertz
IoT	:	Nesnelerin İnterneti
IP	:	İnternet Protokolü
kB	:	Kilobayt
MB	:	Megabayt
MCC	:	Matthews Korelasyon Katsayısı
RAM	:	Rastgele Erişimli Hafıza
SSL	:	Güvenli Yuva Katmanı
SVM	:	Destek Vektör Makineleri
TF-IDF	:	Terim Frekansı – Ters Belge Frekansı
TN	:	Doğru Negatif
TP	:	Doğru Pozitif
TV	:	Televizyon
URL	:	Tekdüzen Kaynak Bulucu
XGBoost	:	Ekstrem Gradyan Arttırma
XML	:	Genişletilebilir İşaretleme Dili

ÖZET

DOKTORA TEZİ

MAKİNE ÖĞRENMESİ YAKLAŞIMI İLE ANDROİD UYGULAMALARIN GÜVENLİK ANALİZİ

Gökhan ÖZOĞUR

İstanbul Üniversitesi-Cerrahpaşa

Lisansüstü Eğitim Enstitüsü

Bilgisayar Mühendisliği Anabilim Dalı

Bilgisayar Mühendisliği Programı

Danışman : Doç. Dr. Muhammed Ali AYDIN

II. Danışman : Dr. Öğr. Üyesi Mehmet Ali ERTÜRK

Android, akıllı telefon pazarındaki en baskın işletim sistemidir. Çeşitli marketlerde milyonlarca uygulama bulunması nedeniyle zararlı uygulamaların kısa sürede tespit edilmesine ihtiyaç duyulmaktadır. Bu tezde, TF-IDF yöntemini kullanarak Android uygulamalarından bayt düzeyinde özellikler çıkaran ve XGBoost sınıflandırıcısı ile zararlı uygulamaları kısa sürede tespit eden benzersiz bir model geliştirilmiştir. Model bir uygulamayı dengeli bir veri setinde %99,05 F1-skor ile 2,75 saniyede ve gizlenme teknikleri uygulanmış uygulamaları içeren dengesiz bir veri setinde %99,35 F1-skor ile 3,30 saniyede zararlı veya zararsız olarak sınıflandırmıştır. Literatürdeki çalışmalar genellikle akıllı telefonlar için zararlı Android uygulamaların tespitini araştırmaktadır ve Android TV uygulamaları hakkında yalnızca birkaç çalışma bulunmaktadır. Önerilen model, internetten toplanan Android TV uygulamalarında da test edilmiştir. Herkese açık olarak paylaşılan çok fazla zararlı Android TV uygulaması olmadığı için zararsız uygulamalara zararlı kod parçaları eklenerek zararlı Android TV uygulamaları oluşturulmuştur. Deneysel sonuçlar, akıllı telefon uygulamaları üzerinde eğitilen modelin zararlı Android TV uygulamalarını tespit etmede yeterince iyi olmadığını göstermiştir. Ancak hem akıllı telefon hem de TV uygulamaları üzerinde eğitilen model, %96,28 F1-skor ile zararlı Android TV uygulamalarını başarıyla tespit etmiştir.

Mayıs 2023 , 80 sayfa.

Anahtar kelimeler: Makine Öğrenmesi, Zararlı Yazılım Tespiti, Android, Siber Güvenlik

ABSTRACT

Ph.D. THESIS

Security Analysis of Android Applications with Machine Learning Approach

Gökhan ÖZOĞUR

İstanbul University-Cerrahpaşa

Institute of Graduate Studies

Department of Computer Engineering

Computer Engineering Programme

Supervisor : Assoc. Prof. Dr. Muhammed Ali AYDIN

Co-Supervisor: Assist. Prof. Dr. Mehmet Ali ERTÜRK

Android is the most dominant operating system in the smartphone market. Since there are millions of applications in various markets, it is necessary to detect malwares in a short time. In this thesis, a unique model has been developed that extracts byte-level features from Android applications using the TF-IDF method and detects malwares in a short time with the XGBoost classifier. The model classified an application as malicious or benign with 99.05% F1-score in 2.75 seconds on a balanced dataset and in 3.30 seconds with 99.35% F1-score in an imbalanced dataset containing obfuscated applications. The studies in the literature generally investigate the detection of Android malwares for smartphones and there are only a few studies on Android TV malwares. The proposed model has also been tested on Android TV applications collected from the internet. Since there are not many Android TV malwares shared publicly, they have been created by injecting malicious payload into benign applications. Experimental results showed that the model trained on smartphone applications is not good enough to detect Android TV malwares. However, the model trained on both smartphone and TV applications successfully detected Android TV malwares with an F1 score of 96.28%.

May 2023, 80. pages.

Keywords: Machine Learning, Malware Detection, Android, Cybersecurity

1. GİRİŞ

Uluslararası Veri Kurumu'nun 2020 yılı için yayınladığı rapora göre akıllı telefonların %84,1'inde Android işletim sistemi kullanılmaktadır [1]. Android işletim sisteminin pazar payına hâkim olması milyonlarca uygulamanın çeşitli marketlerde yayınlanmasına neden olmuştur. Ancak bu uygulamaların hepsi iyi niyetli yazılımlar değildir ve birçok zararlı uygulama (malware) kötü niyetli olduğu fark edilmeden milyonlarca cihaza yüklenmektedir. G Data siber güvenlik şirketi 2019 yılında Android işletim sistemi için 4,18 milyon zararlı uygulama bulunduğunu raporlamıştır [2].

Zararlı yazılımların tespit edilebilmesi için yapılan çalışmalar temel olarak statik ve dinamik güvenlik analiz yöntemlerini kullanmaktadır. Statik analiz, uygulamayı çalıştırmadan yazılım paketi üzerinde yapılan güvenlik incelemeleridir. Tersine mühendislik yöntemleri ile elde edilen kaynak kodlar analiz edilerek uygulamanın çalışma mekanizması anlaşılabilmektedir. Dinamik analizde ise uygulamanın davranışları yürütme zamanı içerisinde takip edilerek analiz edilmektedir. Statik analiz uygulamanın hızlıca incelenmesinde avantajlı olmasına rağmen gelişmiş gizlenme yöntemleri kullanılarak atlatılabilmektedir. Gelişmiş zararlı yazılımların tespit edilebilmesi için yapılan analizlere makine öğrenmesi gibi gelişmiş yaklaşımlar dahil edilmektedir. Literatürde Android işletim sistemini kullanan akıllı telefonlar için geliştirilen uygulamaları makine öğrenmesi ve güvenlik analiz yöntemlerini kullanarak inceleyen birçok çalışma [3-20] bulunmaktadır.

Literatürdeki çalışmalardan faklı olarak tez çalışması kapsamında Android işletim sisteminin çalıştığı akıllı telefon uygulamalarının yanı sıra akıllı televizyon uygulamalarının da statik analiz yöntemi ile incelenmesi ve zararlı yazılımların tespit edilmesi hedeflenmiştir. Akıllı telefonlar için hazırlanan kötü niyetli uygulamalar sıklıkla telefon rehberine erişim, çağrı yapma ve SMS gönderme gibi API çağrılarını kullanarak zarar vermektedir. Android işletim sisteminde API kullanımı belirli izinler üzerinden yapılmaktadır. Statik analiz ile zararlı uygulama tespiti yapan çalışmalar analizlerinde bu API ve izinlerin kullanımını inceleyerek uygulamaların niyetini belirlemektedir. Akıllı televizyonlarda telefonlara özgü API ve izinler kullanılmadığı için sadece bu bilgileri analiz ederek karar veren çalışmalar televizyonlar için zararlı uygulamaların tespit edilmesinde yetersiz kalacaktır. Bu nedenle Android TV için

geliştirilen uygulamaların zararlı olup olmadığının tespit edilebilmesi için cihaz tipine bağlı olmayan davranışların incelenmesine ihtiyaç vardır. Lei ve diğ. [21] tarafından yapılan EveDroid adlı çalışmada akıllı televizyon ve buzdolabı gibi IoT cihazları için zararlı yazılımları tespit eden bir sistem önerilmiş ancak akıllı telefonlar dışında bir uygulama yapılmamıştır. Bu kapsamda literatürde Android TV uygulamaları için statik analiz yöntemiyle zararlı yazılımları tespit eden bir çalışmaya rastlanmamıştır.

Android uygulamalarının güvenlik analizinin makine öğrenmesi yaklaşımı ile gerçekleştirilmesi için tez kapsamında deneyler yapılarak özgün bir model geliştirilmiştir. Geliştirilen model Android TV uygulamaları üzerinde test edilerek zararlı Android TV uygulamalarının tespit edilmesi için çalışmalar yapılmıştır. Tez çalışması kapsamında yapılan ana katkılar şu şekildedir:

- TF-IDF yöntemini kullanarak Android uygulamalarından bayt düzeyinde öznitelik çıkaran ve XGBoost yöntemi ile zararlı uygulamaları tespit eden özgün bir model geliştirilmiştir.

- Gizlenme tekniklerinden etkilenmemek ve işlem süresini kısaltmak amacıyla uygulamayı temsil eden öznitelikler sadece deşifre edilmiş AndroidManifest dosyasından ve deşifre edilmemiş kaynak kodların ikili dosya formatında yer aldığı DEX dosyalarından bayt düzeyinde okunarak elde edilmiştir.

- Geliştirilen model literatürde paylaşılan farklı veri setlerinde test edilerek performansı ölçülmüş ve diğer çalışmalarla karşılaştırması yapılmıştır.

- Modelin performansını etkileyen faktörler incelenerek bulgular paylaşılmıştır.

- Web sayfalarını dolaşarak Android TV uygulamalarını indiren bir program yazılarak Android TV uygulama koleksiyonu oluşturulmuştur.

- Yeterli sayıda zararlı Android TV uygulaması bulunmaması nedeniyle modelin eğitimi için zararsız Android TV uygulama paketlerine zararlı kod parçaları eklenerek zararlı Android TV uygulamaları oluşturulmuştur.

- Mobil uygulamalar ile eğitilmiş modelin performansı TV uygulamaları üzerinde incelenmiş ve performansın arttırılması için çözüm önerileri yapılmıştır.

Tezin geri kalan kısmı şu şekilde organize edilmiştir: Bölüm 2'de Android işletim sistemi, zararlı yazılım çeşitleri, güvenlik analiz yöntemleri, makine öğrenmesi tipleri ve doğal dil işleme konularında teorik bilgiler sunulmuş ve literatürdeki benzer çalışmalara yer verilmiştir. Bölüm 3'te Android uygulama paketlerinin elde edilmesi, zararlı uygulamaların oluşturulması, uygulamaların etiketlenmesi, uygulamalardan öznitelik çıkarılması, öznitelik seçimi, zararlı uygulamaların tespit edilmesi, modelin uygulanması ve performans ölçümü konularında uygulanan yöntemler açıklanmıştır. Tez kapsamında yapılan deneyler sonucunda modelin performansını etkileyen faktörler ve Android TV uygulamalarının güvenlik analizleri ile ilgili bulgular Bölüm 4'te sunulmuştur. Elde edilen bulguların sonuçları Bölüm 5'te tartışılmıştır. Bölüm 6'da ise tez çalışması sonucunda elde edilenler değerlendirilmiş ve gelecek çalışmalar için önerilere yere verilmiştir.

2. KAVRAMSAL ÇERÇEVE

Tez çalışmasında makine öğrenmesi yaklaşımı ile Android uygulamalarının güvenlik analizleri gerçekleştirilmiştir. Kullanılan kavramların anlaşılabilir olması için bu bölümde konu ile alakalı teorik bilgilere ve benzer çalışmalara yer verilmiştir. Bölüm 2.1'de Android işletim sistemi hakkında temel bilgiler sunulmuştur. Bölüm 2.2'de zararlı yazılım çeşitleri tanıtılmıştır. Yazılımların analizlerinde kullanılan güvenlik analiz çeşitlerine Bölüm 2.3'te yer verilmiştir. Makine öğrenmesi çeşitleri Bölüm 2.4'te anlatılmıştır. Bölüm 2.5'te doğal dil işleme yöntemleri paylaşılmıştır. Android uygulamalarının güvenlik analizlerini makine öğrenmesi yöntemleri ile gerçekleştiren literatürdeki çalışmalardan Bölüm 2.6'da bahsedilmiştir.

2.1. ANDROİD İŞLETİM SİSTEMİ

Android işletim sistemi, Google tarafından cep telefonları, tabletler ve televizyonlar için geliştirilmiş Linux tabanlı açık kaynak kodlu bir işletim sistemidir. Bu işletim sistemi Çekirdek, Android Runtime (ART), Kütüphaneler, Uygulama çatısı ve Uygulama katmanı olmak üzere toplam 5 katmandan oluşmaktadır. Çekirdek katmanı, Linux çekirdeği üzerine inşa edilmiş Android mimarisindeki en alt katmandır. Güvenlik, hafıza yönetimi, süreç yönetimi, ağ yığınları ve sürücü modelleri bu katmanda bulunmaktadır. ART katmanında hafıza yönetimi, donanım sürücüleri gibi alt seviye işler yürütülmekte olup temel Java kütüphaneleri, Dalvik sanal makinesi (DVM - Dalvik Virtual Machine) ve çekirdek kütüphaneleri bu katmanda yer almaktadır. Çekirdeğin üstünde yer alan kütüphane katmanı genelde C++ ve C programlama dilleri ile yazılmış sistem kütüphanelerini içermektedir. Bu kütüphaneler içerisinde internet tarayıcısı motorlarının çalışması için Webkit, görüntüleme kontrolünü yapan Surface Manager, grafik işlemleri için OpenGL, ses ve video işlemleri için gereken Media Framework, veri yapıları kontrolü ve düzenlenmesi için SQLite gibi yapıları içerir. Uygulama Çatısı katmanında Android yazılım geliştiricilerine işletim sisteminin tüm özelliklerine erişim sağlayan servisler yer almaktadır. Bu servislere örnek olarak Aktivite Yöneticisi, Görünümler, Uyarı Yöneticisi, İçerik Sağlayıcılar ve Kaynak Yöneticisi servisleri verilebilir. Son katman olan uygulama katmanında doğrudan Java programlama dili ile geliştirilmiş uygulamalar yer almaktadır. Bu katman, bu mobil uygulamaların düzgün bir şekilde çalışabilmesi için diğer tüm katmanları

kullanır. Android işletim sisteminde desteklenen uygulama uzantısı ".apk"'dır. Kullanıcılara ücretsiz ve açık kaynak kodlu şekilde sunulmasından dolayı birçok alanda Android işletim sistemi tercih edilmekte ve akıllı saatler, akılı ev aletleri, otomotiv sektörü, yazıcılar gibi birçok teknolojik üründe ön plana çıkmaktadır.

2.1.1. Uygulama Paket Yapısı

Android uygulama paketi (APK) uygulamayı oluşturan "META-INF", "assets", "res" ve "lib" dizinleri ile "resources.arsc", "classses.dex" ve "AndroidManifest.xml" dosyalarının sıkıştırılması ile oluşan bir dosyadır. Şekil 2.1'de gösterilen APK dosya yapısını oluşturan komponentlerin açıklamaları şu şekildedir:

- "META-INF" dizini uygulamanın geliştirici tarafından paketlendikten sonra değişmediğinin doğrulanması için kullanılan imza dosyaları gibi uygulamaya ait meta verileri içeren dizindir.

- "assets" dizini AssetManager objesi yardımıyla uygulama tarafından erişilen statik dosyaları barındıran dizindir.

- "res" dizini derleme sırasında "resources.arsc" dosyası içerisine alınmayan kaynakların bulunduğu dizindir. Ekran çözünürlüğü, ekran yönü ve çoklu dil desteği gibi farklı konfigürasyonlar için kullanılan alternatif kaynak dosyaları bu dizinde yer almaktadır.

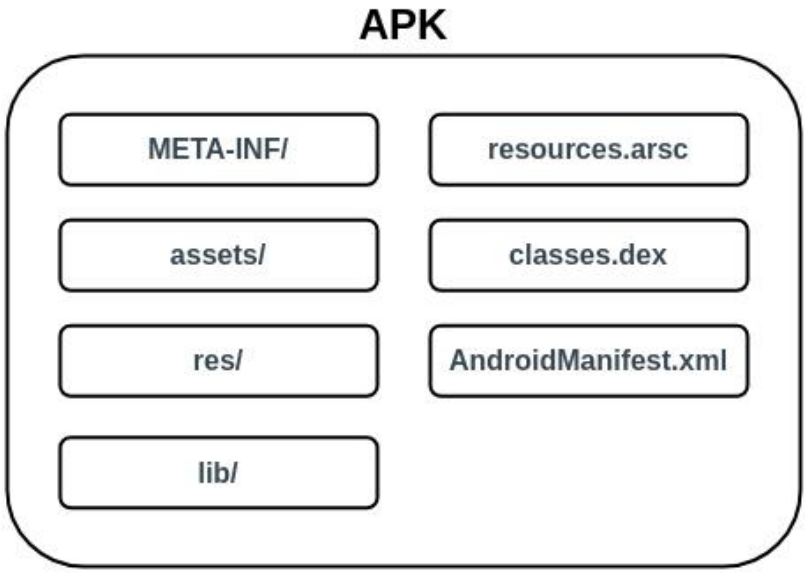
- "lib" dizini uygulamada kullanılan yazılım kütüphanelerinin çalıştırılacak cihazın işlemcisine göre derlenmiş dosyalarının bulunduğu dizindir. Desteklenen işlemci tiplerine göre "armeabi", "armeabi-v7a", "arm64-v8a", "x86", "x86_64", ve "mips" gibi alt dizinleri içermektedir.

- "resources.arsc" dosyası uygulamada kullanılan kaynakların derlenmiş halidir. Uygulamada ihtiyaç duyulan kaynak dosylarının tanımlayıcı isimleri verildiğinde istenilen konfigürasyona uygun olan dosyalara erişilmesini sağlamaktadır.

- "classes.dex" dosyası uygulamanın kaynak kodlarının Dalvik sanal makinesine göre derlenmesi sonucu oluşan byte kodu dosyasıdır.

- "AndroidManifest.xml" dosyası uygulamaya ait temel bilgilerin XML formatında tutulduğu dosyadır. Bu dosya uygulamaya ait isim, versiyon, erişim yetkileri gibi

bilgileri barındırmaktadır. Ayrıca uygulamada kullanılan kütüphane dosyaları, özellikler ve uygulamayı oluşturan komponentler bu dosyada belirtilmektedir.



Şekil 2.1: Android uygulama paketinin yapısı

2.1.2. Desteklenen Cihaz Tipleri

Android işletim sistemi cep telefonu, tablet, saat, televizyon ve araba gibi birçok akıllı cihazda çalışmaktadır. Bölüm 2.1.1'de bahsedildiği üzere uygulama paketleri farklı konfigürasyonlar için hazırlanmış alternatif kaynak dosyaları içermektedir. Benzer olarak uygulamanın gereksinim duyduğu özellikleri destekleyen cihazların filtrelenmesi de uygulama paketi içerisinden yapılabilmektedir. Android işletim sistemi uygulamaların sadece desteklenen cihazlara yüklenmesini ve cihazın konfigürasyonlarına uygun kaynak dosyalarının kullanılmasını sağlamaktadır.

2.1.3. Güvenlik Tehditleri

Android işletim sistemi üzerinde çalışan uygulamalardaki zafiyetler nedeniyle güvenlik açısından çeşitli tehditler barındırmaktadır. Liu ve diğ. [22], Google Play uygulama marketinde sunulan Android TV uygulamalarının VirusTotal sitesindeki tarama sonuçlarını, talep ettiği izinleri, güvenlik açıklarını ve gizlilik sızıntılarını incelemiştir. Elde ettikleri 3.163 uygulamaya ait VirusTotal analiz raporlarında en az 1 anti-virüs tarayıcısı tarafından zararlı olarak sınıflandırılan uygulama sayısını 89 olarak tespit etmişler ve bunların 30'unun oyun kategorisinde olduğunu belirtmişlerdir. Uygulama izinleri hakkında yaptıkları analizde tüm uygulamalar tarafından talep edilen toplam 1.103 izninden 299'unun TV'ye özgü olduğunu

tespit etmişlerdir. Yaptıkları çalışmada SSL güvenliği, WebView kullanımı ve alıcısı belli olmayan çağrı (implicit intent) kullanımının en fazla görülen 3 güvenlik açığı olduğunu gözlemlemişlerdir. Ayrıca dinamik analizler sonucunda uygulama veri tabanında bulunan veriler ve coğrafi konum gibi hassas verilerin ağ üzerinden sızdırıldığı ya da geliştiriciler tarafından hata ayıklama amacıyla konsola yazdırıldığı tespit edilmiştir. Yaptıkları çalışmada Android TV uygulamalarının da akıllı telefonlar gibi güvenlik zafiyetleri barındırdığı ve bu alanda daha fazla çalışma yapılmasına ihtiyaç olduğu belirtilmiştir.

2.2. ZARARLI YAZILIM ÇEŞİTLERİ

Zararlı yazılımlar cihaz ve servislere zarar vermek ya da onlardan faydalanmak üzere tasarlanmış kötü niyetli programlardır. Siber suçlular kurbanlarından veri elde etmek için çeşitli türlerde zararlı yazılımlar kullanmaktadır. Oluşturulan zararlı yazılımların birden fazla saldırı yöntemini birlikte kullanması sıklıkla karşılaşılan bir durumdur. Bu nedenle bir zararlı yazılım birden fazla kategoride zararlı olarak sınıflandırılabilir. Başlıca zararlı yazılım çeşitleri olarak virüsler, korkutucu yazılımlar, casus yazılımlar, reklam yazılımları, fidye yazılımları, solucanlar, truva atları ve dosyasız zararlı yazılımlar görülmektedir [23].

2.2.1. Virüsler

Biyolojik olarak virüsler canlı hücreler sayesinde kendilerini kopyalar. Bilgisayar virüsleri de kendilerini çoğaltarak diğer dosyalara ve ağ üzerinden diğer bilgisayarlara yayılmaları nedeniyle bu şekilde isimlendirilmiştir. Diğer dosyaların arasına gizlenen virüsler etki edebilmek için çalıştırılabilir dosyalara ihtiyaç duymaktadır. Bağlı bulunan dosyanın çalışması ile birlikte virüs çalışmaya başlar ve kendini çoğaltır. Yapılmak istenilen saldırının bir kaynaktan değil de daha fazla sayıda farklı kaynaktan başlatılabilmesi nedeniyle sistem performansının düşürülmesi ve Dağıtılmış Hizmet Reddi (Distributed Denial of Service - DDos) saldırılarında sıklıkla virüsler tercih edilmektedir.

2.2.2. Korkutucu Yazılımlar

Korkutucu yazılımlar, kullanıcıyı gerçek olmayan bir duruma inandırıp ücret talep eden zararlı yazılımlardır. Bu tür yazılımlar genellikle kullanıcının mobil cihazında veya bilgisayarında virüs bulaştığına dair açılır pencerede görüntülenen mesajlar veya kişisel bilgilerin ve mali durumun risk altında olduğunu bildiren büyük bir uyarı düğmesi olan ekran görüntüsü ile görüntülenir. Platformdan bağımsız bir şekilde sunulan bu yazılımlar kullanıcıları şoka,

endişeye ve korkuya sürükleyerek bir kullanıcıları program veya bir hizmeti satın almaya zorladıkları için bir sosyal mühendislik taktiğidir.

2.2.3. Casus Yazılımlar

Casus yazılım, bilgisayara yüklü kullanıcıya ait kişisel bilgileri, sosyal aktiviteleri ve alışkınları içeren bilgileri kullanıcının onayı olmadan ele geçirip sahibine ileten kötü amaçlı yazılım türüdür. Bu tür yazılımlar virüsler gibi kendilerini herhangi bir yolla bir bilgisayardan diğerine kopyalamaz. Casus yazılımlar kullanıcıların cihaz üzerindeki her tür iletişimi izlemelerini sağlayıp genellikle kredi kartı ve banka hesap bilgilerini ele geçirmeyi amaçlamaktadır.

2.2.4. Reklam Yazılımları

Reklam yazılımları, kullanıcının izni olmadan pazarlama verilerini toplamak amacıyla istenmeyen reklamlar gönderen zararlı programlardır. Bu tür yazılımlar genellikle virüslü web siteleri ziyaret edildiğinde, ücretsiz yazılım veya paylaşılan bir yazılım ile kullanıcı bilgisayarlarına yüklenmektedir. Reklam yazılımları istenmeyen reklamları görüntülemenin yanı sıra yüklendiği cihazın performansını yavaşlatma, web tarayıcısına istenmeyen eklentileri yükleme, ana sayfayı değiştirme gibi tarayıcıların çalışmasını etkilemektedir.

2.2.5. Fidye Yazılımları

Fidye yazılımları bir kullanıcının bilgisayarına yüklenerek diskte depolanan verileri şifreler ve kullanıcıya bu verileri geri vermek için bir fidye talebi görüntüleyen zararlı yazılımlardır. Bu yazılımlar genellikle kimlik avı e-postaları veya drive-by indirmeler aracılığıyla kullanıcıların bilgisayarlarına iletilmektedir. Kimlik avı, güvenilir görünümlü e-postalar yardımıyla kullanıcıdan bir bağlantıya tıklamaya veya bir dosya ekini açmaya yöneliktir. Drive-by indirmeler kullanıcının bir izni olmadan internetten otomatik bir şekilde kullanıcının bilgisayarına indirilip yüklenen yazılımlardır. Farklı yöntemlerle fidye yazılımı kullanıcının bilgisayarına yüklenerek işletim sistemi, dosyalar, disk ve kullanıcı şifreleri gibi kullanıcının tüm erişimini ele geçirir.

2.2.6. Solucanlar

Solucanlar ağ trafiği ve sistem güvenliğini tehlikeye atan genellikle e-posta, anlık mesajlaşma, dosya paylaşımı ve internet gibi çeşitli yollarla bilgisayarlara bulaşan zararlı yazılımlardır. Virüslerin bir alt türü olarak bilinen solucanlar, bulaştıkları ağ ve sistemlerde etkin kalıp kendi kendilerini kopyalamaya çalışır. Bu çoğalma ile bilgisayarlara yayılabilen solucanlar arka planda sistem kaynaklarını tükettikleri için bilgisayarlarda performansın düşmesine neden

olabilir. Bu solucanlar yardımıyla siber saldırganlar istedikleri IP ve URL bilgilerine ulaşarak DDos saldırısı yapabilir, ele geçirilen bilgiler ile haksız gelir elde etmek için ücretli bağlantılara tıklayabilir.

2.2.7. Truva Atları

Truva atları, zararsız görünümlü yazılım gibi davranan ve kullanıcının kendi isteği ile indirip kullanmasını sağlayan en sık rastlanan kötü amaçlı yazılımlardır. Truva atları, diğer bilgisayar virüsleri ve solucanlarının aksine kendi kendilerine çoğalamaz. Bir bilgisayara yüklenen zararlı yazılım cihazın kontrolünü ele geçirmek, kullanıcı verilerini ve şifrelerini almak ve saldırgana göndermek, cihaza farklı zararlı yazılımları yüklemek ve çalıştırmak, cihazın ve bilgisayar ağlarının performansını düşürmek, verileri kopyalamak, silmek, değiştirmek, engellemek gibi birçok kötü amaç için kullanılmaktadır.

2.2.8. Dosyasız Zararlı Yazılımlar

Dosyasız zararlı yazılımlar, dosya sistemini doğrudan kullanmayan kötü amaçlı bir yazılım türüdür. Bu yazılımlar bir cihaza girmek için yasal uygulamaları istismar ederek fark edilmeden kodları çalıştırmaya veya verileri çıkarmaya başlar. Bu yazılımların dosya sistemi üzerinde bir iz bırakmaması bir antivürüs programı tarafından bile fark edilip kaldırılmasını zorlaştırmaktadır. Dosyasız zararlı yazılım saldırıları, kullanıcıları sosyal mühendislik yöntemleri ile ele geçirmeyi hedeflemektedir.

2.3. GÜVENLİK ANALİZLERİ

Zararlı yazılımların tespit edilebilmesi için öncelikle analiz edilmeleri gerekmektedir. Yazılımın nasıl çalıştığını ve niyetini anlamak için uzman kişi ve sistemler tarafından çeşitli yöntemlerle gerçekleştirilen güvenlik analizleri statik, dinamik ve hibrit analiz olmak üzere 3 kategoriye ayrılmaktadır.

2.3.1. Statik Analiz

Statik analiz, yazılımın çalıştırılmasına gerek duyulmadan yazılım dosyaları üzerinde yapılan güvenlik incelemeleridir. Genellikle derleme işlemi sonrasında ikili dosya formatında sunulan çalıştırılabilir yazılım dosyalarını oluşturan konfigürasyonlar, komutlar ve kaynak kodlar tersine mühendislik yöntemleri kullanılarak elde edilmeye çalışılmaktadır. Yazılımı oluşturan bu dosyalar analiz edilerek yazılımın çalışma mekanizması ve amacı anlaşılabilmektedir.

Örneğin; kaynak kodlar incelenerek yazılımın bilgisayardaki hangi dosyalara erişim sağladığı ve kullanıcı verilerini hangi internet sitelerine gönderdiği görülebilmektedir.

Yazılımın çalıştırılmasına ihtiyaç duyulmadığı için statik analiz işlemleri hızlıca yapılabilmektedir ancak gelişmiş gizlenme yöntemleri kullanılarak atlatılabilmektedir. Tersine mühendislikle kaynak kodların elde edilmesini engellemek amacıyla zararlı yazılımı oluşturan kötü niyetli kişiler sıklıkla kaynak kodu karmaşıklaştırma (obfuscating) yöntemini kullanmaktadır. Kaynak kodun karmaşıklaştırılması yazılımın çalışma mekanizmasını etkilememesine rağmen anlaşılabilirliğini zor hale getirmektedir. Kod karmaşıklaştırma işlemi yazılım içerisinde kullanılan değişken, fonksiyon, sınıf ve dosya isimlerini anlamsız ya da rastgele karakterlerden oluşan kelimeler ile değiştirerek anlaşılmasını ve takip edilmesini zorlaştırmaktadır. Fonksiyon çağrılarının aracı başka fonksiyonlar üzerinden yapılması ve kaynak kod içerisine kullanılmayan kod parçaları eklenmesi gibi farklı kod karmaşıklaştırma yöntemleri kullanılarak zararlı yazılımların gizlenmesi sağlanabilmektedir.

2.3.2. Dinamik Analiz

Dinamik analiz, yazılımın çalışma sırasındaki davranışlarını takip eden güvenlik incelemeleridir. Yazılımın çalıştığı cihaza ve ağa zarar verme ihtimali nedeniyle dinamik analiz işlemleri izole edilmiş korumalı bir ortamda yapılmaktadır. Dinamik analiz yöntemi ile incelenecek yazılım çalıştırılmadan önce ve çalışma sırasında ortam davranışları kayıt altına alınarak anormal durumlar tespit edilmektedir. Örneğin; ağ üzerinden gerçekleştirilen paket trafiği takip edilerek kullanıcı verilerinin hangi adreslere gönderildiği anlaşılabilir.

Zararlı yazılımı oluşturan kötü niyetli kişiler yazılımın çalıştığı izole ortamın normal bir kullanıcıya ait olup olmadığını anlamak ve dinamik analizden kaçınmak için çeşitli yöntemler kullanmaktadır. Örneğin; dinamik analiz testlerinin zaman kısıtı nedeniyle çok uzun sürmeyeceği varsayılarak yazılım çalışmaya başladığı anda zararlı davranışları sergilemek yerine belli bir süre gizlendikten sonra saldırıya başlayabilir.

2.3.3. Hibrit Analiz

Hibrit analiz, statik ve dinamik analiz yöntemlerinin avantajlarından yararlanmak amacıyla birlikte uygulandığı incelemelerdir. Statik analiz, dinamik analize kıyasla daha kısa sürede sonuç üretmekte ve yazılımın tümüne erişim imkânı sağlamaktadır ancak gelişmiş kod karmaşıklaştırma yöntemlerine karşı etkisiz kalmaktadır. Dinamik analiz ise statik analiz ile anlaşılması güç olan ya da tersine mühendislikle elde edilemeyen durumlarda yazılımın

davranışlarının incelenmesini mümkün kılmaktadır. Ancak dinamik analiz ile tüm koşulların test edilmesi çok uzun zaman gerektirmekte, hatta bazı durumların test edilmesi mümkün olmamaktadır.

Hibrit analiz ile incelenecek yazılıma öncelikle statik analiz yöntemleri uygulanarak genel bilgiler edinilmekte, sonrasında dinamik analiz yöntemleri uygulanarak yazılımın davranışları çıkarılmaktadır. Statik ve dinamik analiz sonucunda elde edilen bilgiler harmanlanarak yazılımın niyeti belirlenmektedir.

2.4. MAKİNE ÖĞRENMESİ

Yapay zekâ bilim dalının alt kümesi olan makine öğrenmesi bilgisayarların veri yardımıyla öğrenen bir sistem olmasını konu almaktadır. Bilgisayarların öğrenerek işlem yapabilmesi sayesinde açık bir şekilde tüm koşulları programlamaya gerek kalmadan istenilen işlemlerin yapılabilmesi sağlanmaktadır. Makine öğrenmesi yöntemleri alışveriş sitelerinde müşterinin tercihlerine uygun ürünlerin önerilmesi, sosyal medyada benzer kişilerin tavsiye edilmesi, sağlık verilerinin analiz edilerek erken uyarı verilmesi ve yazılımların güvenlik analizlerinin gerçekleştirilmesi gibi birçok alanda kullanılmaktadır. Makine öğrenmesi yöntemleri öğrenme şekline göre gözetimli, gözetimsiz ve pekiştirmeli öğrenme olmak üzere 3 kategoriye ayrılmaktadır [24].

2.4.1. Gözetimli Öğrenme

Gözetimli öğrenme etiketlenmiş veriyi kullanarak makine öğrenmesi modelinin eğitim işlemini gerçekleştirmektedir. Bu yöntem ile eğitim işlemi gerçekleştirilirken veri kümesi eğitim ve test verisi olarak ikiye bölünür. Öğrenme işleminde eğitim verilerindeki girdi değişkenleri ve çıktı değişkenleri arasındaki bağlantı öğrenilir ve bu bağlantı bilinmeyen test verilerinin çıktılarını tahmin etmek için kullanılır [25]. Literatürde bu öğrenme yöntemi denetimli öğrenme, danışmanlı öğrenme ve eğiticili/eğitmenli öğrenme olarak da bilinmektedir. Gözetimli öğrenme eğitilen modelin verdiği çıktıların türlerine göre sınıflandırma veya regresyon problemleri olarak ikiye ayrılmaktadır. Sınıflandırma problemleri girdi değişkenlerini bir sınıfa atamayı amaçlar [26]. Örneğin; kullanıcıların cihazlarına yükleyeceği yazılımların zararlı olup olmadığının tespitini gerçekleştirmek için eğitilen bir modelin çıktılarının zararlı ve zararsız olduğu sınıflandırma problemleri örnek verilebilir. Regresyon problemleri ise iki veya daha fazla girdi değişkeni arasındaki ilişkiyi ölçmeyi sağlar [26]. Bu problemlerde girdi

değişkenlerinin çıktı sonuçları gerçek değerler olarak temsil edilir. Modelin başarımı gerçek değer ve tahmin edilen çıktı değeri arasındaki farkın büyüklüğüne göre ölçülür. Örneğin; bir ilin bir haftalık hava durumu bilgilerine göre sonraki günlerde hava durumunun tahmini regresyon problemi olarak örnek verilebilir.

2.4.2. Gözetimsiz Öğrenme

Gözetimsiz öğrenme etiketsiz veriden girdi değişkenlerinin arasındaki ilişki ve yapıların öğrenilmesi işlemini gerçekleştirir [27]. Eğitim işleminde etiketlenmemiş veri kümesinin makine öğrenmesi yöntemleri kullanılarak analizi ve kümelenmesi gerçekleştirilir. Gözetimsiz öğrenme yöntemi kümeleme, boyut azaltma ve ilişkilendirme gibi görevlerde kullanılabilir. Kümeleme işleminde veri kümesindeki özniteliklerin benzerlik veya farklılıklarına göre etiketlenmemiş veri gruplandırılır. İlişkilendirme işleminde veri kümesindeki değişkenlerin arasındaki ilişkiler belirli kurallara göre tespit edilir. Boyut azaltmada ise veri kümesindeki özniteliklerin sayısı fazla olduğunda veri bütünlüğü bozulmadan giriş değişkenlerinin sayısı makul bir sayıya indirilir.

2.4.3. Pekiştirmeli Öğrenme

Pekiştirmeli öğrenme onaylanan davranışların ödüllendirildiği ve onaylanmayan davranışların cezalandırıldığı sistemine dayalı bir öğrenme yöntemidir [28]. Makine öğrenmesi modeline verilen test verilerin çıktılarının tespiti girdi ve çıktıların bilindiği etiketli verilerin verildiği gözetimli öğrenme veya hiçbir etiket bilgisinin verilmediği gözetimsiz öğrenme yöntemlerinden farklı bir şekilde tespit edilir. Pekiştirmeli öğrenme yönteminde test verilerine ait girdilerin çıktıları çevre ile etkileşimler dikkate alınarak deneyimlere dayalı olarak ödül ve cezaları kullanarak tespit edilir. Örneğin; evlerde kullanılan bir robot süpürge bir sonraki odayı süpürmek için şarj seviyesine ve o anda bulunduğu konumla gideceği oda arasındaki mesafeye göre odayı temizleyip temizlememeye karar vermesi pekiştirmeli öğrenme yöntemine bir örnek olarak verilebilir.

2.5. DOĞAL DİL İŞLEME

Yapay zekâ ve dil bilim dallarının kesişim kümesinde olan doğal dil işleme insanların birbirleriyle iletişim kurmak amacıyla kullandıkları doğal dillerin bilgisayarlar tarafından işlenmesini konu almaktadır. Doğal dillere ait yapı ve kuralların bilgisayarlar tarafından daha iyi çözümlenmesini sağlayan modellerin geliştirilmesiyle duygu analizi, metin içerisindeki

yazım hatalarının düzeltilmesi, diller arası otomatik çeviri yapılması, dokümanların özetlenmesi ve soruların otomatik cevaplanması gibi uygulamalar hayatımıza girmiştir.

Yapay zekâ alanında bir metin verisi ile çalışabilmek için öncelikle verileri sayısal olarak ifade etmek gerekmektedir. Böylece, makine öğrenmesi algoritmaları bu verileri tanıyabilecek ve veriler hakkında çıkarımlar yapabilecektir. Bir metin içerisinde bulunan kelimelerin tek tek kelimenin sayısal temsili olan bir vektöre dönüştürüldüğü tekniklere kelime temsil (word embedding) yöntemleri denilmektedir. Başlıca kelime temsil yöntemleri olarak kelime çantası modeli, terim frekansı - ters belge frekansı yöntemi, n-gram ve word2vec yöntemi gösterilmektedir.

2.5.1. Kelime Çantası Modeli

Kelime çantası modeli, orijinal olarak bir belgenin temsili için kullanılan bir doğal dil işleme yöntemidir [29]. Temelde bu yöntem doküman analizinde bir metinden öznitelik çıkarmak için kullanılmaktadır. Bu yaklaşım, bir belgenin vektör olarak temsilini elde etmek için belge içerisindeki her bir ögeyi ayrı ayrı eşleştirir ve sayar. Bunu gerçekleştirirken kelimelerin sırası, aralarındaki bağıntı ve dilbilgisi detayları ihmal edilir. Model sadece bilinen kelimelerin belgede yer alıp almadığı ile ilgilenir, belgede yer aldığı yer ile ilgilenmez.

2.5.2. Terim Frekansı - Ters Belge Frekansı Yöntemi

Genellikle doğal dil işleme çalışmalarında tercih edilen Terim Frekansı - Ters Belge Frekansı Yöntemi (TF-IDF) yönteminde belgeler terimlerin (kelimelerin) kullanım durumuna göre temsil edilmektedir ve terimlerin belge içerisindeki konumları dikkate alınmamaktadır. Kelime temsilinin elde edilmesi için belgelerde bulunan terimlerin listelenmesi, terimlerin her bir belgede kaç defa kullanıldığının sayılması ve terimlerin kullanım durumuna göre öneminin (ağırlığının) belirlenmesi gerekmektedir.

TF-IDF yöntemine göre çoğu belgede yer alan terimler daha az önemli olarak ele alınmaktadır. TF-IDF ağırlık değerinin hesaplanması Formül 2.1'de gösterilmektedir. Bir belgede (d) bulunan terimin (t) ağırlığı (tfidf) terim frekansı (tf) ve ters belge frekansı (idf) değerlerinin çarpılmasıyla elde edilmektedir. Formül 2.2'de gösterilen terim frekansı, bir terimin belge içerisindeki kullanım sayısının (f) belgedeki tüm terimlerin kullanım sayılarının toplamına bölünmesi ile hesaplanmaktadır. Formül 2.3'te gösterilen ters belge frekansı, toplam belge sayısının (n) seçilen terimi içeren belgelerin sayısına (df) oranının logaritmasına 1 değerinin eklenmesi ile hesaplanmaktadır. Bir terimin hiçbir belgede bulunmaması durumunda sıfıra

bölme işleminin gerçekleşmemesi için toplam belge sayısı ve terimi içeren belge sayısına 1 değerleri eklenerek oran hesaplanması yapılmıştır. Son olarak ham değerlerin olduğu vektöre Formül 2.4'de gösterilen Öklid (L2) normu uygulanarak normalizasyon sağlanmıştır.

$$tfidf(t,d) = tf(t,d)x\ idf(t) \tag{2.1}$$

$$tf(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}} \tag{2.2}$$

$$idf(t) = \log\frac{1+n}{1+df(t)} + 1 \tag{2.3}$$

$$V_{norm} = \frac{v}{\sqrt{v_1^2+v_2^2+\cdots+v_n^2}} \tag{2.4}$$

2.5.3. N-gram

N-gram, n adet elemana sahip ardışık dizilerden oluşan doğal dil işleme ve hesaplamalı dilbilim alanlarında kullanılan istatiksel bir dil modelidir [30]. N-gram modelleri, ardışık bir dizinin bir sonraki elemanını bulmak için (n-1) formatlı Markov zincirinden yararlanır. N-gram modellerinin unigram, bigram, trigram ve n'inci gram veya karakteri içeren farklı gösterimleri vardır [31]. Bu gösterimlere n sayısı 1 ise unigram, 2 ise bigram, 3 ise trigram olarak n sayısının büyüklüğüne göre isimlendirmeler verilir. Eğer n sayısı 3'ten büyük ise n-gram olarak temsil edilir. N-gram modelleri birçok uygulama alanında kullanılmaktadır. Örneğin; mobil cihazlarda mesaj yazarken faydalandığımız kelime tahmin özelliği ve otomatik düzeltme n-gram modellerinden faydalanmaktadır.

2.5.4. Word2Vec

Word2Vec uygulaması Mikolov ve diğ. [32] tarafından kelimeleri vektör uzayında sayısal olarak temsil etmeyi sağlayan bir gözetimsiz öğrenme ve tahmin tabanlı bir yaklaşımdır. Bu model Sürekli Kelime Çantası (Continuous Bag of Words - CBOW) ve Skip-gram olmak üzere iki mimariyi kullanır [33]. Sürekli Kelime Çantası modeli, ileri beslemeli bir sinir ağı mimarisine benzemektedir. Bu model, girdi olarak pencere boyutu merkezinde olmayan kelimeleri alır ve bu merkezde olan kelimeleri çıktı olarak tahmin için kullanır [33]. Skip-Gram modeli ise girdi olarak pencere boyutu merkezinde olan kelimeleri alır ve verilen kelimenin etrafındaki kelimeleri çıktı olarak tahmin etmek için kullanır [33]. Word2Vec modelde her bir kelimeyi temsil eden vektör bir sinir ağına benzeyen bir şekilde öğrenilir ve bu vektörler yardımıyla o kelimenin çeşitli özelliklerini metin içerisinden elde etmeye çalışır. Bu vektörler

yardımıyla kelimeler arasında benzerlikler, farklılıklar belirlenebilir ve birçok doğal dil işleme çalışmasında kullanılabilir.

2.6. LİTERATÜR TARAMASI

Literatürde Android işletim sisteminde çalışan uygulamaların güvenlik analizlerini makine öğrenmesi yöntemleri ile gerçekleştiren birçok çalışma bulunmaktadır. Badhani ve Muttoo [3], CENDroid adını verdikleri çalışmada kümeleme ve topluluk öğrenmesi yöntemlerini bir arada kullanarak zararlı ve zararsız uygulamaları ayrıştırmıştır. Uygulamalar, statik analizle edilen API izin ve etiket bilgilerine göre K-Ortalama yöntemiyle kümelere bölündükten sonra Karar Ağacı, Aşırı Öğrenme Makineleri, Lojistik Regresyon, JRip ve Destek Vektör Makineleri (SVM) sınıflandırıcılarının birlikte kullanıldığı Topluluk Öğrenmesi yöntemi ile sınıflandırılmıştır. Önerdikleri yöntem 3 farklı veri seti (D1: 906 zararlı, 1.776 zararsız; D2: 2.929 zararlı, 7.166 zararsız; D3: 21.715 zararlı, 7.166 zararsız uygulama) üzerinde test edilmiş ve F1-skor metriğine göre sırasıyla %99, %95 ve %97 başarı elde edilmiştir.

Rathore ve diğ. [4], makine öğrenmesi yöntemlerinin düşmancıl saldırılara karşı sağlamlığını incelemek amacıyla Lojistik Regresyon, SVM, Karar Ağacı, Rastgele Orman, Ekstra Ağaçlar, Adaptif Yükseltme, Gradyan Yükseltme ve Derin Öğrenme yöntemlerini kullanarak Android işletim sisteminde çalışan zararlı uygulamaları tespit eden sistemlerin performansını 5.569 zararlı ve 5.721 zararsız uygulamadan oluşan bir koleksiyonda analiz etmiştir. Önerilen Pekiştirmeli Öğrenme yöntemi ile uygulama izinlerinin değiştirilmesi sonucunda zararlı uygulamaların tespit sistemleri tarafından zararsız olarak sınıflandırıldığı gösterilmiştir. Ayrıca değişiklik yapılmış uygulamalara doğru etiketler verilerek eğitim işleminin tekrarlanması durumunda tespit sistemlerinin sağlamlığının arttığı sonucuna varılmıştır.

Yuan ve diğ. [5], statik analiz ile elde ettikleri uygulama izinlerine TF-IDF yöntemini kullanarak risk puanı vermiştir. Uygulama izin sayısını ve risk puanını kullanarak uygulamaların zararlı olup olmadığı analiz edilmiştir. 9.419 zararlı ve 6.070 zararsız uygulama üzerinde Naive Bayes, Bayes Ağları, Karar Ağacı, Rastgele Ağaç, Rastgele Orman ve K-En Yakın Komşu (K-Nearest Neighbors - KNN) yöntemleri kullanılarak sınıflandırma yapılmış ve F1-skor metriğine göre en başarılı yöntem %98'in üzerinde başarı gösteren Karar Ağacı yöntemi olmuştur.

Kumar ve diğ. [6], statik analizle elde ettikleri uygulama izinlerini zararlı ve zararsız uygulamalarda kullanılma oranlarına göre puanlamış ve uygulamaları kullandıkları izinlere göre sınıflandırmıştır. Diğer çalışmalardan farklı olarak izinlerin kullanılma durumları öznitelik vektöründe ikili (binary) yerine kayan noktalı (floating) sayı ile temsil edilmiştir. 5.553 zararlı ve 5.818 zararsız uygulamadan oluşan bir koleksiyon üzerinde Karar Ağacı, Naive Bayes, KNN, SVM, Rastgele Orman ve Adaptif Yükseltme yöntemleri kullanılarak yapılan sınıflandırma sonucunda en iyi sonuç F1-skor metriğine göre %95 başarı ile Rastgele Orman yöntemi ile elde edilmiştir. 100 ağaçtan oluşan Rastgele Orman yönteminde kayan noktalı sayı gösterimi ikili gösterime kıyasla %2 daha iyi sonuç vermiştir.

Feizollah ve diğ [7], Androdialysis adını verdikleri çalışmada uygulama içi ve uygulamalar arası mesajlaşma için kullanılan Intent yapısının zararlı yazılım tespit edilmesindeki etkisini incelemiştir. Yaptıkları çalışmada 5.560 zararlı ve 1.846 zararsız uygulama Bayes Ağları yöntemi kullanılarak sınıflandırılmıştır. Doğru Pozitif Oran metriğine göre sadece uygulama izinleri kullanıldığında %83, sadece Intent bilgisi kullanıldığında %91, ikisi birlikte kullanıldığında %95'in üzerinde başarı elde edilmiştir.

Milosevic ve diğ. [8], tersine mühendislikle elde ettikleri uygulama kodlarından oluşturdukları Kelime Çantası modelini ve uygulama izinlerini kullanarak 200 zararlı ve 200 zararsız uygulamadan oluşan bir koleksiyon üzerinde sınıflandırma ve kümeleme çalışması yapmıştır. Zararlı ve zararsız uygulamaların sınıflandırılması için Karar Ağacı, Rastgele Orman, Bayes Ağları, Sıralı Minimal Optimizasyon ile SVM, JRip, Lojistik Regresyon ve bunların birlikte kullanıldığı Topluluk Öğrenmesi yöntemleri; kümeleme için ise K-Ortalama, En Uzak İlk ve Beklenti Maksimizasyonu yöntemleri uygulanmıştır. Çalışma sonucunda F1-skor metriğine göre %95'in üzerinde başarı gösteren Kelime Çantası modelinin Topluluk Öğrenmesi ile birlikte kullanılması en iyi yöntem olarak belirlenmiştir.

Mehtab ve diğ. [9], AdDroid adını verdikleri çalışmada tersine mühendislikle elde ettikleri API çağrılarını, izinleri ve ilgili komutları analiz ederek bunların çeşitli kombinasyonlarda birlikte bulunduğu 63 adet kural belirlemiş ve bu kuralların zararlı ve zararsız uygulamalardaki kullanımını incelemiştir. 910 zararlı ve 510 zararsız uygulamanın önişleme ve örnekleme işlemlerinden geçirilmesi sonucu elde edilen 1.133 zararlı ve 1.000 zararsız uygulama Adaptif Yükseltme ve Karar Ağacı yöntemleri kullanılarak sınıflandırılmış ve F1-skor metriğine göre %99 başarı elde edilmiştir.

Wongwiwatchai ve diğ. [10], kişisel verileri sızdıran uygulamaları tespit etmek amacıyla tersine mühendislikle elde ettikleri uygulama izinlerini, uygulama kodlarını, uygulama bileşenlerini, sertifikaları Yapay Sinir Ağları, Lojistik Regresyon, SVM, Naive Bayes, KNN ve Rastgele Orman yöntemleri ile kullanmıştır. 3.907 sızdırma yapan ve 4.888 zararsız uygulamadan oluşan bir koleksiyonda yapılan sınıflandırma sonucunda F1-skor metriğine göre %74 başarı ile Rastgele Orman yöntemi en iyi yöntem olarak belirlenmiştir.

Arp ve diğ. [11], Drebin adıyla yayınladıkları çalışmada statik analiz yöntemini kullanarak uygulamalardan API çağrıları ve izinlerini, donanım erişim izinlerini, uygulamayı oluşturan bileşenlerin isimlerini ve Intent kullanım bilgisini elde etmiştir. 5.560 zararlı ve 12.3453 zararsız uygulamadan elde edilen bilgiler ile yaklaşık 545.000 öznitelik içeren bir veri seti oluşturulmuştur. SVM yöntemini kullanılarak sınıflandırma yapıldığında zararlı uygulamalar %94 doğruluk ile tespit edilmiştir.

Demontis ve diğ. [12], zararlı yazılımların sınıflandırılmasında kullanılan makine öğrenmesi yöntemlerinin düşmancıl saldırılara (adversarial attacks) karşı yeteri kadar sağlam olmadığından bahsetmiştir. Yaptıkları çalışmada Arp ve diğ. tarafından yapılan Drebin çalışmasının [11] düşmancıl saldırılara karşı sağlamlığının arttırılması amacıyla SVM yöntemi geliştirilerek Sec-SVM adı verilen daha güvenli bir yöntem önerilmiştir. Zararlı uygulamalarda yapılan bazı değişiklikler sonrasında tespit sistemlerinde kullanılan sınıflandırma algoritmalarının yanıltılabildiği gösterilmiş ve fazla sayıda özelliğin değiştirilmesi sonucunda uygulamanın çalışma bütünlüğünün bozulacağından ve fark edilme olasılığının artacağından bahsedilmiştir. SVM yönteminin zararlı yazılımları tespit etme başarısının %60'a düşmesi için 2 adet özelliğin değiştirilmesi yeterliyken, Sec-SVM yöntemi için 15 adet özelliğin değiştirilmesi gerektiği sonucuna varılmıştır.

Sahs ve Khan [13], statik analiz ile elde ettikleri uygulama izinleri ve Kontrol Akış Çizelgesi bilgisini SVM yöntemi ile kullanarak zararlı uygulamaları tespit etmiştir. 91 zararlı ve 2.081 zararsız uygulama üzerinde yaptıkları sınıflandırma sonucunda zararlı uygulamaların zararsız olarak sınıflandırılma oranı düşük çıkmıştır ancak zararsız uygulamaların zararlı olarak sınıflandırılma oranı yüksek çıkmıştır ve geliştirme yapılmasını gerektirmektedir.

Mariconti ve diğ. [14], MaMaDroid adını verdikleri çalışmada uygulamaların Kontrol Akış Çizelgesi bilgisini ve API çağrılarını Markov Zinciri şeklinde temsil ederek bir model oluşturmuştur. Rastgele Orman ve KNN yöntemleri kullanılarak 35.493 zararlı ve 8.447

zararsız uygulamadan oluşan bir koleksiyonda yapılan sınıflandırma sonucunda F1-skor metriğine göre %98'in üzerinde başarı elde edilmiştir.

Martinelli ve diğ. [15] zararlı Android uygulamalarını tespit etmek için 3.564 zararlı ve 3.536 zararsız uygulamayı dinamik olarak analiz etmiş ve bunları bir Evrişimli Sinir Ağları (Convolutional Neural Network - CNN) modeli kullanarak sınıflandırmıştır. Uygulamalardan elde edilen sistem çağrıları, word2vec [32] kelime temsil yöntemi kullanılarak vektörel olarak ifade edilmiştir. Önerdikleri modelin doğruluğu test setinde %75-80 arasında başarı göstermiştir.

Karbab ve diğ. [16], MalDozer adını verdikleri çalışmada 37.627 zararlı ve 33.066 zararsız uygulamadan oluşan bir koleksiyon oluşturmuştur. API çağrıları word2vec [32] yöntemi kullanılarak vektörel olarak kodlanmış ve Yapay Sinir Ağı kullanılarak sınıflandırılmıştır. Çalışma sonucunda zararlı yazılımlar dengesiz ve dengeli veri setlerinde F1-skor metriğine göre sırasıyla %99,22 ve %99,3 başarı ile tespit edilmiştir. Karbab ve Debbabi [17], MalDy adını verdikleri farklı bir çalışmada dinamik analiz yöntemiyle elde edilen güvenlik raporlarındaki kelimeleri TF-IDF ve Özellik Karması (Feature Hashing) yöntemleri ile vekörel olarak kodlamış ve Topluluk Öğrenmesi yöntemiyle Android ve Windows işletim sistemleri için hazırlanmış zarar uygulamaları tespit ederek aile tipine göre sınıflandırmıştır. Sınıflandırma ve Regresyon Ağacı (Classification And Regression Tree - CART), Aşırı Rastgeleleştirilmiş Ağaçlar (Extremely Randomized Trees - ETrees), KNN, Rastgele Orman, SVM ve Ekstrem Gradyan Arttırma (Extreme Gradient Boosting - XGBoost) yöntemlerinin karşılaştırıldığı çalışmada TF-IDF ve XGBoost yöntemlerinin kullanıldığı model zararlı Android uygulamaları F1-skora göre %100 başarı ile tespit etmiştir.

Yousefi-Azar ve diğ. [18], word2vec [32] kelime temsil yönteminden ilham almış ve tersine mühendislik kullanmadan uygulama paketlerindeki DEX ve XML dosyalarından n-gram özniteliklerini çıkaran Byte2vec modelini ortaya koymuştur. Önerdikleri modelde öznitelikler Ortak Bilgi yöntemi kullanılarak seçilmiş ve XGBoost yöntemi kullanılarak zararlı uygulamalar sınıflandırılmıştır. Drebin [11] veri setinde test edilen modelin F1-skoru %97.43 olarak elde edilmiştir. Yousefi-Azar ve diğ. [19] daha sonra yaptıkları benzer bir çalışmada ise farklı uzunluklardaki n-gram özniteliklerini kullanarak %99,06 F1-skoru elde eden daha başarılı bir model geliştirmiştir.

Narayanan ve diğ. [20], kaynak koddan elde ettikleri Prosedürler Arası Kontrol Akış Grafiklerini vertörel olarak ifade ederek modifiye edilmiş bir skip-gram yöntemi ortaya koymuştur. 10.600 zararlı ve 10.000 zararsız uygulamanın bulunduğu bir veri seti üzerinde SVM kullanarak yaptıkları sınıflandırma %74 doğruluk göstermiştir.

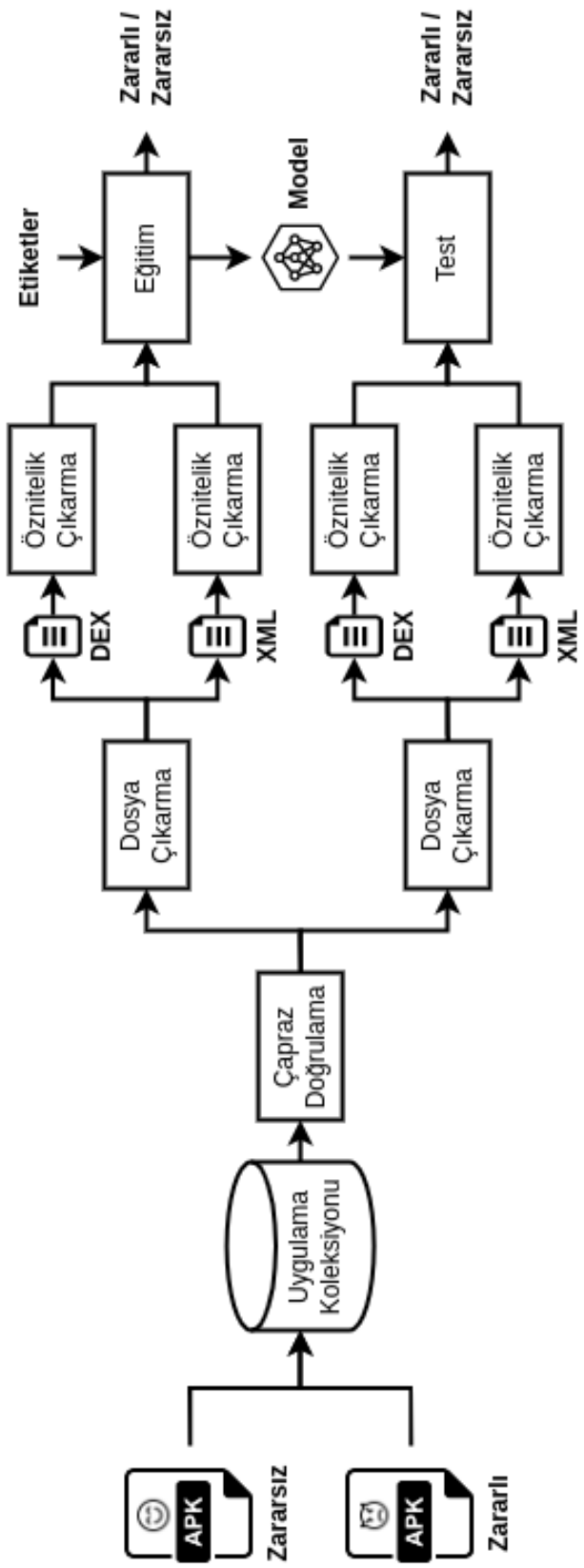
Lei ve diğ. [21], EveDroid adıyla yayınladıkları çalışmada tersine mühendislikle uygulama kodlarından elde ettikleri API çağrılarını doc2vec yöntemini kullanarak vektörel olarak kodlamış ve K-Ortalama yöntemini kullanarak kümelemiştir. API çağrı bilgisine göre vektörel olarak temsil edilen uygulamalar Yapay Sinir Ağları yöntemi kullanılarak sınıflandırılmış ve zararlı olanlar tespit edilmiştir. 28.848 zararlı ve 14.956 zararsız uygulama üzerinde yaptıkları çalışma sonucunda F1-skor metriğine göre %99'a varan başarı elde edilmiştir. Diğer çalışmalardan farklı olarak yapılan kümeleme sayesinde zaman içerisinde değişen API kullanımına karşı daha sağlam bir sistem oluşturulmuştur. Önerdikleri sistemin IoT cihazlarında kullanılabileceğinden bahsedilmiştir ancak sadece telefon uygulamaları üzerinde çalışma yapılmıştır.

3. YÖNTEM

Literatürde statik ve dinamik analiz yöntemleri ile Android işletim sisteminde çalışan uygulamaların güvenliğini inceleyen birçok çalışma bulunmaktadır. Dinamik analiz yöntemi ile uygulamaların çalışması sırasında bilgi toplanması mümkündür ancak karmaşık senaryoların test edilmesi zaman almaktadır. Uygulamaların çalıştırılmasına ihtiyaç duyulmadığı için statik analiz kullanılarak uygulamalardan hızlıca bilgi elde edilebilmektedir. Tersine mühendislik yöntemleri ile uygulamanın kaynak kodlarının çözülmesi statik analizde sıklıkla kullanılan bir yöntemdir. Fakat gelişmiş gizlenme yöntemleri kullanılarak tersine mühendislik yapılması engellenebilmektedir.

Gizlenme yöntemlerinden etkilenmemek için uygulamaların kaynak kodlarını tersine mühendislikle çözme ihtiyacı duymadan ikili dosya formatındaki çalıştırılabilir dosyaları analiz eden alternatif yaklaşımlar bulunmaktadır. Yousefi-Azar ve diğ. [19] tersine mühendislik kullanmadan Android uygulama paketlerindeki DEX ve XML dosyalarından n-gram özniteliklerini çıkararak zararlı uygulamaları tespit etmiştir. Tez kapsamında Yousefi-Azar ve diğ. [19] tarafından önerilen model geliştirilerek farklı bir model oluşturulmuştur. Tez çalışmasında oluşturulan özgün modelde TF-IDF yöntemi kullanılarak Android uygulamalarının kaynak kodları çözülmeden DEX ve XML dosyalarından öznitelik çıkarımı yapılarak zararlı uygulamalar tespit edilmektedir. Tez kapsamında geliştirilen özgün model dergi makalesi olarak yayınlanarak zararlı Android uygulamalarının tespit edilmesi konusunda literatüre katkı yapılmıştır [35].

Tez çalışmasında önerilen sistem zararlı ve zararsız Android uygulamalarından oluşan bir koleksiyondaki uygulama paketlerinden statik analiz uygulanacak dosyaların çıkartılması, elde edilen dosyaların işlenmesiyle uygulamayı temsil eden özniteliklerin çıkartılması, elde edilen öznitelikler arasından etkili olanların seçimi, belirlenen özniteliklere göre uygulamaları sınıflandıran modelin oluşturulması ve modelin çapraz doğrulama yöntemi ile test edilmesinden oluşmaktadır. Sisteme ait akış diyagramı Şekil 3.1'de gösterilmektedir.



Şekil 3.1: Sistem akış diyagramı

Uygulama paketlerinin elde edilme yöntemleri Bölüm 3.1'de açıklanmaktadır. Zararsız uygulamalar kullanılarak zararlı uygulamaların oluşturulması Bölüm 3.2'de açıklanmaktadır. Bölüm 3.3'te uygulamaların etiketlenmesi için kullanılan yöntemden bahsedilmektedir. Uygulamalardan öznitelik çıkarılması ve seçimi sırasıyla Bölüm 3.4 ve 3.5'te açıklanmaktadır. Zararlı uygulamaları tespit etme yönteminden 3.6'da bahsedilmektedir. Bölüm 3.7'de modelin uygulanması kaba-kod üzerinden açıklanmaktadır. Son olarak Bölüm 3.8'de önerilen modelin performansının ölçülmesi için kullanılan metriklerden bahsedilmektedir.

3.1. UYGULAMA PAKETLERİNİN ELDE EDİLMESİ

Android işletim sisteminde çalışan uygulamalar APK uzantılı dosya paketi olarak dağıtılmaktadır. Resmi olarak Google Play uygulama marketinden sunulan uygulamalar alternatif market ve web siteleri üzerinden de paylaşılmaktadır. Google Play uygulama marketi zararlı uygulamaların tespit edilip marketten kaldırılması için çalışmaktadır ancak zararlı uygulamaların genellikle resmi olmayan alternatif yollar üzerinden paylaşılarak yayıldığı gözlenmektedir.

Zararlı uygulamaların tespit edilebilmesi için tez kapsamında geliştirilen modelin hem zararsız hem de zararlı uygulamalardan oluşan karma bir koleksiyon üzerinde eğitilmesi gerekmektedir. Zararlı yazılımların uygulama marketlerinden kaldırılması nedeniyle zararlı uygulamaların bulunması zorlaşmaktadır. Tez çalışması kapsamında çeşitli market ve web siteleri üzerinden sunulan uygulama koleksiyonları incelenerek uygulama paketleri elde edilmiştir. Ayrıca literatürdeki veri setleri incelenerek zararlı ve zararsız uygulama paketlerini paylaşan çalışmalardaki uygulamalar kullanılmıştır.

3.1.1. Akıllı Telefon Uygulamaları

Tez kapsamında farklı tarihlerde toplanmış uygulamaları içeren 3 farklı veri seti kullanılmıştır. Bazı veri setleri uygulamalara ait APK dosyalarının yanı sıra uygulamalardan elde ettikleri özellikleri de paylaşmaktadır. Ancak tez kapsamında yalnızca uygulamalara ait APK dosyaları kullanılmış olup sağlanan özellikler kullanılmamıştır. Veri setlerindeki zararlı (m) ve zararsız (b) uygulamaların sayıları ve Formül 3.1 yardımıyla hesaplanan zararlı uygulama oranları Tablo 3.1'de sunulmuştur.

$$Zararlı\ Uygulama\ Oranı = \frac{m}{m+b} \tag{3.1}$$

Tablo 3.1: Veri setlerindeki zararlı ve zararsız uygulama sayıları

Veri seti	Zararlı Uygulama	Zararsız Uygulama	Zararlı Uygulama Oranı
Drebin&Androzoo	5.551	5.561	%50
CICAndMal2017	425	1.697	%20
CICMalDroid2020	12.685	4.029	%76

Tablo 3.2: Veri setlerindeki zararlı ve zararsız uygulamalara ait dosya boyutu istatistikleri

Veri seti	Dosya tipi	Zararlı uygulama boyutu			Zararsız uygulama boyutu		
		En küçük	En büyük	Ortalama	En küçük	En büyük	Ortalama
Drebin&Androzoo	APK	5,3 kB	23,6 MB	1,3 MB	1,4 MB	13,5 MB	5,4 MB
	DEX	2,3 kB	5,7 MB	0,3 MB	3,8 kB	9,6 MB	3,4 MB
	XML	0,5 kB	27,4 kB	3,1 kB	0,3 kB	0,3 MB	5,3 kB
CICAndMal2017	APK	18,4 kB	62,7 MB	4,1 MB	12,7 kB	52,4 MB	13,1 MB
	DEX	1,2 kB	9 MB	1,6 MB	1,8 kB	36,3 MB	5,1 MB
	XML	0,7 kB	0,2 MB	10,3 kB	0,4 kB	0,2 MB	10,6 kB
CICMalDroid2020	APK	0,3 kB	0,1 GB	2 MB	44,4 kB	48,7 MB	14 MB
	DEX	0,9 kB	21,5 MB	0,6 MB	6,8 kB	36,3 MB	5,7 MB
	XML	0,3 kB	0,3 MB	5,8 kB	0,7 kB	1,9 MB	12,6 kB

Zararlı ve zararsız uygulamaların dosya boyutları her veri seti için analiz edilerek minimum, maksimum ve ortalama dosya boyutları gibi istatistiki bilgiler Tablo 3.2'de sunulmuştur. Uygulamalar APK dosya boyutu açısından karşılaştırıldığında zararsız uygulamaların ortalama boyutunun zararlı uygulamalardan 2-6 kat daha büyük olduğu görülmüştür. DEX dosyaları incelendiğinde ise zararsız uygulamaların ortalama boyutunun zararlı uygulamalardan 2-10 kat daha büyük olduğu görülmektedir. XML dosyalarının ortalama boyutu incelendiğinde ise zararsız uygulamalardaki dosyaların zararlı uygulamalardaki dosyaların en fazla iki katı kadar olduğu görülmüştür.

3.1.1.1. Drebin&Androzoo Veri Seti

Tez kapsamında zararlı ve zararsız uygulamalardan oluşan bir koleksiyon elde etmek amacıyla öncelikle zararlı Android uygulamaların tespit edilmesi konusunda oldukça popüler bir veri seti olan Drebin [11] veri setinde yer alan zararlı uygulamalar kullanılmıştır. Drebin [11] veri setinde Ağustos 2010 ile Ekim 2012 tarihleri arasında toplanmış 179 farklı aileden 5.560 zararlı uygulama bulunmaktadır. Bu veri setindeki uygulamalara https://www.sec.tu-bs.de/~danarp/drebin/ adresinden erişim sağlanabilmektedir. Veri setindeki uygulamalar incelendiğinde hem DEX hem de XML dosyalarının elde edilebildiği 5.551 uygulama olduğu

görülmüştür. Veri setinde sağlanan APK dosyalarının sadece zararlı uygulamalara ait olması nedeniyle zararsız uygulamaların toplanmasına ihtiyaç duyulmuştur.

Zararsız uygulamalar için Androzoo [36] uygulama koleksiyonundan yararlanılmıştır. Bu koleksiyonda sunulan uygulama paketleri ile birlikte VirusTotal (https://www.virustotal.com) sitesindeki antivirüs tarayıcılarından kaçı tarafından zararlı olarak görüldüğü bilgisi sağlanmaktadır. Tez kapsamında oluşturulan koleksiyonda hiçbir tarayıcı tarafından zararlı olarak algılanmamış uygulamalar zararsız olarak kabul edilmiştir. Koleksiyonun oluşturulması sırasında Androzoo [36] uygulama koleksiyonunda 18 milyondan fazla uygulama bulunmaktaydı. Bu uygulamalara https://androzoo.uni.lu/ adresinden erişim sağlanabilmektedir. Yousefi-Azar ve diğ. [19] tarafından oluşturulan koleksiyona benzer olması açısından tez kapsamında oluşturulan koleksiyonda 2006 ile 2016 yılları arasında Google Play uygulama marketinde yayınlanan 1,4 MB ile 13,5 MB aralığında bulunan zararsız uygulama paketleri seçilmiştir. Zararsız uygulamalar, Kushniarou tarafından geliştirilen az [37] isimli yazılım kullanılarak elde edilmiştir. Zararsız uygulamalar incelendiğinde hem DEX hem de XML dosyalarının elde edilebildiği 5.561 uygulamanın olduğu görülmüştür.

3.1.1.2. CICAndMal2017 Veri Seti

Lashkari ve diğ. [38], 2011 ile 2017 yılları arasında çeşitli kaynaklardan topladıkları 4354 zararlı uygulamayı ve 2015 ile 2017 yılları arasında Google Play uygulama marketinde yayınlanan 6.500 zararsız uygulamayı bir araya getirerek bir koleksiyon oluşturmuştur. APK dosyalarındaki hatalar ve etiketleme sırasında karşılaştıkları sorunlar nedeniyle fiziksel olarak akıllı telefonlara sadece 429 zararlı ve 5.065 zararsız uygulamayı yüklemeyi başarabilmişlerdir. 42 farklı aileden 426 zararlı ve 1.700 zararsız uygulamanın APK dosyaları https://www.unb.ca/cic/datasets/andmal2017.html adresinden herkesin erişimine açık olarak paylaşılmıştır. Elde ettiğimiz uygulamalar için dosya çıkarma işlemleri yapıldıktan sonra hem DEX hem de XML dosyalarını içeren kullanılabilir 425 zararlı ve 1.697 zararsız uygulamanın olduğu görülmüştür.

3.1.1.3. CICMalDroid2020 Veri Seti

Mahdavifar ve diğ. [39], Aralık 2017 ile Aralık 2018 arasında çeşitli kaynaklardan reklam, sahte bankacılık, sms dolandırıcılığı, riskli olabilecek ve zararsız olmak üzere 5 kategoride 17.341 Android uygulamasını toplamış ve uygulama paketlerini herkese açık olarak https://www.unb.ca/cic/datasets/maldroid-2020.html adresinden paylaşmıştır. Tez kapsamında zararlı uygulamaların tespit edilmesi amaçlandığı için bu veri setindeki zararsız olmayan 4

kategorideki uygulamalar zararlı olarak kabul edilmiştir. Hem DEX hem de XML dosyalarını içeren uygulamalar ele alındığında 12.685 zararlı ve 4.029 zararsız uygulamanın kullanıma uygun olduğu görülmüştür.

Mahdavifar ve diğ. [39], elde ettikleri uygulama paketlerini paylaşmanın yanı sıra uygulamalardan statik ve dinamik analiz yöntemleri ile çıkardıkları bilgileri de veri setinin içerisinde paylaşmıştır. Statik analiz yöntemi ile elde ettikleri bilgiler arasında uygulamaların kaynak kodunun ifşasını önlemek amacıyla gizlenme tekniğinin uygulanıp uygulanmadığı bilgisi yer almaktadır. Kaynak kod üzerinde uygulanan gizlenme tekniği özellikle zararlı yazılımların kötü amaçlı faaliyetlerini gizlemek ve tespit edilmemek başvurdukları bir yöntemdir. Tez kapsamında oluşturulan modelde sadece veri setlerinde sunulan uygulama paketleri kullanılmaktadır. Ancak bu veri setinde bulanan uygulamalar arasında kaynak kodlarını gizleyen uygulamaların var olduğu bilmek tez kapsamında oluşturulan modelin gizlenme tekniklerine karşı dayanıklılığını test etmek için faydalı olmuştur.

3.1.2. Akıllı TV Uygulamaları

Tez kapsamında akıllı TV uygulama paketlerinin elde edilmesi için ApkMirror websitesi ve Androzoo uygulama koleksiyonu kullanılmıştır.

3.1.2.1. APKMirror Web Sitesi

ApkMirror (https://www.apkmirror.com/) web sitesinde sunulan Android uygulamaları arasında TV uygulamalarının olduğu saptanmıştır. Yapılan incelemelerde bu siteden elde edilen uygulamalar ile root yetkisine sahip bir cihazdaki Google Play uygulama marketi üzerinden indirilen uygulamaların aynı olduğu doğrulanmıştır. ApkMirror sitesinde yapılan aramalarda TV'de çalışacak uygulamaların filtrelenmesinin mümkün olduğu görülmüştür. Bu sitede arama yapıp TV uygulamalarının isimlerini ve bağlantılarını liste olarak saklayan bir program (crawler) yazılmıştır. Elde edilen bu liste kullanılarak yaklaşık 600 adet Android TV uygulama paketi toplanmıştır. İndirilen paketler arasında aynı uygulamaya ait farklı versiyonlardaki uygulama paketlerinin ve APK formatında olmayan paketlerin elemesi yapılarak 372 TV uygulamasından oluşan bir koleksiyon oluşturulmuştur. Koleksiyondaki uygulamalar tersine mühendislik yapılarak incelendiğinde 370 uygulamanın tez kapsamında geliştirilen model tarafından kullanıma uygun olduğu saptanmıştır.

Elde edilen uygulama paketleri VirusTotal sitesinde sunulan yaklaşık 60 adet anti-virüs tarayıcısı yardımıyla analiz edilmiş analiz raporları incelendiğinde 8 uygulama paketinin en

fazla 1 anti-virüs tarayıcısı tarafından zararlı olarak sınıflandırıldığı, diğer anti-virüs tarayıcıları tarafından ise zararsız olarak sınıflandırıldığı görülmüştür. Diğer uygulama paketleri ise tüm anti-virüs tarayıcıları tarafından zararsız olarak sınıflandırılmıştır. Tez kapsamında en az 3 anti-virüs tarayıcısı tarafından zararlı olarak sınıflandırılan paketler zararlı olarak etiketlenmiştir. Buna göre ApkMirror web sitesinden elde edilen koleksiyondaki uygulamaların tamamı zararsız olarak değerlendirilmiştir.

3.1.2.2. Androzoo TV Uygulama Koleksiyonu

Liu ve diğ. [22] tarafından yapılan çalışmada Google Play uygulama marketinde sunulan 3.163 Android TV uygulaması incelenerek VirusTotal sitesindeki tarama sonuçları, talep ettikleri izinler, güvenlik açıkları ve gizlilik sızıntıları analiz edilmiştir ve uygulama paketlerinin sha256 algoritması ile hesaplanmış hash değerleri liste olarak https://github.com/DannyGooo/Android-TV-apps-found-in-Google-Play-Store adresinden herkese açık olarak paylaşılmıştır. Tez çalışması kapsamında gelişilen modelin farklı bir veri setinde test edilebilmesi için paylaşılan bu listedeki uygulamalar Androzoo [36] uygulama koleksiyonu üzerinden taranarak indirilmiştir ancak listelenen 3.163 Android TV uygulamasından sadece 1.107 uygulama paketine erişim sağlanmıştır.

Elde edilen uygulama paketleri VirusTotal sitesinde sunulan yaklaşık 60 adet anti-virüs tarayıcısı yardımıyla analiz edilmiş analiz raporları incelendiğinde 1.070 uygulama paketinin tüm anti-virüs tarayıcıları tarafından zararsız olarak sınıflandırıldığı görülmüştür. Koleksiyondaki 31 uygulama paketi 1 anti-virüs tarayıcısı tarafında, 3 uygulama paketi ise 2 anti-virüs tarayıcısı tarafından zararlı olarak sınıflandırılmıştır. Geriye kalan 3 uygulama paketi ise daha fazla anti-virüs tarayıcısı tarafından zararlı olarak tespit edilmiştir. Tez kapsamında en az 3 anti-virüs tarayıcısı tarafından zararlı olarak sınıflandırılan paketler zararlı olarak değerlendirilmiş ve bu koleksiyondaki uygulamalardan 1.104'ü zararsız ve 3'ü zararlı olarak etiketlenmiştir.

3.2. ZARARLI UYGULAMALARIN OLUŞTURULMASI

Zararlı Android TV uygulamalarının sayısının az olması veri setinde dengesizliğe yol açmaktadır ve tez kapsamında geliştirilen modelin öğrenmesini kısıtlamaktadır. Modelin eğitilmesi ve test edilmesi sırasında kullanılmak üzere tez kapsamında zararlı uygulamaların oluşturulmasına karar verilmiştir. Zararlı uygulamaların oluşturulmasında zararlı yazılım geliştiricilerin sıklıkla kullandığı tekrar paketleme (repackaging) yöntemi uygulanmıştır. Bu

yöntemde zararsız uygulama paketleri açıldıktan sonra içerisine zararlı kod parçaları (payload) eklenip uygulamalar tekrar paketlenmektedir. Uygulama paketi üzerinde çok az değişiklik yapılarak aynı isimde tekrar paketleme işlemi yapılması sonucunda gerçek uygulamaya çok benzer ancak zararlı olan bir uygulama elde edilebilmektedir. Farkında olmayan birçok kullanıcı gerçek uygulama yerine tekrar paketlenmiş zararlı uygulamayı yükleyerek saldırganların hedefi olmaktadır.

Zararlı uygulamaların oluşturulması sürecinde birçok işletim sistemi ve platform için güvenlik açıkları sağlayan açık kaynaklı bir penetrasyon testi ve geliştirme aracı olan Metasploit Framework [40] kullanılmıştır. Metasploit Framework [40] içerisinde sunulan Meterpreter zararlı kod parçası hedef cihazın hafızasında (RAM) çalışan bir komut satırı programıdır. Meterpreter zararlı kod parçası kullanılarak hedef cihazda istenilen komutların çalıştırılması sağlanabilmektedir. Android işletim sistemi için reverse_http, reverse_https ve reverse_tcp olmak üzere 3 farklı tipte zararlı kod parçası üretilebilmektedir. Liu ve diğ. [22] tarafından paylaşılan Android TV uygulama listesinde yer alan zararsız uygulamaların bazıları olduğu gibi bırakılırken diğerlerine tez kapsamında tekrar paketleme yöntemi kullanılarak zararlı kod parçaları eklenmiştir. Bu sayede 3 farklı tipte elde edilen zararlı Android TV uygulamalarının sayısı Tablo 3.3'te gösterilmektedir. Androzoo TV uygulama koleksiyonunda 567 zararsız ve 471 zararlı uygulama bulunmaktadır ve zararlı uygulama oranı %45 olarak hesaplanmıştır. Zararlı Android TV uygulamalarının 154'ü reverse_http, 154'ü reverse_https ve 163'ü reverse_tcp zararlı kod parçalarını içermektedir.

Tablo 3.3: Androzoo TV koleksiyonundaki uygulamaların türüne göre dağılımı

Uygulama Türü	Zararlı Kod Tipi	Uygulama Sayısı
Zararsız	-	567
Zararlı	reverse_http	154
	reverse_https	154
	reverse_tcp	163

Metasploit Framework [40] ile üretilen zararlı kod parçalarındaki sınıf ve fonksiyon isimlerinde rastgele değerler kullanılmaktadır. Ayrıca tekrar paketleme işlemi sırasında farklı geliştirici imzalarının kullanılması sayesinde zararsız bir uygulamanın zararlı hale getirilmesinde birbirinden farklı uygulama paketleri elde edilebilmektedir. Tez kapsamında zararsız bir uygulamadan sadece 1 adet zararlı uygulama paketi oluşturulmuştur. Zararlı uygulama paketlerinin oluşturulmasında farklı zararsız uygulama paketleri kullanılması ve her bir zararlı

kod parçasının birbirinden farklı üretilmesi ile elde edilen zararlı uygulama paketlerinin birbirine benzeme olasılığı düşüktür.

3.3. UYGULAMALARIN ETİKETLENMESİ

Akıllı telefon uygulamalarını içeren Drebin&Androzoo, CICAndMal2017 ve CICMalDroid2020 veri setlerindeki uygulama paketlerinin zararlı ya da zararsız olduğu paylaşıldığı için etiketleri bilinmektedir. Ancak akıllı TV uygulamaları için toplanan uygulama paketlerinin zararlı olup olmadığı bilinmemektedir. Bu bilginin elde edilip uygulamaların etiketlenmesi için VirusTotal sitesinden yararlanılmıştır. Bu sitede yaklaşık 60 adet anti-virüs tarayıcısı Android uygulama paketlerinin analiz edilmesi için hizmet vermektedir. VirusTotal sitesinde sunulan anti-virüs tarayıcıları Tablo 3.4'te listelenmiştir.

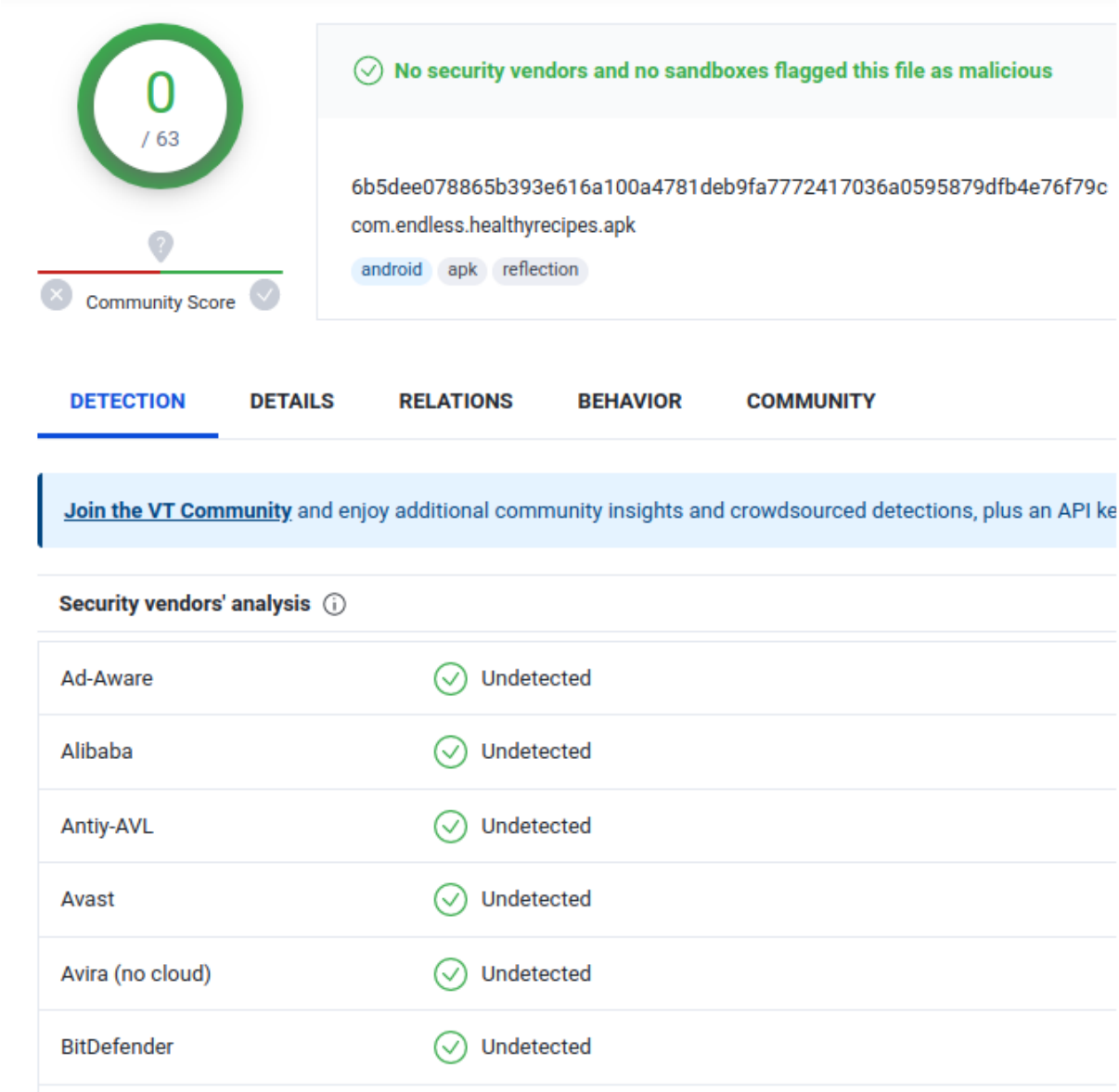
Tablo 3.4: VirusTotal sitesinde sunulan anti-virüs tarayıcıları

Ad-Aware	Emsisoft	NANO-Antivirus
AhnLab-V3	ESET-NOD32	Panda
Alibaba	F-Secure	Rising
ALYac	FireEye	Sangfor
Antiy-AVL	Fortinet	Sophos
Arcabit	GData	SUPERAntiSpyware
Avast	Gridinsoft	Symantec
Avast-Mobile	Ikarus	SymantecMobileInsight
Avira	Jiangmin	TACHYON
Baidu	K7AntiVirus	Tencent
BitDefender	K7GW	TrendMicro
BitDefenderFalx	Kaspersky	TrendMicro-HouseCall
BitDefenderTheta	Kingsoft	Trustlook
Bkav	Lionic	VBA32
CAT-QuickHeal	Malwarebytes	VIPRE
ClamAV	MAX	VirIT
CMC	MaxSecure	ViRobot
Comodo	McAfee	Yandex
Cynet	McAfee-GW-Edition	Zillya
Cyren	Microsoft	ZoneAlarm
DrWeb	MicroWorld-eScan	Zoner

VirusTotal sitesi web arayüzünün yanı sıra geliştiriciler için uygulama programı arayüzü (API) üzerinden hizmet vermektedir. Uygulama paketlerine ait analiz raporlarını elde etmek ve paketleri etiketlemek için VirusTotal API kullanılarak bir program (script) hazırlanmıştır. Tez

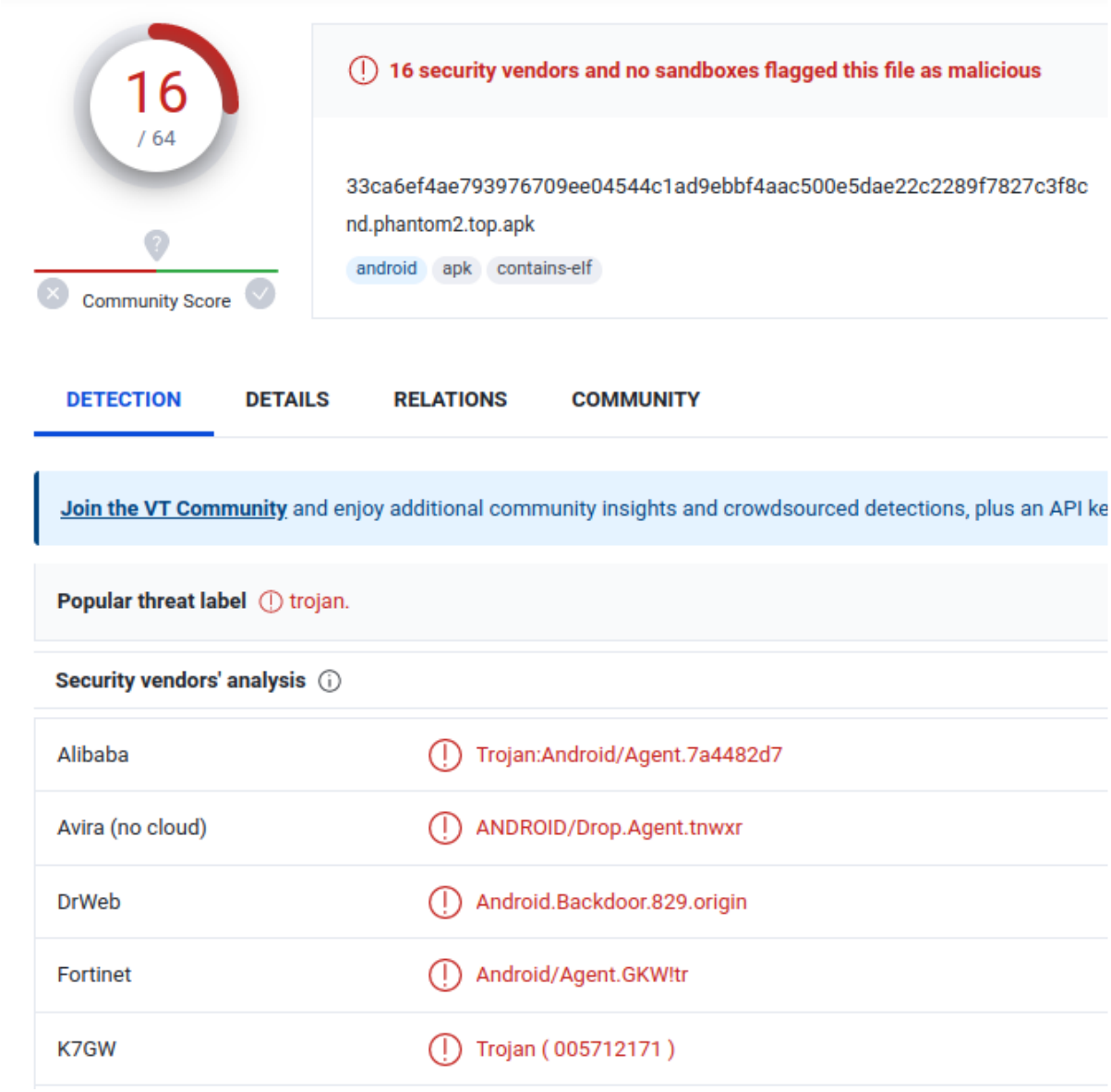
kapsamında geliştirilen program ile uygulama paketlerinin sha256 hash değerleri hesaplanarak VirusTotal analiz raporları elde edilmektedir. Program uygulama paketine ait analiz raporlarından uygulamayı zararlı olarak değerlendiren anti-virüs tarayıcılarının sayısını hesaplayarak belirlenen eşik değerinin üzerinde bulunması durumunda uygulamayı zararlı olarak etiketlemektedir.

Şekil 3.2'de zararsız bir uygulama paketine için VirusTotal sitesinden elde edilen analiz sonucu gösterilmektedir. Zararsız uygulama için elde edilen analiz raporuna göre uygulama paketi hiçbir anti-virüs tarayıcısı tarafından zararlı olarak sınıflandırılmamıştır.



Şekil 3.2: Zararsız bir uygulama paketi için VirusTotal sitesiden elde edilen tarama sonucu örneği

Şekil 3.3'te zararlı bir uygulama paketine için VirusTotal sitesinden elde edilen analiz sonucu gösterilmektedir. Elde edilen sonuçlara göre zararlı uygulama paketi 16 anti-virüs tarayıcısı tarafından zararlı olarak sınıflandırılmıştır. Liu ve diğ. [22] tarafından yapılan çalışmada VirusTotal analiz raporlarına göre en az 1 anti-virüs tarayıcısı tarafından zararlı olarak sınıflandırılan uygulamalar zararlı olarak değerlendirilmiştir. Ancak sadece 1 anti-virüs tarayıcısının kararına göre uygulamaları etiketlemek hatalı olabileceği için tez kapsamında en az 3 anti-virüs tarayıcısı tarafından zararlı olarak sınıflandırılan paketler zararlı olarak etiketlenmiştir.



Şekil 3.3: Zararlı bir uygulama paketi için VirusTotal sitesiden elde edilen tarama sonucu örneği

3.4. UYGULAMALARDAN ÖZNİTELİK ÇIKARILMASI

Android uygulama paketi içerisinde sıkıştırılmış olarak bulunan AndroidManifest.xml ve classes.dex dosyaları uygulamaya ait önemli barındırmaktadır. AndroidManifest dosyası uygulamanın komponentleri, ihtiyaç duyduğu izinler ve kullandığı özellikler gibi önemli bilgileri XML formatında saklamaktadır. Classes dosyası ise uygulama kodlarının derlenmesi ile elde edilmiş Dalvik Executable (DEX) formatındaki dosyadır. Uygulama kodlarının tersine mühendislikle elde edilememesi amacıyla bazı uygulamalar uygulama paketinin derlenmesi sırasında çeşitli gizlenme teknikleri kullanmaktadır. Gizlenme tekniklerinden etkilenmemek ve daha kısa sürede sınıflandırma yapmak amacıyla tez çalışmasında uygulama kodlarının elde edilmesine ihtiyaç duymayan bir model geliştirilmiştir ve derlenmiş kaynak kodları barındıran DEX dosyası üzerinde tersine mühendislik uygulanmadan doğrudan analiz yapılmıştır. Uygulama hakkında bilgiler içeren AndroidManifest dosyasının ise ikili dosya formatından XML formatına deşifre edilmesi sağlanmıştır. DEX ve XML dosyalarının uygulama paketi içerisinden çıkartılmasında Apktool [41] yazılımı kullanılmıştır.

Uygulamalardan elde edilen dosyaların modele girdi olarak verilebilmesi için sayısal değerlere dönüştürülmesi gerekmektedir. Her bir dosya öznitelik olarak adlandırılan bir sayısal değerlere dönüştürülmekte ve dosyalardan elde edilen öznitelikler birleştirilerek uygulamayı temsil etmektedir. Tez çalışmasında uygulanan öznitelik çıkarma işlemi uygulama paketlerinden elde edilen DEX ve XML dosyalarının bayt düzeyinde okunup işlenmesini kapsamaktadır. Örnek olarak bir DEX dosyasının ilk 8 baytı Tablo 3.5'te ikili, onaltılık ve karakter gösterim biçimlerinde ifade edilmiştir. İlk 8 bayt "d", "e", "x", "new line", "0", "3", "5" ve "null" karakterlerinden oluşmaktadır. Diğer dosya formatlarına benzer olarak DEX dosya formatı belirli bir özgün bayt dizisiyle başlamaktadır. DEX dosya formatındaki dosyaları diğer dosya formatlarından ayrıştırmak için kullanılan bu diziye Dex File Magic adı verilmektedir [42]. Dosya bozulmasını da kontrol etmek için kullanılan bu yönteme göre bir DEX dosyasındaki ilk 8 bayt "dex" karakterleri, "new line" karakteri, 3 basamaklı format sürüm numarası ve boş karakter olmalıdır. Tablo 3.5'te gösterilen baytlara göre örnek olarak alınan dosya 035 format sürümüne sahip geçerli bir DEX dosyasıdır.

Tez kapsamında bayt düzeyinde okunan dosyalardaki her bir bayt n-gram dizisindeki bir öge olarak ele alınmıştır. (N) bayt uzunluğundaki bir pencerenin dosyanın başından sonuna kadar 1 bayt kaydırılarak okunması sonucu n-gram terimleri elde edilmiştir. Elde edilen n-gram değerleri onaltılık gösterime göre "0x" ön eki ve baytlar arasında boşluk olmadan saklanmıştır.

Tablo 3.5'te gösterilen örnek DEX dosyasından 2 bayt uzunluğundaki (2-gram) terimler hesaplanmak istendiğinde "6465", "6578", "780a", "0a30", "3033", "3335" ve "3500" değerleri elde edilmektedir.

Tablo 3.5: Örnek bir DEX dosyasının ilk 8 baytı

İkili gösterim	Onaltılık gösterim	Karakter gösterimi
0b01100100	0x64	d
0b01100101	0x65	e
0b01111000	0x78	x
0b00001010	0x0a	New Line
0b00110000	0x30	0
0b00110011	0x33	3
0b00110101	0x35	5
0b00000000	0x00	Null

Uygulama paketleri içerisinde yer alan DEX ve XML dosyalarının bayt düzeyinde okunmasıyla n-gram terimleri elde edilmiş ve bu terimlerin kullanım oranları uygulamayı oluşturan öznitelikler olarak ele alınmıştır. Bir uygulamadan elde edebilecek benzersiz terimlerin sayısı, terimi oluşturan bayt sayısıyla (n) ilişkilidir ve $[0, 2^{8*n}]$ aralığındadır. Elde edilen tüm terimlerin bellekte saklanması terim uzunluğu arttıkça daha fazla alan gerektireceği için önemli olanların belirlenip saklanması daha uygundur. Tez çalışması kapsamında önemli öznitelikleri belirlemek amacıyla özellikle Doğal Dil İşleme alanında metin belgelerinden öznitelik çıkarmak için yaygın olarak benimsenen Terim Frekansı - Ters Belge Frekansı (TF-IDF) kelime temsil yöntemini kullanılmıştır. TF-IDF yöntemi hesaplama karmaşıklığı açısından ele alındığında terim sayısı (s) ve ortalama terim uzunluğu (L) ile ilişkili olarak O(sL log sL) karmaşıklığına sahip olduğu görülmektedir.

Tez kapsamında analiz edilen dosyalardan TF-IDF yöntemi kullanılarak öznitelik çıkarımı yapılmasında Scikit-learn kütüphanesindeki [43] TfidfVectorizer dönüştürücüsü kullanılmıştır. Bu dönüştürücüye verilen dosyadan belirlenen kriterlere uygun olarak terimler (n-gram) ve kullanım oranları matris formatında çıkartılmaktadır. Elde edilen kullanım oranları TF-IDF yöntemi kullanılarak ilgili dosyayı temsil edecek vektörel değere dönüştürülmekte ve normalize edilmektedir. DEX ve XML dosyalarından her biri TF-IDF yöntemine göre bir belge olarak ele alınarak dosyanın içindeki her bayt TF-IDF yönteminde bir kelimeyi oluşturan harfi temsil edecek şekilde işlenmektedir. DEX ve XML dosyalarından elde edilen tüm terimler bellekte saklanmaktadır ve terim uzunluğu arttıkça daha fazla alana ihtiyaç duyulmaktadır. Scikit-learn

kütüphanesinin [43] sağladığı önemli bir avantaj seyrek matris kullanımını mümkün kılmasıdır. Seyrek matrisler sıfır olmayan elemanları saklayan, kalan elemanları ise sıfır kabul eden veri yapılardır. Hesaplamalarda seyrek matris kullanılması sonucunda daha az bellek ile daha hızlı işlem yapılabilmesi sağlanmıştır.

3.5. ÖZNİTELİK SEÇİMİ

Uygulamayı temsil eden öznitelikler DEX ve XML dosyalarından TF-IDF yöntemi ile bayt düzeyinde elde edilen terimlerdir. Daha fazla sayıda bayt içeren uzun terimlerin kullanılması elde edilecek öznitelik sayısını arttırmaktadır ancak çok fazla öznitelik olması modelin başarısını ve hesaplama süresini doğrudan etkilemektedir. Öznitelik olarak 2 bayt uzunluğundaki terimler kullanıldığında 2^{16} (65.536) farklı terim elde edilebilmektedir. Bayt uzunluğu 3 olduğunda ise eşsiz terim sayısı 2^{24} (16.777.216) olmaktadır. Terim uzunluğunun performans üzerindeki etkisini incelemek amacıyla farklı uzunluklarda terimler için modeller oluşturulmuştur. Terim (n-gram) uzunluğunun performansa etkisi ile ilgili bulgular Bölüm 4.1.2'de sunulmuştur.

Tez kapsamında geliştirilen modelde XML dosyalarından 3 bayt uzunluğundaki terimler (3-gram) kullanılmasına karar verilmiştir. Performans açısından XML dosyalarından en fazla 100 bin terim çıkarımı yapılacak şekilde limit koyulmuştur. DEX dosyalarının boyutu büyük olduğu için sadece 2 birim uzunluğundaki terimler kullanılarak analiz yapılmış ve en fazla 5 bin terim çıkarımı olacak şekilde limit koyulmuştur.

Modelin eğitilmesi sırasında eğitim kümesinde yer alan uygulamalara ait dosyalardan TF-IDF yöntemi ile tüm terimler elde edilmiştir. Elde edilen terimler kullanım oranlarına göre sıralanmış ve en yüksek değere sahip terimler alınarak öznitelik kümesi belirlenmiştir. Test aşamasında ise uygulamalara ait dosyalardan tüm terimlerin elde edilmesi yerine sadece öznitelik kümesinden yer alan terimlerin kullanım oranları hesaplanmıştır. Bu sayede modelin zararlı uygulamaları daha hızlı tespit edilmesi sağlanmıştır.

3.6. ZARARLI UYGULAMALARIN TESPİT EDİLMESİ

Uygulamalardan elde edilen öznitelikler tez kapsamında geliştirilen modelin eğitim aşamasında ve test aşamasında uygulamaların zararlı ya da zararsız şeklinde sınıflandırılması için kullanılmaktadır. Geliştirilen modelde Chen ve Guestrin [34] tarafından önerilen ve gradyan

artırıcı karar ağacı olan XGBoost sınıflandırıcısı kullanılmıştır. Gradyan artırma, daha iyi sonuçlar elde etmek için birden fazla sınıflandırıcıyı birleştiren bir topluluk öğrenme algoritmasıdır. Gradyan artırıcı karar ağaçlarında toplu olarak güçlü bir model elde etmek için zayıf ağaçlar diğer zayıf ağaçlarla birleştirilerek iteratif olarak iyileşme sağlanmaktadır.

XGBoost [34] paralel işleme, ağaç budama, eksik değerleri işleme ve yanlılığı önlemek için düzenlileştirme gibi avantajları nedeniyle popüler bir algoritmadır. Ayrıca sıfır değerine sahip olmayan ögelerin depolandığı seyrek veriler üzerinde kullanım için optimize edilmiştir. Seyrek veriye duyarlı algoritmalar, daha az bellek ve disk alanı kullanarak daha kısa bir zamanda hesaplama yapılmasına olanak sağlamaktadır. Tez çalışmasında kullanılan girdiler bayt düzeyinde çıkarılan n-gram değerlerinin normalizasyonu sonucu elde edilmiş TF-IDF ağırlıklarıdır. Bir uygulamayı temsil eden eşsiz özniteliklerin sayısının olası tüm bayt dizilerinin sayısından çok daha küçük olması nedeniyle birçok özniteliğin değeri sıfır olarak hesaplanmaktadır. Tez kapsamında seyrek veriye duyarlı bir algoritma tercih edilerek hesaplama sırasında büyük boyutlarda bellek kullanımına olan ihtiyaç ortadan kaldırılmıştır. XGBoost yöntemi hesaplama karmaşıklığı açısından ağaç sayısı (K), ağaç derinliği (d), sıfır değere sahip olmayan öznitelik sayısı (x) ve örnek sayısı (q) ile ilişkili olarak O(Kdx log q) karmaşıklığına sahiptir [34].

3.7. MODELİN UYGULANMASI

Tez çalışmasında geliştirilen model Python dilinde Scikit-learn [43] ve XGBoost [44] yazılım kütüphaneleri kullanılarak uygulanmıştır. Modelin kaba-kodu Şekil 3.4'te gösterilmektedir. Satır 1 ile 10 aralığında TF-IDF yöntemi kullanılarak XML ve DEX özelliklerini çıkaran ve XGBoost yöntemi kullanılarak uygulamayı sınıflandıran modeli oluşturan fonksiyon yer almaktadır. Geliştirilen model, Scikit-learn [43] yazılım kütüphanesinde yer alan “make_pipeline”, “TransformerMixin”, “TfidfVectorizer” ve “FeatureUnion” yöntemlerinden oluşan bir veri hattı kullanmaktadır. Satır 2 ve 4'te modele girdi olarak verilen XML ve DEX dosya konumları “TransformerMixin” yöntemi kullanılarak ayrıştırılmaktadır. Satır 3 ve 5'te “TfidfVectorizer” yöntemi kullanılarak XML ve DEX dosyalarından TF-IDF değerleri elde edilerek uygulamaya ait öznitelikler seyrek matris formatında saklanmaktadır. Sıfır değerli özniteliklerin hafızada tutulmaması, modelin daha az hafıza kullanarak daha kısa sürede hesaplama yapmasını sağlamaktadır. “make_pipeline” yöntemi kullanılarak DEX ve XML dosyaları için oluşturulan veri hatlarından gelen öznitelik verileri Satır 6'da “FeatureUnion”

yöntemi kullanılarak birleştirilmektedir. Satır 7'de XGBoost [44] yazılım kütüphanesindeki "XGBClassifier" yöntemi kullanılarak bir XGBoost sınıflandırıcısı oluşturulmaktadır. Satır 8'de öznitelik verilerini sınıflandırıcıya girdi olarak veren bir veri hattı "make_pipeline" yöntemi kullanılarak oluşturulmaktadır. Oluşturulan model Satır 9'da fonksiyonun geri dönüş değeri olarak döndürülmektedir. Modelin oluşturulmasında kullanılan hiperparametreler Tablo 3.6'da sunulmuştur.

```
1  function create_model
2      get xml file paths from the input
3      extract tfidf features as feature_xml
4      get dex file paths from the input
5      extract tfidf features as feature_dex
6      combine feature_xml and feature_dex as list_feature
7      create an xgboost classifier as classifier
8      feed list_feature as input into classifier
9      return classifier
10 end function
11 initialize empty lists list_app, list_label
12 for each apk file in collection do
13     extract XML and DEX binary files from apk
14     initialize empty list list_path
15     append path of the XML file into list_path
16     append path of the DEX file into list_path
17     append list_path into list_app
18     if application is malicious then
19         append 1 into list_label
20     else
21         append 0 into list_label
22     end if
23 end for
24 for each fold in kfold do
25     split list_app into list_app_train and list_app_test
26     split list_label into list_label_train and list_label_test
27     get model pipeline as pipe using create_model function
28     fit pipe using list_app_train and list_label_train
29     get predictions as pred using list_app_test
30     compare pred with list_label_test
31 end for
```

Şekil 3.4: Zararlı uygulamaları tespit eden modelin kaba-kodu

Uygulama koleksiyonundaki uygulama paketlerinden dosya çıkarma işlemleri Şekil 3.4'te Satır 12 ile 23 aralığında gösterilmektedir. Satır 13'te Apktool [41] yazılımı ile uygulama paketlerine tersine mühendislik yapılarak DEX ve XML dosyaları elde edilmektedir. Tersine mühendislik işlemleri sırasında DEX dosyalarından kaynak kodların elde edilmesi yerine ikili dosya

formatında saklanması amacıyla yazılıma ait parametreler "kaynak kodları deşifre etme" şeklinde ayarlanmıştır. Tersine mühendislik işlemleri sadece ikili dosya formatından insan tarafından okunabilir dosyalar elde etmek için XML dosyaları üzerinde uygulanmıştır. Satır 14 ile 16 aralığında uygulama paketlerinden elde edilen “AndroidManifest.xml” ve “classes.dex” dosyalarının konumları birinci öge XML ve ikinci öğe DEX dosyasına ait olacak şekilde bir listeye eklenmektedir. Satır 17 ile 22 aralığında eğitim veri setinde yer alan uygulamaların etiket bilgileri zararlıysa 1 ve zararsızsa 0 etiket değeriyle temsil edilecek şekilde bir listeye eklenmektedir.

Tablo 3.6: Modelde kullanılan hiperparametreler

Yöntem	**Hiperparametre**	**Açıklama**	**Değer**
TfidfVectorizer	input	Girdi tipi	filename
TfidfVectorizer	analyzer	Özellik çıkarma düzeyi	char
TfidfVectorizer	lowercase	Tüm karakterleri küçük harfe çevir	false
TfidfVectorizer	decode_error	Bilinmeyen bayt dizisi için uygulanacak yöntem	replace
TfidfVectorizer	ngram_range	n-gram uzunluğunun alt ve üst sınırı	XML: (3,3), DEX: (2,2)
TfidfVectorizer	max_features	Maksimum öznitelik sayısı	XML: 100000, DEX: 5000
XGBClassifier	n_estimators	Gradyan artırıcı karar ağacı sayısı	500
XGBClassifier	max_depth	Temel öğreniciler için maksimum ağaç derinliği	8
XGBClassifier	eta	Öğrenme hızı	0,1
XGBClassifier	colsample_bytree	Her ağacı oluştururken örneklerin alt örnek oranı	0,8
XGBClassifier	objective	Öğrenme amaç fonksiyonu	binary:logistic
XGBClassifier	eval_metric	Değerlendirme metriği	logloss
XGBClassifier	n_jobs	Paralel iş parçacığı sayısı	1
XGBClassifier	base_score	Tüm örneklerin ilk tahmin puanı	0,5
XGBClassifier	booster	Yükseltici	gbtree
XGBClassifier	colsample_bylevel	Her düzey için sütunların alt örnek oranı	1
XGBClassifier	colsample_bynode	Her bölme için sütunların alt örnek oranı	1
XGBClassifier	gamma	Daha fazla bölüm yapmak için gereken minimum kayıp azaltma	0
XGBClassifier	importance_type	Öznitelik önem türü	gain
XGBClassifier	max_delta_step	Her ağacın ağırlık tahmini için maksimum delta adımı	0
XGBClassifier	min_child_weight	Bir çocuk düğümde gereken minimum örnek ağırlığı toplamı	1
XGBClassifier	reg_alpha	Ağırlıklarda L1 düzenleme terimi	0
XGBClassifier	reg_lambda	Ağırlıklarda L2 düzenleme terimi	1
XGBClassifier	scale_pos_weight	Pozitif ve negatif ağırlıkların dengelenmesi	1
XGBClassifier	subsample	Eğitim örneğinin alt örneklem oranı	1
XGBClassifier	tree_method	Ağaç yöntemi	exact

Uygulama paketlerinden elde edilen öznitelik ve etiket verileri modelin eğitilmesinde ve test edilmesinde kullanılmaktadır. Şekil 3.4'te Satır 24 ile 31 aralığında verilerin eğitim ve test

kümelerine ayrılması ve modelin çapraz doğrulama yöntemi kullanılarak performansının ölçülmesi gösterilmektedir. Uygulama koleksiyonundaki uygulamalar 5 katlamalı çapraz doğrulama yöntemi ile bölünerek 5 adet deney gerçekleştirilmiştir. Uygulama koleksiyonu 5 kümeye bölündükten sonra 1 küme test için ayrılmış ve kalan 4 küme ile model eğitilmiştir. Aynı işlem sırayla diğer parçalara uygulanarak 5 deney tamamlanmış ve performansları ölçülmüştür. Modelin performansı 5 deneyin ortalaması alınarak hesaplanmıştır. Uygulamaların test ve eğitim kümelerine ayrıştırılması sırasında veri setinin geneline ait zararlı uygulama oranının korunmasına dikkat edilmiştir. Bu sayede dengesiz bir küme oluşumu engellenerek her bir kümenin genel veri setini daha iyi temsil etmesi sağlanmıştır. Bir çapraz doğrulma iterasyondaki test ve eğitim kümelerinde yer alacak uygulamalara ait öznitelik ve etiket verileri sırasıyla Satır 25 ve 26'da ayrıştırılmıştır. Satır 27'de modeli oluşturan fonksiyon çağrılarak model objesi elde edilmiştir. Satır 28'de modelin eğitim kümesindeki öznitelik ve etiket verileri ile eğitilmesi gerçekleştirilmiştir. Satır 29'da modelin test kümesindeki öznitelik verilerini kullanarak uygulamaları sınıflandırması sağlanmıştır. Modelin eğitilmesi ve test edilmesi sırasında Scikit-learn [43] yazılım kütüphanesindeki “fit” ve “predict” yöntemleri kullanılmıştır. Satır 30'da modelin gerçekleştirdiği sınıflandırma sonuçları Bölüm 3.8'de açıklanan performans ölçümü metriklerine göre gerçek etiket değerleri ile karşılaştırılmak üzere saklanmıştır.

Tez kapsamında yapılan tüm hesaplamalar Intel Core i5-8365U CPU @ 1.60 GHz (4 çekirdekten 1'i kullanılarak), 16 GB RAM ve 64-bit Linux Mint 20.1 Cinnamon işletim sistemine sahip bir dizüstü bilgisayarda gerçekleştirilmiştir. Performans kriteri olarak hesaplama süreleri ölçüldüğü için hesaplama sırasında çok çekirdekli ölçeklendirmenin etkisinden etkilenmemek adına işlemci kullanımı tek bir çekirdekle sınırlandırılmıştır. Hesaplama için işlemcinin birden fazla çekirdeğinin kullanılmaması için cpulimit [45] yazılımı ile sınırlandırma yapılmıştır.

3.8. PERFORMANS ÖLÇÜMÜ

Tez çalışması kapsamında önerilen modelin performansının değerlendirilmesinde Yanlış Negatif Oranı (FNR), Yanlış Pozitif Oranı (FPR), F1-skor ve Matthews Korelasyon Katsayısı (MCC) metrikleri kullanılmıştır. Bu metrikler Tablo 3.7’de verilen karışıklık matrisine göre hesaplanmıştır. Zararlı uygulamalar pozitif sınıf, zararsız uygulamalar ise negatif sınıf ile temsil edilerek hesaplamalar yapılmıştır.

Tablo 3.7: Karışıklık matrisi

	Tahmini Negatif (PN)	**Tahmini Pozitif (PP)**
Gerçek Negatif (N)	Doğru Negatif (TN)	Yanlış Pozitif (FP)
Gerçek Pozitif (P)	Yanlış Negatif (FN)	Doğru Pozitif (TP)

Karışıklık matrisinde verilen Pozitif skorlar (P) pozitif sınıfa, Negatif skorlar (N) ise negatif sınıfa ait gerçek örnekleri gösterirken, model tarafından tahmin edilen Tahmini Negatif skorlar (PN) ve Tahmini Pozitif skorlar (PN) ile gösterilmiştir. Matriste yer alan TN skoru gerçekte negatif olan ve negatif olarak tahmin edilen değerleri, FP skoru gerçekte negatif olan ancak pozitif olarak tahmin edilen değerleri, FN skoru gerçekte pozitif olan ancak negatif olarak tahmin edilen değerleri, TP skoru gerçekte pozitif olan ve pozitif olarak tahmin edilen değerleri temsil etmektedir. Pozitif, Negatif, Tahmini Negatif ve Tahmini Pozitif değerlerinin hesaplanması sırasıyla Formül 3.2, 3.3, 3.4 ve 3.5'te gösterilmiştir.

$$Pozitif\ (P) = FN + TP \tag{3.2}$$

$$Negatif\ (N) = TN + FP \tag{3.3}$$

$$Tahmini\ Negatif\ (PN) = TN + FN \tag{3.4}$$

$$Tahmini\ Pozitif\ (PP) = FP + TP \tag{3.5}$$

Karışıklık matrisindeki ölçüm değerleri oran olarak ifade edilmektedir. Bu oranlar Doğru Pozitif Oranı (TPR), Yanlış Pozitif Oranı (FPR), Yanlış Negatif Oranı (FNR) ve Doğru Negatif Oranı (TNR)'dır. TPR, doğru sınıflandırılmış pozitiflerin veri setindeki toplam pozitif örnek sayısına oranını ifade eder. Ayrıca bu oran Duyarlılık (Recall) olarak da bilinmektedir. TNR, doğru sınıflandırılmış negatiflerin veri setindeki toplam negatif örnek sayısına oranını ifade eder. FPR, yanlış sınıflandırılmış negatiflerin veri setindeki toplam negatif örnek sayısına oranını ifade eder. FNR, yanlış sınıflandırılmış pozitiflerin veri setindeki toplam pozitif örnek sayısına oranını ifade eder. Değerlendirilecek modelin iyi bir performans gösterebilmesi için TPR, TNR skorlarının yüksek ve FNR, FPR skorlarının düşük olması beklenmektedir. TPR, TNR, FPR ve FNR değerleri Formül 3.6, 3.7, 3.8 ve 3.9'da verilen denklemle elde edilmektedir.

$$TPR = \frac{TP}{TP + FN} \tag{3.6}$$

$$TNR = \frac{TN}{TN + FP} \tag{3.7}$$

$$FPR = \frac{FP}{TN + FP} \tag{3.8}$$

$$FNR = \frac{FN}{FN + TP} \tag{3.9}$$

Tez kapsamında modelin performansının değerlendirilmesinde FNR ve FPR metriklerinin dışında F1-skor ve Matthews Korelasyon Katsayısı (MCC) metrikleri de kullanılmıştır. Bu metriklerden F1-skor değeri Kesinlik (Precision) ve Duyarlılık (Recall) değerlerinin harmonik ortalamasını göstermektedir. Kesinlik her zaman pozitif tahminlere odaklanır ve bu oran pozitif tahmin değeri olarak da adlandırılır. MCC metriğinde çıktı -1 ile +1 arasında değişmektedir. 0 değeri rastgele sınıflandırma durumunu gösterirken -1 değeri ise var olan gerçek değerler ile sınıflandırıcının verdiği tahminlerin tamamen zıt olduğunu göstermektedir. Çıktı değeri +1 değerine ne kadar yakınsa sınıflandırma modelinin o kadar iyi olduğunu göstermektedir. İkili sınıflandırma problemlerinde MCC metriği Kesinlik, Duyarlılık ve F1-skor metriklerine daha güvenilir sonucu verebilmektedir. Kesinlik, Duyarlılık, F1-skor ve MCC metriklerini hesaplanması sırasıyla Formül 3.10, 3.11, 3.12 ve 3.13'te gösterilmiştir.

$$Kesinlik\ (Precision) = \frac{TP}{TP + FP} \tag{3.10}$$

$$Duyarlılık\ (Recall) = \frac{TP}{TP + FN} \tag{3.11}$$

$$F1 - skor = 2 \times \frac{Precision \times Recall}{Precision + Recall} \tag{3.12}$$

$$MCC = \frac{TP \times TN - FP \times FN}{\sqrt{N \times P \times PN \times PP}} \tag{3.13}$$

4. BULGULAR

Tez çalışması kapsamında geliştirilen modelin belirlenmesi ve performans iyileştirmelerinin yapılması için farklı hiperparametre değerlerine sahip modeller oluşturulmuştur. Bölüm 4.1'de modelin performansını etkilen faktörler ile ilgili bulgular paylaşılmıştır. Mobil uygulamalar üzerinde başarılı performans gösteren modelin performansı AndroidTV uygulamaları üzerinde test edilmiştir. AndroidTV uygulamalarının güvenlik analizleri ile ilgili bulgular Bölüm 4.2'de sunulmuştur.

4.1. PERFORMANSI ETKİLEYEN FAKTÖRLER

Tersine mühendislikte kullanılan kod çözme seviyesi ve öznitelik çıkarımında kullanılan n-gram uzunluğu gibi hiperparametreler için çeşitli değerler kullanılarak modelin performansı üzerinde etkisi incelenmiştir. Farklı dosyalardan elde edilen özniteliklerin etkileşimi gözlemlenerek en iyi performansı gösteren model belirlenmiştir. Ayrıca veri setindeki uygulama sayısı ve zararlı uygulama oranının modelin performansı üzerindeki etkisi incelenmiştir.

4.1.1. Kod Çözme Seviyesi

Tersine mühendislik yöntemi ile uygulamalardan bilgi edinme sürecinde uygulama paketi içerisinde ikili dosya formatında bulunan kaynak dosyaları ile birlikte kaynak kodun deşifre edilmesi sıklıkla kullanılmaktadır. Ancak kaynak kodun deşifre edilmesi derleme esnasında gelişmiş gizlenme yöntemleri kullanılan uygulamalarda her zaman başarılı olmamaktadır. Kaynak kodun deşifre edilebildiği durumlarda ise deşifre işlemi belli bir zaman aldığı için zararlı olan uygulamaların tespit edilme süresinin uzamasına neden olmaktadır. Tersine mühendislik için kullanılan Apktool [41] yazılımı kod çözme seviyesinin ayarlanabildiği parametrelere sahiptir. Bu parametreler kullanılarak kaynak dosyalarının ve kaynak kodun deşifre edilmesi ya da deşifre edilmeden ikili dosya formatında bırakılması sağlanabilmektedir.

Kod çözme seviyesinin performans üzerindeki etkisini ölçmek amacıyla tez kapsamında 4 farklı senaryo hazırlanmıştır. Kaynak dosyalarının ve kaynak kodun deşifre edilme durumlarına göre belirlenen senaryolar Tablo 4.1’de gösterilmektedir. Senaryo-1'de herhangi bir kısıtlama olmadan hem kaynak dosyaları hem de kaynak kod üzerinde deşifre işlemleri gerçekleştirilmektedir. Senaryo-2'de “AndroidManifest.xml” dosyası gibi kaynak dosyaları ikili dosya formatından insan tarafından okunabilir XML dosyalarına deşifre edilirken kaynak

kodun yer aldığı “classes.dex” dosyaları deşifre edilmeden ikili dosya formatında saklanır. Senaryo-3'te kaynak dosyaları deşifre edilmezken kaynak kodun bulunduğu dosyalar ikili dosya formatı olan DEX dosya formatından Assembly komutları içeren Smali dosya formatına deşifre edilmektedir. Senaryo-4'te ise kaynak kod ve kaynak dosyalarına hiçbir deşifre işlemi yapılmadan ikili dosya formatındaki dosyalar sıkıştırılmış uygulama paketinden çıkarılmaktadır.

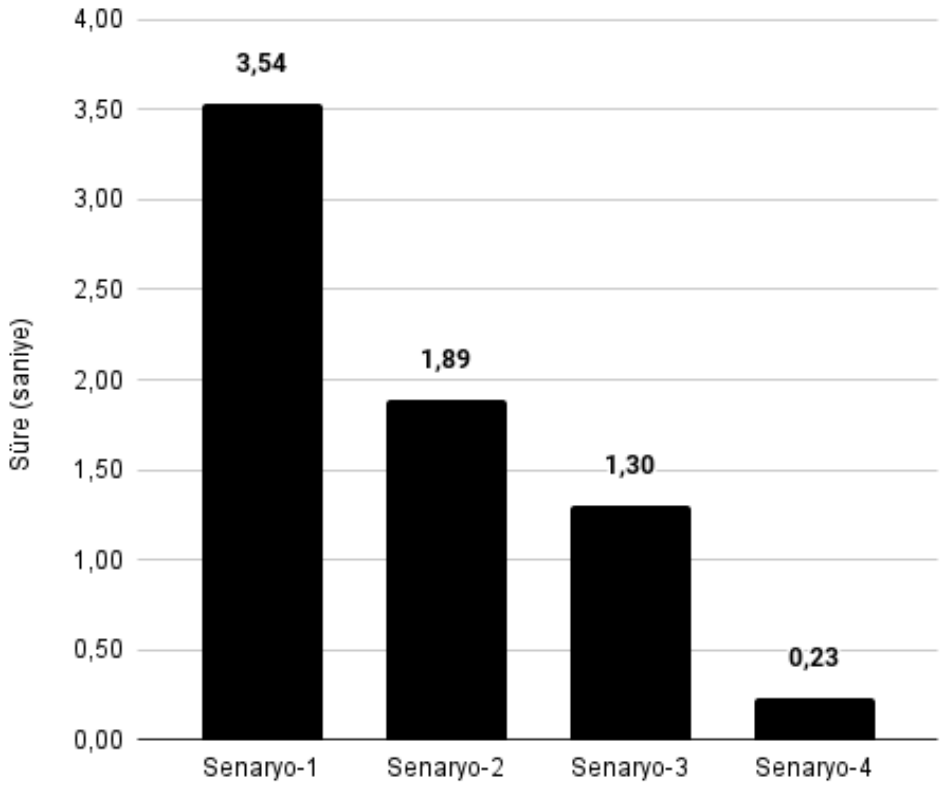
Tablo 4.1: Kod çözme seviyelerine göre senaryolar

Senaryo	**Kaynak Dosyaları**	**Kaynak Kod**
Senaryo-1	Deşifre edilir	Deşifre edilir
Senaryo-2	Deşifre edilir	Deşifre edilmez
Senaryo-3	Deşifre edilmez	Deşifre edilir
Senaryo-4	Deşifre edilmez	Deşifre edilmez

Farklı kod çözme seviyelerinin uygulandığı senaryolar Drebin&Androzoo veri setindeki uygulamalar üzerinde test edilerek kod çözme seviyesinin hesaplama süresi üzerindeki etkisi incelenmiştir. Belirlenen senaryolarda bir uygulamadan dosya çıkarımı için harcanan ortalama süre Şekil 4.1'de gösterilmektedir. Kaynak dosyalarının ve kaynak kodun deşifre edildiği Senaryo-1 için ortalama işlem süresi 3,54 saniye olarak ölçülmüştür. Kaynak dosyalarının deşifre edildiği ancak kaynak kodun deşifre edilmediği Senaryo-2'de ortalama işlem süresi 1,89 saniye olmuştur. Kaynak dosyalarının deşifre edilmediği ancak kaynak kodun deşifre edildiği Senaryo-3 için ortalama işlem süresi 1,30 saniye olarak hesaplanmıştır. Kaynak kod ve kaynak dosyalarına hiçbir deşifre işlemi yapılmadığı Senaryo-4'te işlem süresi 0,23 saniyeye düşmüştür.

Tersine mühendislik uygulanmadan dosya çıkarımı yapılan durumda dosya çıkarımı ortalama 0,23 saniye gibi kısa bir sürede tamamlanmaktadır ancak bu durumda uygulama hakkında fazla bir bilgi edinmek mümkün olmamaktadır. Hem kaynak dosyalarına hem de kaynak koduna tersine mühendislik uygulanan durumda uygulama hakkında daha fazla bilgi edinmek mümkündür. Fakat deşifre işlemleri için uygulama başına ortalama 3,31 saniye fazladan zaman harcanmaktadır. Sadece kaynak kodun deşifre edilip kaynak dosyalarının deşifre edilmediği durumda her ikisinin deşifre edilmesine göre uygulama başına ortalama 2,24 saniye zaman avantajı bulunmaktadır. Ancak kaynak kod üzerinde gizlenme yöntemlerinin uygulandığı durumlarda deşifre işlemleri yetersiz kaldığından uygulama kodu hakkında yeterli bilgi elde edilememektedir. Sadece kaynak dosyalarının deşifre edilip kaynak kodun deşifre edilmediği

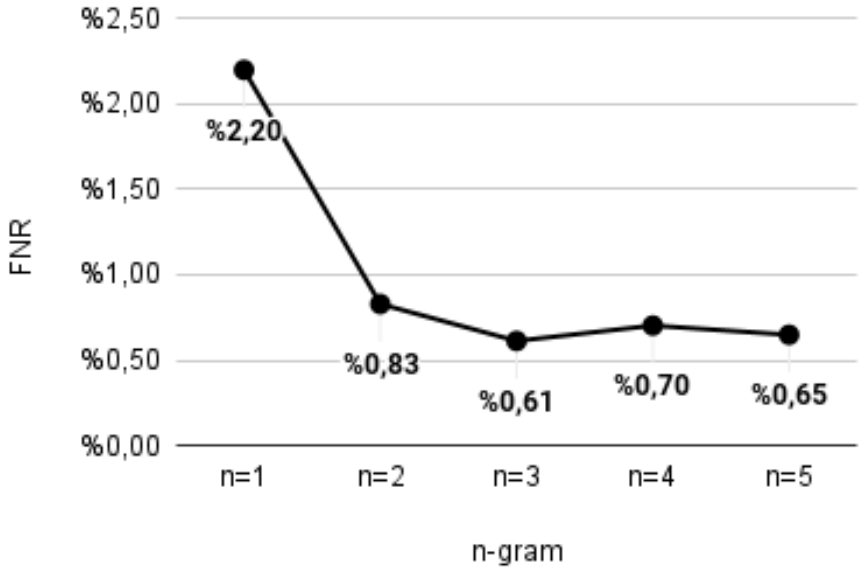
durumda dosya çıkarımı için işlem süresi her ikisinin deşifre edilmesine göre uygulama başına ortalama 1,65 saniye daha kısa sürmektedir. Tez çalışması kapsamında kaynak kodun deşifre edilmesine ihtiyaç duymadan ikili dosya formatındaki DEX dosyalarından öznitelik çıkaran bir model geliştirilmiştir. Bu sayede hem dosya çıkarımının daha kısa sürede tamamlanması sağlanmış hem de kaynak kod üzerinde uygulanması muhtemel gizlenme yöntemlerinden etkilenmeden öznitelik çıkarımı yapılması hedeflenmiştir. Kaynak dosyaları içerisinde "AndroidManifest.xml" gibi uygulama hakkında önemli bilgiler barındıran dosyaların yer alması ve bu dosyalarda gizlenme uygulanmıyor olması nedeniyle kaynak dosyalarının deşifre edilmesi öznitelik çıkarımı için avantaj sağlamaktadır. Tüm bulgular göz önüne alındığında sağladığı avantajlar nedeniyle tez çalışmasındaki uygulama paketlerinden dosya çıkarım işlemlerinde uygulama kaynak dosyalarının deşifre edilip kaynak kodun deşifre edilmediği Senaryo-2 tercih edilmiştir.



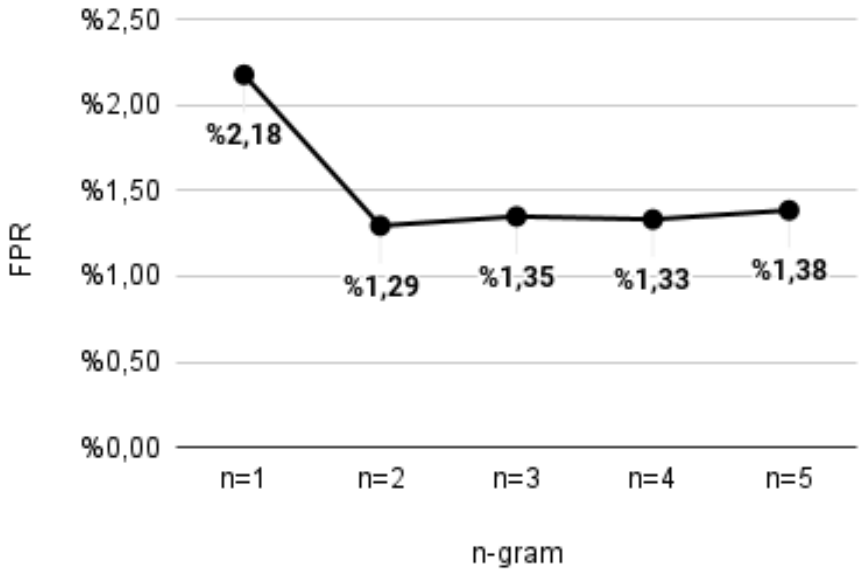
Şekil 4.1: Kod çözme düzeyine göre bir uygulamadan dosya çıkarımı için harcanan ortalama süre

4.1.2. N-gram Uzunluğu

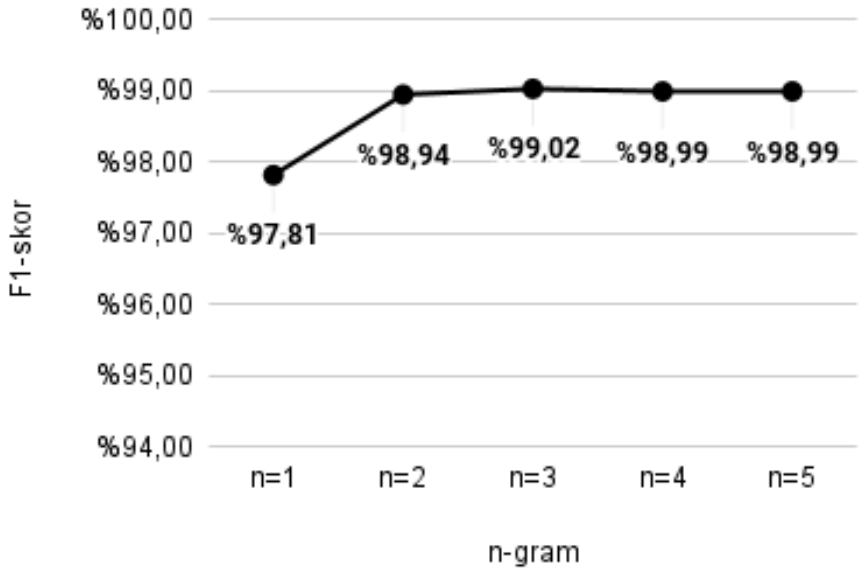
Tez çalışmasında uygulama paketlerinden elde edilen DEX ve XML dosyalarının bayt düzeyinde okunmasıyla n-gram terimleri çıkarılmakta ve bu terimlerin kullanım oranları hesaplanarak öznitelikler elde edilmektedir. Bir uygulamadan elde edebilecek benzersiz n-gram terimlerinin sayısı, terimi oluşturan bayt sayısıyla (n) ilişkilidir $[0, 2^{8*n}]$ aralığındadır. Terim uzunluğu arttıkça oluşacak terimlerin sayısının artması nedeniyle terim uzunluğu modelin başarısını doğrudan etkileyen bir parametredir. Terim uzunluğunun performans üzerindeki etkisini incelemek amacıyla XML dosyalarından farklı uzunluklarda çıkarılmış n-gram terimlerini girdi olarak alan 5 farklı model oluşturulmuştur. Drebin&Androzoo veri setindeki uygulama paketlerine ait XML dosyaları sırasıyla 1 - 5 bayt uzunluğunda okunarak 1-gram, 2-gram, 3-gram, 4-gram ve 5-gram terimlerin olduğu 5 farklı modelin eğitim ve test işlemleri gerçekleştirilmiştir. 5 katlamalı çapraz doğrulama yöntemi ile yapılan hesaplama işlemleri sonucunda test veri kümesindeki uygulamaların performans sonuçlarının ortalamaları alınmıştır. Farklı uzunluklardaki terimleri kullanan modellerin FNR, FPR, F1-skor ve MCC metriklerine göre test kümesindeki performans sonuçları sırasıyla Şekil 4.2, 4.3, 4.4 ve 4.5'te sunulmuştur.



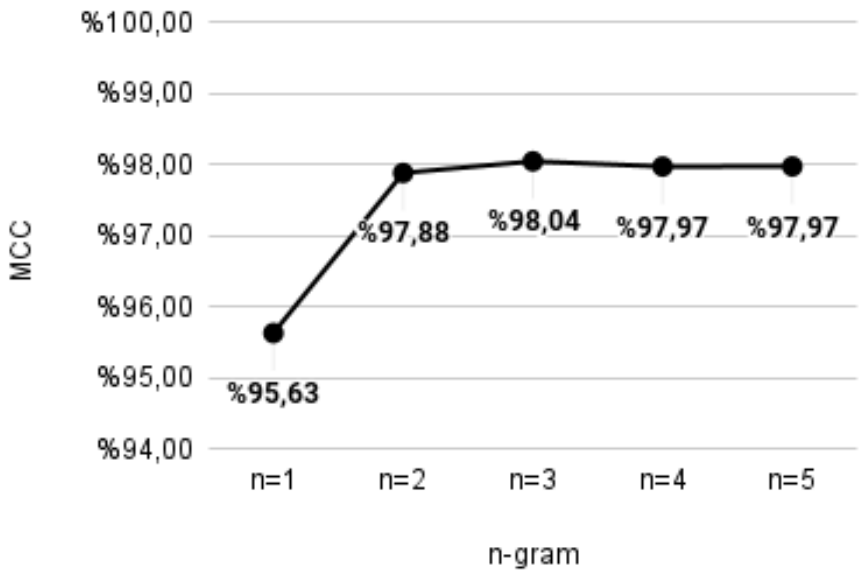
Şekil 4.2: Farklı n-gram uzunluğundaki terimleri kullanan modellerin FNR metriğine göre performans grafiği



Şekil 4.3: Farklı n-gram uzunluğundaki terimleri kullanan modellerin FPR metriğine göre performans grafiği



Şekil 4.4: Farklı n-gram uzunluğundaki terimleri kullanan modellerin F1-skor metriğine göre performans grafiği



Şekil 4.5: Farklı n-gram uzunluğundaki terimleri kullanan modellerin MCC metriğine göre performans grafiği

Öznitelik olarak XML dosyalarından elde edilmiş 1-gram terimler yerine 2-gram terimlerin kullanıldığı model tüm metrikler açısından daha başarılı olmuştur. Zararlı uygulamaların yanlış sınıflandırılma oranını gösteren FNR metriğine göre hata oranı %2,20'den %0,83'e düşmüştür. Zararsız uygulamaların yanlış sınıflandırılma oranını gösteren FPR metriğine göre hata oranı %2,18'den %1,29'a düşmüştür. Modelin F1-skoru %97,81'den %98,94'e yükselirken MCC metriğine göre başarısı %95,63'ten %97,88'e çıkmıştır. 1-gram terimleri kullanan modelde elde edilebilecek maksimum öznitelik sayısı 256 ile sınırlıyken, 2-gram terimleri kullanan model için maksimum öznitelik sayısı 65.536'dır. Öznitelik sayısının yetersiz kalmasının modelin performansını doğrudan etkilediği görülmektedir.

N-gram terim uzunluğu olarak 2 birim yerine 3 birim kullanılan modelin performansı FNR, F1-skor ve MCC metriklerine göre daha iyi çıkmıştır ancak performanstaki artış miktarı 1-gram ve 2-gram terimlerini kullanan modellerin arasındaki farktan daha az olmuştur. Terim uzunluğunun 3 birimden fazla olduğu modellerin performansı ise tüm metriklerine göre daha kötü ya da yakın çıkmıştır. FNR, F1-skor ve MCC metriklerine göre en iyi performans 3-gram terimlerin kullanıldığı model ile elde edilmiştir. Elde edilen sonuçlara göre 16,7 milyondan fazla özniteliğin olduğu durumda arama uzayının boyutunun çok geniş olması nedeniyle sınıflandırıcının performansının kötüleştiği çıkarımı yapılmıştır.

N-gram uzunluğuna göre en iyi performans 3-gram terimlerin kullanıldığı model ile elde edildiği için tez kapsamında XML dosyalarından öznitelik çıkarımı için 3-gram terimlerin kullanılmasına karar verilmiştir. DEX dosyalarının boyutu XML dosyalarına göre çok daha büyük olması nedeniyle 2 birimden daha uzun terimlerin elde edilmesi için çok fazla hafıza ve işlem gücüne ihtiyaç vardır. 1-gram terimlerin kullanıldığı durumda ise yeterli bilgi elde edilememektedir. Bu nedenle tez kapsamında DEX dosyalarından öznitelik çıkarımı için 2-gram terimler kullanılmıştır.

4.1.3. Farklı Özniteliklerin Birlikte Kullanımı

Farklı dosyalardan elde edilen özniteliklerin performans üzerindeki etkisini incelemek amacıyla XML ve DEX dosyalarından elde edilen öznitelikleri kullanan modele ek olarak sadece XML ve sadece DEX dosyalarından elde edilen öznitelikleri kullanan 2 farklı model oluşturulmuştur. "2-gram DEX", "3-gram-XML" ve "2-gram DEX + 3-gram XML" özniteliklerini kullanan 3 farklı modelin performansları karşılaştırılarak farklı özniteliklerin etkisi gözlemlenmiştir. Drebin&Androzoo veri setinde yer alan uygulama paketleri kullanılarak 5 katlamalı çapraz doğrulama gerçekleştirilmiştir. Modellerin FNR, FPR, F1-skor ve MCC metriklerine göre başarıları ve 1 uygulama için eğitim ve test aşamalarında harcanan ortalama süre Tablo 4.2'de gösterilmektedir. Eğitim süresi, eğitim kümesindeki uygulamalar için dosya çıkarma, öznitelik çıkarma ve model eğitimi aşamalarında harcanan toplam sürenin eğitim setindeki uygulama sayısına bölünmesiyle hesaplanmıştır. Benzer şekilde test setindeki uygulamalar için dosya çıkarma, öznitelik çıkarma ve sınıflandırma aşamalarında harcanan toplam süre test setindeki uygulama sayısına bölünerek test süresi hesaplanmıştır.

Tablo 4.2: Farklı öznitelikler kullanan modellerin FNR, FPR, F1-skor ve MCC metriklerine göre performans sonuçları

Model	FNR	FPR	F1-skor	MCC	Eğitim Süresi	Test Süresi
2-gram DEX	%0,79	%2,28	%98,47	%96,94	0,84 saniye	0,85 saniye
3-gram XML	%0,61	%1,35	%99,02	%98,04	1,91 saniye	1,89 saniye
2-gram DEX + 3-gram XML	%0,61	%1,29	%99,05	%98,09	2,76 saniye	2,75 saniye

FNR metriği açısından sadece DEX dosyalarını kullanan modelin hata oranı %0,79 olarak elde edilirken sadece XML dosyalarını kullanan modelin hata oranı %0,61 olarak elde edilmiştir. DEX ve XML dosyalarından elde edilen öznitelikler karşılaştırıldığında "AndroidManifest.xml" dosyasından elde edilen bilgilerin zararlı uygulamaların tespit edilmesinde daha etkili olduğu görülmektedir. DEX ve XML dosyalarının birlikte kullanıldığı

modelin FNR performansı ise sadece XML dosyalarının kullanıldığı model ile aynı sonuçlanmıştır.

FPR metriği dikkate alındığında, yalnızca DEX dosyalarının kullanıldığı modelde %2,28 hata oranı elde edilmiştir. Sadece XML dosyalarının kullanıldığı modelde ise %1.35 hata oranı ile daha iyi bir sonuç alınmıştır. DEX ve XML dosyalarının birlikte kullanıldığı modelde ise %1.29 hata oranı elde etmiş ve sadece tek bir dosya tipini kullanan modellere göre daha iyi sonuçlar alınmıştır. Bu durumdan DEX ve XML dosyalarından elde edilen özniteliklerin birlikte kullanılmasının daha iyi bir etkileşim oluşturduğu ve modelin düşük bir hata oranı ile performans gösterdiği sonucuna varılmıştır.

F1-skor metriğine göre en iyi performans %99,05 başarı ile DEX ve XML dosyalarının birlikte kullanıldığı modelden elde edilmiştir. Yalnızca XML dosyalarının kullanıldığı model %99,02 başarı gösterirken yalnızca DEX dosyalarının kullanıldığı model %98,47 başarı göstermiştir. Modellerin MCC metriğine göre performans sonuçları F1-skor metriğine ile elde edilen sonuçlara benzerdir. En iyi MCC performansı %98.09 ile DEX ve XML dosyalarının birlikte kullanıldığı modelden elde edilirken, bunu sırasıyla %98.04 ve %96.94 değerleri ile sadece XML ve sadece DEX dosyalarının kullanıldığı modeller takip etmektedir.

Sadece DEX dosyalarını kullanan modelin eğitim ve test hatlarında harcadığı zaman 1 uygulama için ortalama olarak sırasıyla 0,84 saniye ve 0,85 saniye olarak ölçülmüştür. DEX dosyalarının çıkarımında tersine mühendislik yöntemleri uygulanmadığı için dosya çıkarma aşaması kısa sürmektedir ancak tüm süreçte geçirilen zamanın çoğu DEX dosyalarından özniteliklerin çıkarılması için harcanmaktadır. XML dosyalarından özniteliklerin çıkarılması çok kısa bir sürede gerçekleşmesine rağmen dosya çıkarma aşamasında tersine mühendislik yöntemlerinin uygulanması nedeniyle sadece XML dosyalarını kullanan modelin eğitim ve test hatlarında 1 uygulama için harcadığı zaman ortalama olarak sırasıyla 1,91 saniye ve 1,89 saniye olarak ölçülmüştür. DEX ve XML dosyalarını birlikte kullanan modelin eğitim ve test hatlarında harcadığı zaman 1 uygulama için ortalama olarak sırasıyla 2,76 saniye ve 2,75 saniye olarak ölçülmüştür.

4.1.4. Veri Setindeki Uygulama Sayısı ve Zararlı Yazılımların Oranı

Tez kapsamında geliştirilen modelin performansı farklı uygulama sayıları ve zararlı yazılım oranlarına sahip 3 ayrı veri setinde incelenmiştir. DEX ve XML dosyalarından elde edilen öznitelikleri kullanan modelin 5 katlamalı çapraz doğrulama yöntemi ile 3 veri setinde ayrı ayrı

eğitim ve test işlemleri gerçekleştirilmiştir. Drebin&Androzoo, CICAndMal2017 ve CICMalDroid2020 veri setlerindeki uygulama sayıları, zararlı uygulama oranları ve modelin FNR, FPR, F1-skor ve MCC metriklerine göre başarıları Tablo 4.3'te gösterilmektedir.

Tablo 4.3: Modelin Drebin&Androzoo, CICAndMal2017 ve CICMalDroid2020 veri setlerindeki performans sonuçları

Veri Seti	Uygulama Sayısı	Zararlı Yazılım Oranı	FNR	FPR	F1-skor	MCC
Drebin&Androzoo	11.112	%50	%0,61	%1,29	%99,05	%98,09
CICAndMal2017	2.122	%20	%20,71	%1,18	%86,72	%83,41
CICMalDroid2020	16.714	%76	%0,84	%1,44	%99,35	%97,34

Drebin&Androzoo veri setinde toplam 11.112 uygulama bulunmaktadır ve zararlı uygulamaların oranı %50'dir. F1-skor ve MCC metriklerine göre sırasıyla %99,05 ve %98,09 başarı gösteren model zararlı ve zararsız uygulamaları FNR ve FPR metriklerine göre sırasıyla %0,61 ve %1,29 hata oranıyla tespit etmiştir. FNR metriğine göre diğer veri setlerine kıyasla en iyi performans Drebin&Androzoo veri setinde ölçülmüştür. Bu durum modelin az sayıdaki zararlı uygulamayı zararsız olarak sınıflandırdığını göstermektedir.

CICAndMal2017 veri setinde toplam 2.122 uygulama bulunmaktadır ve zararlı uygulamaların oranı %20'dir. FNR ve FPR metriklerine göre modelin CICAndMal2017 veri setindeki hata oranı hesaplandığında sırasıyla %20,71 ve %1,18 hata oranları elde edilmiştir. FNR metriğine göre modelin yüksek hata oranına sahip olması modelin fazlaca zararlı uygulamayı zararsız olarak sınıflandırdığını göstermektedir. FPR metriğine göre düşük hata oranına sahip olması modelin zararsız uygulamaların çoğunu başarılı bir şekilde tespit ettiğini göstermektedir. F1-skor ve MCC metriklerine göre sırasıyla %86,72 ve %83,41 başarı gösteren modelin performansı zararlı uygulamaların tespiti için yeterince iyi değildir. CICAndMal2017 veri setindeki uygulamaların yalnızca %20'sinin zararlı olduğu göz önüne alındığında modelin dengesiz bir veri setinde zararlı uygulamaları başarılı bir şekilde tespit etmeyi öğrenemediği sonucuna varılmıştır. Ayrıca veri setinde bulunan 425 zararlı uygulamanın 5 katlamalı çapraz doğrulama yönteminin uygulandığı model eğitimi için yeterli olmadığı görülmüştür. Modelin daha fazla sayıda zararlı uygulama içeren bir veri kümesinde eğitilmesi gerekmektedir.

CICAndMal2017 veri setindeki performans sonuçlarının aksine toplam 16.714 uygulama bulunan ve %76 zararlı uygulama oranına sahip olan CICMalDroid2020 veri setinde model başarılı bir performans göstermiştir. FNR ve FPR metriklerine göre sırasıyla %0,84 ve %1,44

hata oranına sahip olan model F1-skor ve MCC metriklerine göre %99,35 ve %97,34 başarı göstermiştir. Model zararlı ve zararsız uygulamaları düşük hata oranları ile sınıflandırmıştır. Ancak CICAndMal2017 veri setindeki hata oranları Drebin&Androzoo veri setine kıyasla daha yüksektir. Buna rağmen F1-skor metriğine göre model en iyi performansını CICMalDroid2020 veri setinde göstermiştir. FNR, FPR ve MCC metriklerine göre değerlendirildiğinde ise modelin Drebin&Androzoo veri setindeki performansının daha iyi olduğu görülmektedir.

F1-skor metriğine göre veri setlerindeki zararlı yazılım oranları modelin zararlı uygulamaları tespit etme başarısını etkilemektedir. F1-skor metriği, gerçek negatifleri dikkate almayan kesinlik ve duyarlılık metriklerine göre hesaplanmaktadır. Bu nedenle yüksek oranda zararlı uygulamaya sahip dengesiz veri kümelerinde F1-skor metriğine göre performans sonuçları iyi çıkmaktadır. Dengesiz veri setlerindeki performansı adil olarak değerlendirmek için daha iyi bir metrik olan MCC metriği kullanılmaktadır. MCC metriğine göre performansın hesaplanmasında karışıklık matrisindeki tüm ögeler dikkate alınmaktadır. Veri setlerindeki zararlı uygulama oranlarındaki dengesizlikler göz önüne alındığında MCC metriğine göre modelin en iyi performansı %98.09 başarı ile Drebin&Androzoo veri setinde görülmüştür.

4.2. ANDROİD TV UYGULAMALARININ GÜVENLİK ANALİZİ

Tez kapsamında geliştirilen modelin performansı önceki bölümlerde mobil uygulamalar içeren farklı veri setleri üzerinde değerlendirilmiştir. Ancak Android işletim sistemi sadece akıllı telefonlarda değil akıllı televizyonlar gibi farklı cihaz tiplerinde de çalışmaktadır ve zararlı uygulama geliştirenlerin hedefi olmaktadır. Bu kapsamda geliştirilen modelin zararlı TV uygulamalarını tespit etme başarısı incelenmiştir. Bölüm 4.2.1'de mobil uygulamalar ile eğitilmiş modelin performansı TV uygulamaları kullanılarak test edilmiştir. Bölüm 4.2.2'de TV uygulamaları ile eğitilmiş modelin TV uygulamaları ile test edilerek değerlendirilmiştir. Bölüm 4.2.3'te mobil ve TV uygulamalarının birlikte bulunduğu bir koleksiyon üzerinde eğitim ve test işlemleri uygulanarak performans sonuçları incelenmiştir.

4.2.1. Mobil Uygulamalar ile Eğitilmiş Modelin TV Uygulamaları Üzerindeki Performansı

Akıllı telefonlar için geliştirilen mobil uygulamalardan oluşan Drebin&Androzoo veri setinde eğitim ve test işlemleri yapılan modelin başarısı önceki bölümlerde incelenmiş ve F1-skor ve MCC metriklerine göre sırasıyla %99,05 ve %98,09 başarı gösterdiği görülmüştür. Mobil uygulamalar ile eğitilmiş modelin TV uygulamaları üzerindeki performansını incelemek için

567 zararsız TV uygulamasından ve tez kapsamında oluşturulan 471 zararlı TV uygulamasından oluşan Androzoo TV uygulama koleksiyonu kullanılmıştır. Literatürde paylaşılan uygulamalar arasında yeterli sayıda zararlı TV uygulaması olmaması nedeniyle tez kapsamında zararsız TV uygulamalarının içerisine zararlı kod parçaları yerleştirilerek zararlı TV uygulamalarının elde edilmesinden Bölüm 3.2'de bahsedilmiştir. Drebin&Androzoo veri setindeki mobil uygulamalar ile eğitilen model Androzoo TV koleksiyondaki uygulamalar üzerinde test edildiğinde 471 zararlı uygulamadan sadece 20'sinin tespit edilebildiği ve 567 zararsız uygulamadan 565'inin doğru sınıflandırıldığı görülmüştür.

Tablo 4.4'te gösterilen karışıklık matrisine göre mobil uygulamalar ile eğitilmiş modelin TV uygulamaları üzerindeki başarısı F1-skor ve MCC metriklerine göre sırasıyla %8,11 ve %13,46 olarak hesaplanmıştır. Elde edilen sonuçlar Drebin&Androzoo veri setindeki mobil uygulamalar ile eğitilmiş modelin Androzoo TV koleksiyondaki zararlı uygulamaları tespit etmekte yetersiz kaldığını göstermektedir. Modelin başarısının artırılması için zararlı Android TV uygulamalarının bulunduğu bir koleksiyonda eğitilmesi gerekmektedir.

Tablo 4.4: Mobil uygulamalar ile eğitilmiş modelin TV uygulamaları üzerindeki başarısı

	Tahmini Zararsız	**Tahmini Zararlı**
Gerçek Zararsız	565	2
Gerçek Zararlı	451	20

4.2.2. TV Uygulamaları ile Eğitilmiş Modelin TV Uygulamaları Üzerindeki Performansı

Tez kapsamında geliştirilen modelin zararlı Android TV uygulamalarını tespit edebilmesi için Androzoo TV koleksiyonundaki uygulamalar kullanılarak 5 katlamalı çapraz doğrulama yöntemi ile eğitim ve test işlemleri gerçekleştirilmiştir. 567 zararsız TV uygulamasından ve reverse_http, reverse_https ve reverse_tcp olmak üzere 3 farklı tipte zararlı kod parçası içeren 471 zararlı TV uygulamasından oluşan koleksiyonda model %100 doğruluk elde ederek tüm zararlı ve zararsız uygulamaları tam başarı ile ayrıştırmıştır. Bu durum modelin Meterpreter zararlı kod parçasını içeren uygulamaları zararsız uygulamalardan tam doğruluk ile ayırt etmeyi öğrendiğini göstermektedir.

Androzoo TV koleksiyonunda yer alan 471 zararlı TV uygulaması tez kapsamında zararsız TV uygulamalarının içerisine zararlı kod parçaları yerleştirilerek oluşturulmuştur. Ancak zararsız uygulamalardan dönüştürülen 471 zararlı TV uygulamasının dışında Liu ve diğ. [22] tarafından paylaşılan Android TV uygulamaları listesinde yer alan 3 adet TV uygulaması VirusTotal

sitesindeki anti-virüs tarayıcıları içerisinden en az 3'ü tarafından zararlı olarak etiketlenmiştir. Modelin TV uygulamaları üzerindeki eğitim ve test performansının ölçülmesinde bu 3 orijinal zararlı TV uygulaması kullanılmamıştır. TV uygulamaları ile eğitilmiş modelin başarısı 3 orijinal zararlı TV uygulaması üzerinde test edildiğinde hiçbirini zararlı olarak sınıflandırmadığı görülmüştür. Bu durum modelin sadece Meterpreter zararlı kod parçasını içeren zararlı TV uygulamalarını ayrıştırmayı öğrendiğini ve test edilen 3 TV uygulamasının Meterpreter zararlı kod parçasını içermediğini göstermektedir. Modelin kapsayıcı olabilmesi için farklı tiplerdeki zararlı uygulamaları tespit edebilmesi gerekmektedir.

4.2.3. Mobil ve TV Uygulamaları ile Eğitilmiş Modelin Performansı

Mobil uygulamalar ile eğitilmiş model zararlı TV uygulamalarını tespit etmekte yetersiz kalmıştır. Yeterli sayıda zararlı TV uygulaması bulunmaması nedeniyle tez kapsamında oluşturulan zararlı TV uygulamaları ile eğitilmiş model ise sadece Meterpreter zararlı kod parçasını içeren zararlı TV uygulamalarını tespit edebilmiş ve farklı zararlı TV uygulamalarını tespit edememiştir. Kapsayıcı olabilmesi için modelin farklı tiplerdeki zararlı uygulamalar ile eğitilmesine ihtiyaç duyulmaktadır. Bu kapsamda mobil ve TV uygulamalarının birlikte bulunduğu hibrit bir uygulama koleksiyonu oluşturulmuştur. CICAndMal2017 veri setindeki 2.122 Android uygulaması ile Androzoo TV koleksiyonundaki 1.038 TV uygulamasından oluşan hibrit koleksiyondaki zararlı ve zararsız uygulamaların sayıları Tablo 4.5'te sunulmuştur.

Tablo 4.5: Hibrit uygulama koleksiyonundaki zararlı ve zararsız uygulama sayıları

Veri Seti	Zararlı Uygulama Sayısı	Zararsız Uygulama Sayısı	Zararlı Uygulama Oranı
CICAndMal2017	425	1.697	%20
Androzoo TV	471	567	%45
Hibrit	896	2.264	%28

Hibrit koleksiyonundaki uygulamalar üzerinde 5 katlamalı çapraz doğrulama yöntemi kullanılarak modelin eğitim ve test işlemleri gerçekleştirilmiştir. F1-skor ve MCC metriklerine göre sırasıyla %96,28 ve %94,84 başarı gösteren model zararlı ve zararsız uygulamaları FNR ve FPR metriklerine göre sırasıyla %4,79 ve %1,02 hata oranıyla tespit etmiştir. FPR metriğine göre düşük hata oranına sahip olması modelin zararsız uygulamaların çoğunu başarılı bir şekilde ayrıştırdığını göstermektedir. Ancak FNR metriğinin görece daha yüksek olması modelin zararlı uygulamaların bazılarını zararsız olarak sınıflandırdığını göstermektedir.

Hibrit koleksiyondaki uygulamaların çoğu CICAndMal2017 veri setindeki mobil uygulamalardır. Bu nedenle Hibrit veri setinde CICAndMal2017 veri setine benzer sonuçlar alınması beklenen bir durumdur. CICAndMal2017 veri setinde eğitilen modelin F1-skor ve MCC metriklerine göre sırasıyla %89,72 ve %83,41 performans gösterdiği göz önüne alındığında Hibrit veri setinde modelin performansının yükseldiği görülmektedir. CICAndMal2017 veri setindeki zararlı uygulama sayısının ve oranının düşük olması nedeniyle model zararlı uygulamaları tespit etmede yetersiz kalmıştı. CICAndMal2017 veri setine kıyasla hibrit veri setinde daha fazla sayıda zararlı uygulama bulunması ve zararlı uygulama oranının görece daha yüksek olması F1-skor ve MCC metrikleri açısından daha iyi bir başarı elde edilmesini sağlamıştır.

Hibrit koleksiyonda eğitilen model Meterpreter zararlı kod parçası içermeyen 3 orijinal zararlı TV uygulaması ile test edildiğinde 3 TV uygulamasının da zararlı olarak sınıflandırıldığı görülmüştür. Sadece Androzoo TV koleksiyonundaki uygulamalar ile eğitilen modelin 3 orijinal zararlı TV uygulamasının hiçbirini tespit edemediği göz önüne alındığında mobil ve TV uygulamalarından oluşturulan hibrit koleksiyonda eğitilen modelin daha kapsayıcı olduğu ve farklı tiplerdeki zararlı uygulamaları tespit etmeyi öğrendiği görülmektedir.

Hibrit koleksiyonda eğitilen modelin farklı bir veri setinde test edilmesi için ApkMirror web sitesinden tez kapsamında Android TV uygulamaları toplanmıştır. Toplanan uygulamalar VirusTotal sitesindeki anti-virüs tarayıcıları ile analiz edildiğinde 370 uygulamanın hiçbirinin zararlı olmadığına karar verilmiştir. 182 uygulama zararsız olarak bırakılırken sırasıyla 67 uygulamaya reverse_http, 64 uygulamaya reverse_https ve 57 uygulamaya reverse_tcp zararlı kod parçaları eklenerek 3 farklı tipte 188 zararlı TV uygulamasından oluşan bir veri seti oluşturulmuştur. ApkMirror TV koleksiyonundaki uygulamalar hibrit koleksiyonda eğitilen model ile test edildiğinde 188 zararlı uygulamanın tamamının zararlı olarak tespit edildiği, 182 zararsız uygulamadan ise 178'inin zararsız olarak sınıflandırıldığı görülmüştür. ApkMirror TV koleksiyonunda test edilen modelin başarısı F1-skor ve MCC metriklerine göre sırasıyla %98,95 ve %97,86 olarak hesaplanmıştır. Elde edilen sonuçlar hibrit koleksiyonda eğitilen modelin farklı bir veri setinde bulunan zararlı ve zararsız Android TV uygulamalarını başarı ile tespit ettiğini göstermektedir.

5. TARTIŞMA

Akıllı telefon pazarında en popüler işletim sistemi olan Android işletim sistemi milyonlarca cihazda çalışmaktadır. Cep telefonu, tablet, saat, televizyon ve araba gibi birçok farklı tipteki akıllı cihazda çalışan Android işletim sistemi uygulama geliştiricilerin yanı sıra kötü niyetli kişilerin de hedefindedir. Zararlı Android uygulamaların tespit edilmesi konusunda literatürde birçok çalışma mevcuttur. Ancak bu çalışmaların çoğu popülaritesinden dolayı akıllı telefonlara odaklanmıştır. Android TV işletim sisteminin çalıştığı akıllı televizyonları inceleyen ise az sayıda çalışma vardır. Halbuki zararlı uygulamalar cep telefonları gibi televizyonlar için de önemli bir risk unsurudur.

Literatürde Android uygulamalarını paylaşan veri setlerinin çoğu cep telefonları için geliştirilen mobil uygulamaları içermektedir. Televizyonlar için geliştirilen Android TV uygulamalarını içeren veri setlerinde ise zararlı uygulama sayısının yetersiz olduğu görülmüştür. Televizyonda çalışan zararlı uygulamaları tespit edebilmek için zararlı ve zararsız TV uygulamalarından oluşan bir veri setine ihtiyaç duyulması nedeniyle tez kapsamında zararlı TV uygulamaları üretilmiştir. Zararsız TV uygulama paketlerinin içerisine Meterpreter zararlı kod parçaları eklenerek zararlı TV uygulamaları elde edilmiştir. Oluşturulan zararlı TV uygulamaları üzerinden hedef cihazda istenilen komutların çalıştırılabildiği görülmüştür.

Zararlı Android uygulamalarının tespit edilebilmesi için tez kapsamında özgün bir model geliştirilmiştir. Android uygulama paketleri içerisinde sıkıştırılmış olarak bulunan “AndroidManifest.xml” ve “classes.dex” dosyalarının bayt düzeyinde okunup TF-IDF yöntemi ile işlenmesi sonucunda uygulamaları temsil eden öznitelikler belirlenmiştir. Uygulama hakkında önemli bilgiler içeren AndroidManifest dosyasının ikili dosya formatından XML formatına deşifre edilmesiyle öznitelik çıkarımı yapılmıştır. Uygulamaya ait derlenmiş kaynak kodları barındıran DEX dosyası üzerinde ise tersine mühendislik uygulanmadan doğrudan öznitelik çıkarımı yapılmıştır. Uygulama kodlarının tersine mühendislikle elde edilememesi için bazı uygulamalar uygulama paketinin derlenmesi sırasında çeşitli gizlenme teknikleri kullanmaktadır. Zararlı uygulamaların tespit edilmesinde gizlenme tekniklerinden etkilenmemek adına tez kapsamında geliştirilen modelin uygulama kodlarının elde edilmesine ihtiyaç duymaması tercih edilmiştir. Zararlı ve zararsız uygulama paketlerinden bayt düzeyinde

elde edilen öznitelikler kullanılarak XGBoost sınıflandırıcısının eğitimi gerçekleştirilmiştir. 5 katlamalı çapraz doğrulama yöntemi ile gerçekleştirilen deneylerin sonucuna göre model F1-skor ve MCC metriklerine göre sırasıyla %99,05 ve %98,09 başarı ile Drebin&Androzoo veri setindeki zararlı ve zararsız uygulamaları tespit etmiştir. Gizlenme tekniğini kullandığı bilinen uygulamaların olduğu CICMalDroid2020 veri setinde ise F1-skor ve MCC metriklerine göre %99,35 ve %97,34 başarı elde edilmiştir.

Tez çalışmasında geliştirilen modelin performansı Drebin [11] veri seti üzerinde statik analiz yöntemi ile zararlı uygulamaları tespit eden literatürdeki diğer çalışmalarla karşılaştırılmış ve FNR, FPR ve F1-skor metriklerine göre karşılaştırma sonuçları Tablo 5.1'de sunulmuştur. FNR metriğine göre karşılaştırıldığında %0,50 hata oranı ile en düşük hata Yuan ve diğ. [5] ve %6,37 hata oranı ile en yüksek hata Grosse ve diğ. [46] tarafından yapılan çalışmalarda elde edilmiştir. Tez çalışmasında geliştirilen model ise %0,61 hata oranı ile karşılaştırılan çalışmalar arasında ikinci en iyi sonucu elde etmiştir. FNR hata oranının düşük olması modelin az sayıdaki zararlı uygulamayı zararsız olarak sınıflandırdığını göstermektedir ve zararlı yazılımların tespit edilmesinde kritik bir ölçümdür. FNR hata oranı yüksek olan çalışmalarda birçok zararlı uygulamanın tespit edilemeyerek zararsız olarak sınıflandırıldığı görülmektedir. Gizlenme tekniğine karşı dirençli olması adına bayt düzeyinde öznitelik çıkarımı yapılan tez çalışmasındaki modelin zararlı uygulamaları gözden kaçırma olasılığını azalttığı görülmektedir.

Tablo 5.1: Modelin performansının FNR, FPR ve F1-skor metriklerine göre literatürdeki diğer çalışmalar ile karşılaştırması

Model	**FNR**	**FPR**	**F1-skor**
Tez çalışmasında geliştirilen model	%0,61	%1,29	%99,05
Yuan ve diğ. [5]	%0,50	%2,10	%99,50
Kumar ve diğ. [6]	%5,00	-	%95,00
Feizollah ve diğ. [7]	%4,50	%4,40	-
Arp ve diğ. [11]	%6,10	%1,00	-
Mariconti ve diğ. [14]	%3,00	-	%96,00
Karbab ve diğ. [16]	%0,78	%0,45	%99,22
Yousefi-Azar ve diğ. [19]	%0,68	%1,21	%99,06
Grosse ve diğ. [46]	%6,37	%3,96	-
Zhang ve diğ. [47]	%6,00	%7,90	-

FPR metriğine göre karşılaştırıldığında %0,45 hata oranı ile en düşük hata Karbab ve diğ. [16] ve %7,90 hata oranı ile en yüksek hata Zhang ve diğ. [47] tarafından yapılan çalışmalarda elde edilmiştir. Tez çalışmasında geliştirilen model ise %1,29 hata oranı ile karşılaştırılan çalışmalar arasında en iyi dördüncü model olduğu görülmektedir. FPR hata oranının yüksek olduğu çalışmalarda zararsız uygulamalar sıklıkla zararlı olarak sınıflandırmakta ve ek bir inceleme yapılmasını gerektirmektedir. F1-skor metriğine göre en iyi sonuç Yuan ve diğ. [5] tarafından yapılan çalışmada %99,50 başarı ile elde edilmiştir. Tez çalışmasında geliştirilen model ise %99,05 başarı ile karşılaştırılan çalışmalar arasında en iyi dördüncü model olmuştur.

Zararlı uygulamaların tespit edilmesinde harcanan zaman karşılaştırıldığında Mariconti ve diğ. [14] tarafından yapılan çalışmada sadece öznitelik çıkarma için uygulama başına ortalama 33,83 saniye harcandığı görülmüştür. Yousefi-Azar ve diğ. [19] tarafından yapılan çalışmada öznitelik çıkarma ve sınıflandırma işlemleri uygulama başına 11,18 saniye tutmuştur. Karbab ve diğ. [16] tarafından yapılan çalışmada öznitelik çıkarma ve sınıflandırma işlemleri için uygulama başına 2,3 saniye harcanmıştır. Diğer çalışmalarda öznitelik çıkarma süresi belirtilmediği için karşılaştırılmaya dahil edilmemiştir. Tez çalışmasında geliştirilen model dosya çıkarma, öznitelik çıkarma ve sınıflandırma işlemlerini zararlı uygulama oranı açısından dengeli olan bir veri setinde uygulama başına 2,75 saniyede tamamlamıştır. Zararlı uygulama oranı yüksek olan ve gizlenme tekniklerinin uygulandığı bir veri setinde ise tüm işlemler 3,3 saniyede tamamlanmıştır. Ortalama 1,89 saniye süren dosya çıkarma işlemi çıkarıldığında tez çalışmasında geliştirilen modelin öznitelik çıkarma ve sınıflandırma işlemlerini 0,86 saniyede tamamladığı görülmektedir. Literatürdeki diğer çalışmalar ile karşılaştırıldığında tez çalışmasında geliştirilen modelin daha kısa sürede zararlı uygulamaları tespit edebildiği görülmüştür.

Tez kapsamında mobil uygulamalar ile eğitimi yapılmış modelin TV uygulamaları üzerindeki performansı incelendiğinde Meterpreter zararlı kod parçasını içeren zararlı TV uygulamalarının tespit edilemediği görülmüştür. Mobil uygulamalar ile eğitilmiş model TV uygulama koleksiyonunda F1-skor ve MCC metriklerine göre sırasıyla %8,11 ve %13,46 başarı göstermiştir. Mobil ve TV uygulamalarından oluşan hibrit bir uygulama koleksiyonu oluşturularak 5 katlamalı çapraz doğrulama yöntemi ile eğitim ve test işlemleri yapılan modelin performansı F1-skor ve MCC metriklerine göre sırasıyla %96,28 ve %94,84 olarak ölçülmüştür. Elde edilen sonuçlar zararlı Android uygulamalarını tespit eden modellerin eğitiminde kullanılacak uygulama koleksiyonunda sadece mobil uygulamaların bulunması

durumunda modelin başarısının yetersiz kaldığını ve televizyon gibi farklı cihaz tipleri için oluşturulmuş uygulamaların koleksiyona eklenmesinin modelin kapsayıcılığını arttırdığını göstermektedir.

Zararlı uygulamaları tespit eden modeli çalıştıran Python programının bellek kullanımı ölçüldüğünde DEX dosya boyutu arttıkça bellek kullanımının arttığı ve modelin O(n) uzay karmaşıklığına sahip olduğu görülmüştür. Yapılan deneylerde DEX dosya boyutunun yaklaşık 80 katı kadar bellek alanının kullanıldığı ölçülmüştür. Gün geçtikçe dosya boyutlarının arttığı göz önüne alındığında modelin uç cihazlar yerine daha fazla bellek alanına sahip sunucular üzerinde çalıştırılması olası bellek sorunlarının önüne geçecektir.

6. SONUÇ VE ÖNERİLER

Android uygulamalarının sayılarının hızla artması ve karmaşık gizlenme yöntemlerinin uygulanması nedeniyle uygulamaların güvenlik analizlerini kısa sürede tamamlayan otonom sistemlere ihtiyaç duyulmaktadır. Bu amaçla tez kapsamında Android uygulamalarının güvenlik analizlerini makine öğrenmesi yaklaşımı ile kısa sürede gerçekleştiren ve zararlı uygulamaları tespit eden bir model geliştirilmiştir. Geliştirilen modelin gerçekleştirdiği statik analizler sırasında uygulamaların kaynak kodlarının deşifre edilmesine ihtiyaç duyulmadan bayt düzeyinde işlem yapılması sayesinde gizlenme yöntemlerinden etkilenmeden zararlı uygulamalar tespit edilebilmektedir.

Tez kapsamında geliştirilen modelin belirlenmesinde kod çözme seviyesi, öznitelik çıkarımında kullanılan bayt uzunluğu, özniteliklerin elde edilmesinde kullanılan dosyalar, veri setindeki zararlı uygulamaların sayısı ve zararlı uygulama oranı gibi faktörler incelenerek modelin performansı üzerindeki etkileri ölçülmüştür. Geliştirilen model literatürde paylaşılan farklı veri setleri üzerinde test edilmiş ve diğer çalışmalar ile karşılaştırıldığında daha kısa sürede yüksek başarı ile zararlı uygulamaları tespit ettiği görülmüştür. Uygulama paketinden dosya çıkarma, öznitelikleri elde etme ve sınıflandırma işlemlerini uygulama başına 2,75 saniyede gerçekleştiren model Drebin&Androzoo veri setinde F1-skor metriğine göre %99,05 başarı göstermiştir. Gizlenme yöntemlerini kullanan uygulamaların bulunduğu CICMalDroid2020 veri setinde ise 3,3 saniyede işlemleri tamamlayan model F1-skor metriğine göre %99,35 başarı elde etmiştir. Tez kapsamında geliştirilen model ve mobil uygulama veri setlerinde elde edilen bulgular makale formatında "The Computer Journal" dergisinde yayınlanmıştır [35].

Android işletim sistemi cep telefonu dışında televizyon gibi farklı cihaz tiplerinde çalışmasına rağmen literatürde paylaşılan veri setleri popülaritesinin etkisiyle mobil uygulamalara odaklanmıştır. Tez kapsamında yapılan araştırmalarda Android TV uygulama güvenliğini inceleyen az sayıda çalışmanın bulunduğu ve zararlı Android TV uygulamalarının yetersiz olduğu gözlemlenmiştir. Literatürdeki bu eksikliği kapatmak adına tez çalışmasında zararlı Android TV uygulamalarının tespit edilmesi üzerine çalışmalar yapılmıştır. Zararlı Android uygulamaların paylaşan veri setleri olmasına rağmen zararlı Android TV uygulama sayısı yetersiz olduğundan tez çalışması kapsamında zararsız Android TV uygulamalarının içerisine

Meterpreter zararlı kod parçaları eklenerek zararlı TV uygulamaları oluşturulmuştur. Mobil uygulamalar ile eğitilen model TV uygulamaları üzerinde test edildiğinde zararlı TV uygulamalarını tespit etmede yetersiz kaldığı görülmüştür. Mobil ve TV uygulamalarından oluşan hibrit bir uygulama koleksiyonu oluşturularak modelin eğitimi gerçekleştirildiğinde ise F1-skor metriğine göre %96,28 başarı elde edilmiştir.

Zararlı Android uygulamalarının tespit edilmesi amacıyla kullanılan modelin eğitilmesinde farklı cihaz tipleri için oluşturulmuş uygulamaların eğitim setine dahil edilmesine ihtiyaç duyulduğu sonucuna varılmıştır. Yapılan deneylerde zararlı Android TV uygulama sayısının yetersiz olması nedeniyle Meterpreter zararlı kod parçalarını içeren zararlı TV uygulamaları oluşturulmuştur. Gelecek çalışmalarda farklı tiplerdeki zararlı TV uygulamaları elde edilerek modelin kapsayıcılığının arttırılması planlanmaktadır. Ayrıca hibrit uygulama koleksiyonunda cep telefonu ve TV cihazları için geliştirilmiş uygulamalar kullanılmıştır. Gelecekte yapılacak çalışmalara Android işletim sistemini destekleyen saat ve araba gibi farklı cihaz tipleri için geliştirilmiş Android uygulamaları koleksiyona dahil edilecektir.

KAYNAKLAR

[1]. IDC, 2021, IDC - Smartphone Market Share, https://www.idc.com/promo/smartphone-market-share, [Ziyaret Tarihi: 30 Nisan 2022].

[2]. G DATA, 2020, G DATA Mobile Malware Report 2019: New high for malicious Android apps, https://www.gdatasoftware.com/news/g-data-mobile-malware-report-2019-new-high-for-malicious-android-apps, [Ziyaret Tarihi: 17 Nisan 2022].

[3]. Badhani, S., ve Muttoo, S.K., 2019, CENDroid - A cluster-ensemble classifier for detecting malicious Android applications, *Computers & Security*, 85, 25-40.

[4]. Rathore, H., ve diğ., 2021, Robust android malware detection system against adversarial attacks using q-learning, *Information Systems Frontiers*, 23, 867-882.

[5]. Yuan, H., ve diğ., 2020, A detection method for android application security based on TF-IDF and machine learning, *Plos one*, 15 (9), e0238694.

[6]. Kumar, A., ve diğ., 2018, FAMOUS: Forensic Analysis of MObile devices Using Scoring of application permissions, *Future Generation Computer Systems*, 83, 158-172.

[7]. Feizollah, A., ve diğ., 2017, Androdialysis: Analysis of android intent effectiveness in malware detection, *Computers & Security*, 65, 121-134.

[8]. Milosevic, N., ve diğ., 2017, Machine learning aided Android malware classification, *Computers & Electrical Engineering*, 61, 266-274.

[9]. Mehtab, A., ve diğ., 2020, AdDroid: rule-based machine learning framework for android malware analysis, *Mobile Networks and Applications*, 25, 180-192.

[10]. Wongwiwatchai, N., ve diğ., 2020, Detecting personally identifiable information transmission in android applications using light-weight static analysis, *Computers & Security*, 99, 102011.

[11]. Arp, D., ve diğ., 2014, DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket, *Symposium on Network and Distributed System Security (NDSS)*, 22 Şubat 2014 San Diego, The Internet Society, 23-26.

[12]. Demontis, A., ve diğ., 2017, Yes, machine learning can be more secure! a case study on android malware detection, *IEEE Transactions on Dependable and Secure Computing*, 16 (4), 711-724.

[13]. Sahs, J., ve Khan, L., 2012, A machine learning approach to android malware detection, *2012 European Intelligence and Security Informatics Conference*, 22-24 Ağustos 2012 Odense, IEEE, ISBN: 978-1-4673-2358-1, 141-147.

[14]. Mariconti, E., ve diğ., 2017, MAMADROID: Detecting Android Malware by Building Markov Chains of Behavioral Models, *Symposium on Network and Distributed System Security (NDSS)*, 26 Şubat – 1 Mart 2017 San Diego, The Internet Society, ISBN: 1-891562-46-0, 1-15.

[15]. Martinelli, F., ve diğ., 2017, Evaluating convolutional neural network for effective mobile malware detection, *Procedia Computer Science*, 112, 2372-2381.

[16]. Karbab, E.B., ve diğ., 2018, MalDozer: Automatic framework for android malware detection using deep learning, *Digital Investigation*, 24, S48-S59.

[17]. Karbab, E.B., ve Debbabi, M., 2019, Maldy: Portable, data-driven malware detection using natural language processing and machine learning techniques on behavioral analysis reports, Digital Investigation, 28, S77-S87.

[18]. Yousefi-Azar, M., ve diğ., 2020, Byte2vec: malware representation and feature selection for Android, *The Computer Journal*, 63 (8), 1125-1138.

[19]. Yousefi-Azar, M., ve diğ., 2021, Mutual Information and Feature Importance Gradient Boosting: Automatic byte n-gram feature reranking for Android malware detection, *Software: Practice and Experience*, 51 (7), 1518-1539.

[20]. Narayanan, A., ve diğ., 2016, subgraph2vec: Learning Distributed Representations of Rooted Sub-graphs from Large Graphs, *12th International Workshop on Mining and Learning with Graphs (MLG 2016)*, 14 Ağustos 2016 San Francisco, ACM, ISBN: 978-1-4503-2138-9.

[21]. Lei, T., ve diğ., 2019, EveDroid: Event-aware Android malware detection against model degrading for IoT devices, *IEEE Internet of Things Journal*, 6 (4), 6668-6680.

[22]. Liu, Y., ve diğ., 2021, A first look at security risks of android tv apps, *36th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW)*, 15-19 Kasım 2021 Melbourne, IEEE, ISBN: 978-1-6654-3583-3, 59-64.

[23]. McAfee, 2020, Zararlı Yazılım (malware) nedir?, https://www.mcafee.com/tr-tr/antivirus/malware.html, [Ziyaret Tarihi: 4 Mart 2023].

[24]. Russell, S.J., ve Norvig, P., 2010, *Artificial intelligence a modern approach*, Pearson Education, New Jersey, ISBN: 978-0-13-604259-4.

[25]. Cunningham, P., ve diğ., 2008, *Supervised learning*, Machine Learning Techniques for Multimedia, In: Cord, M. (ed), ve Cunningham, P. (ed), Chapter 2, Springer, Berlin Heidelberg, ISBN: 978-3-540-75170-0, 21-49.

[26]. Nasteski, V., 2017, An overview of the supervised machine learning methods, *Horizons*, 4, 51-62.

[27]. Hinton, G., ve Sejnowski, T.J., 1999, *Unsupervised learning: foundations of neural computation*, The MIT Press, USA, ISBN: 9780262581684.

[28]. Collins, A.G.E., ve Cockburn, J., 2020, Beyond dichotomies in reinforcement learning, *Nature Reviews Neuroscience*, 21 (10), 576-586.

[29]. Yan, D., ve diğ., 2020, Network-based bag-of-words model for text classification, *IEEE Access*, 8, 82641-82652.

[30]. Garner, J., 2019, N-gram measures and L2 writing proficiency, *System*, 80, 176-187.

[31]. Bharadwaj, P., ve Shao, Z., 2019, Fake news detection with semantic features and text mining, *International Journal on Natural Language Computing (IJNLC)*, 8 (3), 17-22.

[32]. Mikolov, T., ve diğ., 2013, Distributed Representations of Words and Phrases and Their Compositionality, *Proceedings of the 26th International Conference on Neural Information Processing Systems – Volume 2*, 5-10 Aralık 2013 Nevada, New York, Curran Associates Inc., 3111-3119.

[33]. Jatnika, D., ve diğ., 2019, Word2vec model analysis for semantic similarities in english words, *Procedia Computer Science*, 157, 160-167.

[34]. Chen, T., ve Guestrin, C., 2016, XGBoost: A Scalable Tree Boosting System, *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 13 Ağustos 2016 San Francisco, New York, ACM, ISBN: 9781450342322, 785–794.

[35]. Ozogur, G., ve diğ., 2023, Android Malware Detection in Bytecode Level Using TF-IDF and XGBoost, *The Computer Journal*, bxac198.

[36]. Allix, K., ve diğ., 2016, Androzoo: Collecting millions of android apps for the research community, *IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*, 14-15 Mayıs 2016 Austin, New York, ACM, ISBN: 978-1-4503-4186-8, 468-471.

[37]. Kushniarou, A., 2018, GitHub - ArtemKushnerov/az: Downloads apks from androzoo repository https://androzoo.uni.lu/, https://github.com/ArtemKushnerov/az, [Ziyaret Tarihi: 15 Mayıs 2022].

[38]. Lashkari, A., ve diğ., 2018, Toward developing a systematic approach to generate benchmark android malware datasets and classification, *International Carnahan Conference on Security Technology (ICCST)*, 22-25 Ekim 2018 Montreal, IEEE, ISBN: 978-1-5386-7931-9, 1-7.

[39]. Mahdavifar, S., ve diğ., 2020, Dynamic android malware category classification using semi-supervised deep learning, *IEEE Intl Conf on Dependable, Autonomic and Secure Computing, Intl Conf on Pervasive Intelligence and Computing, Intl Conf on Cloud and Big Data Computing, Intl Conf on Cyber Science and Technology Congress (DASC/PiCom/CBDCom/CyberSciTech)*, 17-22 Ağustos 2020 *Calgary*, IEEE, ISBN: 978-1-7281-6609-4, 515-522.

[40]. Rapid7, 2023, Metasploit | Penetration Testing Software, Pen Testing Security, https://www.metasploit.com, [Ziyaret Tarihi: 13 Nisan 2022].

[41]. Ibotpeaches, 2010, Apktool - A tool for reverse engineering 3rd party, closed, binary Android apps., https://ibotpeaches.github.io/Apktool/, [Ziyaret Tarihi: 15 Mayıs 2022].

[42]. Android Open Source Project, Dalvik Executable format | Android Open Source Project, https://source.android.com/devices/tech/dalvik/dex-format#dex-file-magic, [Ziyaret Tarihi: 2 Mayıs 2022].

[43]. Pedregosa, F., ve diğ., 2011, Scikit-Learn: Machine Learning in Python, *J. Mach. Learn. Res.*, 12, 2825-2830.

[44]. Chen, T., ve Guestrin, C., 2016, XGBoost: A Scalable Tree Boosting System, *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 13-17 Ağustos 2016 San Francisco, New York, ACM, ISBN: 978-1-4503-4232-2, 785-794.

[45]. LimitCPU, http://limitcpu.sourceforge.net/, [Ziyaret Tarihi: 15 Mayıs 2022].

[46]. Grosse, K., ve diğ., 2017, Adversarial Examples for Malware Detection, European Symposium on Research in Computer Security, 11-15 Eylül 2017 Oslo, Springer, ISBN: 978-3-319-66398-2, 62-79.

[47]. Zhang, J., ve diğ., 2018, Dalvik opcode graph based android malware variants detection using global topology features, *IEEE Access*, 6, 51964-51974.

İNTİHAL RAPORU İLK SAYFASI

Gökhan Özoğur

ORİJİNALLİK RAPORU

%8	%7	%3	%3
BENZERLİK ENDEKSİ	İNTERNET KAYNAKLARI	YAYINLAR	ÖĞRENCİ ÖDEVLERİ

BİRİNCİL KAYNAKLAR

1	academic.oup.com İnternet Kaynağı	%2
2	Submitted to The Scientific & Technological Research Council of Turkey (TUBITAK) Öğrenci Ödevi	%2
3	acikbilim.yok.gov.tr İnternet Kaynağı	%1
4	dergipark.org.tr İnternet Kaynağı	<%1
5	Gokhan Ozogur, Mehmet Ali Erturk, Zeynep Gurkas Aydin, Muhammed Ali Aydin. "Android Malware Detection in Bytecode Level Using TF-IDF and XGBoost", The Computer Journal, 2023 Yayın	<%1
6	Submitted to Eskisehir Osmangazi University Öğrenci Ödevi	<%1
7	adlibilisimhizmetleri.com İnternet Kaynağı	<%1