T.C.
YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

ASSEMBLY KODU ÜZERİNDEN DOĞAL DİL İŞLEME VE YAPAY ZEKA İLE ZARARLI YAZILIM TESPİTİ

Alper EĞİTMEN

DOKTORA TEZİ
Bilgisayar Mühendisliği Anabilim Dalı
Bilgisayar Mühendisliği Programı

Danışman
Prof. Dr. Sırma YAVUZ

Eş-Danışman
Prof. Dr. A. Gökhan YAVUZ

Temmuz, 2024

T.C.
YILDIZ TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

ASSEMBLY KODU ÜZERİNDEN DOĞAL DİL İŞLEME VE YAPAY ZEKA İLE ZARARLI YAZILIM TESPİTİ

Alper EĞİTMEN tarafından hazırlanan tez çalışması 24.07.2024 tarihinde aşağıdaki jüri tarafından Yıldız Teknik Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalı Bilgisayar Mühendisliği Programı **DOKTORA TEZİ** olarak kabul edilmiştir.

Prof. Dr. Sırma YAVUZ
Yıldız Teknik Üniversitesi
Danışman

Prof. Dr. A. Gökhan YAVUZ
Türk-Alman Üniversitesi
Eş-Danışman

Jüri Üyeleri

Prof. Dr. Sırma YAVUZ, Danışman
Yıldız Teknik Üniversitesi _____

Prof. Dr. Tolga SAKALLI, Üye
Trakya Üniversitesi _____

Dr. Öğr. Üyesi Erkan USLU, Üye
Yıldız Teknik Üniversitesi _____

Dr. Öğr. Üyesi Göksel BİRİCİK, Üye
Yıldız Teknik Üniversitesi _____

Doç. Dr. Tuğba DALYAN, Üye
İstanbul Bilgi Üniversitesi _____

Danışmanım Prof. Dr. Sırma YAVUZ ve eş danışmanım Prof. Dr. A. Gökhan YAVUZ sorumluluğunda tarafımca hazırlanan Assembly Kodu Üzerinden Doğal Dil İşleme VE Yapay Zeka ile Zararlı Yazılım Tespiti başlıklı çalışmada veri toplama ve veri kullanımında gerekli yasal izinleri aldığımı, diğer kaynaklardan aldığım bilgileri ana metin ve referanslarda eksiksiz gösterdiğimi, araştırma verilerine ve sonuçlarına ilişkin çarpıtma ve/veya sahtecilik yapmadığımı, çalışmam süresince bilimsel araştırma ve etik ilkelerine uygun davrandığımı beyan ederim. Beyanımın aksinin ispatı halinde her türlü yasal sonucu kabul ederim.

Alper EĞİTMEN

İmza

Bu çalışma, “Yıldız Teknik Üniversitesi Bilimsel Araştırma Proje Koordinatörlüğü’nün FBA-2022-4904” numaralı projesi ile desteklenmiştir.

TEŞEKKÜR

Doktora tezim boyunca bana hep destek olan ailem, danışman hocalarım ve arkadaşım Taygun Kekeç'e teşekkürü bir borç bilirim.

Alper EĞİTMEN

İÇİNDEKİLER

SİMGE LİSTESİ

ℓ_C	Cross-Entropy hatası
λ	Hata ağırlık parametresi
θ, ϕ	Model parametreleri
ℓ_M	Mse hatası
$H_m(.)$	Sinyal entropisi

KISALTMA LİSTESİ

API	Application Programming Interface
APT	Advanced Persistent Threat
BIOS	Basic Input Output System
BOTNET	Robot Network
C&C	Command and Control
CBOW	Continuous Bag of Words
CFG	Control Flow Graph
CFG	Context Free Grammar
CISC	Complex Instruction Set Computer
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DAE	Denoising Auto-Encoder
DBN	Deep Belief Network
DDoS	Distributed Denial of Service
DLL	Dynamic-Link Library
DNN	Deep Neural Network
DT	Decision Tree
FNR	False Negative Rate
FPR	False Positive Rate
GAN	Generative Adversarial Nets
GCN	Graph Convolutional Networks
GPT	Generative pre-trained transformer
GPU	Graphic Processing Unit

GRU	Gated Recurrent Neural Networks
IP	Internet Protocol
ISA	Instruction Set Architecture
KNN	K Nearest Neighbor
LLM	Large Language Model
LSTM	Long-Short Term Memory
MALWARE	Malicious Software
MLP	Multi Layer Perceptron
MNN	Multiview Neural Network
MSE	Mean Square Error
OOP	Object Oriented Programming
PE	Portable Executable
RA	Random Append
RAB	Random Append Beginning
RAE	Random Append End
RISC	Reduced Instruction Set Computer
RNN	Recurrent Neural Network
SVM	Support Vector Machine
TF-IDF	Term Frequency - Inverse Document Frequency
URL	Uniform Resource Locator
USB	Universal Serial Bus
XAI	Explainable Artificial Intelligence
XGBOOST	Extreme Gradient Boosting

ŞEKİL LİSTESİ

TABLO LİSTESİ

ÖZET

Assembly Kodu Üzerinden Doğal Dil İşleme VE Yapay Zeka ile Zararlı Yazılım Tespiti

Alper EĞİTMEN

Bilgisayar Mühendisliği Anabilim Dalı
Bilgisayar Mühendisliği Programı
Doktora Tezi

Danışman: Prof. Dr. Sırma YAVUZ
Eş-Danışman: Prof. Dr. A. Gökhan YAVUZ

Zararlı yazılımlar, dijital iş akışları için önemli bir tehdittir. Geleneksel modelleme yöntemleri, çeşitli algoritmalar ya da bir uzman tarafından çıkarılan özelliklerin kullanılmasına odaklanırken, son zamanlardaki gelişmeler öğrenme tabanlı metodolojilerin gerekliliğini ortaya koymuştur. Nitelik ve nicelik evrim geçiren zararlı yazılımlarla, uzmanlar tarafından yapılan geleneksel "yakala- izole et- parmak izi çıkar" yöntemleri, mücadelede yetersiz kalmıştır. Çalışmamız statik analiz ile elde edilen Assembly kodları üzerinde, doğal dil işleme metotlarının kullanılarak karmaşık bir siber güvenlik problemini belge sınıflandırma problemine indirgeyen öğrenme tabanlı bir metodoloji sunmaktadır. Önerdiğimiz model, birden fazla davranışsal hedef değişkeni ile regüle edilen konvolüsyonel sinir ağlarının Assembly kodları ile eğitilerek sınıflandırma yapmaktadır. Bu model, birbirinden farklı bilgisayar sistemleri ve veri setleri üzerinde çeşitli gerçek dünya dayanım testlerine de sokularak doğrulanmıştır. Çalışmalarımız zararlı yazılım araştırmacılarının karşılaştığı problemler ve bunların önerilen modellere etkisini de araştırarak, model cevaplarının gerçek dünya senaryolarında nasıl tepkiler verebileceği üzerine simülasyonlar gerçekleştirmiştir. Metodolojimiz, zararlı yazılımların yetkinlik ve davranışını etiketleyen hedef değişkenlerin, baz derin öğrenme modellerini regüle etmesinin, sınıflandırma performansını hemen artırdığını göstermiştir. Metodolojimizin katkıları, AMDARGUS (Android) ve MOTIF (Windows) veri setleri üzerinde yapılan kapsamlı testler ile doğrulanmıştır.

Ortalama sınıflandırma başarısı ve F1 skorları, modelimizin diğer konvolüsyon tabanlı mimarilere kıyasla girdilerde yapılan manipülasyon saldırılarına karşı daha dayanıklı olduğunu göstermiştir. Çalışmamız, modelimizin genelleştirme başarısını ölçmek adına, sıfırıncı gün zararlı yazılımları testine sokulmuş ve baz modellere göre başarımı oldukça arttırdığı görülmüştür.

Anahtar Kelimeler: Zararlı yazılım, assembly, derin öğrenme, siber güvenlik

ABSTRACT

Malware Detection Through Natural Language Processing and Artificial Intelligence on Assembly Code

Alper EĞİTMEN

Department of Computer Engineering
Program of Computer Science
Doctor of Philosophy Thesis

Supervisor: Prof. Dr. Sırma YAVUZ
Co-supervisor: Prof. Dr. A. Gökhan YAVUZ

Malware is a significant threat to digital workflows. While traditional modeling methods focus on using various algorithms for fingerprinting or features extracted by an expert, recent developments have proved the necessity of learning-based methodologies. Evolving quality and quantity of malware, the conventional "capture-isolate-fingerprint" approach by experts have proven inadequate in combating. Our study presents a deep learning method that reduces a complex cybersecurity problem to a simplier “document classification” problem by using natural language processing methods on Assembly code obtained by static analysis. The proposed model performs classification by training convolutional neural networks which regulated by multiple behavioral weak target variables. This model has been validated through various real-world tests on different computer architectures and datasets. Our research also investigates the challenges faced by malware researchers and the impact of these challenges on the proposed models over real-world scenarios. Our methodology demonstrates that the regulation of basic deep learning models by weak target variables immediately improve classification performance. The contributions of our methodology have been validated through extensive tests on the AMDARGUS (Android) and MOTIF (Windows) datasets. The average classification accuracy and F1 scores show that our model is more resilient to manipulation attacks on inputs compared to other

convolutional-based architectures. To measure the generalization success of our model, zero-day malware testing conducted and it showed significant improvement over baseline models.

Keywords: Malware, assembly, deep learning, cyber security

1
GİRİŞ

1.1 Giriş

Zararlı yazılımlar (Malware) ve bunlara bağlı olarak bilgisayar sistemlerindeki kesinti, aksama veya istismarlar önemli bir sosyo-ekonomik tehdit oluşturmaktadır. Gerek kurumsal gerekse bireysel seviyede elektronik sistemlere sürekli ve ayrım gözetmeyen saldırılar, kaynak istismarı, servis bozulmaları ve hırsızlıklara neden olmaktadır. Yıllar öncesine kıyasla günümüz dünyası mobil cihazlar, Nesnelerin İnterneti (IoT), akıllı araçlar ile doludur. Buna paralel olarak bu cihaz ve sistemler üzerinden verilen hizmetlerin çeşitliliğinin artması ile hızlı ve ulaşılabilir internet ağı ile birleşince sistemlerin ve kullanıcıların zararlı yazılımlarla olan olası temasını oldukça arttırmıştır.

İş akışlarının hızlı bir şekilde elektronik akışlara dönüştüğü günümüzde, zengin içerik ve servis sunan yazılımların sayısı üstel olarak artmış, kabiliyetleri ise evrim geçirmiştir. Buna bağlı olarak bu sistemleri suistimal ederek haksız kazanımlar elde etmeye çalışan tehdit unsurları da aynı oranda artmıştır. Bu tehdit unsurlarının kullandığı en önemli araç "Malicious Software" teriminin kısaltması olan Malware (zararlı yazılı) yazılımlardır. Ciddi oranlarda haksız kazanç sağlayan tehdit unsurları beraberinde bununla mücadele eden siber güvenlik pazarını da oldukça büyütmüştür.

Karşılıklı olarak süre gelen bu mücadele sonucunda Malware gerek nicelik gerekse nitelik olarak evrim geçirerek günümüzde avantajlı duruma gelmiştir. Artık çeşitli analiz ve tespit yöntemlerine karşı kaçınma ve korunma metotları Malware yazılımlarında standart hale gelmiştir. Buna karşılık günümüzde tespit sistemleri

çoğunlukla nitelikli Malware araştırmacılarının tespit etme, izole etme, tersine mühendislik ile kullanılan teknikleri belirleme ve nihayetinde imza ve kural setlerini çıkarma döngüsünü kullanmaktadır. Günler sürebilen bu işlem, Malware sayısının üstel olarak artması ile baş edilebilir olmaktan çıkmıştır.

Literatürde bu kapsayıcılık ve erişebilirlikten kaynaklanan Malware sorunlarının yalnızca doğru şekilde tasarlanmış otomatik analiz yöntemleri ile aşılabileceği kabul edilmektedir. Ancak, bu tür analizlerin doğruluğu, hızlı çıkarım yapabilen yeni teknikler ve makul yaklaşımlar gerektirmektedir. Bu modern yaklaşımlar öğrenme tabanlı metodolojileri kullanarak Malware analiz etmek ve problemi genelleştirmeye çalışmaktadır. Bu metodolojilerin birçoğu yapay sinir ağları mimarisi kullanmakta ve analizleri dinamik, statik ya da hibrit yaklaşımlar ile yapılmaktadır. Statik analiz, zararlı yazılımın çalışma zamanı davranışına erişim olmaksızın programın meta verilerinin ve yazılımın kodlarının incelenerek özellik çıkarımına dayanır. Buna karşılık, dinamik analiz, programın çalışma zamanı davranışını gözlemlemek ve incelemek için kontrollü bir ortam gerektirir.

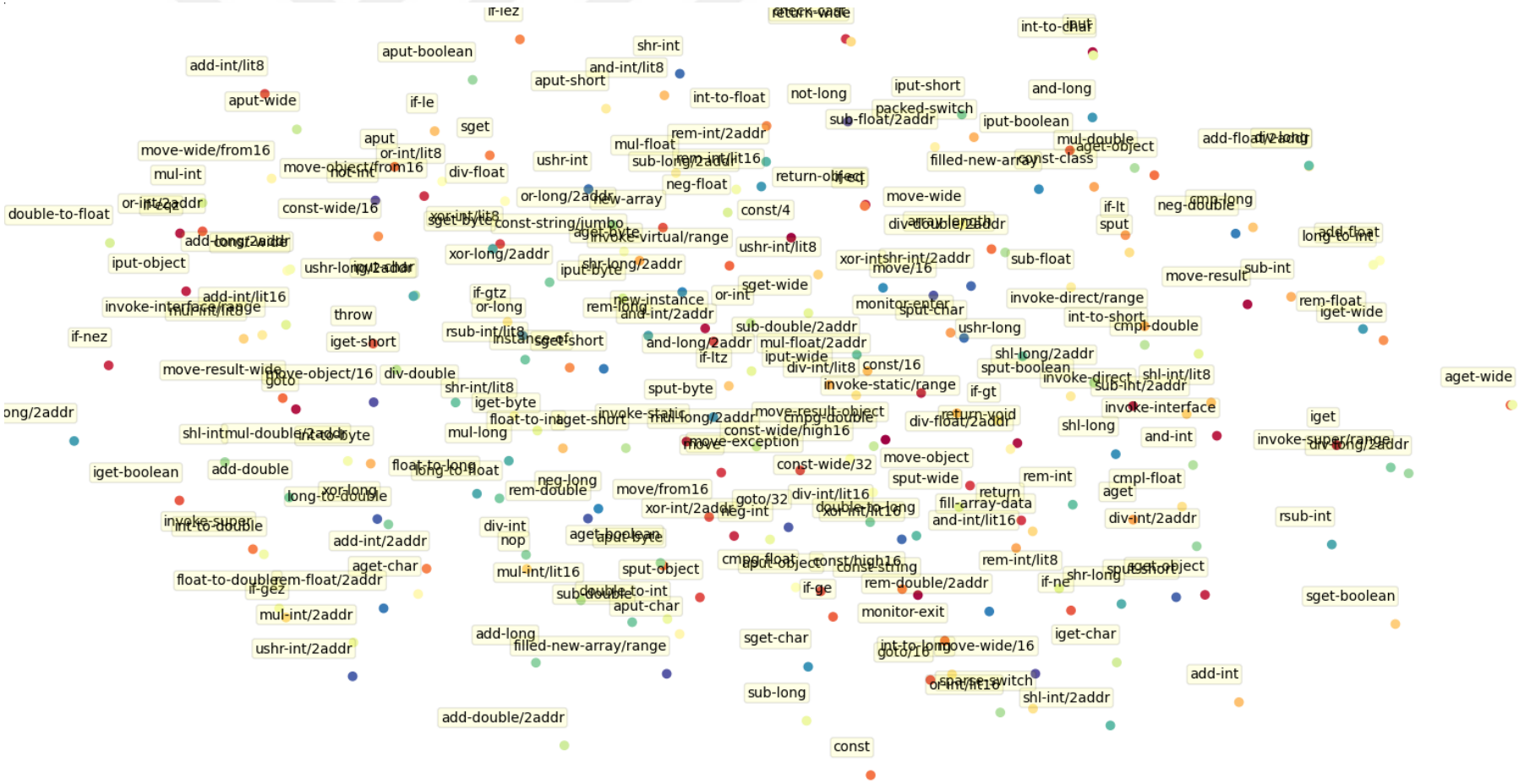
Bu çalışmada statik analiz ile sınıflandırma görevine yönelik yeni bir metodoloji önerilmektedir. Zararlı yazılım sınıflandırma görevini modelleyen çeşitli yaklaşımlar olmasına rağmen, Assembly kod dizilerini etkili bir şekilde düzenleyen ve geniş bir ortamda tatmin edici performans gösteren bir model eksiktir. Bu çalışma da bu sorunu ele alıyor ve derin öğrenme tabanlı yeni bir konvolüsyonel mimari öneriyoruz. Önerdiğimiz mimari, yazılımların Assembly kodlarını ve az güvenilir ancak birbiri ile ilişkili birden fazla davranışsal hedef değişkenlerini ortak formüle eden ortam bağımsız derin öğrenme tabanlıdır. Bu mimari gerçek dünya testlerine tabi tutularak test edilmiş ve literatürdeki mevcut yöntemlere katkı yapmıştır.

1.2 Programlama Dilleri ve Doğal Diller Arasındaki Benzerlikler

Her ne kadar kapsamı çok daha dar olsa da yazılım dilleri, doğal diller ile benzerlikler içermektedir. Bu benzerlikler arasında grammar (dil bilgisi),

vocabulary (sözlük), syntax (söz dizimi) ve semantics (anlambilim) bulunmaktadır. Hatta temel düzeyde context sensitivity'de (bağlam duyarlılığı) doğal dil ve yazılım dilleri arasındaki benzerliklerden biridir. Programlama dillerindeki açık ve kesinlik doğal dillerde yoktur. Sözlük boyutu çok daha küçük, anlamlı cümleler olarak düşünebileceğimiz kod blokları da doğal dillere göre çok daha uzundur. Bu diller arasında Assembly dilinin ayrı bir yeri bulunmaktadır. Çalışacağı makine koduna en yakın okunabilir programlama dillerine verilen genel isim olan Assembly, çalıştırıldığı gerçek ya da sanal donanım kodundan kolaylıkla elde edilebilir. Bununla beraber ortamlar farklı olsa da kullanılan mnemonicler (hafıza yardımcısı) ortamlar arasında oldukça benzerdir. Dört işlem, register (yazmaç), branch (dallanma) operatörleri gibi çeşitli gruplar altında toplayabileceğimiz bu mnemonicler farklı ortamlar için ki buna ara sanal katmanlarda dahil, neredeyse aynıdır.

Bu benzerlik ve farklılıkları göz önüne alırsak günümüzde doğal dil işleme için geliştirilen başarılı metodolojilerin programlama dilleri üzerinde de çalışması oldukça olasıdır. Kod sekanslarını kelimeler, anlamlı kod bloklarını da cümleler şeklinde kabul edersek bir yazılımı oluşturan tüm kodları bir belge ya da kitap olarak kabul edebiliriz. Fakat burada dikkat edilmesi gereken bazı noktalar vardır.



Şekil 1.1 Android Mnemonic kelime temsil vektörlerinin t-SNE ile görselleştirilmesi

1.3 Programlama Dilleri ve Doğal Diller Arasındaki Farklılıklar

Assembly dilinde bulunan mnemonic (add, jump, mov vb) sayısı ki buna sözlük boyutu da diyebiliriz, oldukça küçüktür. Örneğin Android ortamında, Dalvik sanal makinesi için tüm sözlük 1 byte, x86-64 mimarisi için ise 2 byte ile ifade edilebilmektedir. Bu da Android için en fazla 255, x86-64 mimarisi için 4096 kelime anlamına gelmektedir ki veri setimizden çıkardığımız sözlükte (adres ve operandlar çıkartıldıktan sonra) x86-64 mimarisi için benzersiz 1500 kelimeye rastlanmıştır. Bu kelimeleri kendi içinde anlamlarına göre grupladığımızda ise sayı daha da düşmektedir. Örneğin yapılan bir toplama işleminde operandların boyutuna göre ilgili mnemonic değişse de aslında davranışsal olarak yaptığı iş toplamadır.

Mnemonicler, register işlemleri, dört işlem, branch komutları, interrupt (kesme) gibi alt başlıklarda önemli bir bilgi kaybı olmadan gruplanabilirler. Skip-Gram algoritması ile vektörize edilen mnemoniclerin, bu vektörler kullanılarak hesaplanan birbirlerine olan yakınlıklarını Şekil 1.1'de görebiliriz. Dolayısıyla bu aşamada bu mnemonicleri doğal dillerdeki kelimelere benzetmek doğru bir yaklaşım olmayacaktır. Doğal dillerdeki kelimeler daha kesin bir anlam ya da kavramı ifade etmekte ve kendisiyle birlikte bağdaştırılabilen yüzlerce belki de binlerce kelime içermektedir. Assembly dilinde ise bir anlam ifade edilebilmesi için mnenomicler bir sekans haline gelmelidir. Örneğin çarpanlara ayırma (factorization) işlemi doğal dillerde iki kelime ile ifade edebilirken bu işi yapan Assembly kod bloğu yüzlerce satır olabilir.

Bir diğer nokta ise doğal dillerde anlamlı cümle uzunlukları arasındaki varyans'ın Assembly diline oranla çok daha küçük ve yönetilebilir olmasıdır. Anlamlı bir iş yapan kod blokları onlarca satırdan on binlerce satıra kadar değişmektedir. Bununla beraber program kodu boyutları kilo byte seviyelerinden rahatlıkla mega byte seviyelerine ulaşabilmektedir. Bu normalde doğal dillerde nispeten yönetilebilir olan sparse data problemini katlayarak karşımıza getirmektedir. Özetle, bir anlam ifade etmeyen ya da bağlamdan kopuk kod bloklarının da Assembly dilinde mevcut olması nedeniyle, bir program kodunu yüksek gürültülü, yüksek varyanslı oldukça

kirli bir dokümana benzetmek çok yanlış olmayacaktır.

Son olarak programlama dillerinde location invariance (konum bağımsızlığı) söz konusudur. Anlamlı kod bloğu memory üzerinde parçalanmış bir şekilde farklı noktalarda bulunabilir. Bu konumlar rahatlıkla yer değiştirilebilir. Dolayısıyla programlama dillerinin doğal dil dokümanları gibi iyi tanımlanmış bir akış ile sunulacağının bir garantisi yoktur

1.4 Motivasyon

Bütün bu benzerlik ve farklılıkları göz önüne aldığımızda bir kod parçasından yüksek seviyede anlam çıkarabilme ve bu anlamların ilişkilerine göre bir sınıflandırma yapabilmenin teoride mümkün gözüktüğü inancı ile son yıllarda doğrudan programlama dili kodları üzerinden doğal dil işleme yöntemlerinin etkinliği üzerine çalışmalar hızlanmıştır. Bizde aynı motivasyon ile çalışmalarımıza başlamış, öncelikle metodolojiyi ispat eden bir ön çalışma sonrasında derin öğrenme tabanlı güncel bir sınıflandırıcı ile literatürde bulunan state-of-art metodolojilere bir katkı yapmış bulunmaktayız.

1.5 Problem Tanımı

Bu çalışmanın konusu olan öğrenme tabanlı metodolojiler ile Malware tespitine giriş yapmadan önce buna neden ihtiyacımız olduğundan bahsetmek gerekecektir. Burada öncelikle istatistiki bazı verileri incelememiz gerekmekte. Siber saldırılar 2020 sonu itibariyle toplam 6 trilyon dolar hasara neden olmuş, 2025 yılında ise bu rakamın 10,5 trilyon dolara çıkacağı öngörülmüştür [1]. Günlük 16 milyar dolar zarara neden olan bu saldırıların akümülatörü olan Malware sayısı ise 2014 ile 2019 yılları arasında 750 milyon artmıştır. 2020 yılında ilk defa karşılaşılan (zero-day) Malware ailelerin sayısı %88 oranında artmış, 500 bin ilk defa karşılaşılan Malware tespit edilmiştir [2]. 2020'nin son çeyreğinde her üç siber saldırıdan birinin yeni bir Malware ile yapıldığı ve mevcut güvenlik sistemlerinin bunları tespit edemediği görülmüştür. Kobilere yapılan siber saldırıların %50'si başarılı olmakta, siber saldırıya uğrayan kobilerin %60'sı ise saldırıya uğradıktan 6 ay sonra kapanmıştır.

Ransomware olarak bilinen Malware nedeniyle sadece 2020 yılında 11,5 milyar dolar fidye ödenmiştir [3] [4] [5] [6] [7].

Buna karşılık geleneksel yöntemler ile çalışan Anti-Malware uygulamaları için yeni bir örneğin tespit edilip, izole edilerek analizinin yapılması ve parmak izinin çıkartılma süresi yaklaşık 8 gündür [6]. Nicelik olarak üstel şekilde artan zararlı yazılımların kabul edilebilir bir sürede yakalanıp, analiz edilip, parmak izini çıkartmak devlet organizasyonu seviyesinde dahi mümkün görülmemektedir. Geleneksel yaklaşımdaki yetersizlik, araştırmacıları doğru şekilde tasarlanmış otomatik analiz ve tespit metodolojilerini araştırmaya itmiştir.

1.6 Tezin Akışı

Bu çalışmada öncelikle motivasyon, problem detayı ve hedef açıklanacaktır. Malware tanımı detaylı şekilde yapıldıktan sonra Malware sınıflandırma çalışmalarında karşılaşılan zorluklardan bahsedilecektir. Sonrasında kapsamlı bir literatür özeti verilecektir. Literatür özeti konu ile ilgili erken çalışmalar kısmında kronolojik olarak ilerleyecek, güncel çalışmalar ise analiz yaklaşımı, kullanılan özellikler ve kullanılan modeller olarak alt başlıklarda gruplanarak verilecektir. Sonrasında önerdiğimiz yöntemler detaylı olarak anlatılacak ve sonuçlar paylaşılacaktır. Tartışma bölümünden sonra gelecek çalışmalar konuşulacaktır.

Tez akışında sadelik ve netliği yakalamak adına önerdiğimiz yöntemde kullanılan ve literatürde genel olarak bilinen metodoloji, model ve kavramlar (örneğin LSTM gibi) detaylı olarak anlatılmayacak özet şekilde bahsedildikten sonra ilgili akademik çalışmaya doğrudan referans verilecektir.

Tez dili Türkçedir. Kullanılan terimlerin akademik camiada benimsenmiş Türkçe karşılıkları kullanılacaktır. Fakat tezin anlaşılmasını zorlaştıracak, benimsenmemiş çeviriler yerine daha çok bilinen İngilizce karşılıkları ile kullanılmaya devam edecektir. Bu terimlerin Türkçe karşılıkları terimler sözlüğü olarak ek kısmında verilecektir.

Çalışmadan kullanılan bütün veri seti, kod blokları ve ek materyaller tez sahibi ya da danışmanları aracılığı ile iletişime geçilerek temin edilebilir olacaktır.

2
MALWARE TANIMI VE MODERN MALWARE

Bu bölüm Modern Malware tanımı, çeşitleri, özellikleri, yetkinlikleri ve son olarak Malware araştırması konusunda günümüzdeki zorluk ve problemleri anlatacaktır.

Siber Saldırılar amaçlarına göre çeşitli yöntem ve metodolojiler izlemektedir. Bu yöntemler arasında en çok kullanılanlardan bir tanesi Malware kullanmaktır. Malware, sayısal sistemleri çeşitli amaçlar doğrultusunda hedef alan farklı yetenek ve metotlara sahip bütün zararlı yazılımları kapsayan genel bir terimdir. Dolayısıyla Malware amaçlarına (davranışsal) göre ya da bunu nasıl yaptığına göre (metodoloji) çeşitli alt kırılımlara ayrılır. Bu alt kırılımlar günümüzde sayısal iş akışlarının çeşitlenmesi ve geniş bant internet ile evrimleşmiştir.

Trilyon dolar seviyelerinde zarara neden olan Malware saldırıları, beraberinde oldukça büyük bir siber güvenlik pazarı yaratmış gerek sistem geliştiricilerinin gerekse Anti-Malware geliştiricilerinin karmaşık güvenlik mekanizmaları geliştirmesine ön ayak olmuştur. Etki, tepkiyi doğurarak Malware geliştiricilerinin de bu güvenlik mekanizmalarını atlatmak adına karmaşık metodolojiler geliştirerek basit Malware'ler yerine analiz sistemlerini tespit edebilen, güvenlik sistemlerinden kaçınabilen ve kendi yapısını değiştirebilen yeni nesil Malware türleri geliştirmelerine sebep olmuştur. Bu karmaşık mekanizmaları içeren zararlı yazılımlara bu çalışmada Modern Malware denmektedir.

Geleneksel Anti-Malware sistemleri imza tabanlı ve sezgisel kurallar ile çalışmaktadır. İmza, bir yazılımın çalıştırılabilir dosyası üzerinde çeşitli algoritmalar koşturularak o program adına benzersiz bir iz oluşturma adımıdır. Sezgisel yöntemler ise bir siber güvenlik uzmanının, bir Malware yazılımını

izole ve monitör edilen bir ortamda analiz edip davranışlarına yönelik bir kural seti tanımlaması ve bu kural setinin ilgili güvenlik yazılımında işletilmesi üzerinedir. İmza tabanlı yöntemlerin etkisini kaybetmesi üzerinden oldukça uzun bir zaman geçmiştir. Program kodu üzerinde yapılan basit morfolojik dönüşümler ile zararlı yazılımlar imzalarını oldukça hızlı ve masrafsız şekilde değiştirebilmektedir. Sezgisel yöntemler ise ilgili Malware üzerinde detaylı analiz yapılmasını gerektirmektedir. Modern Malware'ler bu analizin yapılmasını zorlaştıracak birçok basit veya kompleks yöntemler içermektedir. Her ne kadar bu yöntemler deneyimli bir Malware analiz uzmanını tamamen kandırmasa da oldukça yavaşlatmaktadır.

2.1 Malware Çeşitleri

Malware'leri amaçlarına, nasıl çalıştıklarına ve yetkinliklerine göre sınıflayabiliriz. Amaçlarına göre gruplama bize yüksek seviye bir temsil sağlar. İlgili amaç için farklı tiplerde birden çok Malware kullanılabilir. Sırayla bunlardan bahsetmek gerekirse:

2.1.1 Amaçlarına Göre Malware Çeşitleri

Amaçlarına göre Malware tiplerini aşağıdaki şekilde gruplayabiliriz.

- **Veri Hırsızlığı:** Kişisel bilgiler, finansal veriler ya da fikri mülkiyet gibi hassas verileri çalmaya odaklanmış bu Malware tipi, sistemlere zarar vermez. Bulaştığı sistemlerde kendini gizleyerek hassas verileri toplar ve uzak sunucuya gönderir. Bu tip bir saldırı davranış olarak veri transferi yapan zararsız bir yazılım gibi davranacaktır. Literatürde Stealer olarak bilinmektedir.
- **Finansal Kazanç:** Hedef sistem veya kişilerden maddi kazanç elde etmeyi amaçlayan saldırı şeklidir. Bu saldırılar hedef sistemdeki banka uygulamalarını hedef alabileceği gibi, taklitte edebilir. Bir diğer yöntem ise hassas verileri şifreleyerek kullanıcıdan bu verileri tekrar çözümleme

karşılığında fidye istemektir. Literatürde Banker ve Ransomware olarak bilinirler.

- **Sisteme Hasar Verme:** Bulaştığı sistemde kritik dosyaları veya konfigürasyonları bozarak ya da silerek sistem kesintisi sağlayan saldırı tipidir.

- **Casusluk ve Sabotaj:** Bireyler, organizasyonlar ya da devlet kurumlarından keşif ve casusluk amacıyla bilgi toplayan veyahut kritik altyapısal sistemleri bozarak hedef devleti zor durumda bırakma amacıyla geliştirilmiş Malware çeşididir. En bilinenlerinden bir tanesi Stuxnet'tir.

- **Kesinti Sağlamak:** Normal şekilde ilerleyen dijital iş akışlarını kesintiye uğratma amacı ile geliştirilmiş Malwarelerdir. Denial of Service (DoS) buna örnek verilebilir.

- **Reklam Geliri Sağlamak:** Kullanıcıları istenmeyen reklamları izlemek ya da ilgili bağlantıya yönlendirmek için zorlayan ve buradan gelecek olan reklam gelirinden kazanç sağlama amacı ile geliştirilmiş Malware çeşitleridir. Literatürde Adware olarak bilinirler.

- **BotNet Oluşturmak:** BotNet olarak bilinen, birden çok sisteme bulaşmış uyku halinde bekleyen bu Malware'ler uzaktan aktive edildikleri taktirde amaçlandıkları işi topluca gerçekleştirirler. Bu bir Distributed DoS (DDoS) saldırısı olabileceği gibi, sistemlerin kaynaklarını kullanarak belirli bir görevi yerine getirmede olabilir. Buna örnek olarak kripto para madenciliği gösterilebilir.

- **İzinsiz Kaynak Kullanımı:** Belirli bir görev için hedef sistemin çeşitli kaynaklarını (CPU, Memory, Disk veya network) gizlice izinsiz olarak kullanmak için tasarlanmış Malwarelerdir.

- **Yetkisiz Giriş:** Sistemlere veya ağlara yasa dışı bir şekilde giriş yaparak kaynakları veya verileri sömürme üzerine tasarlanmışlardır.

- **Kullanıcı İzleme:** Kullanıcı faaliyetlerini izleyip kaydederek ilgili bilgileri yetkisiz kişilere ulaştırmak üzerine tasarlanmış Malwarelerdir.

- **Malware Dağıtım:** Diğer Malwareleri parmak izlerini değiştirecek şekilde çeşitli dönüşümlere tabi tutarak uzak sistemlere dağıtan ya da bulaştıran yazılımlardır.
- **Sahtekarlık:** Hedef kullanıcı ve sistemleri kandırarak sahtekarlık saldırılarına zemin hazırlamak için geliştirilmiş Malwarelerdir
- **Politik Manipülasyon:** Sistemlere doğrudan zarar vermeyen bu zararlı yazılımlar, kullanıcıların sosyal medya ile olan etkileşimini onlara fark ettirmeden manipüle ederek belirli bir politik ajandayı öne çıkartırlar. Bu tip yazılımlar kullanıcıların bilgi kaynaklarını manipüle edebilir, gördüğü reklamları ya da sosyal medyada karşısına çıkan haber, kişi, yorum gibi bilgi ve etkileşim kaynaklarını ilgili politik ajandaya destek olacak şekilde değiştirebilir. Bununla beraber sistemleri bozabilir ya da hassas verileri sızdırabilirler.
- **Data Manipülasyonu:** Hedef sistemdeki veri setlerini, sonuçları, konfigürasyonları bilinçli bir şekilde bozup değiştirme amacı ile geliştirilmiş Malwarelerdir. Bu sayede hedef sistemin ve kişilerin doğru çıkarım yapıp doğru metodolojiler geliştirmesini engeller ya da ciddi oranda yavaşlatır.

2.1.2 Çalışma Şekillerine Göre Malware Çeşitleri

Çalışma şekillerine göre Malware tiplerini aşağıdaki şekilde gruplayabiliriz.

- **Virus:** Kendini bir ana dosyaya ekleyip, enfekte dosya çalıştırıldığında yayılan Malware türüne verilen gelen isimdir.
- **Worm:** Bir ana dosyaya eklenmeye ihtiyaç duymadan network aracılığıyla yayılan kendini kopyalayan Malware türüdür.
- **Trojan Horse:** Kullanıcıları kandırarak kendini kurdurmak için meşru bir yazılım gibi davranan, ardından kullanıcıya sunduğu işlemler haricinde gizlice kötü amaçlı işlemler gerçekleştiren yazılımdır.
- **Ransomware:** Hedef sistemin hassas verilerini şifreleyerek, şifre çözme anahtarı için fidye talep eden yazılımdır.

- **Spyware:** Kullanıcının bilgisi olmadan kullanıcı bilgilerini gizlice izleyen ve toplayan yazılımdır.
- **Adware:** Kullanıcının cihazında otomatik olarak istenmeyen ya da manipülatif reklamlar gösteren veya indiren yazılımdır.
- **Rootkit:** Diğer kötü amaçlı yazılımları gizleyen ve bir bilgisayar sistemine yetkisiz erişim sağlayan yazılımdır. Bu yazılımlar çeşitli donanım ve yazılım seviyelerinde çalışabilirler. Kendini BIOS ile işletim sistemi arasında bir katmanda çalıştırabilir dolasıyla sistemlerde kalıcılığı ve tespiti oldukça zor olabilir.
- **Keylogger:** Klavye tuş vuruşlarını kaydeden ve bunları uzak sunucuya gönderen yazılımdır. Bu sayede hassas verileri çalmaktadır.
- **Backdoor:** Meşru yazılımların içine uygulama geliştiricisinin bilgisi dahilde ya da gizlice çeşitli yöntemler ile gizlenmiş olan bir kod bloğudur. Bu kod bloğu ile uygulamanın çalıştığı sistemlere yetkisiz ve izinsiz giriş yapılabilir olmaktadır.
- **BotNet:** Uyku halinde bulunan bu Malware tipi, enfekte ettiği sistemleri, bir uzaktan kontrol merkezine bağlayan ve belirli bir amaç adına topluca aktif edilebilen hale getirir.
- **Rougeware-Scareware:** Kendini meşru bir güvenlik yazılımı olarak tanıtarak sistemde sahte tehditler bildirerek bunları temizleme adına ödeme talep eden Malware tipidir.
- **Fileless:** Sadece bilgisayarın belleğinde çalışan ve işletim sisteminin dosya sisteminde bir dosyası olmayan bu nedenle tespiti zor olan Malware tipidir. Kalıcılıkları olmamasına rağmen, nadir kapanan ve iyi korunan sistemlerde oldukça etkilidir. Genellikle aracı bir uygulama üzerinden kendilerini çalıştırırlar, örneğin Windows Powershell ya da Browser uygulamaları gibi.
- **APT:** Advanced Persisted Threat olarak bilinen ve doğrudan belirli bir hedefe odaklanmış saldırılar ve bu saldırıları yapan kişi veya organizasyonlara verilen isimdir. Bu tip saldırılar önceden detaylıca planlanmış ve

gerçekleştirilmesi uzun yıllar sürebilen saldırılardır. Bu saldırılar esnasında kullanılan Malwareler hedef sisteme özel geliştirilmiştir.

2.2 Malware Analizi Yaklaşımları

Gerek öğrenme tabanlı metodolojiler gerekse geleneksel yöntemlerin kullanacağı özellik çıkarımları için iki farklı yaklaşım metodolojisi bulunmaktadır. Bunlardan ilki, hedef yazılımı çalıştırmadan elde edilen özelliklerin çıkarılması, diğeri ise kontrollü ve izlenen bir ortamda ilgili yazılımın koşturularak, çalışma zamanında (runtime) oluşan metrik ve davranışsal özelliklerini çıkarmak üzerinedir. Statik ve Dinamik olarak adlandırılan bu yöntemlere ek olarak bu iki yöntemin birleştirildiği hibrit metodolojilerde bulunmaktadır.

2.2.1 Dinamik Analiz

Dinamik analiz, hedef programın kontrollü bir ortamda çalışmasını gözlemleyerek, çalıştığı ortam ile girdiği etkileşimler üzerinden özellik çıkarımı ve davranış analizi yapılmasıdır. Analiz sonrasında elde edilen özellikler, öğrenme tabanlı bir sınıflandırıcıya ya da geleneksel güvenlik yazılımlarına kural olarak eklenebilir. Bu özellikleri aşağıdaki şekilde özetleyebiliriz:

- **Dosya Sistemi ile Etkileşim:** Dosyalara erişme, yaratma, değiştirme ve silme işlemlerinin takip edilerek hassas ya da anormal erişimlerin takip edilmesidir.
- **İşletim Sistemi Konfigürasyon Dosyaları ile Etkileşim:** İşletim sisteminin sağlıklı ve düzgün şekilde çalışmasından sorumlu konfigürasyon dosyalarına erişim ve değiştirme işlemlerini takip etme. Buna örnek olarak Windows ortamı için Registry ayarları verilebilir.
- **Process ve Thread Aktiviteleri:** Hedef yazılımın kendi ve diğer process ve threadler ile olan etkileşimini takip ederek, "process injection" gibi anormal olayları tespit etmektir.
- **Network Aktiviteleri:** Hedef yazılımın hangi uzak sunuculara hangi protokollerle eriştiği ve gönderdiği veriler incelenerek yazılımın yetkisiz

noktalara hassas veri gönderip göndermediğini belirlemek.

- **Sistem ve API Çağrımları:** Hedef yazılımın İşletim Sistemi ile etkileşime girmek adına çağırdığı sistem çağrıları, bu çağrıların gerekliliği ve amacının takip edilmesi, API çağrımlarının hangi sıra ile ne amaçla kullanıldığının takip edilmesi ve son olarak sistem API çağrılarına yapılan modifikasyon ya da kancalama (hooking) işlemlerinin takip edilmesi

- **Memory Analizi:** Hedef yazılımın sistem belleğini nasıl kullandığı, yetkisiz yerlere kod yerleştirip yerleştirmediği ve özellikle şifrelenmiş ya da paketlenmiş zararlı kod parçalarını tespit etmek adına anormal memory kullanımların gözlemlenmesi.

- **Davranışsal Analiz:** Hedef yazılımın analizi atlatmak adına kullanabileceği kaçınma tekniklerinin incelenerek, zararlı kod parçasının aktivasyon parametrelerinin tespit edilmesi (belirli bir süre bekleme ya da kullanıcı ile etkileşim gibi) ek olarak hedef sistemin kaynaklarının ne derecede kullanıldığının tespit edilmesi

- **Komut Satırı ve Script Aktiviteleri:** Hedef yazılımın kendini gizlemek adına zararlı kod bloğunu bir başka uygulama üzerinden çalıştırıp çalıştırmadığının (örneğin Windows Powershell) incelenmesi.

- **Kernel Aktiviteleri:** Hedef yazılımın Kernel ile olan aktivitelerinin incelenmesi (örneğin sürücü (driver) yüklenmesi gibi).

Dinamik analiz programın çalışma zamanındaki özelliklerinin çıkartılmasına dayandığı için, incelenen Malware yazılımının kontrollü ve güvenli bir ortamda çalıştırılması gerekir. Bunun için farklı yöntemler bulunmaktadır. Bunlardan ilki Sandbox olarak bilinen sanal ve izole bir ortamda bu uygulamanın çalıştırılmasıdır. Farklı Sandbox metotları bulunmaktadır. Bunların bir kısmını modern işletim sistemleri, çeşitli sistem çağrılarını monitör ederek sağlarken, bir kısmı uygulamayı L2 (layer-2) ya da L3 (layer-3) bir sanal makinede çalıştırma üzerine dayalıdır. Bu sayede analiz ev sahibi sistemin enfekte olmasını engeller.

Diğer bir yöntem olan Debug ise özelleşmiş programların hedef yazılımı adım adım çalıştırmasına dayanır. Her adımda hedef sistemin "Architectural State" olarak bilinen, hedef sistemin o anki donanım ve yazılımın durumunun incelenmesine olanak tanır.

Emülasyon olarak bilinen yöntem ise analiz edilen yazılımın hedeflediği sistemi taklit eden özelleşmiş bir yazılımın üzerinde programın çalıştırılmasıdır. Buna QEMU'yu örnek verebiliriz.

Bahsedebileceğimiz bir diğer yöntemde Honeypot olarak bilinen gerçek ama tuzak sistemlerdir. Bu sistemlerde izleme out-of-band şekilde yapılır. Yani izlenen kaynaklar hedef işletim sistemi üzerinde doğrudan değil hedef işletim sisteminin kaynaklarının dışarıdan izlenmesi ile sağlanmaktadır. Günümüzde Honeypot sistemler, yeni Malwarelerin yakalanmasında önemli bir yöntemdir.

Fark edileceği üzere dinamik analiz yöntemleri bir zararlı yazılım hakkında potansiyel olarak oldukça detaylı bilgiler sunabilir. Fakat buna karşı zaman ve kaynak gereksinimi oldukça yüksektir. Dolayısıyla gerçek zamanlı ve düşük gecikme gerektiren tespit sistemlerinde kullanılmaları oldukça güçtür. Ek olarak Modern Malware, dinamik bir şekilde analiz edildiğini hissedebilecek etkin yöntemlere sahiptir. Proactive ve Reaktive yöntemler ile otomatize edilmiş dinamik analiz süreçleri atlatabilmektedir.

2.2.2 Statik Analiz

Statik analiz hedef programın çalıştırılmadan özellik çıkartılması ile yapılan bir analiz türüdür. Bu analiz türü dinamik analize kıyasla hızlı, az kaynak tüketen ve güvenli bir analiz türüdür. Bu özellikleri nedeniyle bu yaklaşım, otomatik olarak özellik çıkartan sistemlere daha uygundur. Statik analiz ile çıkarılan özellikleri aşağıdaki şekilde özetleyebiliriz:

- **Metadata Verileri:** Hedef yazılımın dosya seviyesinde sahip olduğu özelliklerdir. Bunlar çeşitli zaman damgaları, dosya ismi ve boyutu, hash imzaları varsa paketleme sırasında kullanılan imzalar gibi özelliklerdir.

Metadata, hedef ortama göre çeşitlilik gösterebilir. Örneğin Android işletim sistemi için ilgili yazılımın talep edeceği izinlerin bulunduğu Xml dosyası da bir metadata verisidir.

- **Dosya Yapısı ve Formatı:** Hedef yazılımın başlık (header) bilgisi (örneğin Windows yazılımları için PE header) ve hedef sistemin talep ettiği yapılar bütünüdür. (Örneğin PE bölümleri, .text, .data gibi kod ya da data verilerinin bulundukları ayrı bölümler)

- **String Analizi:** Yazılım içinde sabit şekilde bulunan ve insan tarafından okunabilen cümle ve betimler, URL veya IP adresleri ile karıştırılmış ama belirli bir örüntüye (patern) uyan sabit betimler (string) örnek verilebilir.

- **Kullanılan API ve DLL'ler:** Yazılım tarafından kullanılan ve import edilen çeşitli API'ler, dinamik linklenen kütüphaneler (DLL) gibi özelliklerdir.

- **Kod Analizi:** Yazılımın gerek Disassemble edilerek çıkartılan Assembly kodları, gerekse CFG olarak bilinen kod kontrol akışlarının çıkartılmasıdır.

- **Entropi ve İstatistiksel özellikler:** Dosya ya da bölge bazlı entropi ve varyans değerleri, byte ve işlem kodu(opcode), frekans değerleri, n-gram analizi, ilgili bölümlerin boyutları ve varyansları gibi istatistiksel değerlerdir.

- **Sezgisel Kural Analizi:** Hedef yazılımın kullandığı, şifreleme, paketleme, dinamik yükleme, kod karıştırma, yük (payload) şifreleme, kritik sistem çağrıları yapma gibi siber güvenlik uzmanlarının Malwarelar'da sıklıkla görülen çeşitli paternlerin tespit edilip kurallaştırılmasıdır.

Statik analiz yöntemleri hedef programı çalıştırmadığı için yazılımın dinamik analize karşı aldığı önlemlerden muaftır. Ayrıca bu analizde, dinamik analiz için alınan önlemler çeşitli paternler ile tespit edilebilmektedir. Yine de statik analiz bize hedef yazılımın çalışma zamanı hakkında tam bir davranışsal analiz raporu veremez. Benzer şekilde Malware statik analizi zorlaştıracak önlemler alabilir.

Bu iki yöntemi birleştiren hibrit tabanlı yaklaşımlarda mevcuttur. Bu sayede öğrenme tabanlı sınıflandırma modeli için seçilen özelliklerin kapsamı

genişletilerek daha doğru ve kararlı bir sınıflandırma yapılması amaçlanmıştır. Kısıtlı bir veri seti ile akademik çalışmalar özelinde iyi sonuçlar elde edilse de uygulanabilirlik ve ölçeklenebilirlik noktasında bu yöntemlerin gerçek hayatta bir karşılığı olup olmayacağı sorusu çoğu zaman göz ardı edilmiştir. Yine de teorik olarak bahsetmek gerekirse, hedef bir uygulama için hem statik hem de dinamik özellikleri çıkartıp harmanlayan bir otomasyon sistemi geliştirilmesi mümkündür.

2.3 Malware Yetkinlikleri ve Kaçınma (Evasion) Yöntemleri

Modern Malware tanımı yaparken, günümüzde Malware geliştiricileri ve buna karşı olarak güvenlik yazılımı geliştiricileri arasındaki mücadele nedeniyle Modern Malwarelerin proaktif ve reaktif korunma yöntemleri içerdiğinden bahsedilmişti. Günümüz Malware yazılımları hem geleneksel güvenlik yazılımlarını atlatarak hem de bahsettiğimiz analiz yöntemlerinden kaçınmaya çalışarak davranışını gizlemeye çalışmaktadır. Bu metodolojilerin bir kısmı doğrudan yazılımın imzasını değiştirecek şekilde basit yöntemler ile olabilirken, bir kısmı dinamik ve statik analizde kullanılan yöntemleri doğrudan hedef almaktadır.

Malware yetkinliklerini (capabilities) yüksek seviyede anlamlandırmak istersek, dinamik ve statik analiz için geliştirilmiş korunma yöntemlerini Anti-Analiz grubu altında toplayabiliriz. Bunun haricinde Malware, dağılma ve bulaşma yöntemi, aktivasyon şartları, kalıcılık (Persistence), tespit edilememezlik (görevini tamamladıktan sonra izlerini temizleme anlamıyla), komuta ve kontrol yapısı, yetki yükseltme gibi alt başlıklarda incelebilir.

Dağıtım ve bulaşma Malware yazılımının nasıl dağıtılacağı ve sistemlere nasıl bulaşacağı ile ilgili yetkinlikleridir. Dağılma yöntemi elden ele dolaşan USB bellekler ile olabileceği gibi, zararlı yazılımın çeşitli yöntemlerle imzasını değiştirerek yeni ve farklı kopyalarını üretip hedef sistemlere gönderen otomatize edilmiş sunucular gibi sofistike yöntemler ile de olabilir. Bulaşma yöntemleri hedef kullanıcıyı kandırarak kullanıcının Malware yazılımını farkında olmadan yüklemesinden, kullanıcı ile herhangi bir etkileşime ihtiyaç duymadan bulaşabilen yazılımlara kadar (zero-click Malware) çeşitlilik göstermektedir.

Aktivasyon şartları, bulaşan yazılımın zararlı faaliyetlere hangi aşamada başlayacağını belirleyen yetkinliklerdir. Bunlar, zamanlanmış bir tarih, kullanıcı etkileşimi, belirlenmiş bir olay (örneğin bir akıllı telefonda kayıtlı kişi sayısının 50'ye ulaşması gibi) ya da uzak bir sistem üzerinden tetikleme ile olabilir.

Kalıcılık yetkinliği iki farklı anlamda kullanılabilir. Bunlardan ilki hedef işletim sistemi yeniden kurulsa ya da ilgili sisteme hard-reset denilen fabrika ayarlarına döndürme işlemi yapılsa dahi Malware varlığının sürdürülebilmesidir. Bir diğeri de hedef sistem çalışır durumdayken ilgili zararlı yazılımın sonlandırılamaması anlamındadır.

Tespit edilememezlik ise, ilgili yazılımın görevini tamamladıktan sonra ya da dışarıdan bir komut veya olay sayesinde bulaştığı sistemde, izi bulanamayacak şekilde kendini sistemden temizlemesidir. Buna son zamanlarda popüler olan Pegasus isimli Malware örnek verilebilir.

2.3.1 Evasion Yöntemleri

Evasion metotları statik ve dinamik analiz için farklı yöntemler içerse de bunları Anti-Analiz başlığı altında inceleyebiliriz. Bunlardan bazılarından bahsetmek gerekirse:

- **Renaming:** Oldukça basit ama geleneksel sistemleri aldatmada başarılı bir yöntemdir. Yazılım kendi ismini ya da kod içerisinde bulunan değişkenlerinin ismini değiştirerek imzasını değiştirir.
- **Configuration and Metadata Alteration:** Yazılım imzasını değiştirmek adına bazı metadata, konfigürasyon ve header bilgilerini değiştirir. Benzer şekilde örneğin Android ortamı için uygulama her paketlendiğinde çeşitli metadata verileri değiştiği için yazılımın imzası değişmektedir
- **String Encryption:** Yazılım içinde kullanılan sabit ama belirleyici string, url, ip adresi gibi veriler şifrelenerek kullanılır. Şifreleme anahtarı değiştirilerek kendini çoğaltan zararlı yazılım aynı veriler için her seferinde farklı girdiler oluşturarak imzasını değiştirir ve önceden tespit edilmiş kuralları aşar.

- **Code Obfuscation:** Yazılımın kod yapısını ve lojik akışını çeşitli metotlarla değiştirerek kodu okunması zor hale getirmek amacıyla uygulanan tekniklerdir.

- **Packing & Code Encryption:** Yazılımda zararlı kod bloğunu gizlemek adına ilgili bölümün şifrelenmesi ya da paketlenerek kod dışı bir alana (payload) yazılmasıdır. Genel olarak literatürde "Packing" olarak bilinen bu yöntem çok katmanlı olabilir. Burada amaç zararlı kod bloğunu çalışma zamanında kademeli olarak çözerek (unpacking işlemi) kullanmaktır. Bu sayede otomatik statik ve dinamik analiz araçları normal şartlar altında unpacking işlemi yapmadan zararlı kod bloğuna ulaşamaz ve buradaki değerli özellikleri çıkaramaz. Fakat yapılan çalışmalarda, paketleme işleminde kullanılan ve "packer" olarak bilinen bu kütüphanelerin, statik analiz esnasında tespit edilebilecek şekilde kendilerine dair izler bırakabildikleri görülmüştür. Bu izler kullanılarak program çalıştırılmadan unpack edilebilmektedir [8]. Bununla beraber paketlenen veri üzerinde belirli paternler oluşabilmekte ve bu paternler öğrenme tabanlı sınıflandırıcılar tarafından yakalanabilmektedir.

- **Code Insertation:** Zararlı yazılıma gereksiz kod ekleyerek ya da zararlı kodu büyük bir zararsız bir kod bloğunun içine saklamak sıklıkla kullanılan bir yöntemdir. Derlenmiş ve çalışmaya hazır bir programa, programın çalışma yapısını bozmadan önüne ya da arkasına alakasız bytelar eklenebilir. Bununla beraber kod içerisine işlevsiz (dummy) ya da amaçsız (dead) kod blokları eklenerek gerek dinamik analiz gerekse statik analiz yöntemleri kandırılmaya çalışılır. Bu işlemler özellikle Assembly dilinin birkaç istisna haricinde CFG (Context Free Grammar) olması özelliğini kullanarak, hiçbir iş yapmayan ve tamamlandığında donanımı ve yazılımı, başladığı Architectural State durumuna döndüren yapay kod üretici otomatlar ile sağlanabilir.

- **Call Indirection, Reflection, Self-Modifying Code:** Kod akışının değerleri çalışma zamanında belirlenen bazı değişkenlere göre dinamik olarak değişmesi nedeniyle, sadece çalışma zamanında belirli olan akışların doğru tespiti için, analizciyi kodu çalıştırmaya zorlayan yöntemler bütünüdür.

- **Dynamic Loading:** Zararlı kod bloğunun çalışma anında uzak sunucudan ya

da hedef bilgisayardaki farklı bir dosyadan yüklenmesini sağlayan yöntemdir. Statik analiz sırasında dışarıdan yüklenecek zararlı kod, çalışma anından önce bulunmayacağı için ayırt edici özelliklerin dinamik analiz yapılmadan elde edilmesi engellenir. Yine de belirli statik analiz özelliklerin bir araya gelmesi ile (istatistiksel özellikler ile dinamik yüklemeyi yapan kod bloğu gibi) yazılımın bu davranışı hakkında bilgi çıkarımı yapılabilir.

- **Anti-Disassembly:** Anti-Disassembly'den bahsetmeden önce "Decompiling" ve "Disassembling" farkından bahsetmek gerekmektedir. Disassemble, makine kodunun, çalıştığı ortama göre insan tarafından okunabilen en alt seviye programlama diline dönüştürülmesidir. Makine kodu, doğrudan ilgili Assembly dilinin yönerge (instruction) (opcode, operand vb alanları içeren yönergeler) dönüştürülür. Decompiling ise çeşitli araç ve yöntemler ile kodun yüksek seviye dildeki haline getirilme işlemidir. Compile aşamasında compiler'ın türü, ayarları ve yaptığı optimizasyonlara ek olarak code obfuscation metotları nedeniyle decompile işlemi her zaman orijinal koda geri döndürülemez. Bu fark anlaşılır olduktan sonra Anti-Disassembly metotlarından bahsetmek gerekirse, hedef ortam için gerek mevcut disassembler araçlarındaki yazılım hatalarından faydalanarak gerekse çeşitli teknikler ile program akışını bozmayacak şekilde yazılıma hatalı ya da yasak opcodelar ekleyerek disassemble programını hata vermeye zorlayarak süreci bozmaya amacı güderler.

- **Anti-Debugging:** Malware yazılımının Debug edildiğini anlamaya yönelik aldığı önlemlerdir. Debug işlemi, işletim sistemi çağrıları ve çeşitli bayraklar ile sağlanmaktadır. Malware bu işletim sistemi çağrılarını takip edebilir, bayrakları tespit edebilir ya da kod bloklarının birbiri ardına çalışma süresini ölçerek Debug edildiğini anlayabilir ve davranışını değiştirebilir.

- **Anti-Sandbox:** Programın, gözetimli ya da sanal bir ortamda çalıştırıldığını anlamasını sağlayan metotlardır. Sandbox ya da sanal ortamlar, ortamın sanal olduğuna dair doğrudan ya da dolaylı olarak birtakım izler bırakabilirler. Bu izler donanımsal özellikler, yazılım özellikleri, ya da "side channel" olarak bilinen kaynak sistemden istemsizce sızdırılan bazı önemli bilgiler olabilir.

Bu özellikler tespit edildiğinde zararlı yazılım davranışını değiştirerek zararsız bir uygulama gibi davranarak analizi atlatır.

- **Event-Waiting Evasion:** Malware zararlı faaliyetlere başlamadan önce belirli bir zamanı, olayı ya da etkileşimi bekleyebilir. Bu da zararlı davranışların otomatik analiz yapan bir sistemin analiz zamanı içinde tetiklenmeyerek davranışını gizlemesini sağlayacaktır. Bu tarz yaklaşımlar deneyim siber güvenlik uzmanları tarafından kodun normal akışına müdahale edilerek tespit edilebilir.

2.4 Malware Araştırmalarında Zorluklar ve Problemler

Literatür bölümüne geçmeden önce özellikle akademik alanda Malware analizi konusunda çalışmaların ve araştırmacıların yaşadığı zorluklardan bahsedilmelidir.

Öncelikle araştırmacıların standart, kapsayıcı ve güncel veri setlerine erişimleri bulunmamaktadır. Literatürde mevcut ve açık erişimde olan veri setleri, çok eski ve Modern Malware metotlarını içermeyen veri setleridir. Bununla beraber bu veri setleri oldukça küçük ve gerçek dünyaya kıyasla kapsamı çok kısıtlıdır. Diğer bir problem ise erişilebilen veri setlerinin Benign (zararsız) örnekler içermemesidir. Benign örnekler fikri mülkiyet ihlali korkusu nedeniyle araştırmacılar tarafından paylaşılmamaktadır. Halka açık olan veri setlerinde bir diğer problem ise ilgili yazılımların yalın (raw) hallerinin paylaşılmamasıdır. Bu veri setleri, araştırmacı tarafından önceden belirlenmiş bazı özellikler çıkartılarak, yazılımın çalıştırılabilir versiyonuna geri döndürülemeyecek şekilde sunulmaktadır. Bütün bunlar göz önüne alındığında, araştırmacılar kendi veri setini oluşturmaya ya da bulabildikleri farklı veri setlerini birleştirerek çalışma yapmak durumundadır.

Veri seti oluşturma aşamasında uygulanan çeşitli yaklaşımlar mevcuttur. Malware örnekleri çeşitli açık kaynak (vxshare, vxunderground gibi) ortamlarından temin edilebilirken, Benign örnekler sadece ve sadece araştırmacıların kendi çabaları ile elde edilebilmektedir. Bu veri setini doğrulamak ve temizlemek için VirusTotal benzeri üçüncü parti yazılımlar kullanılmaktadır. Fakat bu işlem, veri setini "kolay" hale getirecektir. Bunun sebebi bu doğrulama işleminin zaten var olan geleneksel

güvenlik yazılımlarının sınamasından geçirilerek yapılmasıdır. Geleneksel güvenlik yazılımlarının, Modern Malware yazılımlara karşı yetersiz olduğu hipotezi altında, bu yazılımların kolayca doğrulayabildiği uygulamalar ile zor bir veri seti elde edilemez. Bir diğer yaklaşım ise hedef işletim sisteminde temiz bir kurulum yapıldıktan sonra işletim sistemi içindeki çalıştırılabilir dosyaları zararsız kabul ederek toplamaktır. Fakat burada toplanan yazılımlar aynı kod blokları, aynı kütüphaneler ve aynı frameworkler kullanılarak yazıldığı için çok açık benzerlikler içermekte, dolayısıyla bu benzerlikler sınıflandırıcıları aldatmaktadır.

Bir diğer problem toplanan veri setlerindeki temsil ve denge sorunudur. Mevcut veri setleri bir Malware ailesi için binlerce örnek içerebilirken, başka bir aile için birkaç tane içerebilir. Bu sınıflandırıcı modellerin ön yargı (bias) oluşturmasına sebep olacaktır. Bununla beraber bir diğer problem temsil edilen örneklerin birbirlerinin çok benzeri olup kolaylıkla gruplanabilir olmasıdır. Bunun nedeni örneklerin aslında tekil bir örnekten sadece imza değiştirme amacıyla çok küçük değişiklikler yapılarak oluşturulmuş klonlar olmasıdır.

Özetle az gürültülü, çeşitliliği ve kapsamı geniş olan ve Modern Malware metotlarını içeren dengeli bir referans veri seti bulmak ya da oluşturmak oldukça zordur. Bu tarz veri setlerine sadece çeşitli güvenlik firmaları ile organik bağı olan araştırmacılar erişebilmektedir.

Modellerin Malware sınıflandırma problemini genelleştirebilme yeteneği zorluklardan bir diğeridir. Bu genelleştirme bir programın amacını ve niyetini anlama gibi yüksek seviye çıkarımlar yapılmasını gerektirmektedir. Her ne kadar her gün birbirinden farklı ve yeni zararlı yazılımlar ortaya çıksa da bu yazılımların amaçları sınırlı sayıdadır. Bu amaçlar özelinde tespit yapabilen sınıflandırıcılar, problemi genelleştirmiş olacak ve yeni zararlı yazılımlara karşı daha gürbüz olacaktır. Problemi genelleştirebilmiş bir sınıflandırıcı modelin, değerlendirme aşamasında hiç görmediği bir aileye ait örneğe karşı gürbüz olması beklenir.

Karşılaşılan bir diğer sorun, önerilen modellerin gerçek hayata uygulanabilirliği ve ölçeklenebilirliği konusudur. Akademik çalışmalar genellikle önerilen sistemin performans yükünden çok başarımına odaklanmışlardır. Kapsamlı akademik

çalışmalar nadiren 100 bin örnekten fazla örnek ile çalışırken, gerçek dünyada kabul edilebilir bir tespit sistemi on milyonlarca örnek ile eğitilmelidir. Bununla beraber gerçek dünyada Malware tespiti konusunda False-Positive ve False-Negative değerleri çok önemlidir. Akademik çalışma için kabul edilebilir FPR ve FNR değerleri gerçek dünya için kabul edilebilir olmayabilir. Bunun başlıca sebeplerinden bir tanesi, çalışmada kullanılan veri seti boyutu ile gerçek dünyadaki örnek sayısı arasındaki uçurumdur. Bir diğer performans kriteri, sınıflandırma süresi ve ihtiyaç duyulan kaynaktır. Akademik çalışmalar az örnekle başarım odaklı olduğu için oldukça karmaşık ve hesaplama yükü fazla modeller kullanabilir, fazlaca kaynak tüketen ön işleme adımları içerebilir. Fakat bu modeller gerçek zamanlı sistemler ve hedef cihazların işlem kapasitesi ve kriterleri göz önüne alındığında uygulanabilir olmayacaktır. (Örneğin korunmak istenen cihaz basit bir IoT cihazı da olabilir)

Son olarak "Adversarial Attacks" olarak bilinen, hedef derin öğrenme modelinin çalışma mantığını analiz edip, onu aldatmaya yönelik yapılan girdilere karşı modellerin gürbüzlüğünün ölçülmesi gerekmektedir.

3
LİTERATÜR ARAŞTIRMASI

Bu çalışmada literatür bölümü erken dönem çalışmalar ve yakın dönem çalışmalar olarak ayrılmıştır. Erken dönem klasik doğal dil işleme yöntemlerinin kullanıldığı, yakın dönem çalışmalar ise gerek araştırmacıların yeterli kaynağa erişimi gerekse bu dönemde yapay zeka çalışmalarının hızlanması ile daha kapsamlı modellerin oluşturulduğu dönemi kapsamaktadır. Bununla beraber çalışmamız statik analiz metodolojisini benimsediğinden erken dönem çalışmalar genellikle statik analiz temelinde yapılan çalışmaları konu almaktadır. Ayrıca çalışmalarda kullanılan veri setleri erişebilir olmadığından bu çalışmada önerilen metodolojiler, ilgili çalışmalardaki veri setleri ile test edilememiştir.

3.1 Erken Dönem Yapılan Çalışmalar

Moskovitch ve arkadaşları [9], Windows çalıştırılabilir dosyalarını disassemble ederek opcodelar üzerinde n-gram analizi yapmışlardır. Burada Term-Frequency ve Fisher Score kullanarak, 2-gramlar üzerinde özellik seçerek ağaç tabanlı bir sınıflandırıcıya sokmuşlardır. Bu çalışma 5677 Malware ve 20146 Benign örnekten oluşan veri setinde %94,43 lük doğruluk rapor etmiştir.

McLaughlin ve diğer araştırmacılar [10], Opcodelardan elde edilen kelime vektörlerine dayalı bir yöntem önermişlerdir. Öncelikle opcodelar üzerinden 3-gram özellik çıkarımı yapılmış, ardından bu 3-gram özellikler kelime vektörlerine dönüştürülerek CNN [11] sınıflandırıcıya sokulmuştur. Bu yöntem 24 bin Malware, 24 bin Benign içeren bir veri setinde %69 doğruluk vermiştir.

Karbab ve arkadaşları [12], API çağrım sekanslarını çıkartarak, bunlar üzerinden

kelime vektörleri elde etmişlerdir. Bu kelime vektörleri daha sonra CNN tabanlı bir sınıflandırıcı mimari ile eğitilerek 33 bin Malware 37 bin Benign içeren veri setinde %95 doğruluk oranı vermiştir.

Awad ve arkadaşları [13], word2vec [14] özellik çıkarma yöntemlerinde kullanılan Skip-Gram ve CBOW algoritmalarının etkinliğini araştırmak üzerine 9 farklı aile sınıfına ait 10 bin Windows Malware örneği üzerinde çalışma yapmışlardır. Opcodelar üzerinden çıkardıkları kelime vektörlerini ağaç tabanlı sınıflandırıcıya sokarak CBOW için %98, Skip-Gram için %76 başarım elde etmişledir. Bu çalışma dönemin etkili Doğal Dil İşleme metotlarının programlama dillerinde de etkili olduğunu göstermiştir.

Xu ve arkadaşları [15], metadata verileri ve opcode özellik çıkarımı kullanan katmanlı bir sınıflandırıcı yöntemi önermişlerdir. Metadata verileri içeren XML özellik vektörü bir MLP ile sınıflandırılmıştır. İkinci katman, opcodeların 15 farklı gruba ayrılmasının ardından kelime vektörlerinin çıkartılarak yine MLP tabanlı bir sınıflandırıcı da kullanmaktadır. İki sistemin birleşimiyle oluşan tespit sistemi 62 bin Malware 47 bin Benign örnek içeren veri setinde %97,74 doğruluğa ulaşmıştır.

Yousefi-Azar [16] ve diğer araştırmacılar, McLaughlin'e ait çalışmaya benzer şekilde opcodelardan önce n-gram özelliklerinin çıkarılması sonrasında ise n-gram özelliklerinden kelime vektörleri çıkartarak ML tabanlı sınıflandırıcılar ile deneyler yapmıştır. Farklı olarak, kullandığı veri seti PDF üzerinden bulaşan zararlı yazılımları içeren çalışma %92 ile %98 arasında değişen başarımlara ulaşmıştır.

Ye ve arkadaşları [17], graf tabanlı ve gerçek zamanlı bir tespit algoritması çalışması yapmışlardır. Bu çalışmaya göre öncelikle incelenen Android örneklerinden API çağrım grafları çıkartılmış, daha sonra bu çağrım grafları arasındaki ilişkileri inceleyerek yüksek seviyeli özellik çıkarımı yapmışlardır. Heterojen Graf Ağı adını verdikleri özellikleri MLP tabanlı bir sınıflandırıcı ile eğiterek %99 başarı elde etmişlerdir.

Zhang ve diğer araştırmacılar [18] kümeleme tabanlı bir metodoloji önermişledir. Bu metodolojide kod analizi, metadata verileri ve zayıf etiket olarak ta mevcut Anti-Malware uygulamalarının sonuçlarını ekleyerek veri seti oluşturmuşlardır.

MLP tabanlı bir sınıflandırıcı oluşturulan veri seti ile eğitilerek %91 başarıma ulaşmıştır. Bu çalışmada Modern Malware özeliklerine sahip örnekler çalışma kapsamı dışında bırakılmıştır.

Ge ve arkadaşları [19], uygulamaların davranışlarını temsil eden fonksiyon çağrısı grafikleri (FCG'ler) kullanan ve FCG'lerden, uygulamaların yapısal anlamını otomatik olarak öğrenmek için grafik kernelleri kullanan AMDroid adlı bir yaklaşım önermiştir. AMDroid'in performansı SVM [20] sınıflandırıcısı ile Genome veri seti üzerinde eğitilerek %97.49 başarıma ulaşmıştır.

Pektaş ve diğer araştırmacılar [21], örneklerde ki API çağrımlarını çıkardıktan sonra uygulamanın takip edebileceği mümkün olan bütün API sekansları çıkartıp bunlar üzerinden graf tabanlı özellikler çıkartmışlardır. Bu graf tabanlı özelikler MLP tabanlı bir sınıflandırıcıya beslendikten sonra oluşan çıktılar benzerlik tabanlı algoritmalara tabi tutularak %98.86 son sınıflandırma başarımı elde edilmiştir.

Tablo 3.1 Erken dönem literatür araştırması Özeti

Araştırma	Acc. (%)	Veri Seti	Metotlar	Özellikler
[9]	94,4	26.093	Boosted DT	Opcode n-grams
[10]	69	48.000	CNN	Opcode Kelime Vektörleri
[10]	98	2.213	CNN	Opcode Kelime Vektörleri
[12]	96	70.693	CNN	Kelime Vektörleri
[13]	98	10.868	KNN	API çağrımları + Opcode Kelime Vektörleri (CBOW)
[15]	97,7	110.438	MLP + LSTM	APK Metadata Kelime Vektörleri
[16]	98	19.979	XGBoost	Skip-Gram, Opcode n-grams
[22]	97	59.749	SVM	HIN metagraph2vec
[23]	98	-	MNN	Skip-Gram
[17]	99	190.696	HG Learning	API çağrım grafı
[18]	91	14.416	Clustering + DNN	Skip-Gram, CBOW
[19]	97,49	2.520	SVM	Skip-Gram
[21]	98,86	58.139	DNN	API çağrım grafı + Skip-Gram

3.2 Yaklaşımlara Göre Gruplanmış Yakın Dönem Çalışmalar

Yakın dönem yapılan çalışmalar gerek veriye gerekse kompleks modellerin çalışması için gerekli donanım kaynaklarına ulaşımın kolaylaşmasından ötürü çoğunlukla görüntü ve doğal dil işleme derin öğrenme modellerini kullanmaktadır. Bununla beraber bu çalışmaları, özellik çıkarımındaki kullanılan analiz yaklaşımına göre, model eğitiminde kullanılan özellik tiplerine göre ve son olarak kullanılan sınıflandırıcı modellerine göre gruplanabilir.

3.2.1 Analiz Yaklaşımına Göre Yakın Dönem Çalışmalar

Önceki bölümlerde bahsedildiği üzere Malware analizi metotları statik veya dinamik özellikler üzerinden çıkarım yapar. Statik analiz, sadece hedef dosyaya ve bir takım metadata verilerine erişim olduğu durumda, dosyayı çalıştırmadan elde edilen özelliklere dayanır. Dolayısıyla ilgili yazılımın gerçek zamanlı davranışı hakkında bilgi sahibi olunmaz. Hedef dosya üzerinde yapılacak özellik çıkarımları farklı temsil seviyelerinde olabilir. Örneğin dosyayı byte seviyesinde işleyebilir ya da bir disassembler kullanarak uygulama yapısı ve kodları hakkında bilgi sahibi olunabilir. Jiang ve arkadaşları [24], sınıflandırıcılara beslemeden önce opcode sekanslarını hassas API çağrımları içerip içermediklerine göre öncelikli ya da öncelikli olmayan diye ayırmıştır.

Buna karşılık dinamik analiz, zararlı yazılımın çalışma zamanındaki davranışlarını incelediği için daha kapsamlı özellikler çıkartabilir. Dinamik analizle, ağ kullanımı, dosya erişimleri, sistem çağrıları ve çalışma zamanı metrikleri gibi yararlı yeni özellikler elde edilebilmektedir. Buna bir örnek olarak Catal ve arkadaşlarının [25] yaptığı çalışma verilebilir. Bu çalışmada, çalışma zamanında gözlemlenen API çağrımlarının grafları oluşturularak Node2Vec adını verdikleri algoritma ile vektörler oluşturulmuştur. Daha sonra bu vektörler Generative Adversarial Networks [26] (GAN) ve Graph Convolutional Networks [27] (GCN) modellerini besleyerek sınıflandırma yapmıştır.

Bazı çalışmalar, modelleme gücünü artırmak için hibrit bir yaklaşımı takip eder. Aktaş ve Şen [28], statik ve dinamik özelliklerin bir araya getirildiği hibrit bir analiz

yöntemi önermişledir.

3.2.2 Özellik Biçimlerine Göre Yakın Dönem Çalışmalar

Statik veya dinamik yaklaşımların her ikisinde de farklı "input modalities" (girdi biçimleri) kullanabilmektedir. Yan ve arkadaşları [29], Android APK dosyalarından çıkardıkları opcode tabanlı bir analiz yapmıştır. Çıkartılan opcode sekansları üzerinden belirli kalıplar çıkartılarak TF-IDF algoritması ile az bilgi içeren kalıpları tespit etmiş ve veri setinden çıkarmışlardır. Özellikle LSTM (Long-Short Term Memory) [30] tabanlı Auto-Encoder modellerinde gerekli olan sabit girdi uzunluğu problemini aşmak içinde, girdi uzunluklarını uygulama içinde bulunan fonksiyon uzunları ile sınırlandırarak standartlaştırmıştır.

Anderson ve Roth [31] , Windows "Portable Executable" (PE) dosyalarından uzmanlar tarafından belirlenen yaygın istatistiksel özellikleri çıkartarak adını EMBER verdikleri özellik setini oluşturmuştur. Daha sonra çeşitli ağaç tabanlı sınıflandırıcıları bu özellik seti kullanılarak elde edilmiş veri seti ile eğiterek başarılı sonuçlar elde etmişlerdir.

Raju ve Wang [32], PE dosyaları üzerinde statik analiz ile Malware yetkinliklerini çıkartan CAPA [33] isimli bir yardımcı program ile API yüklemeleri, entropi ve byte dağılımları gibi önceden belirlenmiş özellikleri çıkartmışlardır. Bu özellikler, EMBER özellikleri ile birleştirilerek sınıflandırıcı başarımı arttırılmıştır.

Kadri ve arkadaşları [34], opcode tabanlı modelleme ile byte tabanlı modellemeyi kıyaslamış ve opcode tabanlı modellemenin daha başarılı olduğunu göstermiştir.

Vinayakumar ve diğer araştırmacılar [35], byte tabanlı modelleme ile EMBER özelliklerini birleştirerek başarımı arttırmayı denemiştir.

Pektaş ve Acarman [36], statik ve dinamik analizi birleştirmek adına, opcode sekansları ve çalışma zamanındaki API çağrım sekanslarını birleştirerek hibrit bir özellik vektörü oluşturmuşlardır.

Saxe ve Berlin [37], byte entropi histogramı, PE header verileri ve diğer byte tabanlı özellikleri birleştirerek karmaşık input vektörleri oluşturmuş ve sınıflandırma

başarımını yükseltmişlerdir.

Raff ve arkadaşları [38] doğrudan opcode vektörleri kullanmak yerine opcode sekanslarından 6-gram'a kadar özellik vektörleri çıkarmıştır. Bu yaklaşım oldukça güçlü olmasına rağmen memory ve disk alanı gereksiniminden ötürü 6-gram yukarısında bir n-gram analizi mümkün değildir.

Yazılımları opcode tabanlı özellikler ile temsil etmede çeşitli varyasyonlar gözlemlenmiştir. Ghezelbigloo ve VafaeiJahan [39] rollerine göre opcode gruplama yapmıştır. Bu sayede çok daha az bir sözlük büyüklüğü ile sınıflandırma yapabilmişlerdir. Zhang ve arkadaşları [40] sistem çağrı sekanslarını opcode sekansları ile birleştirerek hibrit özellik vektörü elde etmiştir. Benzer bir çalışmayı Yesir ve Soğukpınar [41] dinamik analizde tespit edilen API çağrıları üzerinden yapmıştır. Lucas ve diğer araştırmacılar [42], literatürde state-of-art olarak kabul gören MalConv [43] isimli konvolüsyonel modelin hassasiyetini ölçmek adına opcode sekanslarına rastgele değişiklikler yaparak, ilgili modelin, bu gibi değişikliklere karşı oldukça hassas olduğunu raporlamıştır.

Yakın dönemde görüntü işleme modellerinin başarımı nedeniyle oluşan motivasyon ile program binary dosyalarını 2 boyutlu görüntüye çevirerek temsil eden çalışmalar yapılmıştır [44] [45] [46] [47]. Örneğin Xu ve arkadaşları [48] SeqNet adında yazılım binary dosyalarını 2 boyutlu görüntülere çeviren bir metodoloji önermiştir.

Bensaud ve arkadaşları [49] 2 boyutlu görüntüye çevirdikleri binary dosyalarını literatürde çok bilinen VGG16 [50] ve Resnet50 [51] gibi modeller kullanarak sınıflandırmaya çalışmışlardır. Benzer çalışmalar Adversarial networklere uygulanmıştır [44]. Basit Convolutional Neural Network (CNN) yapıları dahi oldukça başarılı sonuçlar rapor etmiştir [52]. Bu çalışmalar genellikle yazılımların yalın (raw) binaryleri üzerinden yapılmasına rağmen Xing ve arkadaşları [53] önce opcode çıkarımı yapıp daha sonra 2 boyutlu görüntü dönüşümüne tabi tutarak sınıflandırma yapmıştır.

Bilgisayar programlarının 2 boyutlu bir görüntüye dönüştürülmesi, standart görüntü sınıflandırma modellerinin kullanılabilmesini sağlasa da bilgisayar programlarının neden bir görüntü olarak ele alınması gerektiğine dair net bir kanıt yoktur.

Bilgisayar programındaki bir byte, kod içindeki konumuna çok bağımlıdır. Ancak, görüntülerde bir piksel, bir renk veya yoğunluk bilgisini yakalar. Bilgisayar programlarını iki boyutlu olarak temsil ettiğimizde, mutlaka normalde geçerli olmayacak bazı yatay ve dikey ilişkileri hesaba katmış oluruz. Bu gibi geçerliliği olmayan ilişkilerin model eğitiminden önce filtrelenmesi gerekmektedir.

3.2.3 Kullanılan Modellere Göre Yakın Dönem Çalışmalar

Malware ya da daha genel olarak bakarsak bir programlama dilini modelleme probleminin birçok farklı yönü olduğu için bu görevi yerine getirmek adına çeşitli kompleks mimariler kullanılmıştır. Bu mimarilerden bazıları günümüz büyük dil modellerinin kullandığı yöntemleri kullanmaktır. Örneğin Dang ve arkadaşları [54] sınıflandırıcı model olarak LSTM [30] kullanmaktadır.

Bu yaklaşımların bir alt kümesi, büyük dil modellerinin (LLM'ler) dahil edilmesini önermektedir. Balıkçıoğlu ve arkadaşları [55] Android opcodelarını modellemek için bir LSTM yöntemi önermiştir. Önerdikleri yöntem doğrudan opcode sekanslarını kullanmak yerine fonksiyon, sınıf ve dosya sevilerinde opcode sekanslarını gruplayarak sınıflandırma yapmışlardır. Bu gruplama modeli Android gibi Object Oriented Programming (OOP) tabanlı ve iyi tanımlanmış yapılarda mümkün olsa da her ortam için uygulanabilir değildir. Demirci ve arkadaşları [56] önceden eğitilmiş büyük dil modelleri üzerinde (GPT-2 [57], distilBERT [58], Stacked-BiLSTM) opcode sekansları kullanarak fine-tuning işlemi ile sistematik bir karşılaştırma yapmıştır. Ding ve arkadaşları [59], veri setlerinin dengesiz olduğu durumlarda, zararlı yazılımların ailelerinin de sınıflandırıldığı çok sınıflı modelleme üzerine yoğunlaşmıştır. Önerdikleri model, self-attention [60] mekanizması ile yüksek entropi içeren bölgeleri tespit ederek sınıflandırma başarısını arttırmıştır.

Her ne kadar konvolüsyonel yaklaşımlar Malware modellemede etkili olsa da bazı durumlara karşı zafiyet gösterebilirler. Bu bağlamda Demetrie ve arkadaşları [61] CNN tabanlı modellerin hangi tarz Adversarial saldırılara karşı zafiyeti olduğu üzerine çalışma yapmışlardır. Kolosnjaji ve diğer araştırmacılar [62] basit konvolüsyonel yaklaşımların Malware sınıflandırmasındaki zayıflıklarını vurgulamışlardır. Uygulamaların Header alanlarında yapılan ufak değişiklikler

sayesinde MalConv modelini kandırmaya çalışmışlardır. Çalışmalarında hedef programların yüzde birinden daha azını oluşturacak şekilde yapılan değişiklikler ile MalConv modelinin test örneklerinin %50'sinden fazlasını yanlış sınıflandırdığı raporlamışlardır. Suciu ve arkadaşları [63], MalConv mimarisinin hassasiyetini Adversarial saldırılar tasarlayarak araştırmış ve bu mimarinin özellik vektöründe konumsal bilgileri (positional encoding) iyi bir şekilde kodlayamadığını dolayısıyla byte ya da opcode tabanlı sekanslara rastgele ekleme yaparak giriş vektörünün değiştirildiği saldırılara karşı savunmasız olduğunu iddia etmişlerdir. Son olarak Song ve arkadaşları [64] Reinforcement Learning [65] kullanan ve Adversarial örnekler üreten bir framework geliştirerek MalConv ve EMBER modellerine saldırılar düzenlemiştir.

Lu ve arkadaşları [66], düşük gecikmeli sınıflandırmaya odaklanan basit ve yüzeysel self-attention transformer modeli önermiştir. Programları hem tek boyutlu dizi hem de 2 boyutlu görüntü olarak temsil ederek modellerini eğitmişlerdir. MalConv, CNN-BiLSTM ve CNN-BiGRU [67] gibi modeller kullanılarak kapsamlı bir karşılaştırma yapmışlardır.

Amin ve arkadaşları [68], CNN [69], DAE [70], DBN [71], RNN [72], LSTM [30] ve BiLSTM [73] gibi çeşitli derin öğrenme mimarilerini karşılaştıran bir değerlendirme yapmış ve bu mimarilerin Android Malware sınıflandırması üzerindeki performansını ve zero-day (sıfırıncı gün) değerlendirmeleri de içerecek bir şekilde rapor etmişlerdir.

Malware sınıflandırma konusunda kapsamlı bir etüt ile ilgilenen okuyucular için, öğrenmeye dayalı yaklaşımlarla ilgili birçok yakın dönem çalışmanın taksonomik bir şekilde ele alındığı [74] çalışmasını incelemelerini tavsiye etmekteyiz.

4
ÖNERİLEN YÖNTEM

Önerdiğimiz metodoloji, programlama dilleri ile doğal diller arasındaki benzerlikleri kullanıp, farklılıkları dikkate alarak Malware sınıflandırma problemini genelleştiren ve indirgeyen, iyi tasarlanmış uçtan uca bir sınıflandırıcı model oluşturmak ve bu modeli gerçek dünya metriklerine göre test etmek üzerinedir. Çalışmamızda öncelikle doğal dil işlemede kullanılan yöntemlerin, programlama dillerine uyarlanabilir olduğu gösteren makine öğrenmesi tabanlı bir model önerilmiş, sonrasında ise ilişkili ama az güvenilir davranışsal hedef değişkenlerini konvolüsyon modellerine tanıtarak, gerçek dünya sorunlarına karşı gürbüz, etkili ve uçtan uca bir derin öğrenme tabanlı sınıflandırıcı önerilmiştir. Önerdiğimiz modeller gerçekçi bir Zero-Day (sıfırıncı gün) testi ile çeşitli muhalif (adversarial) saldırılara karşıda test edilmiştir. Literatürde kullanılan en başarılı yöntemler, oluşturulan güncel veri setlerimiz ile test edildikten sonra Gated-CNN ve CNN-BiLSTM tabanlı iki benchmark modeli ile önerdiğimiz Apk2vec [75] ve TRCONV [76] karşılaştırmalı olarak test edilmiştir. Benchmark modellerinin seçiminde literatürde kullanılanılan yöntemler karşılaştırmalı olarak denenmiş (self-attention, Stacked-BiLSTM, DAE, LSTM-AutoEncoder, Hiyerarşik CNN, Multi-Modal CNN gibi) eğitim süresi, anlamlı başarım farkı, kaynak ihtiyacı gibi metrikler arasından en başarılı modeller seçilmiştir. Seçilen modellerin, gerçek dünyada karşılaşılan problemlere karşı da başarımı ölçülmüştür. Modellerin verdiği kararları nasıl verdiği üzerine temel bir çalışmada yapılarak elde edilen sonuçlar paylaşılmıştır.

4.1 Problem

Önceki bölümlerde nitelik ve nicelik olarak evrim geçirmiş Modern Malwarelere karşı neden öğrenme tabanlı sistemler ile mücadele etmemiz gerektiği anlatılmıştır. Bu bölümde bu hipotezimizden yola çıkarak bu görev için karşılaşılan problemlerden bahsedilecektir.

4.1.1 Hipotez

Hipotezimiz, programlama dillerinin doğal diller ile benzerlikleri nedeniyle başarılı doğal dil işleme metotlarının programlama dillerine uygulanarak uygulama sınıflandırma görevinin yerine getirilebileceği üzerinedir. Bu hipotez ile yazılım aslında belirli bir dilde yazılmış belge olarak ifade edilecek ve kapsamlı bir siber güvenlik operasyonundan, belge sınıflandırma görevine indirgenecektir. Malware sınıflandırmada hayal edilen yöntem, doğal dillerde metin ve bağlam üzerinden yapılan anlamsal analiz gibi yüksek seviye çıkarımları, yazılımın kodları üzerinden yaparak, yazılımın niyetini anlayan genelleşmiş sistemler tasarlamaktır.

4.1.2 Programlama Dilleri Hakkında

Bu noktada hangi programlama dilinin seçileceği konusu sorulacak ilk sorudur. Günümüzde yazılımlar nadiren doğrudan alt seviye programlama dili ile yazılmaktadır. Alt seviye programlama dili yazılımın çalışacağı ortam dilinin (makine kodu) bir üst seviyesinde yer alan ve insanlar tarafından okunabilen ilk seviyedir. Buna genel olarak Assembly denilmektedir. Her ortam için Assembly dili farklılıklar gösterse de makine dili ile birebir eşleştiği için elde edilmesi hızlı ve oldukça kolaydır. Bunun aksine, derlenmiş bir programı yazıldığı yüksek seviye dile geri çevirmek için programın "Decompile" işlemine sokulması gerekmektedir. Her ne kadar üst seviye diller anlamsal olarak daha kıymetli veriler içerse de Decompile işlemi oldukça karmaşık ve pahalı bir işlem olup, günümüz derleyicilerin geri döndürülemez şekilde yaptığı optimizasyonları düzeltemez. Bu sebeple araştırmacılar ilk olarak Assembly diline yönlenmiştir. Assembly dilinin bir artısı, işlemciler farklı mimarilerde tasarlanmış olsa da komut satırı mimarisi (instruction set architecture, ISA) yapılarının birbirine benzer olmasıdır. Burada

RISC ve CISC kavramlarına değinmeyeceğiz fakat, görev olarak temelde yapılan işler benzer olduğundan farklı ISA yapılarında kullanılan mnemonicler (opcodelar) birbirine benzemektedir. Bu da opcodelar üzerinden eğitilen bir dil modelinin ortam bağımsız çalışabilmesine olanak sağlar. Bunu doğal diller ile açıklamaya çalışırsak, Assembly için her ülkenin konuştuğu ortak bir dil diyebiliriz. Şekil 4.1'de farklı ortamlar için örnek komut satırları bulunmaktadır.

```
sget-object v0, String->String;
if-nez v0, :cond_0
invoke-virtual {p1}, String-->String;
move-result-object v0
const v1, 0x7f020005
invoke-static {v0, v1}, String;->String;
move-result-object v0
```

(a) Android

```
140001000 : mov qword ptr [rsp + 0x10], rbx
140001005 : mov qword ptr [rsp + 0x18], rsi
14000100a : push rdi
14000100b : sub rsp, 0x20
14000100f : xor esi, esi
140001011 : mov rbx, rdx
140001014 : mov rdi, rcx
140001017 : and word ptr [rdx], si
14000101a : mov al, byte ptr [rcx]
```

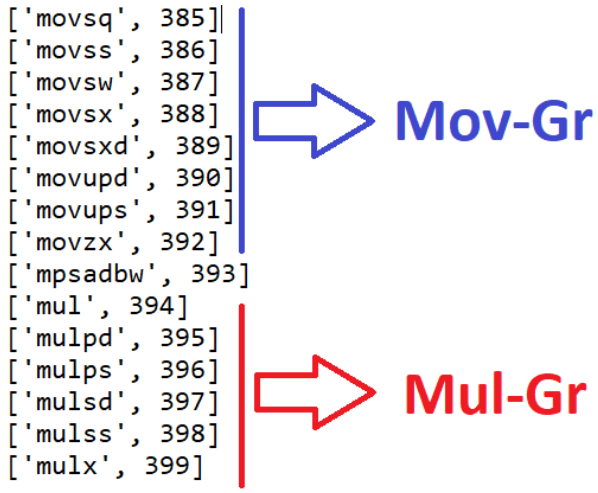
(b) X86-64

Şekil 4.1 X86-64 ve Android ortamları için örnek komut sekansları

4.1.3 Problemler

Hedef programlama dilimizi Assembly seçtiğimizde hemen fark edilecek temel bazı problemler vardır. Bunlardan ilki sözlük büyüklüğünün normal bir doğal dil ile kıyaslanamayacak kadar küçük olduğudur. Örnek vermek gerekirse yaptığımız denemelerde x86-64 ortamı için 1500 farklı kelime (mnemonic), Android ortamı için ise 239 farklı kelime tespit edebildik. Şekil 4.2'de görüleceği üzere tespit edilen bu kelimeler ayrıca gruplanabilir durumdadır. Bunları doğal dillerdeki eş anlamlı kelimelere benzetebiliriz. Görevlerine göre de bu kelimeleri grupladığımızda, kelime sayısı 16-32 arasına kadar düşülebilmektedir. Bu bize çok net bir şekilde doğal dillerin aksine, Assembly dillerinde kullanılan kelimelerin tek başlarına yüksek seviyeli bir anlam ifade etmediğini göstermektedir. Bu kelimelerin sekansları kullanılarak yüksek seviye özellik vektörleri elde edilmelidir.

Bir diğer problem, bilgisayar programların kod uzunlukları arasındaki varyansın çok yüksek olmasıdır. Buna örnek vermek gerekirse aynı aileye ait aynı işi yapan bir Malware kod uzunluğu, bir örnekte 10 kilo byte iken, kendisine ait başka bir klon örnekte 5 mega byte olabilmektedir. Programlama dillerinin özelliklerinden ötürü, anlamsal olarak hiçbir amacı olmayan ama mutlaka olması gereken kod blokları da mevcuttur (örneğin değişken tanımlamaları gibi). Bütün bu özellikler bize verilerin



Şekil 4.2 Opcodelar yaptıkları işe göre anlamda önemli bir kayıp olmaksızın gruplanabilirler

standart bir doğal dil işleme görevine göre önemli derecede daha gürültülü ve kirli olacağını göstermektedir.

Program uzunluklarındaki bu değişkenlik aynı zamanda bize bir sabit giriş uzunluğu (input length) sorunu getirmektedir. Günümüz LLM'lerinde "input length" limiti bellidir. Bu sınırı arttıran ya da sınırsız hale getiren sistemleri geliştirmek günümüzde hala çok popüler ve üzerinde araştırma yapılan bir konudur. Burada bir örnek vermek gerekirse 8'in katları şeklinde ilerleyen giriş uzunlukları, bugün itibariyle en gelişmiş sistemde dahi 64 bin olarak belirlenmiştir. Doğal dillerde 64 bin kelime, istatistiklere göre ortalama bir kitap uzunluğudur. Tekil kelimeler yüksek seviye anlamlar ifade ettiği için 64 bin kelime ile pek çok konu rahatlıkla anlatılabilir. Fakat Assembly dili için bu geçerli değildir. Bir yazılımın Assembly kod uzunluğu rahatlıkla 5 milyon kelimeyi geçebilmektedir (Tablo 4.1 'te kullandığımız veri setinde opcode uzunluklarına göre sayılar verilmiştir). Bilgisayar programlarının yapısında "location-invariance" özelliği olduğu için yani bir kod bloğunun, bütünün içinde yerinin bir önemi olmadığı için (yeri değişebildiği için), önemli anlamlar içeren kod sekansları bu 5 milyon kod bloğunun herhangi bir yerinde olabilir. Günümüz LLM'lerinde kullanılan yöntemler (LSTM, RNN, Transformers, Attention vb) bu uzunlukta girdiler ile çalışacak şekilde tasarlanmamıştır. Bu konu üzerine çalışmalar güncel araştırma konusudur.

Dengeli ve kapsayıcı veri seti oluşturma noktasında da önemli problemler vardır.

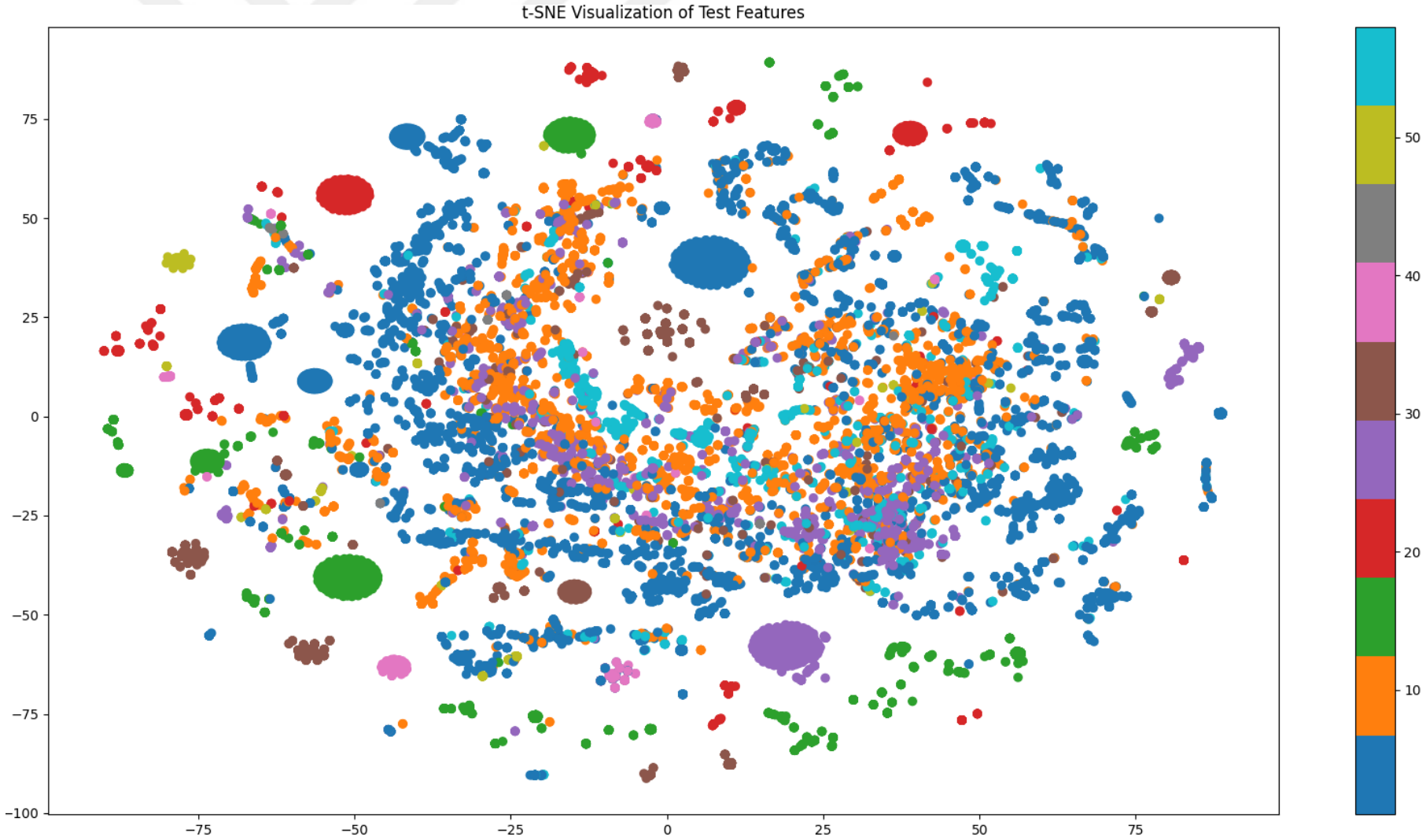
Tablo 4.1 Veri setindeki örneklerin kod uzunluklarına göre adet olarak dağılımı

Kod Uzunluğu	Android	X86-64
64k ve daha az (%)	58%	69%
64k ve daha az	27580	4508
<2k	3101	266
<4k	2847	295
<8k	3468	447
<16k	8088	1021
<32k	4449	1049
<64k	5627	1430
>=64k	19215	1942
TOPLAM	46795	6450

Öncelikle ikili sınıflandırma için kapsayıcı bir zararsız yazılım (Benign) kümesi hiçbir şekilde erişilebilir değildir. Gerek araştırmacılar gerekse güvenlik firmaları ellerindeki bu veriyi fikri mülkiyet hak ihlali korkusu nedeniyle paylaşmamaktadır. Malware veri seti ise bunun tam tersi olarak her çeşit ve her aileden istenildiği kadar örnek açık kaynak oluşumlar [77] [78] [79] [80] üzerinden temin edilebilmektedir. Yine de Benign örnekler, özellikle ikili sınıflandırmada başarıma önemli oranda etki etmektedir. Araştırmacıların doğru (ground-truth) bir Benign veri seti oluşturmak için kullandığı ilk yöntem temiz işletim sistemi kurulumlarından sonra işletim sistemindeki çalıştırılabilir programları almaktır. İlk bakışta çok mantıklı gözüken bu yaklaşımda önemli bir sorun mevcuttur. Bunu bir örnek ile açıklamak gerekirse, Windows üzerinden temin edilen Windows programların tamamı aynı opcode sekansı ile başlamakta ve aynı string parametrelerini içermektedir. Yaptığımız deneyler de gerçekçi bir veri seti ile %75 başarı gösteren bir modelimiz, ikili sınıflandırmada Benign olarak sadece Windows işletim sisteminden alınan dosya örnekleri kullandığımızda başarısı %99.99 olmuştur. Araştırmacıların ellerindeki veri setlerini paylaşmadıkları düşünüldüğünde karşılaştırma ancak yüksek başarım vaat eden modellerin kodlarına erişerek, elde bulunan veri seti ile test yapılması ile mümkün olmaktadır. Bütün bunları göz önüne aldığımızda, araştırmalar arası sonuçlar karşılaştırılabilir ve yeniden üretebilir olmadığı için pek bir anlam ifade etmemektedir.

Veri seti konusunda bir diğer hususta, Apk2vec [75] algoritması ile vektör haline getirilen örneklerin, bu vektörlerin yakınlıklarına göre gruplandıklarında, bazı

örneklerin mükemmel şekilde gruplanmasıdır. Bunu Şekil 4.3'da görebilmekteyiz. Bunun başlıca sebebi ilgili zararlı yazılımın dağıtım aşamasında sadece imzasını değiştirmek için kendisi üzerinde minör değişiklikler yapmasıdır. Bu değişiklikler kod tabanında olabileceği gibi, kod harici (örneğin Header gibi) yerlerde de olabilir. Sınıflar arası dengesizliğin, birbirinin kopyası olan bu örneklerin lehine olduğu durumlarda yüksek başarımlar yanıltıcı olacaktır. Buradaki temsil ve kapsayıcılık eksikliği yüksek başarım gösteren modelin gerçek hayatta uygulanabilir olduğunu sorgulatmaktadır



Şekil 4.3 Android veri setinde örneklerin birbirine yakınlığının TNSE ile görselleştirilmesi

4.2 Kullanılan Veri Setleri

Çalışmada önerilen metodolojinin farklı bilgisayar sistemlerinde benzer performanslarını sergilediğini göstermek adına iki farklı bilgisayar sisteminde deneyler yapılmıştır. Bunlardan ilki Android ortamı için kullanılan veri setidir. Deneylere başlanılan dönemde erişilebilir olan veri setlerinin oldukça eski ve kısıtlı olmasına istinaden (örneğin Drebin gibi) kapsamlı ve Modern Malware tanımına uyacak şekilde bir veri seti arayışına girdik. Aradığımız özellikleri AMDARGUS veri setinde bulduk. Bu veri seti Yıldız Teknik Üniversitesi Bilgisayar Mühendisliği Akıllı Sistemler Laboratuvarı ve COMODO güvenlik firmasının iş birliği ile kapsamlı Benign örnekler ile desteklendi. Windows ortamı için ise Modern Malware veri seti arayışımız Booz Allen Hamilton firması ile Maryland Üniversitesinin, güvenlik firmalarının güncel bültenlerini kullanarak topladıkları 454 aile ve 3095 örnek içeren veri seti ile sonlandı. Veri setlerine ait detaylı bilgiler için Tablo 4.2 incelenebilir.

Tablo 4.2 Veri seti örnek sayıları

	Android	**X86-64**
Malware Kaynak	AMDARGUS	MOTIF, PEMDB
Benign Kaynak	Comodo, AndMal2017, Maldroid2020	PEMDB
Malware Sayısı	22899	3704
Benign Sayısı	22279	2746
Eğitim Sayısı	31624	4165
Test Sayısı	13554	885
Zero-day Sayısı	1617	1400
Aile Sayısı	72	454

4.2.1 AMDARGUS

2017 yılında Güney Florida Üniversitesi Argus araştırma laboratuvarı tarafından yayınlanan ve zararlı yazımların yetkinlikleri hakkında da etiketler içeren AMD [81] veri setini kullanmaya karar verdik. Bu veri seti toplam 71 aile, 25 bin örnekten oluşan ve Modern Malware örnekleri içeren oldukça kapsamlı bir veri setidir. İkili sınıflandırma yapabilmek adına Yıldız Teknik Üniversitesi Akıllı Sistemler Laboratuvarı bünyesinde, Android marketinden çeşitli kategoriler kullanılarak elde edilen ve doğrulanan 12 bin Benign örnek ile COMODO güvenlik firması üzerinden elde edilen 5 bin Benign örnek birleştirilerek toplam 17 bin Benign örnek

toplanmıştır. Final veri seti 72 aileden oluşan 25 bin zararlı ve 17 bin zararsız örnek içermektedir. Bu veri seti çalışmamızın ortasından itibaren ANDMAL ve MALDROID veri setindeki Benign örnekler kullanılarak daha dengeli hale getirilmiştir.

4.2.2 ANDMAL ve MALDROID

Önerdiğimiz modelde özellikle Android ortamı için veri seti dengesizliğini düzeltmek adına New Brunswick Üniversitesi tarafından 2017 ve 2020'de yayınlanan [82] [83] ilgili veri setlerinden sadece Benign örnekler alınarak AMDARGUS seti ile birleştirilmiştir. Hatalı örnekler ayıklandıktan sonra toplamda 22 bin Malware, 22 bin Benign olmak üzere dengeli bir veri seti oluşturulmuştur.

4.2.3 MOTIF

Booz Allen Hamilton firması ile Maryland Üniversitesinin, güvenlik firmalarının güncel bültenlerini kullanarak topladıkları 454 aile ve 3095 örnek içeren veri seti [84] hatalı örneklerden arındırıldıktan sonra en az iki örneğe sahip olan aileler alınarak eğitime hazır hale getirilmiştir. Bu işlemlerden sonra 295 aileye ait toplam 2601 örnek içeren oldukça zorlu bir veri seti elde edilmiştir. Örneklerin tamamı en az bir Evasion yöntemi içermekte olup örneklerin bir kısmı Native (doğrudan hedef donanım koduna dönüştürülerek) bir kısmı sanal ortamlarında (CLR gibi) koşacak şekilde derlenmiştir. Hedef ortam bilgisi Header üstünden alınarak doğru Disassembler kütüphaneleri ile opcodelar çıkarılmıştır.

4.2.4 PEMDB

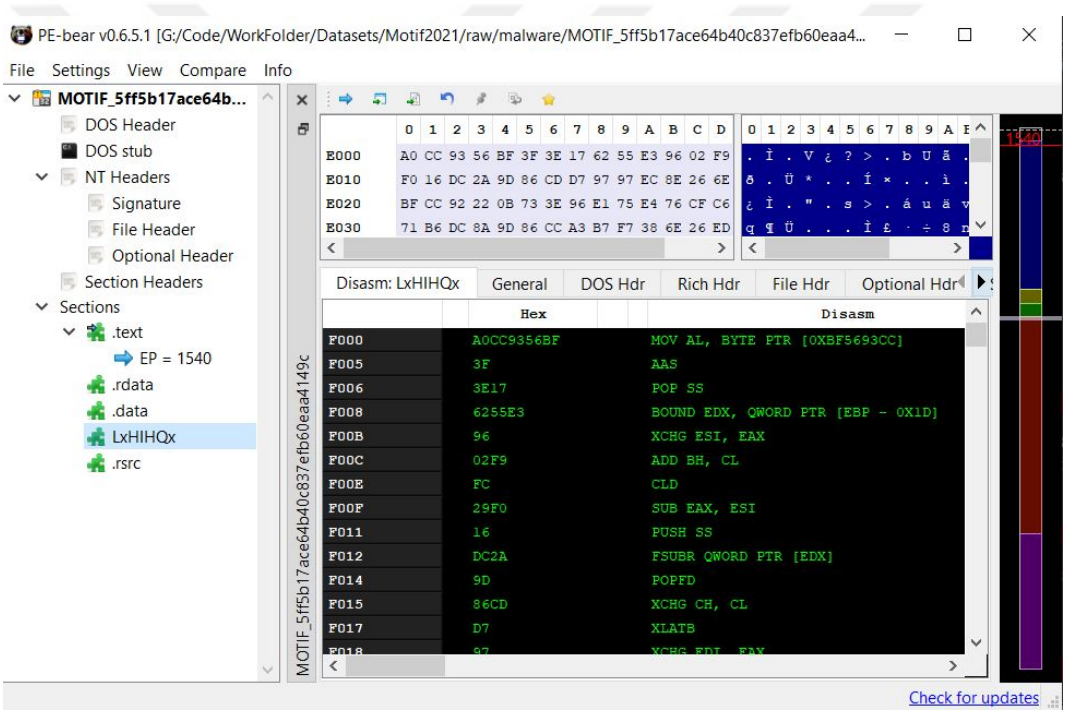
Practical Security Analytics LLC firmasının 2021 yılında yayınladığı 114 bin Malware, 86 bin Zararsız yazılım içeren güncel veri seti temin edilmiştir [85]. Bu veri setinden rastgele 2 bin adet Zararsız yazılım alınarak MOTIF veri seti ile birleştirilmiştir. Aynı zamanda Zero-Day testi için rastgele 1500 Malware örneği temin edilmiştir. Bu örneklerin 100 tanesi eğitim setinde, 1400 tanesi Zero-Day testinde kullanılmıştır.

4.3 Özellik Çıkarımı ve Önişleme

Önerilen yöntemlere özel özellik çıkarımı ve ön işleme bilgileri detaylı olarak ayrıca ilerleyen bölümlerde verilecektir.

Hipotezimiz Assembly dili üzerinde doğal dil işleme yöntemlerini kullanmak olduğundan, öncelikle efektif şekilde yazılımların yalın hallerinden Assembly kodlarının çıkartılması gerekmektedir.

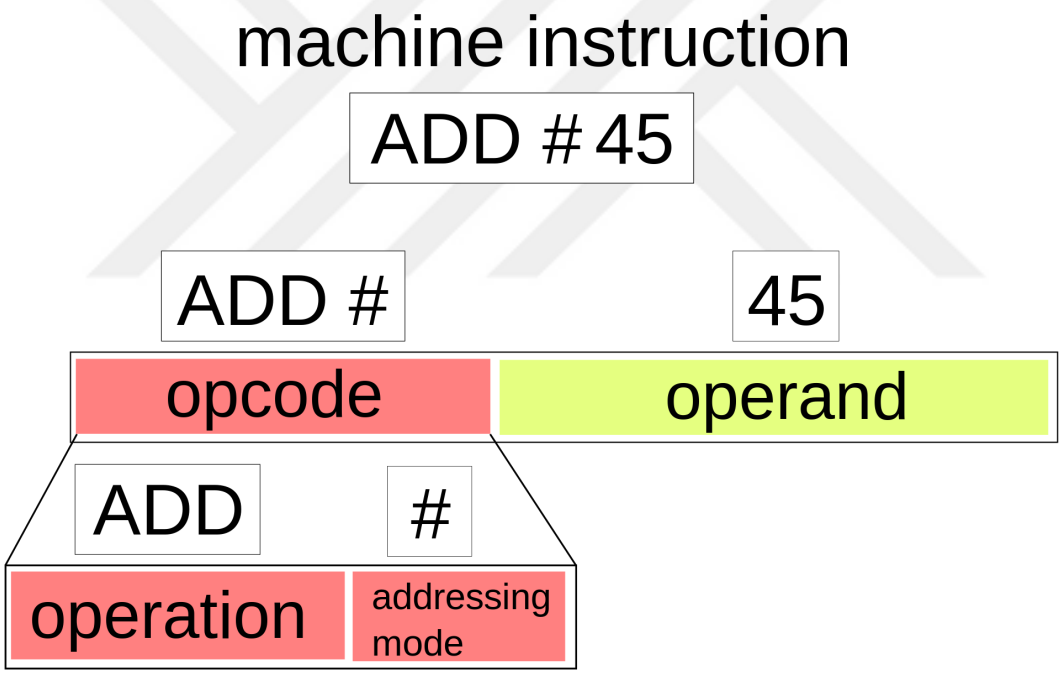
Windows ve Android mimarileri birbirinden oldukça farklıdır. Windows PE dosyaları Header, .text .data gibi bölümler içerirken, bir APK (Android Package Kit) dosyası açıldığında Metadata ve .dex uzantılı dosyalar bulunacaktır.



Şekil 4.4 Farklı bir bölümde faydalı kod saklayan bir Malware örneği

Windows için uygulama kodlarının bulunduğu alan normal şartlar alında .text dosyasıdır. Fakat bir aldatma yöntemi olarak zararlı yazılımlar birden çok farklı isimde bölüm oluşturarak kodları bu bölümlere saklayabilirler. (Şekil 4.4) Android ortamında ise .dex dosyası bir Disassembler aracından geçirildikten sonra uygulamanın Dalvik Assembly kodlarının bulunduğu birden çok .smali dosyayı oluşacaktır. Android mimarisi nesne yönelimli (Object Oriented) bir mimari olduğu için, bu .smali dosyaları aslında uygulama içindeki sınıfları (class) temsil etmektedir.

Yukarıdaki ayrımdan sonra ön işleme adımı iki ortam içinde aynıdır. Öncelikle her bir örnek için bütün kodlar tek bir dosyada toplanır. Örneğin A.exe dosyası için A.txt dosyası oluşur. Bu dosyalar Disassembler çıktısını doğrudan içerdiğinden, içlerinde temizlenmesi gereken alanlar olacaktır. Ayrıca içlerinde opcode mnemonicleri değil instruction setler (adresler ve operand'ın bulunduğu tam bir bir makine yönergesi, Şekil 4.5) bulunacaktır. Kısaca eğer çalışmada sadece opcode sekansları üzerinden gidilmek isteniyorsa gereksiz alanlar (operandlar gibi) atılmalıdır. Eğer operandlar atılmayacaksa burada dikkat edilmesi gereken bir nokta bulunmaktadır. Bu operandlardan bazıları sabit veri içermektedir (örneğin stringler). Bu değerler anonimleştirilmediği taktirde öğrenici model bunları ezberleyebilir. Dolayısıyla o veri seti için oldukça iyi çalışan fakat problemi genelleştirememiş bir model elde edilecektir.



Şekil 4.5 Makine yönergesi örneği

Bütün örnekler için opcode sekansları çıkarıldıktan sonra sözlük oluşturma ve tokenizer işlemleri gerçekleştirilerek her bir kelimeye bir sayı değeri atanır. Elimizdeki veri setinde Windows ortamı için tüm corpusta yaklaşık 1500, Android ortamı için ise yaklaşık 239 tekil kelime bulunmuştur. Daha fazla sıkıştırma için tokenizer işleminde Huffman Coding benzeri bir algoritma kullanılabilir.

Corpus ve Tokenizer işleminden sonra kelimeler arasındaki ilişkiyi daha iyi

ifade etmek adına kelime vektörleri çıkartılır. Burada Word2vec algoritmasında kullanılan iki yöntemden biri olan Skip-Gram kullanılmıştır.

Sınıflandırma yapılacağı zaman kullanılacak modelin input layer uzunluğuna göre ilgili örnekten o uzunlukta tokenize edilmiş opcode sekansları sırayla çekilir. Gerekli padding işlemleri yapıldıktan sonra veri eğitime hazır hale gelmektedir.

4.3.1 Ortam Farklılıkları

Android ortamı için standart tek bir Disassembler aracı yeterliyken, Windows ortamı için makine koduna ait özel Disassembler gerekmektedir. Windows ortamında çalışan bazı programlar doğrudan işlemcinin makine koduna döndürülmeyip, bir ara sanal katmanda çalıştırılabilirler. (JRE, CLR vb). Bu hedef ortam bilgisi PE dosyalarının Header alanında bulunmaktadır (Target Machine) [86]. Dolayısıyla Windows için birden fazla Disassembler gereklidir.

Windows için kullanılabilecek standart Disassembler araçları varsayılan ayarlarında genellikle sadece .text kısmını çözmeye çalışacaktır. Çünkü normal akışta programın kodları sadece .text alanında bulunmaktadır. Bu oldukça dikkat edilmesi gereken bir konudur. Modern zararlılar kodların bir kısmını yeni .section alanları açarak bunların içine gizleyebilmektedir.

4.3.2 Kullanılan Programlar

Android Disassemble için apktool kütüphanesi kullanılmıştır [87]. Windows Disassemble işlemleri için ise PEBear [88] yazılım kullanılmıştır. PEBear Windows yazılımları için programın giriş noktasını belirleyerek bütün sectionlar için Assembly kodlarını çıkarmaktadır.

Word2Vec algoritması için Gensim [89] kütüphanesi, derin öğrenme modellerinin geliştirilmesi için Tensorflow kütüphanesi kullanılmıştır.

Windows ortamında Malware yetkinliklerini çıkartmak için Mandiant CAPA [33] kütüphanesi kullanılmıştır.

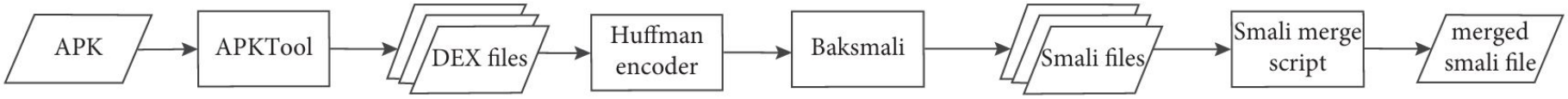
4.4 APK2VEC

Hipotezimizi destekleyen önerdiğimiz ilk model Android veri seti üzerinde kelime vektörleri ve Makine Öğrenmesi metotları ile geliştirdiğimiz Apk2vec isimli modeldir. Bu çalışmamızda bulduğumuz sonuçlar Nisan 2020 yılında Hindawi Security and Communication Networks dergisinin 2020 sayısında yayınlanmıştır [75].

Bu bölümde bu çalışmada önerilen metodoloji, kullanılan veri seti ve ön işleme adımları son olarak bulunan sonuçlar verilecektir. Apk2vec çalışmamızda yapılan literatür araştırması, "erken literatür araştırmaları" bölümünde verilmiştir.

Tablo 4.3 Android veri setinde bulunan örneklerin yetkinlikleri

Komposizyon	Yüklenme	Aktivasyon	Bilgi Çalma	Kalıcılık	Ayrıcalıklık	C&C	Anti Analiz	Paraya Dönüştürme
• Bağımsız • Yeniden Paketleme • Kütüphane	• Zero-Click • Yüklenerek	• Olay • Etkileşim • Zamanlama	• Kişi Bilgileri • Cihaz Bilgileri	• Kanıt Temizleme • Kapatılmayı Önleme	• Admin Yetkisi • Root Exploit	• Net • SMS • Komut Kodlama	• Yeniden İsimlendirme • String Şifreleme • Dinamik Yükleme • Native Payload • Dinamik Analizden Kaçınma	• Premium • Bank • Ransom



Şekil 4.6 Android APK dosyalarında ön işleme adımları

4.4.1 Özet

Önceki bölümlerde bahsedilen hipotezimizi doğrulama adına, çalışmanın yapıldığı dönemde state-of-art doğal dil işleme metotları ve makine öğrenmesi yöntemlerini kullanarak bir doğrulama çalışması yapma gereksinimi doğmuştur. Önceki araştırmalarda eksiğini gördüğümüz hususlardan biri olan deneylerin kapsamlı ve Modern Malware örnekleri içeren bir veri seti üzerinde yapılması gerekliliğini gidermek adına veri seti arayışında bulunduk. 2017'de yayınlanan ve 2012-2016 yılları arasında toplanmış 25 bin adet Modern Malware içeren Argus veri seti ile, bizim oluşturduğumuz ve COMODO güvenlik firması üzerinden temin ettiğimiz Benign veri seti bu çalışmada kullanılmıştır. AMDARGUS veri seti detaylı analiz çalışmasıyla Malware aileleri için çeşitli yetkinlik etiketleri çıkarmıştır. Bu yetkinlikler ilgili Tablo 4.3'da detaylıca verilmiştir. Yaptığımız çalışma, kelime vektörleri üzerinden ağaç tabanlı algoritmalar eğitilerek test verisinde %95.64 başarım elde etmiş, 20 aileyi eğitim setinden çıkartarak oluşturduğumuz Zero-Day testlerinde ise %37.36 başarım sergilemiştir. Kullandığımız test verisi VirusTotal API kullanılarak sektörde en güvenilir 10 anti-malware ürünü ile karşılaştırılmıştır. 2019 Kasım ayında yapılan bu test sonucunda önerdiğimiz model başarım olarak 7 Anti-Malware uygulamasını geride bırakmıştır 4.8. Veri setindeki en genç örneğin 2016 yılından olduğunu hesaba katarak, çok yüksek başarım vermesi beklenen Anti-Malware uygulamalarının önerdiğimiz metodolojiden kötü sonuçlar vermesi hipotezimizi kuvvetlendirmiştir.

4.4.2 Kullanılan Veri Seti

Kullandığımız veri setinde bulunan aile bazlı kaçınma yöntemleri ve yetkinlikler Tablo 4.3'te verilmiştir. Veri setinde bulunan bazı hatalı veriler ve 5 MB kod uzunluğundan fazla olan örnekler ayıklandıktan sonra Malware veri seti boyutu 22.899 adede düşmüştür. Opcodelar çıkartıldıktan sonra oluşan kod uzunlukları 10 KB ile 5 MB arasında değişmektedir. Benzer şekilde Benign örnek sayısı 16.630 olup, kod uzunlukları 2 KB ile 5 MB arasında değişmektedir.

4.4.3 Ön işleme

Dataset, raw halleri ile APK (Android Package Kit) dosyaları içermektedir. Bu dosyalar içerisinde derlenmiş kod bloğu (Dex dosyaları), çeşitli kaynaklar ve metadata.xml gibi dosyalar içermektedir. Dex dosyası, Android Run Time (ART) sanal makinesi üzerinde koşacak şekilde hazırlanmış makine kodlarını içermektedir. Dex dosyalarından çıkartılan opcodelar, text olarak yazıldıklarında veri boyutu oldukça büyüdüğü için bu çalışmada Huffman Coding kullanılarak bu opcodelar tokenize edilmiştir. APK dosyalarından opcode çıkarma işlemi birkaç adımdan oluşmaktadır. Bu adımların şeması için Şekil 4.6 incelenebilir.

Öncelikle APK dosyalarından Dex dosyalarını çıkarmak için Apktool kütüphanesi kullanılır. Bu işlem her bir APK programı için tek bir Dex dosyası üretir. Daha sonra bu Dex dosyası "baksmali" Disassembler kullanılarak birden çok Smali dosyalarına çevrilir. Smali dosyalarının içindeki opcode olmayan bilgiler ve operandlar silindikten sonra her bir APK için birden çok şekilde oluşan Smali dosyaları Huffman Encoding'den geçirilerek tokenize edilir. Son aşamada ise bütün Smali dosyaları tek bir smali dosyasında birleştirilerek her bir APK için içinde sadece tokenize edilmiş opcode sekansları bulunan tek bir özellik dosyası elde edilir. Bu işlemler için baksmali disassembler yerine Apktool'un içinde bulunan disassemblerda kullanılabilir.

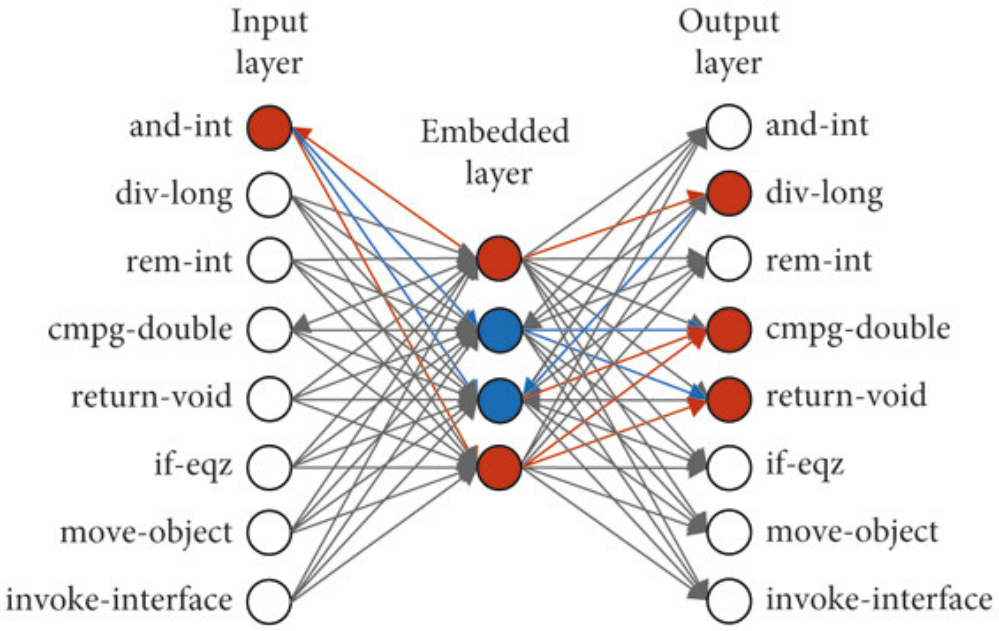
4.4.4 Sınıflandırıcı Model

Android uygulamalarındaki opcode dizilerini yapay bir dil olarak değerlendirdik. Bu dilde, benzersiz opcode'lar dilin kelimeleri, fonksiyonlar cümleleri ve kodun tümü de korpusun bir parçası olarak kabul edilebilir. Opcode'lar arasında anlamsal ilişkiler kurmak için Word2Vec modelinin bir parçası olan Skip-Gram yöntemi kullanılmıştır.

Word2Vec, kelime vektörlerini üretmek için kullanılan bir algoritmadır. Bu modeller, kelimelerin dilbilimsel bağlamlarını yeniden oluşturmak için eğitilen basit ve iki katmanlı yapay sinir ağlarıdır. Word2Vec, büyük bir metin korpusunu girdi olarak alır ve korpustaki her bir benzersiz kelime için genellikle birkaç yüz boyutlu bir vektör uzayı üretir.

4.4.4.1 Skip-Gram

Kelime vektörleri, korpusta ortak bağlamları (context) paylaşan kelimelerin vektör uzayında birbirine yakın konumlandığı şekilde yerleştirilir. Skip-gram, seçilen bir kelimeden önce ve sonra gelebilecek potansiyel kelimeleri tahmin etmek için kullanılan bir özellik çıkarım modelidir. Seçilen kelimeye hedef kelime, belirli bir pencere boyutunda etrafında bulunan kelimelere bağlam olarak atıfta bulunur. Skip-gram modeli, her hedef ve bağlam ikilileri için unsupervised şekilde eğitilir. Eğitim süreci tamamlandıktan sonra, oluşan ağırlıklar her kelime için kelime vektörü olarak kabul edilir. Bu süreç için Şekil 4.7 bakılabilir.

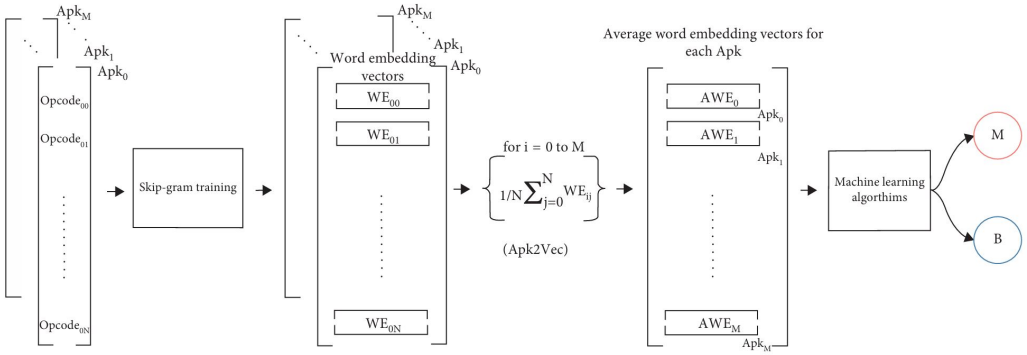


Şekil 4.7 Skip-gram öğrenme sürecinde girdi ve çıktı birbirine yakın olacak şekilde ara katman eğitilir. Daha sonra bu ara katmanın ağırlıkları kelime vektörlerinin elde edilmesinde kullanılır

4.4.4.2 Apk2vec

Veri setindeki uygulamalar, opcode sekans uzunluğu açısından büyük bir varyansa sahiptir. Bu durum, makine öğrenimi tabanlı modellerin eğitimi için tutarsız bir giriş katmanı boyutu sorunu yaratmaktadır. Burada akla gelen ilk çözümlerden bir tanesi, yeterince uzun bir giriş katmanı belirledikten sonra bu uzunluğu sağlamayan sekanslara "padding" eklemektir. Bu yaklaşım varyansın düşük olduğu doğal dillerde işe yaramakla birlikte bizim veri kümemizde, giriş katmanı boyutları birkaç

yüzden, milyonlara kadar değiştiği için bu "sparse" veri sorunu oluşturmaktadır. Bu problemi çözmek adına her Smali dosyasını sabit boyutlu bir vektörle tanımlamak için "Apk2vec" olarak adlandırılan basit ama etkili bir yöntem kullanılmıştır. Smali dosyasındaki her opcode'a karşılık gelen kelime vektörleri o dosya için toplanarak ortalaması alınmıştır. Oluşan vektör ilgili örneğin temsil vektörü olarak kabul edilmiştir. Bu işlem her örnek için tekrarlandıktan sonra oluşan özellik vektörü seçilen sınıflandırıcı modellere beslenerek eğitim sağlanmıştır. Bu süreç için Şekil 4.8 incelenebilir.



Şekil 4.8 Apk2vec Algoritması

4.4.5 Deney ve Sonuçlar

Skip-gram, büyük sözlüklere sahip ancak küçük boyutları olan doğal diller için tasarlanmıştır. Fakat daha önce bahsedildiği gibi Assembly dili tam tersi bir karakteristiğe sahiptir. Dolayısıyla varsayılan Skip-gram hiper parametreleri mevcut duruma uyum sağlayacak şekilde yeniden keşfedilmelidir. Hiper parametre arama işlemini veri setinin tamamında yapmak zaman kaybı olacağından aileleri iyi derecede temsil eden 1500 Malware ve 1500 Benign örnek seçilerek küçük bir veri seti oluşturulmuştur. Çeşitli pencere ve kelime vektör uzunlukları ile çıkarılan özellikler Apk2vec algoritmasından geçirilerek Naive Bayes, SVM, Random Forest ve Desicion Tree basit öğrenciler ile test edilmiştir. Yapılan hiper parametre testinin sonuçlarına Tablo 4.4 bakılabilir.

Tablo 4.4 Skip-gram hiper parametre deneyi

		J48		Naive Bayes		Random Forest		SVM	
Pencere Boyutu		2	3	2	3	2	3	2	3
	50	79.77%	78.91%	60.51%	60.44%	83.89%	83.56%	60.01%	59.78%
Vektör Uzunluğu	300	81.93%	81.50%	60.81%	62.37%	87.08%	87.14%	59.61%	60.04%
	500	82.63%	82.82%	61.60%	65.99%	86.51%	88.54%	59.21%	59.21%

4.4.5.1 Test ve Zero-Day Performansları

Bu çalışmada deneyler 2 farklı senaryo üzerine kurgulanmıştır. Senaryolardan ilki ikili sınıflandırma başarımı ölçerken, Zero-Day başarımını ölçmek için kurguladığımız ikinci senaryo, eğitilen modelin daha önce karşılaşmadığı bir aileden örnek geldiğindeki davranışını ölçmek üzerinedir.

Senaryo-1 için eğitim ve test seti, her aileden bir örnek mutlaka eğitim setinde olacak şekilde %70 ve %30 oranında ayrılmıştır. Senaryo-2 için, eğitim setinden toplamda 21 aile (ve bu ailelere dahil olan toplam 1999 örnek) tamamen çıkarılarak yeni sınıflandırıcılar eğitilmiştir. Testlerde veri seti boyutu yeterince büyük olduğu için cross-validation aşaması atlanmıştır.

Özellikler çıkarılırken en başarılı Skip-Gram hiper parametreleri kullanılmış, Huffman Coding ile toplam özellik boyutu 107 GB'dan 500 MB'ye kadar düşürülmüştür. Senaryolarda kullanılan eğitim, test ve Zero-Day örnek dağılımları için Tablo 4.5 bakılabilir.

Tablo 4.5 Senaryo-1 ve Senaryo-2 deneyleri için veri setindeki örnek sayıları

	Sınıf	**Senaryo-1**	**Senaryo-2**
Eğitim Seti	Malware	16,982	15,593
	Benign	11,641	11,641
Test Seti	Malware	7,322	6,712
	Benign	4,988	4,988
Zero-Day Seti	Malware	0	1,999
	Benign	0	0

4.4.5.2 Deney Sonuçları

Performans değerlendirmesi için F1 ölçüsü, precision, recall ve accuracy metrikleri hesaplandı. Veri setimizde Benign ve Malware yazılım örneklerinin dengeli sayıda bulunması nedeniyle, önerdiğimiz modelin performansını yansıtmak için belirleyici kriter olarak doğruluk metriği seçildi. Tablo 4.6 ve Tablo 4.7, modelin Senaryo-1

ve Senaryo-2 için deney sonuçlarını özetlemektedir. Her iki modelde de RF (Random Forest) algoritması, sırasıyla %95,64 ve %96,12 ile en iyi başarımı sağladı. Zero-Day Malware örnekleri, ikinci senaryodaki eksik aile sayısı ile yeniden eğitilmiş RF sınıflandırıcı ile 1999 örnek arasından 747 örneği başarılı bir şekilde tespit etti.

Ayrıca modelimizin performansını, en popüler 11 anti-malware yazılımının performanslarıyla karşılaştırdık. Karşılaştırmanın sonuçları Tablo 4.8'da verilmiştir ve modelimiz 11 ticari yazılım uygulamasından 7'sini geçmiş ve bir tanesi ile berabere kalmıştır.

Son olarak, modelimizin Evasive Malware yazılımlarına karşı dayanıklılığını değerlendirdik. AMDARGUS veri setini oluşturan araştırma grubu, 71 aileden 48 tanesini anti-analiz metotları içeren aileler olarak işaretlemiştir. Bu da test verimizde toplam 6667 örneğin Anti-Analiz yetkinliğine sahip olduğu anlamına gelmektedir. Tablo 4.9 Önerdiğimiz modelin veri setindeki Evasive Malware tespiti görevindeki başarımı %95 olarak ölçülmüştür.

Tablo 4.6 Senaryo-1 test sonuçları. Ensemble sonucu diğer metotların çoğunluk oyuna göre belirlenmiştir.

	Malware			**Benign**			**Toplam**
	Precision	Recall	F1	Precision	Recall	F1	Doğruluk %
SVM	86.5	79.7	82.9	73.2	81.7	77.2	80.48
Random Forest	97	95.6	96.3	93.7	95.7	94.7	95.64
Decision Tree	91.9	93.6	92.7	90.3	87.9	98.1	91.25
Random Subspace	95.7	94.4	95.1	92	94.8	92.9	94.18
SGD	82.9	93.4	87.8	88.8	71.7	79	84.58
KNN	93.4	96.9	95.1	95.2	90	92.5	94.09
Ensemble	95	96.5	95.7	94.7	92.5	93.7	94.88

Tablo 4.7 Senaryo-2 test sonuçları. Bu senaryoda 20 aile eğitim ve test setinden tamamen çıkarılmıştır. Bu ailelere ait örnekler, Zero-Day testinde kullanılmıştır. Modelin Zero-Day başarımı %37,5 olarak ölçülmüştür.

	Malware			**Benign**			**Toplam**
	Precision	Recall	F1	Precision	Recall	F1	Doğruluk %
SVM	87.6	79.8	83.5	75.8	84.8	80	81.94
Random Forest	97.4	95.8	96.6	94.4	96.6	95.5	96.12
Decision Tree	92.5	93.9	93.2	91.6	89.7	90.6	92.04
Random Subspace	96.4	94.6	95.5	92.9	95.2	94	94.86
SGD	70.4	97.4	81.7	92.7	45	60.6	75.04
KNN	93.5	97.1	95.3	95.9	90.9	93.4	94.48
Ensemble	94.8	96.9	95.8	95.7	92.8	94.2	95.15

4.4.6 Tartışma

Bu çalışma Modern Malware yazılımlarından çıkarılan Assembly opcodeları üzerinde uygulanan doğal dil işleme metotlarını etkinliğini göstermiştir. Yazılımları çalıştırmadan sadece statik analiz yaparak %95.64 başarıma ulaşan model, bize çalışmanın devamı için yeşil ışık yakmıştır. Yine de modelin yüksek başarımına rağmen Zero-Day testinde gerçek dünya için yetersiz olduğu gösterilmiştir.

Tablo 4.8 VirusTotal API üzerinden test veri seti kullanılarak önerilen yöntem, dönemin en popüler Anti-Malware yazılımları ile karşılaştırılmıştır. Önerilen model en popüler 11 yazılımdan 7 tanesini başarım olarak geçmiştir.

Anti-Malware	**TP**	**FN**	**Doğruluk**
Avast	2900	4422	0.39
AVG	2902	4420	0.39
Avira	6797	525	0.92
Comodo	6367	955	0.86
Eset-Nod32	7304	18	0.99
F-secure	6311	1011	0.86
Kaspersky	4144	3178	0.56
McAfee	7265	57	0.99
Microsoft	4075	3247	0.55
Sophos	7074	248	0.96
Symantec	7225	97	0.98
Proposed model	7066	256	0.96

Tablo 4.9 Modelin evasive yöntemlerine karşı dayanıklılığı; bu tablo zararlı yazılım ailelerini, evasive yeteneklerini ve test setindeki örnek sayılarını göstermektedir. Son sütun, ilgili zararlı yazılım ailesine göre modelin tespit performansını göstermektedir.

Family name	Renaming	String Encryption	Dynamic Loading	Native Payload	Anti-Dynamic Analysis	Total count	Accuracy (%)
Airpush	✓					2353	95
Andup	✓					14	92
BankBot		✓				195	99
Bankun		✓				21	90
Boqx						65	75
Boxer		✓				14	100
Cova						4	100
Dowgin	✓					1015	95
DroidKungFu	✓	✓				164	98
FakeAngry	✓					3	66
FakeDoc	✓					7	100
FakeInst		✓				651	100
FakePlayer		✓				12	83
FakeUpdates		✓				2	50
Finspy	✓					3	100
Fobus	✓		✓			383	97
Fusob	✓		✓			11	91
GingerMaster					✓	7	100
GoPro	✓					21	63
Gumen		✓				10	80
Koler		✓				3	100

Tablo 4.9 Modelin evasive yöntemlerine karşı dayanıklılığı; bu tablo zararlı yazılım ailelerini, evasive yeteneklerini ve test setindeki örnek sayılarını göstermektedir. Son sütun, ilgili zararlı yazılım ailesine göre modelin tespit performansını göstermektedir. (devamı)

Family name	Renaming	String Encryption	Dynamic Loading	Native Payload	Anti-Dynamic Analysis	Total count	Accuracy (%)
Ksapp		✓				11	100
Kuguo		✓				360	98
Kyview		✓				48	95
Leech	✓					39	100
Lotoor		✓				95	96
Minimob	✓					61	88
Mseg	✓					71	53
Mtk	✓	✓	✓			21	100
Obad	✓	✓			✓	3	100
Opfake		✓				3	66
Ogel	✓			✓		2	100
Roop	✓					155	99
RuMMS	✓	✓	✓			121	100
SlemBunk		✓	✓	✓		52	100
Simplelocker	✓					48	100
SmsKey	✓					50	98
Stealer	✓					8	100
Svpeng					✓	4	75

Tablo 4.9 Modelin evasive yöntemlerine karşı dayanıklılığı; bu tablo zararlı yazılım ailelerini, evasive yeteneklerini ve test setindeki örnek sayılarını göstermektedir. Son sütun, ilgili zararlı yazılım ailesine göre modelin tespit performansını göstermektedir. (devamı)

Family name	**Renaming**	**String Encryption**	**Dynamic Loading**	**Native Payload**	**Anti-Dynamic Analysis**	**Total count**	**Accuracy (%)**
Tesbo	✓	✓				2	100
Triada	✓	✓	✓		✓	63	95
UpdtKiller	✓			✓		8	100
Utchi	✓					4	100
VikingHorde				✓		3	33
Youmi	✓					390	97
Winge	✓					6	16
Zitmo	✓					8	87
Ztorg	✓	✓	✓			6	199
Başarım %	95	96	94	93	92	-	95 (ortalama)

4.5 TRCONV

İlk çalışmanın kapsamını genişleterek önerdiğimiz TRCONV modeli, konvolüsyon tabanlı bir derin öğrenme metodolojisi içermektedir. Bununla beraber model parametrelerinin güncellenme aşamasında modele, birbiri ile ilişkili davranışsal hedef değişkenleri tanıtılarak modelin problemi genelleştirme yeteneği, Zero-Day saldırılarına karşı başarısı ve çeşitli Adversarial saldırılara karşı dayanımı arttırılmıştır. Bu hedef değişkenleri, Malware ve Benign örnekler için çeşitli yetkinliklerin (örneğin evasive özellikleri) statik analiz yöntemleri ile çıkartılması ile oluşturulmuştur. Bununla ilgili detaylar ilerleyen bölümde verilecektir.

İlgili çalışma Mayıs 2024 yılında IEEE Access dergisinde yayınlanmıştır[76]. Bu bölümde bu çalışmaya ait metodoloji, kullanılan veri seti ve ön işleme adımları son olarak bulunan sonuçlar verilecektir. TRCONV çalışmamızda kullanılan literatür araştırması, "yakın dönem literatür araştırmaları" bölümünde verilmiştir.

Önceki çalışmanın kapsamı aşağıda belirtildiği gibi genişletilmiştir:

- Veri seti kalitesinin iyileştirilmesi
- Ortalama vektörler yerine, kelime sekanslarının kullanılarak bağlam ve yüksek seviye anlam çıkarımı yapılması
- Basit öğrenicilerden, derin öğrenme modellerine geçilmesi
- Literatürde bulunan state-of-art modeller ile kıyaslama yapılması
- Gerçek dünya problemlerine karşı modelin dayanımının ölçülmesi

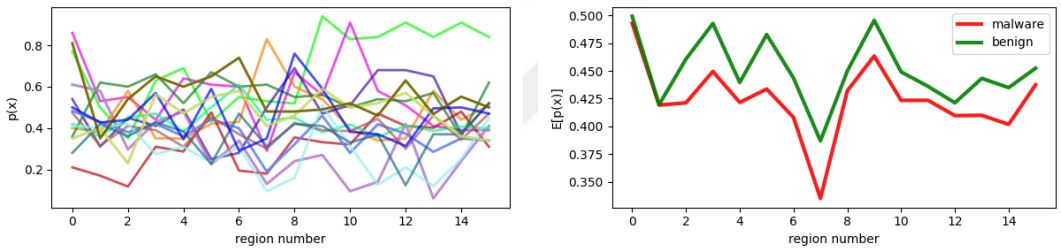
Bu sayede önerilen model literatüre aşağıdaki katkıları yapmıştır:

- Malware sınıflandırması probleminin, ilişkili ancak zayıf güvenilirliğe sahip hedef değişkenlerin ortak öğrenimi olarak formüle edilmesi
- Metodolojinin etkinliğinin hem Windows hem de Android mimarisinde doğrulanması

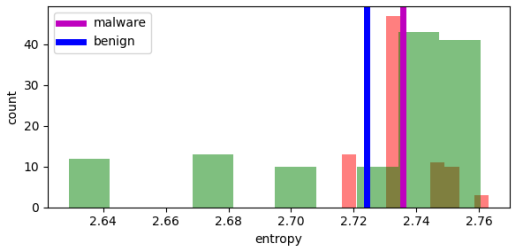
- Modelin performansının çeşitli gerçek dünya senaryolarında değerlendirilmesi
- Kelime vektörlerinin önceden eğitilerek derin öğrenme modeline verilmesi ile model eğitiminin hızlandırılması

4.5.1 Kullanılan Veri Seti ve Ön işlemeler

Bir önceki çalışmada kullanılan AMDARGUS veri setinde bulunan Malware-Benign sayıları arasındaki dengesizlik, halka açık yeni ve güncel veri setlerinden temin edilen Benign örnekler ile dengelenmiştir. Windows ortamı için ise oldukça zorlu ve karmaşık Malware örnekleri içeren MOTIF veri seti temin edilmiş bu veri seti yine halka açık olan PEMDB veri setinde bulunan Benign örnekler ile desteklenmiştir. Bu sayede her iki mimari içinde zorlu ve dengeli iki temel veri seti grubu oluşturulmuştur.



(a) Her bölge için Posterior $p(y = 1|x_r)$ **(b)** Her bölge için Ortalama posterior olasılığı



(c) Posteriorların entropi histogramı

Şekil 4.9 a) Eğitimden sonra her bir örnek bölgesi için sınıflayıcının posterior olasılıkları $p(y = 1|x_r)$. Tepkiler oldukça dağınık, tek bir bölgeye atıf yapan uniform ve zirve yapan bir deseni gözlemlemek oldukça zor. **b)** Hem Malware hem de Benign setler için bölge başına ortalama posterior olasılık. İki sinyal çok benzer özellikler sergiliyor. **c)** Posterior olasılıkların entropilerinin histogramı, Malware (kırmızı) ve Benign (yeşil). Entropi bilgisiyle bile sinyalleri ayırt etmek zor.

4.5.2 Lokal Analiz Problemi

Literatürdeki çalışmaların birçoğu Malware yazılım tespitini bir sınıflandırma problemi olarak ele almaktadır. Bu çalışmaların bazıları, veri kümelerinde oldukça yüksek ve doygun performanslar raporlamaktadır. Bu durum, Malware tespitinin hali hazırda çözülmüş kolay bir problem olup olmadığı veya kullanılan veri kümelerinin görevin zorluğunu gösterecek kadar yeterli temsil gücüne sahip olup olmadığı sorusunu gündeme getirmektedir. Eğer problemin karmaşıklığı gerçekten düşükse, Riber ve arkadaşlarının [90] çalışmasında olduğu gibi sınıflandırıcıların tepkilerini anlamaya çalışmak gereklidir. Buradaki hipotez doğruysa, sınıflandırmalar için açıklanabilirlik sorunsalı oluşmaktadır. Bu bağlamda, açıklanabilir yapay zekâ (XAI) [91] konusuna cevap verebilecek herhangi bir analiz, Malware analistleri için faydalı olacaktır.

Açıklanabilirliği artırmak için, Malware yazılımlarındaki kod bloklarının belirli bölümlerinin diğerlerinden daha bilgilendirici olup olmadığını sorgulamak doğru olabilir. Eğer bazı yerel bölümler diğerlerinden daha ayırt edici ise, Malware örnekleri üzerinde bir sınıflandırıcı eğitmek, sadece yorumlanabilirliği artırmakla kalmayacak, aynı zamanda kötücül özellikler gösteren kod bölgeleri hakkında ipuçları da verecektir. Fakat özellikle akademi bünyesinde birçok araştırmacı için kapsamlı kod analizi yaparak çeşitli kod bölgelerini kötücül olarak etiketleme işlemi oldukça zor ve maliyetlidir. Bu zorluk nedeniyle, yerellik hipotezini test etmek adına Malware ve Benign kod blokları eşit alt bölgelere ayrılarak, bu alt bölgeleri kullanarak yeni sınıflandırıcılar eğittik.

İlk adımda, opcode kelime vektörlerini öğrenildikten sonra, her örnek, eşit yerel bölgelere x_r bölünerek bu bölgeler için Apk2Vec [75] algoritması çalıştırılmıştır. Elde edilen ortalama vektör ile eğitilen basit bir sınıflandırıcının (RF) ardından test setinde ki her yerel bölge sınıflandırılarak posterior olasılık elde edilir $p(y = 1|x_r)$. Şekil 4.9a'da bu posterior olasılıklar gösterilmiştir. Her ne kadar net bir ayrım beklemesek de eğer ilk hipotez doğru ise bazı ayrımlara dair bir iz yakalamamız gerekmektedir. Yaptığımız analiz bize bu bilgiyi sunmamıştır. Buna ek olarak Şekil 4.9b'de beklenen posterior $E[p(y = 1|x_r)]$ değerlerini göstermektedir. Burada Benign ve Malware sınıfları aynı karakteristikleri göstermektedir. Bu bize ilgili

hipoteze dair sınıflandırıcı davranışlarından bir ipucu yakalamanın oldukça zor olduğunu göstermiştir.

Son olarak [92] ve [93] çalışmalarından aldığımız motivasyon ile kelime ve kelime vektörleri arasındaki entropiden kaynaklı bozulmalar ölçülmüştür. Benign ve Malware örnekler için sinyal entropisi $H_m(.)$:

$$H_m[p(y=1|x)] = -\sum_{x_r} p(y=1|x_r) log\left(p(y=1|x_r)\right) \tag{4.1}$$

formülü ile hesaplanmıştır

Şekil 4.9c'de formülasyon sonuçlarına göre her ne kadar Benign örnekler için entropinin standart sapması beklendiği gibi daha yüksek çıksa da Malware ve Benign ortalama entropileri birbirine yakın çıkmıştır. Bu analiz, hangi kod bölgelerinin Malware olmasından sorumlu olduğunu belirlemenin ve yorumlanabilirlik sağlamanın çok zor olduğunu göstererek, yazılımları tek bir varlık olarak öğrenen konvolüsyonel yaklaşımlara ihtiyaç duyulduğunu ortaya koymaktadır.

4.5.3 Ortak Konvolüsyonel Modelleme

Elde ettiğimiz bulgulara dayanarak, Malware sınıflandırma problemini ele almak için yerelleştirilmiş bir sınıflandırıcı eğitmenin kolay olmadığı anlaşılmaktadır. Malware olmayan bölgelerden kaynaklanan herhangi bir gürültü, sınıflandırıcı performansına hemen zarar vermektedir. Bu gözlem, açıklanabilir çözümler geliştirme yolunda bir zorluk ortaya koysa da Raff ve arkadaşlarının [43] çalışmasının bulgularıyla paralellik göstermektedir; onlar da Malware dosyasının bütünsel yapısını ayırırsak herhangi bir analizin düzgün yapılamayacağını öne sürmektedir.

Bu sorunu çözmek için, bir dizi konvolüsyonel filtreden oluşan bir model benimsemekteyiz. Literatürdeki Malware sınıflandırma görevleri için kullanılan bu tür konvolüsyonel modeller bulunmaktadır. MalConv, Windows Malware yazılımlarında byte tabanlı sınıflandırma yapan Gated-CNN [94] mimarisi

önermiştir. Önerdiğimiz mimari yapımız MalConv ile benzerlikler göstermekte, ancak bazı açılardan ondan farklılıklar arz etmektedir.

4.5.3.1 Opcode Tabanlı Sınıflandırma

Malware sınıflandırması gibi karmaşık görevlerde açıklanabilirlik ve doğruluk elde etmek amacıyla, konvolüsyonel mimarilerin ve opcode özelliklerinin kullanılması etkili bir yöntemdir. Pek çok çalışma, raw (ham) bytelar üzerinde çalışan modeller geliştirse de bu yaklaşımlar genellikle büyük miktarda eğitim verisi gerektirmekte ve raw byteların içerdiği gereksiz bilgiler nedeniyle verimsiz olabilmektedir. Ayrıca, raw bytelar ile yapılan eğitimler, doğru şekilde yapılmazsa model kararsız performanslar sergileyebilir [34].

Bu zorlukları aşmak ve eğitim sürecini daha verimli hale getirmek için, opcode sekansları kullanılabilir. Opcode'lar, giriş uzayının boyutunu azaltarak konvolüsyonel filtreler için doğal bir regülasyon sağlar ve eğitim süresini önemli ölçüde kısaltır. Ayrıca, önceden eğitilmiş opcode temsil vektörleri modele hazır verilerek eğitim hızı arttırılabilir.

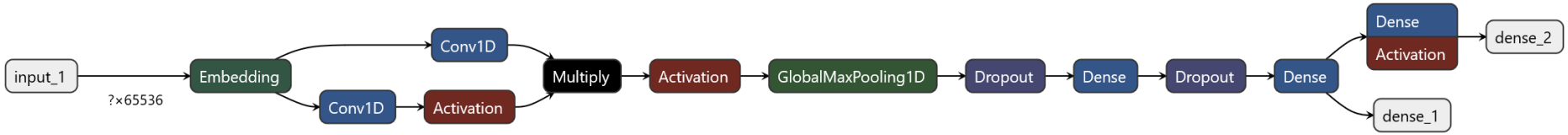
MalConv gibi mimariler, raw bytelar üzerinde çalışarak kötü amaçlı yazılım sınıflandırması yapmaktadır, ancak bu mimariler genellikle header bilgisini ezberler ve dolayısıyla burada yapılan değişikliklere karşı hassas olabilirler [61]. Opcode tabanlı eğitim, bu zayıflıkları aşabilir ve makine yönergelerine odaklanarak daha güvenilir sonuçlar elde edebilir.

4.5.3.2 Davranışsal Değişkenler ile Regüle Edilmiş Sınıflandırma

İkinci stratejimiz, modele "birbiri ile ilişkili ama zayıf davranışsal hedef değişkenlerini" tanıtmaktır. Bu değişkenler Malware'in davranışsal özelliklerini ve kullandığı Anti-Analiz metotlarını içeren etiketlerdir. Her Malware örneğinin belirli bir aileden geldiği ve her ailenin önceden tespit edilmiş sınırlı sayıda davranışsal özelliklere sahip olduğu bilinmektedir. Bu davranışları $\mathbf{z} = \{z_1, z_2, z_3\}$ ile ifade edebiliriz. Örneğin, bazı Malware örnekleri "renaming" işlemi gerçekleştirebilirken, aynı aileden başka bir örnek "string encryption" işlemi

yapabilir. Buradaki temel fikrimiz bu hedef değişkenleri arasındaki ilişkileri ve bağımlılıkları bularak sınıflandırıcı modele tanıtmak, bu sayede modelin çoklu-hedef ile eğitilerek daha doğru çıkarımlar yapmasını sağlamaktır. Model $f_\theta(\mathbf{z}|x)$ ile belirlenen bu değişkenler arasındaki ilişkiyi öğrenir. Burada x örneği, θ parametre setlerini ve $\mathbf{z}$ davranışsal etiketleri göstermektedir. Nihai sınıflandırıcı mimari Şekil 4.10 gösterilmektedir.

Çoklu hedef formülasyonun Malware sınıflandırma modelinde doğal bir regülasyon etkisi yaratacağını öngörüyoruz. Malware davranış hedef değişkenleri ile iyi bir şekilde ilişkilendirilen ancak Malware özelliği göstermeyen bir deseni öğrenmenin, olası olduğunu kabul ediyoruz. Buna ek olarak belirli bir aileden seçilen zararlı yazılımın, aileyle ilişkilendirilen olası davranışsal etiklerin tümüne sahip olmak yerine bir kısmına sahip olması da muhtemeldir. Bu nedenle, belirlediğimiz bu hedef değişkenlerini, daha düşük güvenilirliğe sahip zayıf etiketler olarak düşünmek mantıklı olacaktır. Windows ve Android veri setleri için kullanılan bu davranışsal hedef değişkenlerin tamamı Tablo 4.10 ve Tablo 4.11'de bulunmaktadır.



Şekil 4.10 Önerilen modele genel bir bakış

Yetkinlik
DEFENSE EVASION
DISCOVERY
EXECUTION
PRIVILEGE ESCALATION
ANTI-BEHAVIORAL ANALYSIS
CRYPTOGRAPHY
DATA ACCESS
FILE SYSTEM ACCESS
OPERATING SYSTEM INTERACTION
PROCESS INTERACTION
COLLECTION
IMPACT
PERSISTENCE
ANTI-STATIC ANALYSIS
COMMUNICATION
MEMORY OPERATIONS
COMMAND AND CONTROL
CREDENTIAL ACCESS
HARDWARE

Tablo 4.10 Windows ortamı için seçilen davranışsal hedef etiketleri

Yetkinlik
RENAMING
STRİNG_ENCRYPTİON
DYNAMIC_LOADİNG
NATIVE_PAYLOAD
EVADE_DYNAMİC_ANALYSİS
ADWARE
BACKDOOR
RANSOM
TROJAN-BANKER
TROJAN-DROPPER
TROJAN-SMS
TROJAN
TROJAN-SPY
HACKERTOOL
TROJAN-CLİCKER

Tablo 4.11 Android ortamı için seçilen davranışsal hedef etiketleri

4.5.3.3 Formülasyon

En genel senaryoda, davranışsal özellikler arasında doğrusal olmayan bağımlılıklar olabilir. Ancak çok katmanlı derin öğrenme modelinin bunu öğrenmesi beklenmektedir. Bu doğrultuda, öğrenme sürecinde bu davranışsal özelliklere herhangi bir ağırlık verilmemiştir. Veri setimizde bulunan bazı örnekler için davranışsal hedef etiketleri tespit edilemediyse, örneğin ait olduğu Malware ailesinin tüm davranışsal özelliklerini içerdiğini varsayımsal kabul ettik. Bu sebeple ilgili örnek için davranışsal hedefleri içeren etiket vektörü tamamı 1'den oluşacak şekilde düzenlenmiştir.

Modele sunulan bir örneğin bu hedef davranışsal etiketleri içerip içermediği tespiti model tarafından yapıldığında, bu örneğin Malware olduğuna dair bir pozitif katkı yapması beklenir. Çalışmamızın ilk aşamasında, bu katkıların eşit ağırlıklı ortalamasına karşı bir eşik değeri belirlenerek ilerlediğimizde gördük

ki, davranışsal etiketlerin sayısı arttığında eşik değeri hassasiyeti değişmekte, dolayısıyla anlamsızlaşmaktadır. Burada modele bir katman daha ekleyerek model parametreleri güncellenirken binary (Malware-Benign) sınıflandırma hatasına ek olarak, davranışsal etiket sınıflandırma hatası da göz önüne alacak şekilde geri bildirim yapılması sağlanmıştır. Parametre güncellemeleri, bu iki hata fonksiyonunu minimize aynı anda minimize edecek şekilde aşağıdaki formülde gösterildiği üzere formüle edilmiştir. Binary sınıflandırmada ki hata fonksiyonu için Cross-Entropy, davranışsal etiket hata fonksiyonu içinse MSE (mean-squared error) kullanılmıştır.

$$J(\theta) = \lambda \ell_M(\mathbf{z}, f_\theta(x)) + (1 - \lambda)\ell_C(y, g_\phi(f_\theta(x))) \tag{4.2}$$

Formülde, θ, ϕ model parametrelerini, ℓ_M MSE hatasını, ℓ_C Cross-Entropy hatasını, λ ise binary hata fonksiyonu ile davranışsal etiket hata fonksiyonu arasındaki ağırlık parametresidir. λ değeri tespit edilirken, her iki veri seti içinde hiper parametre keşfi yapılarak en uygun ortak değer belirlenmiştir. Bu değer her iki veri setinde de aynı (0.2) çıkmıştır.

4.5.4 Deney ve Sonuçlar

Deney Ortamı: Önişleme adımı ve deneyler 128gb RAM, Ryzen 9 5900, ve 24gb VRAM'e sahip RTX A5000 ekran kartı içeren bir workstationda yapılmıştır. Kütüphane olarak Tensorflow 2.12 ve Cuda 11.8 kullanılmıştır. Eğitim hızı ve başarımı arasında iyi bir denge sağlamak adına Batch büyüklüğü olarak 16 belirlenmiştir. Nvidia RT sensörlerinin aktif olması için modelin tüm parametreleri 8 ve katları olacak şekilde belirlenmiş, Cudanın yarım hassasiyet (Half Precision) bayrağı set edilmiştir. Embedding katmanındaki ağırlıklar eğitimden önce Gensim kütüphanesi ile hesaplanmış, hesaplanan ağırlıklar yapay zeka modeline verilerek bu katmanın eğitim esnasında güncellemesi engellenmiştir. Deneylerde global seed değeri belirlenmiş, bu seed değeri Tensorflow, Cuda, model ilklendirme ve diğer tüm rastgelelik içeren parametrelerde ortak kullanılmıştır.

Modellerin performansı eğitim, test ve Zero-Day doğrulukları kullanarak

değerlendirildi. Ayrıca, modellerin dayanıklılığını ölçmek için çeşitli sensitivity (hassasiyet) analizi yapılarak test edildi. Her eğitim tekrarında oluşan GPU hesaplamalarındaki rastgeleliği ortadan kaldırmak ve aynı sonuçların yeniden üretilebilirliği sağlamak için her deneyi beş farklı seed (tohum, rastgele sayı) ile tekrarlayıp bunların ortalama doğruluğu rapor ediyoruz. Belirlenen seed global olarak bütün ilgili işlemlere uygulanmıştır. Bunlar:

- Veri seti karıştırma (shuffle) ve bölme (split)
- Tensorflow global rastgelelik değeri
- Model ilklendirme ağırlıkları
- Gpu rastgelelik değeri

şeklindedir.

Kullanılan Modeller: Eğitim ve karşılaştırma için kullanılan modeller aşağıdaki gibidir, parametre ve hiper parametreler ile ilgili bilgi vermek gerekirse, kullanılan her model için hiper parametre keşfi ayrı ayrı yapılmıştır. Seçilen parametreler en iyi başarım ve kısa eğitim süresini sağlayan parametrelerden seçilmiştir. Tablo 4.12'den bu parametreler görülebilir.

- **Apk2Vec [75]:** Önceki çalışmamızda önerilen modeldir. Opcode sekanslarından çıkarılan kelime vektörlerinin ortalamasının ağaç tabanlı makine öğrenmesi algoritmaları ile sınıflandırılması üzerine dayanır.
- **CNN-BiLSTM [54]:** BiLSTM katmaanından önce opcode sekanslarından yüksek seviye anlamlar/özellikler çıkaran bir CNN katmanı ile desteklenen ve denenen LSTM modelleri arasında en yüksek başarım elde eden modeldir. CNN katmanından ötürü, yüksek seviye özelliklerin sayısı, giriş uzunluğuna göre oldukça küçüldüğü için, görece sıralı (sequential) çalışan LSTM katmanı nedeniyle uzayan eğitim süresini önemli ölçüde kısaltması modelin getirdiği bir diğer avantajdır. Bu sayede kabul edilebilir sürelerde giriş uzunluğu ve örnek sayısı fazla veri setleri eğitilebilmektedir.

- **MalConv [43]:** Gated-CNN (Kapılı-CNN) model olan bu model, kullandığı aktivasyon fonksiyonu sayesinde model tarafından belirlenen yüksek seviye özellikleri ağırlıklandırarak sınıflandırma katmanına iletmektedir.
- **TRConv [76]:** Bizim tarafımızdan önerilen, davranışsal hedef değişkenleri ile regüle edilmiş Gated-CNN modelidir.

Tablo 4.12 Kullanılan modellerin hiper parametreleri

MalConv		**TRConv**		**CNN-BiLSTM**		**Apk2Vec**	
Embedding Size	32	Embedding Size	32	Embedding Size	32	Embedding Size	32
Filter Size	8	Filter Size	8	Filter Size	8	Input Len	65536
Filter Stride	1	Filter Stride	1	Filter Stride	1	Classification Algorithm	RandomForest
Filter Count	128	Filter Count	128	Filter Count	128	Seed Count	5
Input Len	65536	Input Len	65536	LSTM Units	128		
Dropout	20%	Dropout	20%	Input Len	65536		
Dense Layer	64	Dense Layer	64	Dropout	10%		
Optimizer	RMSProp	Optimizer	RMSProp	Dense Layer	64		
Classification Loss	Cat.CrossEntropy	Capability Loss	MSE	Optimizer	RMSProp		
Batch Size	16	Classification Loss	Cat.CrossEntropy	Classification Loss	Cat.CrossEntropy		
Mixed Precission	True	Loss Weights	0,2	Loss Weights	0,2		
Epoch Shuffle	True	Batch Size	16	Batch Size	16		
Seed Count	5	Mixed Precission	True	Mixed Precission	True		
		Epoch Shuffle	True	Epoch Shuffle	True		
		Seed Count	5	Seed Count	5		

Zero-Day Analizi: Malware analizi literatüründe, Zero-Day analizi iyi genelleme göstergesi olarak kabul edilmektedir. Bu analizdeki önemli nokta, eğitim sırasında bazı ailelerden gelen örnekleri bilinçli olarak dışarıda bırakmak ve yalnızca diğer ailelerden gelen örneklerle modeli eğitmektir. Sınıflandırıcı problemi genelleştirmiş ise (yazılımları davranışlarına göre sınıflandırabiliyorsa) yeni ailelerden gelen örnekler üzerinde iyi bir tahmin performansı vermesi beklenir.

Önerdiğimiz modelin Zero-Day performansını test ederek model dayanımını ve genelleştirme kapasitesini ölçmekteyiz. Daha önce [95], en fazla örneğe sahip 20 aile için Zero-Day analizi yapmış ve her seferinde bir aile dışarıda kalacak şekilde eğitimden çıkarmıştı. Ancak, böyle bir Zero-Day testinin ön yargı oluşturabileceğine inanıyoruz. Aynı zamanda, sınıflar arasındaki denge problemini göz ardı ederek Zero-Day deneyini yapmanın modelin performansını ölçmek adına fazla iyimser olduğunu düşünüyoruz. Bu nedenle Zero-Day senaryosu için, daha dengeli ve heterojen bir dışarıda bırakma (left-out) stratejisi izleyerek, yeterince fazla sayıda örnek içeren aileleri rastgele seçerek toplu şekilde eğitimden çıkardık.

Bazı çalışmalar eğitim setinde dışarıda bırakılan aileye ait en az bir örneğe izin verilmesi gerektiğini öne sürmektedir. Biz deneylerimizde daha kötümser bir senaryo üzerinde çalışmak adına dışarıda bırakılan ailelere ait hiçbir örneği eğitim yada test setinde kullanmadık.

λ *parametresi*: Önerdiğimiz TRConv modelinde, λ parametresi modelimizin davranışsal hedef değişkenleri tespit etme ve nihai Malware/Benign kararını tahmin etme arasında hangisine ne kadar öncelik vereceğini belirlemektedir. Android deneyi için aynı aileden gelen tüm Malware örneklerini, ilgili ailenin sahip olduğu tüm davranışsal hedef değişkenlerine sahip olduğu şeklinde kötümser bir yaklaşım ele aldık. Windows deneyinde CAPA kütüphanesi ile örnek özelinde bu davranışsal etiketleri elde etsek de hiç etiket elde edilemeyen örnekler için davranışsal vektörler, örneğin Benign ya da Malware olmasına göre hepsi 1 veya hepsi 0 olacak şekilde verildi. Bu varsayımlar, ailenin bazı üyeleri için geçerli olmayabilir, bu nedenle modelimizin bu davranışsal hedef değişkenlerini tam olarak öğrenmesini istemedik. Bu parametreyi belirlemek için ampirik gözlemler ve optimizasyon sonuçlarına dayandırarak, tüm deneyler için λ değerini 0.2 olarak belirledik. Optimizasyon

algoritması için RMSProp kullandık. Bu algoritma, gradyanların karesinin hareketli ortalamasını korumaktadır. RMSProp için öğrenme oranı olarak $1e-4$, önceki gradyan faktörü olarak $\rho = 0.9$, epsilon toleransı olarak $1e-07$ belirledik.

Tablo 4.13 Argus veri setinde ailelere göre örnek sayıları

Airpush	7843	FakeAngry	10	GoldDream	53	Mecor	1820	Roop	48	Ztorg	17
AndroRAT	46	FakeAv	5	Gorpo	32	Minimob	203	RuMMS	402	Triada	210
Andup	43	FakeDoc	21	Gumen	145	Mmarketpay	14	SimpleLocker	165	Univert	10
Aples	21	FakeInst	2168	Jisut	548	MobileTX	17	SlemBunk	174	UpdtKiller	24
BankBot	648	FakePlayer	21	Kemoge	15	Mseg	235	SmsKey	165	Utchi	12
Bankun	70	FakeTimer	12	Koler	69	Mtk	67	SmsZombie	9	Vidro	23
Boqx	215	FakeUpdates	5	Ksapp	36	Nandrobox	76	Spambot	15	VikingHorde	7
Boxer	44	Finspy	9	Kuguo	1198	Obad	9	SpyBubble	10	Vmvol	13
Cova	17	Fjcon	16	Kyview	174	Ogel	6	Stealer	25	Winge	19
Dowgin	3385	Fobus	4	Leech	128	Opfake	10	Steek	12	Youmi	1301
DroidKungFu	546	Fusob	1276	Lnk	5	Penetho	18	Svpeng	13	Zitmo	24
Erop	46	GingerMaster	128	Lotoor	328	Ramnit	8	Tesbo	5	Total	24516

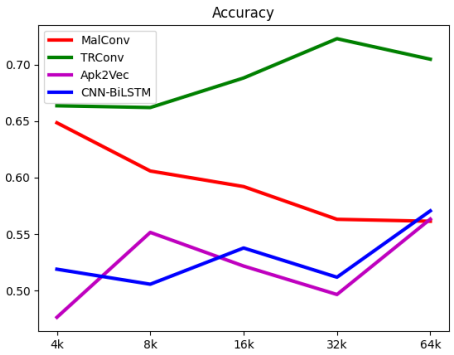
4.5.4.1 Android Deney Sonuçları

Android deneyinde veri seti olarak daha önce kullandığımız AMDARGUS Malware veri setine ve beraberinde Yıldız Teknik Üniversitesi Akıllı Sistemler Lab. ve COMODO iş birliği ile oluşturduğumuz Benign veri seti genişletilerek kullanılmıştır. Genişletme işlemi, AndMal2017 ve MalDroid2020 veri setlerinden Benign örnek alarak yapılmıştır. Veri setinde opcode sekansları çıkartıldıktan sonra 100 opcode'dan az olan ve 5 milyon opcode'dan fazla olan örnekler veri setinden atılmıştır. Nihai veri seti 22899 Malware, 22279 Benign örnek ve 71+1 aile içermektedir. Malware aileleri arasındaki örnek sayı dağılımına Tablo 4.13'dan bakılabilir.

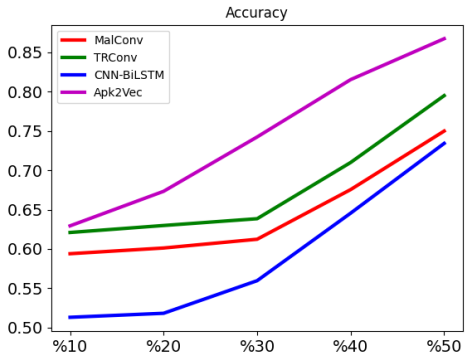
APK dosyalarından opcodeların çıkartılması için Apktool kullanılmıştır. Apktool bize, mnenomic, adres, operand ve kod olmayan tanımlamalar içeren smali dosyaları vermektedir. Mnemonic haricindeki tüm alanlar silindikten sonra bütün opcodelar tek bir dosyada toplanıp ilk 65 bin kod padding veya truncate işlemleri yapılarak alınmıştır.

Referans deney sonuçları Tablo 4.14 ve Zero-Day için Tablo 4.15'da verilmiştir.

OPCODE UZUNLUĞUNA GÖRE SONUÇLAR: Raff ve arkadaşları [96] çok uzun giriş uzunlukları ile Malware sınıflandıran bir yöntem önermiştir. Ancak, BiLSTM gibi birçok mimari için, opcode uzunluğu parametresi hala bir tasarım kriteridir ve opcode uzunluğunun nihai performansı nasıl etkilediği net değildir. Bu deneyde, modellerin opcode dizi uzunluklarındaki varyasyonlara karşı tepkilerini ölçtük. Apktool sonucunda ortaya çıkan birden çok .Smali dosyalarının birleştirme sırası



(a) Opcode uzunluğuna göre Zero-Day performansı



(b) Benign dengesizliğine göre Zero-Day performansı

Şekil 4.11 AMDARGUS veri setinde çeşitli deney sonuçları a) Değişken opcode uzunluklarına göre b) Veri seti Benign aleyhinde dengesizliğine göre

rastgele olduğundan, model giren opcode sekans uzunluğu arttıkça performansta kademeli bir lineer artış gözlemlemeyi bekliyoruz. [54] çalışmasında, opcode uzunluğu parametre aralığını 2k'dan 10k'ya kadar test etmişti. Bizim veri setimizdeki istatistiklere göre, test setinin %38'inde 65k'dan fazla opcode bulunmakta ve Zero-Day Malware setindeki örneklerin %40'ında 65k'dan fazla opcode bulunmaktadır. Bu opcode uzunluklarından hareketle, sınıflandırma için kritik olabilecek etkili bölümleri göz ardı etme şansını en aza indirmek amacıyla bu analizi daha büyük bir parametre aralığında genişletiyoruz.

Şekil 4.11a bize giriş uzunluğunun 4k ile 64k arasında değişimine göre modelin performans cevabını göstermektedir. Burada CNN-BiLSTM ve Apk2Vec modellerinde opcode uzunluğu arttıkça, performansta iyileşme görmekteyiz. MalConv modeli belirgin bir performans düşüşü yaşamaktadır. Opcode uzunluğu arttığında, MalConv ağının muhtemelen filtrelerinin öğrenmekte olduğu özelliklerin değiştiği bir etiket kayması yaşadığını düşünmekteyiz. Önerdiğimiz modelde de küçük bir performans düşüşü yaşanmaktadır. Bu, kısa vadede, giriş boyutu arttıkça özellikleri aynı anda öğrenmenin zorlaştığının bir göstergesi olabilir. Ancak, performans düşüşü Malconv'a göre nispeten daha önemsiz büyüklükte olmuş ve modelimiz ortalama başarım olarak hala temel modellerden daha üstündür.

SINIF DENGESİZLİĞİNE GÖRE SONUÇLAR: Malware sınıflandırma görevlerinde, özellikle çok sınıflı senaryolarda sınıf dengesizliği yaygın bir

durumdur. Allix ve arkadaşları [97], eğitim setinde daha fazla Benign örnek eklemenin precision değerini arttırdığı fakat recall değerini azalttığını savunmaktadır. Bu nedenle, Malware/Benign oranının azalmasının genelde daha kötü performans vereceğini iddia etmektedirler. Bunu, veri setinden [%10, %20, %30, %40, %50] oranlarında Benign örnekleri çıkararak doğruladık ve genel olarak, dengesizlik Benign örneklerin lehine olduğunda modellerin ortalama doğruluğunun azaldığını gördük. Bu olgu, Malware ve Benign örnekler arasındaki entropik farkla açıklanabilir. Eğitim setinde Benign örnek baskın hale geldikçe, eğitim setinin ortalama entropisi nispeten artacak ve bu da daha fazla ön işlem yapılmadan Malware örnekleri sınıflandırmayı zorlaştıracaktır. Bu nedenle, sınırlı veri rejiminde etkili bir öğrenme için Malware örnekleri toplamanın çok daha faydalı olduğuna kanaat getirdik.

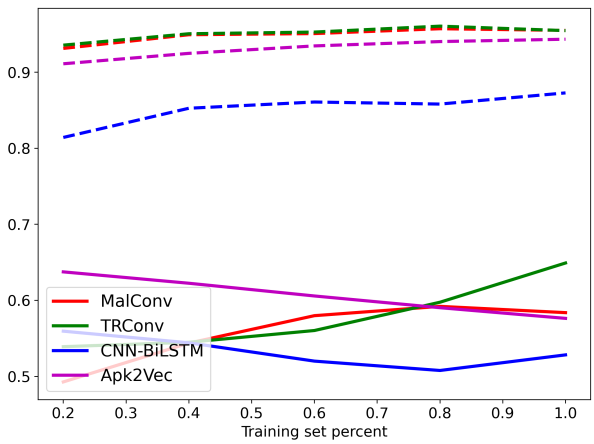
Şekil 4.11b, bu alanda Zero-Day veri seti üzerindeki sonuçlarımızı göstermektedir. Genel bir eğilim olarak, tüm modeller Benign örnek azaldıkça Zero-Day performansında daha iyi sonuçlar vermektedir. Eğitim setinden daha fazla Benign örnek çıkardıkça, modeller bir Zero-Day örneğini Malware olarak etiketlemeye daha yatkın hale gelmektedir. %20'ye kadar çıkarma işlemi, modellerin küçük bir performans artışı göstermesine neden olurken, Apk2vec'in modelinin kararı hızla değişmektedir. Bu performans artışı Apk2vec için bir kazanç olarak yorumlanabilse de, Apk2vec'in Benign örnek çıkarma işlemi ile tanıtılan sınıf dengesizliğini çok hızlı bir şekilde benimsemesinden modelin ezberlemeye yatkın olabileceğinden şüphelendik. %20'ye kadar olan çıkarma işlemlerinde diğer modellerdeki küçük performans artışları bize veri seti dengesizliklerden hemen etkilenmediklerini dolayısıyla problemi genelleştirme noktasında daha başarılı olduklarını göstermektedir.

ALT KÜME ANALİZİ:

Akademik araştırmalarda kullanılan veri setleri ile, gerçek dünyada kullanılan Anti-Malware sistemlerinde kullanılan makine öğrenmesi tabanlı sınıflandırıcıların eğitimi için kullanılan veri seti boyutlarında oldukça büyük farklar bulunmaktadır. Örneğin Mandiant [77] kabul edilebilir FPR ve FNR değerlerine sahip bir modelin eğitimi için en az 20 milyon Malware ve 20 Milyon Benign yazılım kullanılmasın

gerektiğini savunmaktadır. Bizim kullandığımız veri seti literatürde kapsamlı bir veri setlerinden biri olmasına rağmen, ilgili çalışmada iddia edilen veri seti boyutunun 1000'de biri kadardır. Bu deneyde veri seti boyutunun sınıflandırıcı performanslarına etkisini incelemek adına orjinal veri setine göre %20 ile %100 aralığı içinde alt veri setleri oluşturularak eğitim ve sınıflandırma başarıları tekrar ölçülmüştür.

Şekil 4.12, rastgele seçilen alt veri kümelerinden eğitilen modellerin ortalama test ve Zero-Day başarımlarını çizdik. Her adımda, daha büyük bir alt küme kullanılarak modeller eğitildi. Eğitim veri seti boyutu arttıkça, APK2Vec ve CNN-BiLSTM'nin performansının düşmekte oluşu, bu modellerle elde edilen yüksek performansların ezberleme nedeniyle kaynaklanabileceğini önermektedir. Eğitim seti boyutu arttıkça, modelimiz daha iyi bir Zero-Day performansı elde etmekte, bu da genelleme yapabilme yeteneğine bir atıf yapmaktadır. En yüksek performans, tüm veri setini kullandığımızda elde edilmiştir. Temsil yeteneği yüksek bir alt veri seti oluşturma zor bir işlem olduğundan, modelimizde bulunan davranışsal hedef değişkenleri hızlıca başarıma etki etmiştir.



Şekil 4.12 Android ortamı için veri setinin büyümesine karşı Test (kesikli çizgiler) ve Zero-Day (düz çizgiler) başarımları

Tablo 4.14 Android ortamı test veri seti için sınıflandırma sonuçları

Model	ACC	FP	FN	TP	TN	F1
MalConv	0.9551	313	295	6575	6371	0.9551
CNN-BiLSTM	0.8729	1246	476	6394	5438	0.8718
Apk2Vec	0.9435	384	382	6488	6300	0.9435
TRConv	0.9548	430	183	6687	6254	0.9548

Tablo 4.15 Android ortamı Zero-Day veri seti için sınıflandırma sonuçları

Model	ACC	FP	FN	TP	TN	F1
MalConv	0.5614	0	709	908	0	0.3531
CNN-BiLSTM	0.5706	0	694	923	0	0.3583
Apk2Vec	0.5633	0	706	911	0	0.3603
TRConv	0.7049	0	477	1140	0	0.4121

4.5.4.2 Windows Deney Sonuçları

Windows deneyinde veri seti olarak Malware Open-Source Threat Intelligence Family (MOTIF) [84] kullanılmıştır.

Bu veri seti hem açık kaynak güvenlik raporlarından hem de Anti-Malware imzalarından türetilen izleri ve isimleri içeren kapsamlı bir eşleşme sağlayan ilk veri setidir. Oldukça çeşitli ve güncel bir Malware örnek seti içermektedir. Bu veri setindeki Malware örnekleri, diğer veri setlerine kıyasla daha az etiketleme hatasına sahiptir. Örnekler başlangıçta 400'ün üzerinde aileden toplanmıştır. İki örnekten az olan aileleri çıkardıktan ve hatalı örnekleri ayıkladıktan sonra, 295 aileden 2601 örnek kalmıştır. Disassemble işlemi için PE-BEAR yazılımı kullanılmıştır. Bu program .text bölümü haricindeki bölümlerde bulunan kodları da çıkartmıştır.

MOTIF veri setinde Benign örnekler bulunmadığından, ikili sınıflandırma için eğitim setini 2K Benign örnekle genişletmek amacıyla son zamanlarda yayınlanan PEMDB [85] veri setini kullandık. Benzer şekilde, [75] çalışmasında olduğu gibi, PEMDB veri setinden 100 zararlı yazılım örneğini eğitim setine ve 1400 tanesini Zero-Day veri setine ekledik.

MOTIF veri seti, Malware etiketlerini aile seviyesinde sunmaktadır. Davranışsal hedef değişkenleri Mandiant'ın CAPA aracı kullanılarak çıkartılmıştır. Bu araç, PE dosyasını tarar ve PE dosyasını karakterize eden hiyerarşik özellikler listesi çıkarır. Tüm özellik listesi Tablo 4.10 sunulmuştur.

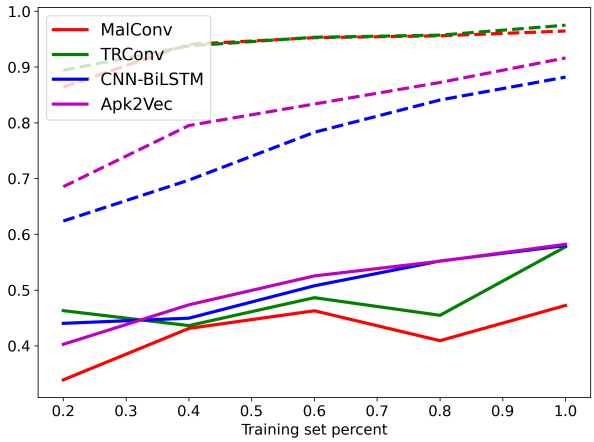
Şekil 4.13'da her model için farklı subset boyutlarında ki doğrulukları çizilmiştir. Windows ortamı için referans deney sonuçları Tablo4.16 ve Zero-Day için Tablo 4.17'da verilmiştir. Genel olarak, TRConv, CNN-BiLSTM ve Apk2vec Zero-Day veri setinde iyi performans göstermektedir. Test veri setinde ise en iyi performansı TRConv ve MalConv göstermiştir. Bu deney, önerdiğimiz modelin çok küçük değişiklikler ile farklı ortamlarda uygulanabilir olduğunu göstermiştir. Önerdiğimiz model MalConv'a kıyasla davranışsal hedef değişkenlerin etkisiyle daha yüksek performans göstermiştir.

Tablo 4.16 Windows ortamı test veri seti için sınıflandırma sonuçları

Model	ACC	FP	FN	TP	TN	F1
MalConv	0.9647	8	23	638	216	0.9550
CNN-BiLSTM	0.8820	13	92	569	211	0.8596
Apk2Vec	0.9164	2	72	589	222	0.8990
TRConv	0.9751	12	10	651	212	0.9668

Tablo 4.17 Windows ortamı Zero-Day veri seti için sınıflandırma sonuçları

Model	ACC	FP	FN	TP	TN	F1
MalConv	0.4823	0	725	675	0	0.3160
CNN-BiLSTM	0.5830	0	584	816	0	0.3672
Apk2Vec	0.5860	0	580	820	0	0.3693
TRConv	0.5954	0	566	834	0	0.3696



Şekil 4.13 Windows ortamı için veri setinin büyümesine karşı Test (kesikli çizgiler) ve Zero-Day (düz çizgiler) başarımları

4.5.4.3 Hassasiyet Analizi

Bu bölümde, test ettiğimiz modellerin çeşitli Adversarial saldırılar karşısında nasıl tepki verdiğini göstereceğiz. Modelin giriş katmanına verilen özellik vektörlerinde yapılan rastgele ekleme ve çıkarmalar ile bu Adversarial saldırılar simüle edilecektir. Bu tip saldırı tasarımlarına odaklanan akademik çalışmaların bir çoğu rastgele yada önceden tasarlanmış sekansları ekleyerek uygulamaktadır [62]. Bizde bu çalışmada bu tip bir saldırı üzerine odaklanacağız. Ekleme stratejisi olarak aşağıdakileri uygulayacağız.

- RA: Bu strateji rastgele opcodeları, girdi opcode sekansında rastgele bölgelere eklemektedir. Bu strateji opcode sekans paternlerini tamamen bozmakta ve diğer stratejilere nazaran model tarafından çözülmesi daha zor bir saldırıdır.
- RAB: Bu strateji eklenen opcodeların orjinal sekansın en başına eklenmesiyle yapılmaktadır. Daha önce bahsedildiği üzere Malware sınıflandırma görevinde konvolüsyonel modellerin sınıflandırma başarımının genellikle girdilerin ilk bölümlerinden çıkarılan paternlere borçlu olduğu iddia edilmektedir. Bu saldırı tipi, bu zafiyetle beraber, modellerin "positional-invarience" (konum önemsizliği)'ni ne kadar doğru kodlayabildiğini gösterecektir.
- RAE: Son strateji, rastgele opcode sekanslarını orjinal girdinin en sonuna eklemektedir. Buradaki amaç girdi sonundaki gereksiz bölümlerin modelleri ne ölçüde kandırabildiğini ölçmektir.

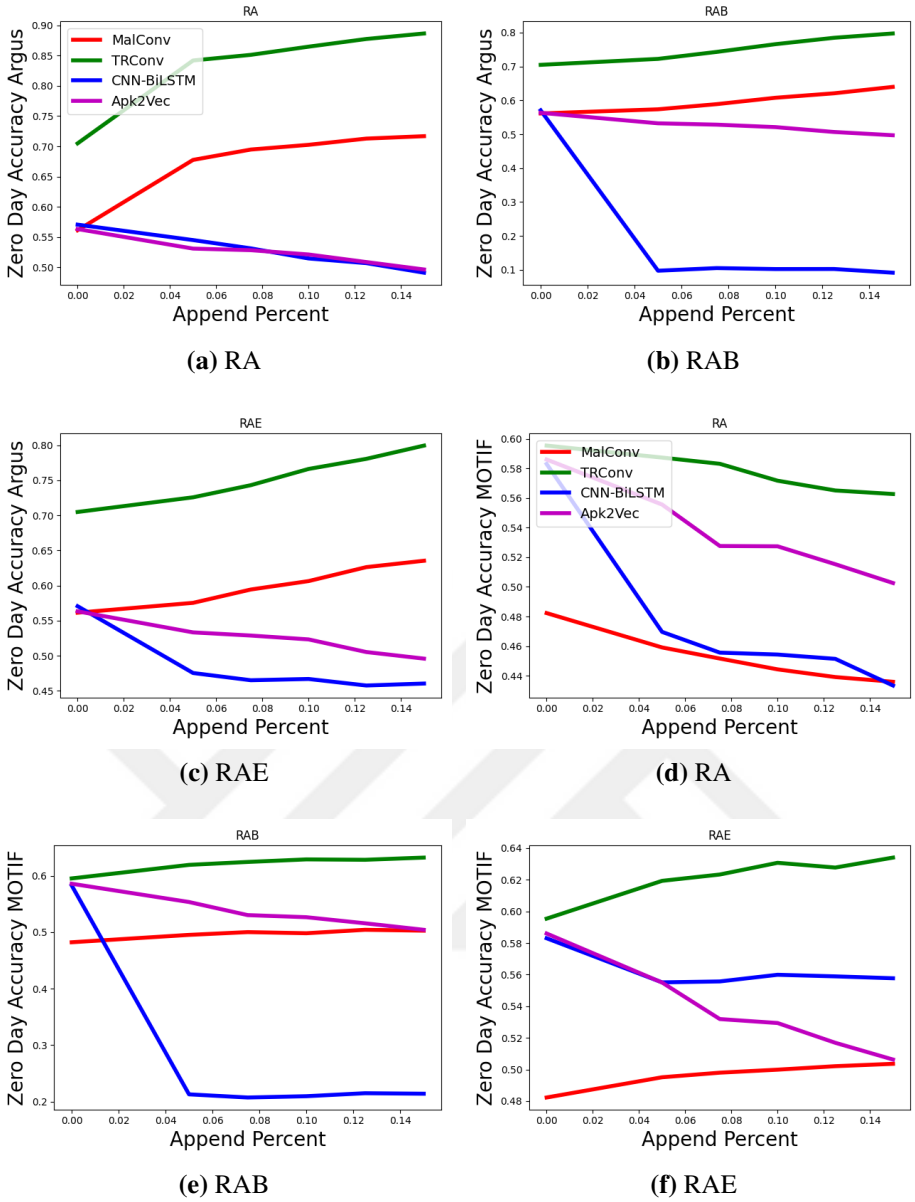
Tüm deneylerde orjinal girdi uzunluğunun %5'i kadar rastgele ekleme ile başlayıp, dizi uzunluğunun %15'ine kadar kademeli olarak arttırdık. Saldırıları basit tutmak için eklenecek opcodeları uniform bir dağılımdan örnekledik. Opcodeların birlikte görülme istatistiklerini dikkate alarak daha karmaşık saldırı teknikleri de mümkündür. Ancak, bu çalışmanın kapsamı dışında olup, bunu gelecekteki çalışmalara bırakıyoruz.

Bulgularımızı Şekil 4.14'de göstermekteyiz. AMDArgus veri setinde, MalConv ve

TRConv, test ve Zero-Day gün setindeki değişimlere karşı dayanıklı görünmektedir. Ekleme oranı arttıkça, bu iki modelin Zero-Day başarımı artmaktadır. Bu, MalConv ve TRConv modelinin ezberleme imkanın diğer karmaşık modellere göre (CNN-BiLSTM gibi) daha zor olmasından kaynaklanıyor olabilir. Apk2Vec küçük bir performans düşüşü yaşarken, CNN-BiLSTM keskin performans düşüşleri yaşamaktadır. Özellikle RAB saldırı için Zero-Day tahmini CNN-BiLSTM için tamamen olanaksız hale gelmiştir. CNN-BiLSTM'nin hafıza nöronlarının kafası çok hızlı bir şekilde karışmaktadır. Unutulmaması gereken bir diğer nokta, LSTM modellerinin yalın hallerinin çok uzun sekanslara karşı başarımı oldukça düşüktür. Bu modeller uzun sekanslarda bilginin büyük bir kısmını sekansın en başından almaktadır.

Şekil 4.14b'de, MOTIF veri seti için saldırı sonuçlarını göstermekteyiz. RA saldırıları altında, tüm modellerin performansı hızla düşmektedir. Daha az eğitim verisine sahip olmak, tüm modelleri olumsuz etkilemektedir. RAE saldırılarında, Apk2Vec önemli ölçüde tahmin gücünü kaybederken, diğer modellerde ise küçük bir performans değişikliği gözlemlenmiştir.

Genel olarak, MalConv ve TRConv'un MOTIF veri setinde AMDArgus veri setine kıyasla rastgele saldırılardan daha fazla etkilendiğini gözlemliyoruz. Ayrıca, her iki senaryoda da kod sekanslarının başına rastgele opcode eklediğimizde, LSTM modeli en çok etkilenen model olmaktadır. Bu gözlem, herhangi bir LSTM eğitim metodolojisinin, eğitim setini varsayılan olarak kaydırılmış (shifted) opcode artırımları ile genişletmesi gerektiğini vurgulamaktadır. Ancak, bu tür genişletmeler görüntü tabanlı inputlarda bütünlüğünü bozmadan yapılabilse de opcode dizileri için doğru olmayabilir. Bu nedenle, TRConv ile bu tür ek artırma mekanizmaları kullanmadan da makul bir performans elde edebilmekteyiz.



(a) RA **(b)** RAB

(c) RAE **(d)** RA

(e) RAB **(f)** RAE

Şekil 4.14 Her model için hem (APK için AMDArgus, Windows için MOTIF) veri seti üzerindeki dayanıklılık sonuçlarının analizi. Üst sıra: AMDArgus veri setindeki a) rastgele ekleme b) başa ekleme c) sona ekleme saldırıları. Alt sıra: MOTIF veri setindeki a) rastgele ekleme b) başa ekleme c) sona ekleme saldırıları.

5
SONUÇ

Yaptığımız çalışmada öğrenme tabanlı yöntemlerin Malware sınıflandırma görevinde oldukça umut vaat edici olduğunu göstermiş bulunmaktayız. Statik analiz ile kolay ve hızlıca çıkartılabilen davranışsal hedef değişkenlerin modellere tanıtılması ile literatürde bulunan state-of-art yöntemlere önemli bir katkı yapmış bulunmaktayız. Bu değişkenler ayrıca modellerin problemi genelleştirme başarısını da arttırmış bulunmaktadır.

Yine de çalışmamız etkili modeller geliştirmek için dikkat edilmesi gereken noktaların altını çizmiştir. Bunlardan ilki temsil yeteneği yüksek kaliteli veri setleri üzerinde çalışması gerektiğidir. Diğer bir nokta byte ya da opcode sekanslarının yalın hallerinin anlamsal olarak bir bilgi içermediğidir. CNN gibi Yüksek seviye özellik çıkarımı yapan katman olmadan doğrudan karmaşık LSTM ağlarına input olarak opcode sekanslarını verdiğimizde modellerin problemi öğrenemediğini gözlemledik. Dolayısıyla kod bloklarından yüksek seviyeli davranışsal çıkarımlar yapabilen bir katman, hedef modelin bir bölümünü işgal etmelidir.

Dikkat edilmesi gereken bir diğer nokta, ağaç tabanlı basit modeller ile elde edilen yüksek başarımların problemi genelleştiremediğinin kanıtıdır. Yaptığımız denemelerde, Apk2Vec modeli, çalıştığı veri setinde oldukça yüksek başarımlar vaat ederken, veri setindeki iterasyon ve değişimlerden en çok etkilenen model de olmuştur. Bu ve benzeri modeller temsil yeteneği düşük veri setlerinde yüksek başarımlar vermektedir. Bunun tam zıttı olarak, Malware sınıflandırma probleminde karmaşık modeller ise veri setini ezberlemeye daha yatkındır. Sonuçlarını bu çalışmada paylaşmadığımız katmanlı LSTM mimarileri, Attention,

Hiyerarşik Multi Modal konvolüsyon ağları gibi modeller, elimizdeki veri seti ve çıkardığımız özelliklere istinaden problemi genelleştirmede anlamlı bir katkı yapamamışlardır.

Önerdiğimiz model, yaklaşık 50 bin örnek ile tek bir GPU üzerinde 5 ile 10 dakika arasında eğitilebilmektedir. Dolayısıyla örneğin 40 milyon örneğin eğitimi için gerekli olan işlem gücü ve zaman kısıtı, bireysel araştırmacıların dahi karşılayabileceği makul seviyelerdedir. Bununla beraber modele tanıtılan davranışsal hedef değişkenleri hızlıca modelin performansını arttırmış ve gerçek dünya problemlerine karşı daha dayanıklı hale getirmiştir. Buda bize hedefi yazılım sınıflandırmasından, davranış sınıflandırmasına kaydırma noktasında önemli ipuçları vermektedir.

GELECEK ÇALIŞMALAR: Yaptığımız çalışmalar sonucunda Malware sınıflandırma konusunda 3 önemli yol gözlemlemekteyiz.

Bunlardan ilki statik analiz yöntemleri ile sadece dinamik analiz de elde edilen özelliklerin elde edilebilmesi için gerekli algoritmaların geliştirilmesidir. Literatürde "Symbolic Execution" olarak bilinen yöntemler günümüzde aktif araştırma konusudur. Özellikle Assembly dilinin deterministik ve kesin oluşu ve gramer yapısından dolayı otomatlar tarafından rahatlıkla tanınabilmesi gibi özellikleri nedeniyle sembolik çalıştırma statik analiz ile sadece dinamik analizde elde edebileceğimiz özellikleri elde etmemize olanak sağlamaktadır.

Bir diğer gidilebilecek yol ise sadece Assembly ile LLM modelleri eğitmek ve bunları çeşitli yöntemler ile fine-tune ederek Malware sınıflandırma görevinde kullanmak olacaktır. Doğal dillere kıyasla kapsam çok daha dar olduğu için Bert ya da GPT büyüklüğünde modellerin eğitilmesi, gözlemlediğimiz kadarıyla oldukça yeterli olacaktır. Buna alternatif olarak açık-kaynak LLM modellerinin programlama dilleri üzerinde uzmanlaşmış varyasyonları (llama-code gibi) üzerinde fine-tune işlemleri ile benzer bir sonuç elde edilebilir. Fakat burada daha işlem gücü ve kaynağa ihtiyaç olacaktır.

Son olarak, alt seviye anlam (döngü, dallanma, kontrol) içeren kod blokları ile yüksek seviye anlam içeren kod blokları (örneğin, asal sayı test etme) matematiksel

olarak ifade edilebilir ve otomatlar tarafından tanınabilir durumdadır. Dolayısıyla kapsamlı ön işleme adımı ile, opcode sekansları kullanmak yerine bu yüksek seviye özellikler kullanılması, başarımı ve genelleştirmeyi önemli ölçüde arttıracaktır.

KAYNAKÇA

[1] T. Mobiles, *Malware statistics 2020: A look at malware trends by the numbers*, Mevcut: https://www.tigermobiles.com/blog/malware-statistics/, Erişim: 19 Haziran 2024. [Çevrimiçi].

[2] S. Dedectives, *Malware and virus statistics, trends & facts*, Mevcut: https://www.safetydetectives.com/blog/malware-statistics/, Erişim: 19 Haziran 2024. [Çevrimiçi].

[3] C. C. Ventures, *Cybercrime to cost the world $10.5 trillion annually by 2025*, Mevcut: https://cybersecurityventures.com/hackerpocalypse-cybercrime-report-2016/, Erişim: 19 Haziran 2024. [Çevrimiçi].

[4] S. S. Dergilik, *Türkiye en çok siber saldırıya uğrayan üçüncü ülke*, Mevcut: https://www.savunmasanayiidergilik.com/tr/HaberDergilik/Turkiye-en-cok-siber-saldiriya-ugrayan-ucuncu-ulke, Erişim: 19 Haziran 2024. [Çevrimiçi].

[5] Comparitech, *Malware attack statistics and facts for 2024: What you need to know*, Mevcut: https://www.comparitech.com/antivirus/malware-statistics-facts/, Erişim: 19 Haziran 2024. [Çevrimiçi].

[6] R. G. LLC, *Traditional anti-malware software effectiveness*, Mevcut: https://riskgroupllc.com/traditional-anti-malware-software-effectiveness/, Erişim: 19 Haziran 2024. [Çevrimiçi].

[7] L. Jobs, *44 worrying malware statistics to take seriously in 2023*, Mevcut: https://legaljobs.io/blog/malware-statistics, Erişim: 19 Haziran 2024. [Çevrimiçi].

[8] J.-W. Kim, Y.-S. Moon, M.-J. Choi, *An efficient multi-step framework for malware packing identification*, 2022. arXiv: 2208.08071.

[9] R. Moskovitch, C. Feher, N. Tzachar, *et al.*, "Unknown malcode detection using opcode representation," in *European Conference on Intelligence and Security Informatics*, Berlin, Germany: Springer, 2008, pp. 204–215.

[10] N. McLaughlin, J. M. del Rincon, B. Kang, *et al.*, "Deep android malware detection," in *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, New York, NY, USA: ACM, 2017, pp. 301–308.

[11] K. O'Shea, R. Nash, *An introduction to convolutional neural networks*, 2015. arXiv: 1511.08458.

[12] E. B. Karbab, M. Debbabi, A. Derhab, D. Mouheb, "Maldozer: Automatic framework for android malware detection using deep learning," *Digital Investigation*, vol. 24, S48–S59, 2018.

[13] Y. Awad, M. Nassar, H. Safa, "Modeling malware as a language," in *Proceedings of the 2018 IEEE International Conference on Communications (ICC)*, Kansas City, MO, USA: IEEE, 2018, pp. 1–6.

[14] T. Mikolov, K. Chen, G. Corrado, J. Dean, *Efficient estimation of word representations in vector space*, 2013. arXiv: 1301.3781.

[15] K. Xu, Y. Li, R. H. Deng, K. Chen, "Deep refiner: Multi-layer android malware detection system applying deep neural networks," in *Proceedings of the 2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, London, UK: IEEE, 2018, pp. 473–487.

[16] M. Yousefi-Azar, L. Hamey, V. Varadharajan, S. Chen, "Learning latent byte-level feature representation for malware detection," in *International Conference on Neural Information Processing*, Cham: Springer International Publishing, 2018, pp. 568–578.

[17] Y. Ye, S. Hou, L. Chen, *et al.*, "Out-of-sample node representation learning for heterogeneous graph in real-time android malware detection," in *Proceedings of the 28th International Joint Conference on Artificial Intelligence*, Macao, China: AAAI Press, 2019, pp. 4150–4156.

[18] Y. Zhang, Y. Sui, S. Pan, *et al.*, "Familial clustering for weakly labeled android malware using hybrid representation learning," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 3401–3414, 2020.

[19] X. Ge, Y. Pan, Y. Fan, C. Fang, "Amdroid: Android malware detection using function call graphs," in *Proceedings of the 2019 IEEE 19th International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, IEEE, Sofia, Bulgaria, 2019, pp. 71–77.

[20] C. Corinna, V. Vladimir, "Support-vector networks," *Springer Machine learning*, vol. 20, pp. 273–297, 1995.

[21] A. Pektas, T. Acarman, "Deep learning for effective android malware detection using api call graph embeddings," *Soft Computing*, vol. 24, pp. 1027–1043, 2020.

[22] Y. Fan, S. Hou, Y. Zhang, Y. Ye, M. Abdulhayoglu, "Gotcha-sly malware! scorpion: A metagraph2vec based malware detection system," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, London, UK: Association for Computing Machinery, 2018, pp. 253–262.

[23] S. Wang, Z. Chen, Q. Yan, *et al.*, "Deep and broad url feature mining for android malware detection," *Information Sciences*, vol. 513, pp. 600–613, 2020.

[24] J. Jiang *et al.*, "Android malware family classification based on sensitive opcode sequence," in *Proc. IEEE Symp. Comput. Commun. (ISCC)*, 2019, pp. 1–7.

[25] C. Catal, H. Gunduz, A. Ozcan, "Malware detection based on graph attention networks for intelligent transportation systems," *Electronics*, vol. 10, p. 2534, 2021.

[26] I. Goodfellow *et al.*, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, vol. 27, Curran Associates, Inc., 2014.

[27] T. N. Kipf, M. Welling, "Semi-Supervised Classification with Graph Convolutional Networks," in *Proceedings of the 5th International Conference on Learning Representations*, (ICLR), ser. ICLR '17, Palais des Congrès Neptune, Toulon, France, 2017.

[28] K. Aktas, S. Sen, "Updroid: Updated android malware and its familial classification," in *Secure IT System (Lecture Notes in Computer Science)*, vol. 11252, 2018, pp. 352–368.

[29] J. Yan, Y. Qi, Q. Rao, "Lstm-based hierarchical denoising network for android malware detection," *Secure Communications Network*, vol. 2018, pp. 1–8, 2018.

[30] S. Hochreiter, J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, pp. 1735–1780, 1997.

[31] H. S. Anderson, P. Roth, *EMBER: An open dataset for training static PE malware machine learning models*, 2018. arXiv: 1804.04637.

[32] A. D. Raju, K. Wang, "Lockboost: Detecting malware binaries by locking false alarms," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, 2022, pp. 1–8.

[33] FireEye Mandiant, *Capa: The flare team's open-source tool to identify capabilities in executable files*, Mevcut: https://github.com/mandiant/capa, Erişim: 20 Haziran 2024. [Çevrimiçi].

[34] M. A. Kadri, M. Nassar, H. Safa, "Transfer learning for malware multi-classification," in *Proceedings of the 23rd International Database Applications & Engineering Symposium*, 2019, pp. 1–7.

[35] R. Vinayakumar, M. Alazab, K. Soman, P. Poornachandran, S. Venkatraman, "Robust intelligent malware detection using deep learning," *IEEE Access*, vol. 7, pp. 46 717–46 738, 2019.

[36] A. Pektaş, T. Acarman, "Learning to detect android malware via opcode sequences," *Neurocomputing*, vol. 396, 2020.

[37] J. Saxe, K. Berlin, "Deep neural network based malware detection using two dimensional binary program features," in *2015 10th International Conference on Malicious and Unwanted Software (MALWARE)*, 2015, pp. 11–20.

[38] E. Raff *et al.*, "An investigation of byte n-gram features for malware classification," *Journal of Computer Virology and Hacking Techniques*, vol. 14, pp. 1–20, 2018.

[39] Z. Ghezelbigloo, M. VafaeiJahan, "Role-opcode vs. opcode: The new method in computer malware detection," in *2014 International Congress on Technology, Communication and Knowledge (ICTCK)*, 2014, pp. 1–6.

[40] N. Zhang, J. Xue, Y. Ma, R. Zhang, T. Liang, Y.-a. Tan, "Hybrid sequence-based android malware detection using natural language processing," *International Journal of Intelligent Systems*, vol. 36, no. 10, pp. 5770–5784, 2021.

[41] S. Yesir, İ. Soğukpinar, "Malware detection and classification using fasttext and bert," in *2021 9th International Symposium on Digital Forensics and Security (ISDFS)*, IEEE, 2021, pp. 1–6.

[42] K. Lucas, M. Sharif, L. Bauer, M. K. Reiter, S. Shintre, "Malware makeover: Breaking ml-based static analysis by modifying executable bytes," in *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security*, 2021, pp. 744–758.

[43] E. Raff, J. Barker, J. Sylvester, R. Brandon, B. Catanzaro, C. K. Nicholas, "Malware detection by eating a whole exe," in *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[44] O. J. Falana, A. S. Sodiya, S. A. Onashoga, B. S. Badmus, "Mal-detect: An intelligent visualization approach for malware detection," *Journal of King Saud University-Computer and Information Sciences*, 2022.

[45] J. Singh, D. Thakur, F. Ali, T. Gera, K. S. Kwak, "Deep feature extraction and classification of android malware images," *Sensors (Basel, Switzerland)*, vol. 20, 2020.

[46] M. Nisa *et al.*, "Hybrid malware classification method using segmentation-based fractal texture analysis and deep convolution neural network features," *Applied Sciences*, vol. 10, no. 14, 2020.

[47] M. Ahmadi, D. Ulyanov, S. Semenov, M. Trofimov, G. Giacinto, *Novel feature extraction, selection and fusion for effective malware family classification*, 2016. arXiv: 1511.04317.

[48] J. Xu, W. Fu, H. Bu, Z. Wang, L. Ying, "Seqnet: An efficient neural network for automatic malware detection," *arXiv preprint arXiv:2205.03850*, 2022.

[49] A. Bensaoud, N. Abudawaood, J. Kalita, "Classifying malware images with convolutional neural network models," *International Journal of Network Security*, vol. 22, no. 6, pp. 1022–1031, 2020.

[50] K. Simonyan, A. Zisserman, *Very deep convolutional networks for large-scale image recognition*, 2015. arXiv: 1409.1556.

[51] K. He, X. Zhang, S. Ren, J. Sun, "Deep Residual Learning for Image Recognition," in *Proceedings of 2016 IEEE Conference on Computer Vision and Pattern Recognition*, ser. CVPR '16, Las Vegas, NV, USA: IEEE, 2016, pp. 770–778.

[52] E. K. Kabanga, C. H. Kim, "Malware images classification using convolutional neural network," *Journal of Computer and Communications*, vol. 6, no. 1, pp. 153–158, 2017.

[53] X. Xing, X. Jin, H. Elahi, H. Jiang, G. Wang, "A malware detection approach using autoencoder in deep learning," *IEEE Access*, vol. 10, 2022.

[54] D. Dang, F. D. Troia, M. Stamp, *Malware classification using long short-term memory models*, 2021. arXiv: 2103.02746.

[55] P. G. Balikcioglu, M. Sirlanci, O. A. Kucuk, B. Ulukapi, R. K. Turkmen, C. Acarturk, "Malicious code detection in android: The role of sequence characteristics and disassembling methods," *International Journal of Information Security*, vol. 22, 1 2022.

[56] D. Demirci, N. Sahin, M. Sirlancis, C. Acarturk, "Static malware detection using stacked bilstm and gpt-2," *IEEE Access*, vol. 10, 2022.

[57] A. Radford, K. Narasimhan, "Improving language understanding by generative pre-training," 2018.

[58] V. Sanh, L. Debut, J. Chaumond, T. Wolf, "Distilbert, a distilled version of bert: Smaller, faster, cheaper and lighter," *ArXiv*, vol. abs/1910.01108, 2019.

[59] Y. Ding *et al.*, "Malware classification on imbalanced data through self-attention," in *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, IEEE, 2020, pp. 154–161.

[60] H. Zhang, I. Goodfellow, D. Metaxas, A. Odena, "Self-attention generative adversarial networks," in *Proceedings of the 36th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 97, PMLR, 2019, pp. 7354–7363.

[61] L. Demetrio, B. Biggio, G. Lagorio, F. Roli, A. Armando, *Explaining vulnerabilities of deep learning to adversarial malware binaries*, 2019. arXiv: 1901.03583.

[62] B. Kolosnjaji *et al.*, "Adversarial malware binaries: Evading deep learning for malware detection in executables," in *2018 26th European signal processing conference (EUSIPCO)*, IEEE, 2018, pp. 533–537.

[63] O. Suciu, S. E. Coull, J. Johns, "Exploring adversarial examples in malware detection," *2019 IEEE Security and Privacy Workshops (SPW)*, 2019.

[64] W. Song, X. Li, S. Afroz, D. Garg, D. Kuznetsov, H. Yin, *Mab-malware: A reinforcement learning framework for attacking static malware classifiers*, 2021. arXiv: 2003.03100.

[65] R. S. Sutton, A. G. Barto, *Reinforcement Learning: An Introduction*, Second. The MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/the-book-2nd.html.

[66] Q. Lu, H. Zhang, H. Kinawi, D. Niu, "Self-attentive models for real-time malware classification," *IEEE Access*, vol. 10, 2022.

[67] J. Chung, C. Gulcehre, K. Cho, Y. Bengio, *Empirical evaluation of gated recurrent neural networks on sequence modeling*, 2014.

[68] M. Amin, T. A. Tanveer, M. Tehseen, M. Khan, F. A. Khan, S. Anwar, "Static malware detection and attribution in android byte-code through an end-to-end deep system," *Future generation computer systems*, vol. 102, pp. 112–126, 2020.

[69] Y. Lecun, Y. Bengio, *Convolutional Networks for Images, Speech and Time Series*. The MIT Press, 1995, pp. 255–258.

[70] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P.-A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion.," *J. Mach. Learn. Res.*, vol. 11, pp. 3371–3408, 2010.

[71] Y. Hua, J. Guo, H. Zhao, "Deep belief networks and deep learning," in *Proceedings of 2015 International Conference on Intelligent Computing and Internet of Things*, 2015, pp. 1–4.

[72] W. Zaremba, I. Sutskever, O. Vinyals, *Recurrent neural network regularization*, 2014.

[73] S. Cornegruta, R. Bakewell, S. Withey, G. Montana, *Modelling radiological language with bidirectional long short-term memory networks*, 2016. arXiv: 1609.08409.

[74] D. Gibert, C. Mateu, J. Planes, "The rise of machine learning for detection and classification of malware: Research developments, trends and challenges," *Journal of Network and Computer Applications*, vol. 153, p. 102 526, 2020.

[75] A. Egitmen, I. Bulut, R. Aygun, A. Gündüz, O. Seyrekbasan, A. Yavuz, "Combat mobile evasive malware via skip-gram-based malware detection," *Security and Communication Networks*, vol. 2020, pp. 1–10, Apr. 2020. doi: 10.1155/2020/6726147.

[76] A. Egitmen, A. Yavuz, S. Yavuz, "Trconv: Multi-platform malware classification via target regulated convolutions," *IEEE Access*, vol. PP, pp. 1–1, Jan. 2024. doi: 10.1109/ACCESS.2024.3401627.

[77] R. Harang, E. M. Rudd, *Sorel-20m: A large scale benchmark dataset for malicious pe detection*, 2020. arXiv: 2012.07634.

[78] V. Underground, *Vx underground: Collection of malware source code, samples, and papers*, Mevcut: https://vx-underground.org/, Erişim: 20 Haziran 2024. [Çevrimiçi].

[79] VirusShare, *Virusshare: Collection of malware samples*, Mevcut: https://virusshare.com/, Erişim: 20 Haziran 2024. [Çevrimiçi].

[80] Abuse.ch, *Bazaar: Malware bazaar - a repository of malware samples*, Mevcut: https://bazaar.abuse.ch/browse/, Erişim: 20 Haziran 2024. [Çevrimiçi].

[81] F. Wei, Y. Li, S. Roy, X. Ou, W. Zhou, "Deep ground truth analysis of current android malware," in *Detection of Intrusions and Malware, and Vulnerability Assessment*, Cham: Springer International Publishing, 2017, pp. 252–276.

[82] A. H. Lashkari, A. F. A. Kadir, L. Taheri, A. A. Ghorbani, “Toward developing a systematic approach to generate benchmark android malware datasets and classification,” in *Proceedings of the 52nd IEEE International Carnahan Conference on Security Technology (ICCST)*, Montreal, Quebec, Canada, 2018.

[83] S. Mahdavifar, A. F. A. Kadir, R. Fatemi, D. Alhadidi, A. A. Ghorbani, “Dynamic android malware category classification using semi-supervised deep learning,” in *The 18th IEEE International Conference on Dependable, Autonomic, and Secure Computing (DASC)*, 2020, pp. 17–24.

[84] R. J. Joyce, D. Amlani, C. Nicholas, E. Raff, *Motif: A large malware reference dataset with ground truth family labels*, 2021. arXiv: 2111.15031.

[85] M. Lester, *Pe malware machine learning dataset*, Mevcut: https : / / practicalsecurityanalytics . com / pe - malware - machine - learning - dataset/, Erişim: 20 Haziran 2024. [Çevrimiçi].

[86] W. contributors, *Portable executable*, Mevcut: https://en.wikipedia.org/wiki/Portable_Executable, Erişim: 20 Haziran 2024. [Çevrimiçi].

[87] Apktool, *Apktool: A tool for reverse engineering android apk files*, Mevcut: https://apktool.org/, Erişim: 20 Haziran 2024. [Çevrimiçi].

[88] Hasherezade, *Pe-bear: A pe file explorer and analysis tool*, Mevcut: https: //github.com/hasherezade/pe-bear, Erişim: 20 Haziran 2024. [Çevrimiçi].

[89] R. Řehůřek, P. Sojka, “Software Framework for Topic Modelling with Large Corpora,” in *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, Valletta, Malta: ELRA, 2010, pp. 45–50.

[90] M. T. Ribeiro, S. Singh, C. Guestrin, “"why should i trust you?",” *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016.

[91] P. Gohel, P. Singh, M. Mohanty, *Explainable ai: Current status and future directions*, 2021. arXiv: 2107.07045.

[92] C. Bentz, D. Alikaniotis, M. Cysouw, R. Ferrer-i-Cancho, “The entropy of words—learnability and expressivity across more than 1000 languages,” *Entropy*, vol. 19, no. 6, 2017.

[93] T. Kekeç, D. Mimno, D. M. Tax, “Boosted negative sampling by quadratically constrained entropy maximization,” *Pattern Recognition Letters*, vol. 125, pp. 310–317, 2019.

[94] Y. N. Dauphin, A. Fan, M. Auli, D. Grangier, *Language modeling with gated convolutional networks*, 2017. arXiv: 1612.08083.

[95] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, "Drebin: Effective and explainable detection of android malware in your pocket," in *Network and Distributed System Security Symposium*, 2014.

[96] E. Raff, W. Fleshman, R. Zak, H. S. Anderson, B. Filar, M. McLean, *Classifying sequences of extreme length with constant memory applied to malware detection*, 2020. arXiv: 2012.09390.

[97] K. Allix, T. F. Bissyandé, Q. Jérome, J. Klein, R. State, Y. Le Traon, "Empirical assessment of machine learning-based malware detectors for android," *Empirical Software Engineering*, vol. 21, no. 1, pp. 183–211, 2016.

A
TERİM SÖZLÜĞÜ

Terim	Çeviri
Accuracy	Doğruluk
Adversarial Attacks	Kötü Niyetli Saldırılar
Anti-Behavioral Analysis	Davranış Analizine karşı kaçınma
Anti-Debugging	Hata Ayıklamayı Önleme
Anti-Disassembly	Çözümlemeyi Önleme
Anti-Sandbox	Sandbox Önleme
Anti-Statik Analysis	Statik Analize karşı kaçınma
Application Programming Interface	Uygulama programlama arayüzü
Architectural State	Mimari Durum
Assembly	Makine Dili
Batch	Yığın
Benchmark	Kıyaslama
Benign	Zararsız Yazılım
Boosted Decision Tree	Güçlendirilmiş Karar Ağaçları
Branch	Dallanma
Byte	Bayt
Call Indirection	Çağrı Dolaylama
Capability	Yetkinlik
Central Processing Unit	İşlemci
Clustering	Kümeleme
Code Insertation	Kod Ekleme
Code Obfuscation	Kod Karıştırma

Collection	Toplama
Command and Control	Komuta Kontrol
Communication	İletişim
Compile	Derleme
Complex Instruction Set Computer	Karmaşık Komut Setli Bilgisayar
Configuration and Metadata Alteration	Konfigurasyon ve Meta Veri Değiştirme
Context Sensitivity	Bağlam Duyarlılığı
Continuous Bag of Words	Sözcük Düzeneği
Convolutional Neural Network	Konvolüsyonel Sinir Ağı
Credential Access	Kimlik bilgilerine erişim
Cross-Entropy	Çapraz Entropi
Cryptography	Kriptografi
Data Access	Verilere erişim
Dead Kod	Herhangi bir işleve sahip olmayan kod döngüsü
Debug	Hata Ayıklama
Decompiling	Geri Derleme
Deep Neural Network	Derin Sinir Ağı
Defense Evasion	Kaçınma
Disassemble	Çözümlemek
Discorvery	Keşif
Disk	Depolama Birimi
Distributed Denial of Service	Dağıtık Hizmet Redd'i
Driver	Sürücü
Dummy	İşlevsiz
Dynamic Loading	Dinamik Yükleme
Embedding	Kelime Temsil
Evasion	Kaçınma
Event Waiting	Olay Bekleme
Execution	Yürütme
Exploit	Sömürü
File System Access	Dosya sistemine erişim

Fine-Tuning	İnce Ayar
Global	Evrensel
Grammar	Dil Bilgisi
Ground-Truth	Gerçek Değer
Hardware	Donanım
Header	Başlık, Üst Bilgi
Hooking	Kancalama Yöntemi
Impact	Etki
Instruction	Yönerge
Instruction Set	Komut Seti
Interrupt	Kesme
K Nearest Neighbor	K en yakın komşu
Kernel	Çekirdek
Large Language Model	Büyük Dil Modeli
Location Invariance	Konum Değişmezliği
Long-Short Term Memory	Uzun-Kısa Dönemli Bellek
Malware	Zararlı Yazılım
Memory	Bellek
Memory Operations	Memory İşlemleri
Mnemonic	Bellek Yardımcısı, Hafıza Yardımcısı
Multi Layer Perceptron	Çok Katmanlı Algılayıcı
Multiview Neural Network	Çoklu Görünüm Yapay Sinir Ağı
Network	Bilgisayar Ağları
Object Oriented Programming	Nesne Yönelimli Programlama
Opcode	İşlam Kodu
Operand	İşlenen
Operating System Interaction	İşletim sistemi ile etkileşim
Out-Of-Scope	Kapsam Dışı
Packing	Paketleme, Kod Şifreleme
Padding	Dolgu
Patern	Örüntü
Payload	Yük

Persistence	Kalıcılık
Positional Encoding	Konumsal Kodlama
Posterior	Sonrasal
Precision	Hassasiyet
Privilege Escalation	Ayrıcalık yükseltme
Process	İşlem
Process Interaction	İşlemler ile etkileşim
Raw	Yalın, Ham
Recall	Hatırlama
Reduced Instruction Set Computer	İndirgenmiş Komut Setli Bilgisayar
Reflection	Yansıma
Register	Yazmaç
Reinforcement Learning	Pekiştirmeli Öğrenme
Renaming	Yeniden İsimlendirme
Runtime	Çalışma Zamanı
Script	Betik
Self-Modifying Code	Kendini Değiştiren Kod
Semantics	Anlambilim
Sequential	Sıralı
Shuffle	Karıştırma
Sparse	Seyrek
State-of-art	En gelişmiş teknik
String	Betim
String Encryption	Betim Şifreleme
Subset	Alt Küme
Support Vector Machine	Destek Vektör Makinesi
Symbolic Execution	Sembolik Çalıştırma
Syntax	Söz Dizimi
Term-Frequency	Terim Frekansı, TF
Thread	İş Parçacığı
Truncate	Sınırlandırma, Kesme
Vocabulary	Sözlük

Workstation	İş İstasyonu
Zero-Click	Etkileşim olmadan yüklenen zararlı yazılımlar
Zero-Day	Sıfırıncı Gün

TEZDEN ÜRETİLMİŞ YAYINLAR

Makale

1. A. Egitmen, I. Bulut, R. Can Aygun, A. Bilge Gunduz, O. Seyrekbasan, A. Gokhan Yavuz, "Combat Mobile Evasive Malware via Skip-Gram-Based Malware Detection", in Security and Communication Networks, 2020, 6726147, 10 pages, 2020, doi: 10.1155/2020/6726147

2. A. Egitmen, A. Gokhan Yavuz and S. Yavuz, "TRConv: Multi-Platform Malware Classification via Target Regulated Convolutions," in IEEE Access, vol. 12, pp. 71492-71504, 2024, doi: 10.1109/ACCESS.2024.3401627

Proje

1. **TUBİTAK 1512, Proje Kodu: 2210557**, *Yapay Zeka ve Doğal Dil İşleme Temelli Zararlı Yazılım (Malware) Tespiti Yapan Siber Güvenlik Yazılımı*, 01.11.2021-31.01.2023

2. **YTÜ GAP, Proje Kodu: FBA-2022-4904**, *Mobil Cihazlar İçin Modern Zararlı Yazılımları Tespit Eden Yapay Zeka Tabanlı Yeni Nesil Tespit Algoritması*, 15.06.2022-05.10.2023