ANKARA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

DOKTORA TEZİ

ZARARLI YAZILIMLARIN GÖSTERMİŞ OLDUKLARI DAVRANIŞLARA GÖRE ANALİZ VE TESPİT EDİLMESİ

Ömer ASLAN

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

ANKARA
2020

ÖZET

Doktora Tezi

ZARARLI YAZILIMLARIN GÖSTERMİŞ OLDUKLARI DAVRANIŞLARA GÖRE ANALİZ VE TESPİT EDİLMESİ

Ömer ASLAN

Ankara Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

Danışman: Prof. Dr. Refik SAMET

Son yıllarda bilgisayar, taşınabilir cihaz ve İnternet teknolojilerinin kullanımında dünya çapında bir artış görülmüştür. Bu teknolojilerin aşırı ve hızlı yaygınlaşması birçok güvenlik sorununu da birlikte getirmiştir. Yapılan araştırmalar güvenlik saldırılarının büyük çoğunluğunun zararlı yazılım kullanarak yapıldığını göstermiştir. Daha önceleri basit amaçlar için gerçekleştirilen saldırılar yerini geniş çaplı, küresel boyutta olan hedef odaklı saldırılara bırakmıştır. Bu nedenle, her gün yeni zararlı yazılımlar yazılmakta ve bu yazılımlar zamanla şekil ve yöntem değiştirmektedir. Zararlı yazılımların sürekli şekil değiştirmesi, güncel olarak kullanılan antivirüs tarayıcılarının bu yazılımları tespit etmede yetersiz kalmasına neden olmaktadır. Bu sebepler yeni yöntemlerin geliştirilmesini zorunlu kılmaktadır.

Tez kapsamında bilgisayarlardaki zararlı yazılımların sistem içinde gösterdikleri davranışlar analiz edilerek bu yazılımları tespit etmek amaçlanmıştır. Bunun için Eksiltici Merkezi Davranış Modeli (EMDM) önerilmiştir. Önerilen modelde, zararlı yazılım davranışları ve davranışların gerçekleştirildiği sistem dosya yolları analiz edilerek davranış ve özellikler oluşturulmuştur. Ayrıca, yeni bir özellik seçim algoritması önerilerek elde edilen özellikler azaltılmıştır. Elde edilen özellikler sınıflandırılarak zararlı yazılımlar tespit edilmiştir. Bu süreçte mevcut sınıflandırma algoritmaları kullanılarak sınıflandırma yapılmıştır. Ayrıca, karar ağaçlarında özellik seçim kriteri ve budama için optimizasyon yapılmıştır. Önerilen model ve yöntemlerin performanslarını değerlendirmek amacıyla çeşitli veri setleri oluşturulmuş ve sonuçlar literatürdeki öncü yöntemlerle karşılaştırılmıştır. Toplamda 6700 zararlı yazılım ve 3000 normal yazılım analiz edilmiştir. Analiz edilen yazılımlar zararlı ya da normal olarak sınıflandırılmıştır. Test sonuçlarına göre önerilen model uygun bir makine öğrenmesi sınıflandırıcı ile birleştirildiğinde tespit oranı, yanlış pozitif oranı ve doğruluk oranı sırasıyla %99.9, %0.2 ve %99.8 olarak ölçülmüştür. Literatürdeki diğer öncü yöntemlerle karşılaştırıldığında daha yüksek sonuçların elde edildiği görülmektedir.

Temmuz 2020, 136 sayfa

Anahtar Kelimeler: Kötü amaçlı yazılım analizi, zararlı yazılım analizi, zararlı yazılım tespiti, davranış analizi, davranış sınıflandırması, eksiltici merkezi davranış modeli, veri madenciliği, makine öğrenmesi

ABSTRACT

Ph.D. Thesis

ANALYSIS AND DETECTION OF MALWARE BASED ON BEHAVIORS

Ömer ASLAN

Ankara University
Graduate School of Natural and Applied Sciences
Department of Computer Engineering

Supervisor: Prof. Dr. Refik SAMET

In recent years, the use of computer, mobile and Internet technologies has increased rapidly worldwide. The proliferation of these technologies has also brought many security problems altogether. Recent studies have shown that the majority of cyber security attacks are performed by malicious software (malware). In the early days, malware was written for simple purposes, but in process of time it was replaced by targeted and persistent attacks. Thus, new malware has been created by using different obfuscation methods; due to this evasion, malware can easily bypass antivirus scanner. Hence, there is a huge demand to develop new methods to detect malware.

This study is aimed at detecting malware by analyzing the malicious behaviors that are carried out on computer systems. For this purpose, Subtractive Central Behavior Model (SCBM) has been proposed. In SCBM, malware behaviors and system paths are taken into consideration when creating a properties. Furthermore, a feature selection algorithm is proposed to reduce the number of features. To classify the obtained properties, well-known classification algorithms are used. In addition, optimization for feature selection criteria and pruning was made for decision trees. To evaluate the performance, various datasets are created and the results are compared with the state-of-the-art methods in the literature. Totally, 6700 malware and 3000 benign samples are tested. The results show that the detection rate, false positive rate, and accuracy are measured as %99.9, %0.2 and %99.8, respectively, which is quite high when comparing to well-known method in the literature.

July 2020, 136 pages

Key Words: Malware analysis, malware detection, behavior analysis, behavior classification, subtractive central behavior model, data mining, machine learning

TEŞEKKÜR

Çalışmalarımı yönlendiren, araştırmalarımın her aşamasında bilgi, öneri ve yardımlarını esirgemeyerek akademik ortamda olduğu kadar beşeri ilişkilerde de engin fikirleriyle yetişme ve gelişmeme katkıda bulunan danışman hocam sayın Prof. Dr. Refik SAMET'e, çalışmalarım süresince teknik anlamda derin bilgilerinden faydalandığım tez izleme komitesi üyeleri sayın Dr. Öğr. Üyesi Ömer Özgür TANRIÖVER (Ankara Üniversitesi Bilgisayar Mühendisliği Anabilim Dalı Öğretim Üyesi) ve Dr. Öğr. Üyesi. Hüseyin ÇAKIR'a (Gazi Üniversitesi Eğitim Fakültesi Bilgisayar ve Öğretim Teknolojileri Eğitimi Öğretim Üyesi), çalışmalarım süresince birçok fedakarlıklar göstererek beni destekleyen eşim ve çocuklarıma en derin duygularla teşekkür ederim.

Ömer ASLAN
Ankara, Temmuz 2020

İÇİNDEKİLER

ÖZET ..... **i**
ABSTRACT ..... **ii**
TEŞEKKÜR ..... **iii**
SİMGELER VE KISALTMALAR DİZİNİ ..... **vii**
ŞEKİLLER DİZİNİ ..... **x**
ÇİZELGELER DİZİNİ ..... **xii**
1. GİRİŞ **1**
1.1 Genel Bilgi **1**
1.2 Tezin Amacı ve Önemi **1**
1.3 Tezin Katkısı **4**
1.4 Tezin Yapısı **6**
2. SİBER SALDIRILAR VE ZARARLI YAZILIM TEMEL KAVRAMLARI **8**
2.1 Siber Saldırılar ve Artma Sebepleri **8**
2.1.1 Var olan sistem hatalarını kullanarak yapılan siber saldırılar **8**
2.1.2 Yeni gelişen teknolojilerden kaynaklanan hataları kullanarak yapılan siber saldırılar **11**
2.1.3 Kritik altyapı sistemleri sayısındaki artıştan kaynaklanan hataları kullanarak yapılan siber saldırılar **14**
2.2 Zararlı Yazılımlar ve Çoğalma Şekilleri **15**
2.2.1 Zararlı yazılımların tanımı, tarihi ve tespiti **15**
2.2.2 Zararlı yazılım türleri ve çoğalma şekilleri **18**
2.3 Zararlı Yazılım Gizlenme Teknikleri **25**
2.3.1 Gizleme yöntemleri **27**
2.3.2 Gizleme sırasında kullanılan teknikler **30**
2.4 Değerlendirme **33**
3. KAYNAK ÖZETLERİ VE LİTERATÜR DEĞERLENDİRMESİ **34**
3.1 Zararlı Yazılım Analizi **34**
3.1.1 Temel (basit) statik analiz **35**
3.1.2 İleri düzey (gelişmiş) statik analiz **35**
3.1.3 Temel (basit) dinamik analiz **36**
3.1.4 İleri düzey (gelişmiş) dinamik analiz **36**
3.2 Zararlı Yazılım Özellik Çıkarma Yöntemleri **37**
3.2.1 *n*-gram modeli **37**
3.2.2 Diyagram (Graph) modeli **38**
3.2.3 Mevcut veri setleri **38**
3.3 Zararlı Yazılımlar ve Sınıflandırma **39**
3.4 Zararlı Yazılım Tespit Yaklaşımları **40**
3.4.1 İmza tabanlı tespit (Signature-based detection) yaklaşımı **43**
3.4.1.1 İmza belirleme süreci ve kullanılan teknikler **44**
3.4.1.2 Literatürde yapılan imza tabanlı çalışmaların özeti **45**
3.4.1.3 İmza tabanlı tespit yaklaşımı ve yöntemlerinin değerlendirilmesi **47**

3.4.2 Davranış tabanlı tespit (Behavior-based detection) yaklaşımı 47
3.4.2.1 Davranış belirleme süreci 48
3.4.2.2 Literatürde yapılan davranış tabanlı çalışmaların özeti 48
3.4.2.3 Davranış tabanlı tespit yaklaşımı ve yöntemlerinin değerlendirilmesi 52
3.4.3 Bulgusal tabanlı tespit (Heuristic-based detection) yaklaşımı 53
3.4.3.1 Literatürde yapılan bulgusal tabanlı çalışmaların özeti 53
3.4.3.2 Bulgusal tabanlı tespit yaklaşımı ve yöntemlerinin değerlendirilmesi 55
3.4.4 Model denetleme tabanlı tespit (Model checking-based detection) yaklaşımı 55
3.4.4.1 Literatürde yapılan model denetleme tabanlı çalışmaların özeti 55
3.4.4.2 Model denetleme tabanlı tespit yaklaşımı ve yöntemlerinin değerlendirilmesi 57
3.4.5 Derin öğrenme (Deep Learning (DL)-based detection) tabanlı tespit yaklaşımı . 57
3.4.5.1 Literatürde yapılan derin öğrenme tabanlı çalışmaların özeti 58
3.4.5.2 Derin öğrenme tabanlı tespit yaklaşımı ve yöntemlerinin değerlendirilmesi 59
3.4.6 Bulut tabanlı tespit (Cloud-based detection) yaklaşımı 60
3.4.6.1 Literatürde yapılan bulut tabanlı çalışmaların özeti 60
3.4.6.2 Bulut tabanlı tespit yaklaşımı ve yöntemlerinin değerlendirilmesi 62
3.4.7 Mobil ve IoT tabanlı tespit (Mobil and IoT-based detection) yaklaşımı 62
3.4.7.1 Literatürde yapılan mobil ve IoT tabanlı çalışmaların özeti 62
3.4.7.2 Mobil ve IoT tabanlı tespit yaklaşımları ve yöntemlerinin değerlendirilmesi .. 64
3.5 Zararlı Yazılım Tespit Yaklaşımları ve Bu Alanda Yapılan Çalışmaların Değerlendirilmesi 64
3.5.1 Yapılan çalışmaların değerlendirilmesi 64
3.5.2 Zararlı yazılım tespit yaklaşımlarının değerlendirilmesi 65
4. DAVRANIŞA GÖRE ZARARLI YAZILIM ANALİZ VE TESPİT SİSTEMİNİN GELİŞTİRİLMESİ 69
4.1 Birleştirmeli Sıralı Tespit Yönteminin (BSTY) Geliştirilmesi 70
4.1.1 Statik analiz ve kullanılan araçlar 70
4.1.2 Dinamik analiz ve kullanılan araçlar 71
4.1.3 BSTY kullanılarak zararlı yazılım analizi 72
4.2 Eksiltici Merkezi Davranış Modelinin (EMDM) Geliştirilmesi 74
4.2.1 EMDM sistem mimarisi 74
4.2.2 EMDM kuralları 76
4.2.3 Özellik seçimi 88
4.3 Karar Ağacı Özellik Seçim Kriterinin Çok Dallanmalı ve Farklı Ağırlıklı Dağıtılmış Olacak Şekilde İyileştirilmesi 89
4.3.1 Karar ağacı özellik seçim kriterinin iyileştirilerek ağaca yerleştirmenin yapılması 90
4.3.2 Etkin karar ağacı budamasının yapılması 92
4.4 Değerlendirme 93
5. ÖNERİLEN ZARARLI YAZILIM TESPİT SİSTEMİNİN UYGULAMASI 95
5.1 Veri Toplama ve Gösterimi 95
5.2 BSTY Kullanılarak Yazılım Analizi 96

5.3 EMDM Kullanılarak Özelliklerin Belirlenmesi ve Zararlı Davranış Kalıplarının Ayrıştırılması 99
5.4 Özellik Seçimi, Makine Öğrenmesi ve Tespit 102
5.5 Önerilen Modelin Performans Değerlendirmesi 104
5.6 Değerlendirme 106
6. BULGULAR, TARTIŞMALAR VE DEĞERLENDİRMELER 107
6.1 EMDM Kullanarak Elde Edilen Sonuçlar 107
6.2 Elde Edilen Sonuçların Mevcut Veri Seti Sonuçlarıyla Karşılaştırılması 111
6.3 Eksiltici Merkezi Davranış Modeli Kullanılarak Elde Edilen Özellikler 114
6.4 Eksiltici Merkezi Davranış Modelinin Çalışma Zamanı Performans Analizi 115
6.5 Zararlı Yazılım Analiz Sonucunda Elde Edilen Bulgular 115
6.5.1 Zararlı yazılım tespit yöntemleriyle ilgili elde edilen bulgular 115
6.5.2 Zararlı yazılım karakteristik özellikleriyle ilgili elde edilen bulgular 116
6.6 Değerlendirme 117
7. SONUÇ 119
7.1 Sonuçlar 119
7.2 Öneriler ve Gelecek Çalışmalar 122
KAYNAKLAR 124
ÖZGEÇMİŞ 136

SİMGELER DİZİNİ

A	Algoritma
a	Sonlu sayı
D	Karar verici
F	Rastgele dosya
$F(x)$	Tespit edici
$G(V, E)$	G diyagram, V sistem çağrıları ve E sistem çağrıları arasındaki ilişki
m	Davranış sayısı
N	Nesne (Bilgisayar kaynağı)
n	Analiz araç sayısı
P	Program
S	Sembol (Bilgisayar komutu)
T_n	Davranış ya da string bloğu
V	Zararlı kodları bulunduran virüs bölümleri (Viral-set)
W	Bilgisayar solucanı
X'	Şüpheli dosya
$\psi_i(x)$	Dosya skoru

Kısaltmalar

ANN	Yapay sinir ağı (Artifical neural network)
API	Uygulama programlama arayüzü (Application programming interface)
APTs	Gelişmiş kalıcı tehditler (Advanced persistent threats)
AVS	Antivirüs tarayıcısı (Antivirus scanner)
BN	Bayes ağı (Bayesian network)
BSTY	Birleştirmeli sıralı tespit yöntemi
CART	Sınıflandırma ve regresyon ağacı (Classification and regression tree)
CFG	Kontrol akış şeması (Control flow graph)
CTL	Hesaplama ağacı mantığı (Computation tree logic)
CTPL	Hesaplama ağacı yüklem mantığı (Computation tree predicate logic)
C4.5/J48	Bir çeşit karar ağacı (C4.5 Tree)
DBN	Derin inanç ağı (Deep belief network)
DL	Derin öğrenme (Deep Learning)
DLL	Windows metotlarını içeren kütüphane modülleri (Dynamic link library)
DNP3	Uzak sistemleri gözlemlemek için kullanılan dağıtılmış ağ protokolü

	(Distributed network protocol version 3)
DNS	Alan adı sistemi (Domain name system)
DoS	Hizmeti engelleme (Denial of service)
EMDM	Eksiltici merkezi davranış modeli
FN	Yanlış negatif (False negative)
FP	Yanlış pozitif (False pozitive)
HKCU	HKEY_CURRENT_USER
HKLM	HKEY_LOCAL_MACHINE
HTTP	Üst Metin transfer protokolü (Hypertext transfer protocol)
HTTPS	Güvenli üst metin transfer protokolü (Hypertext transfer protocol secure)
IDS	Saldırı tespit sistemi (Intrusion detection system)
IDPS	Saldırı tespit ve koruma sistemi (Intrusion detection and prevention system)
ID3	Yinelemeli dikometre 3 (Iterative dichotomiser 3)
IoT	Nesnelerin interneti (Internet of things)
IP	İnternet protokolü (Internet protokol)
IPS	Saldırı önleme/koruma sistemi (Intrusion prevention system)
IPSec	İnternet protokolü güvenliği (Internet protocol security)
KNN	K-en yakın komşu (K-Nearest neighbors)
LMT	Lojistik model ağacı (Logistic model tree)
LR	Lojistik regresyon (Logistic regression)
LTL	Doğrusal zamansal mantık (Linear temporal logic)
ML	Makine öğrenmesi (Machine learning)
MLP	Çok katmanlı algılayıcı (Multi layer perceptron)
NB	Naive bayes
NY	Normal yazılım
Opcode	İşlem kodu (Operation code)
PE	Çalıştırılabilir Windows dosya formatı (Portable executable)
RF	Rastgele orman (Random forest)
SCADA	Denetim kontrolü ve veri toplama (Supervisory control and data acquisition)
SDL	Güvenli kod geliştirme yaşam döngüsü (Security development lifecycle)
SLR	Basit lojistik regresyon (Simple logictic regresssion)
SMO	Sıralı en küçük optimizasyon (Sequential minimal optimization)
SQL	Yapılandırılmış sorgu dili (Structured query language)
SSL	Güvenli soket katmanı (Secure socket layer)
SVM	Destek vektör makinesi (Support vector machine)
TCP	İletişim kontrol protokolü (Transmission control protocol)
TLS	Taşıma katmanı güvenliği (Transport layer security)
VPN	Özel sanal ağ (Virtual private network)

WİS	Winsows işletim sistemi (Windows operating system)
ZY	Zararlı yazılım (Malicious software-Malware)

ŞEKİLLER DİZİNİ

Şekil 2.1 Virüsün şekil değiştirerek yayılması 20
Şekil 2.2 Virüsün arama motoru kullanarak yayılması (Fu vd. 2017) 21
Şekil 2.3 Derleme ve tersine mühendislik işlemleri 26
Şekil 2.4 Metamorphic zararlı yazılımın çoğalması 28
Şekil 2.5 Kod paketleme işlemi 30
Şekil 3.1 Zararlı yazılım tespit yaklaşımları ve özelliklerinin akış şeması 41
Şekil 3.2 ClamAV bayt imzası 44
Şekil 3.3 "90FF1683EE0483EB0175F6" imzasının assembly bayt dizilimi 44
Şekil 3.4 Bayt imzalarının Yara formatında gösterimi 45
Şekil 4.1 Önerilen sistemin genel mimarisi 69
Şekil 4.2 Zararlı yazılım analiz akış diyagramı 72
Şekil 4.3 Zararlı yazılım tespit algoritması 73
Şekil 4.4 EMDM sistem mimarisi 74
Şekil 4.5 EMDM zararlı yazılım tespit özet algoritması 75
Şekil 4.6 Windows API ve direk Windows Native API kullanımı 77
Şekil 4.7 Davranış oluşturma algoritması 79
Şekil 4.8 Özellik oluşturma algoritması I 85
Şekil 4.9 Özellik oluşturma algoritması II 86
Şekil 4.10 Özellik frekansı hesaplama algoritması 87
Şekil 4.11 Özellik seçimi iyileştirilmiş karar ağacı mimarisi 90
Şekil 4.12 Çok dallanmalı ve farklı ağırlıklı dağıtılmış karar ağacı 93
Şekil 5.1 Analiz edilen zararlı yazılımların oransal dağılımı (%) 96
Şekil 5.2 “Process Monitor” kullanarak “InstallMSN.exe” arka kapı analizi (Aktivitelerin listesi kısaltılmıştır) 97
Şekil 5.3 “InstallMSN.exe” arka kapısının başka sisteme (10.36.253.135) bağlanırken yaptığı ağ işlemleri (Aktivitelerin listesi kısaltılmıştır) 98
Şekil 5.4 “svchost.exe” işlemleri (Aktivitelerin listesi kısaltılmıştır) 98
Şekil 5.5 “API Monitor” kullanarak program analizi (API sistem çağrılarının listesi kısaltılmıştır) 99
Şekil 5.6 “5c8dd4561380ba1d7e6f8a03e4279530” zararlı yazılım çalıştırıldığında göstermiş olduğu aktivitelerin listesi (Liste kısaltılarak verilmiştir) 100
Şekil 5.7 EMDM kullanılarak çıkarılan “5c8dd4561380ba1d7e6f8a03e4279530” zararlı yazılım davranışları (Liste kısaltılarak verilmiştir) 101
Şekil 5.8 EMDM kullanılarak elde edilen “5c8dd4561380ba1d7e6f8a03e4279530” zararlı yazılım özellikleri (Liste kısaltılarak verilmiştir) 101
Şekil 5.9 4-gram kullanılarak elde edilen “5c8dd4561380ba1d7e6f8a03e4279530” zararlı yazılım özellikleri ve frekansları (Liste kısaltılarak verilmiştir) 102
Şekil 5.10 EMDM kullanılarak elde edilen “5c8dd4561380ba1d7e6f8a03e4279530” zararlı yazılım özellikleri ve frekansları (Liste kısaltılarak verilmiştir) 102

Şekil 5.11 EMDM kullanılarak seçilen özellikler ve frekansları (Liste kısaltılarak verilmiştir).....103
Şekil 5.12 EMDM kullanılarak elde edilen veri seti özellikleri (Liste kısaltılarak verilmiştir) 104
Şekil 5.13 EMDM kullanılarak elde edilen veri seti özellik frekansları vektörü (Liste kısaltılarak verilmiştir).....104
Şekil 6.1 Sınıflandırıcıların oluşturulan skorlu ve skorsuz veri setleri üzerindeki performansları (1000 program analiz edildiğinde) 109
Şekil 6.2 Sınıflandırıcıların analiz edilen program sayısına göre ortalama doğruluk ve FP oranları 110

ÇİZELGELER DİZİNİ

Çizelge 1.1 Zararlı yazılım türleri ve temel özellikleri 1
Çizelge 2.1 Yeni gelişen teknolojilerin ortak özellikleri ve yaygın saldırı türleri 11
Çizelge 2.2 Zararlı yazılımlarla ilgili yapılan teorik yayınların listesi 15
Çizelge 2.3 Geleneksel ve yeni nesil zararlı yazılımların karşılaştırılması 18
Çizelge 2.4 Yıllara göre bilgisayar solucanları ve karakteristik özellikleri 23
Çizelge 2.5 Chernobyl/CIH virüsünün orjinal ve obfuscated kod bloğu (Dalla Preda 2007) 27
Çizelge 2.6 Chernobyl virüs imzaları (Dalla Preda 2007) 27
Çizelge 2.7 Win32/Evol virüsünün metamorphic teknik kullanarak çoğalması (Szor ve Ferrie 2001) 28
Çizelge 2.8 Registerlerin yer değiştirmesi (Shahid 2014) 30
Çizelge 2.9 Eşdeğeriyle kod değiştirme 31
Çizelge 2.10 Programa boş kod ekleme (Alzarooni 2012) 31
Çizelge 2.11 Program komutlarının yer değiştirmesi (Shahid 2014) 32
Çizelge 2.12 Alt program kodlarını rastgele yer değiştirme 32
Çizelge 2.13 Orjinal “merhaba dunya” programı ve ASCII onaltılık-Base64 kodlama kullanılarak formatı değiştirilmiş kod 33
Çizelge 3.1 Statik-dinamik analiz avantaj ve dezavantajları 37
Çizelge 3.2 Makine öğrenmesi algoritmalarının karşılaştırılması 40
Çizelge 3.3 Zararlı yazılım tespitinde kullanılan yaklaşımlar ve bu yaklaşımlarda kullanılan yöntemlerin özeti 43
Çizelge 3.4 Zararlı yazılım tespit yaklaşımlarının karşılaştırılması 66
Çizelge 3.5 Zararlı yazılım tespit yaklaşımlarının avantaj ve dezavantajları 66
Çizelge 4.1 Statik zararlı yazılım analiz araçları 71
Çizelge 4.2 Dinamik zararlı yazılım analiz araçları 71
Çizelge 4.3 Örnek bir zararlı yazılım çalıştırıldığında göstermiş olduğu aktivitelerin listesi (Liste kısaltılarak verilmiştir) 78
Çizelge 4.4 Çıkarılan zararlı yazılım davranışları (Liste kısaltılarak verilmiştir) 78
Çizelge 4.5 Risk skoru hesaplanırken kullanılan grup türleri ve parametreleri 80
Çizelge 4.6 Çok kullanılan Windows DLL’leri 83
Çizelge 4.7 Windows tarafından yaygın olarak kullanılan metotlar 83
Çizelge 4.8 Windows kaynakları ve yapılan davranışlar 84
Çizelge 4.9 Çıkarılan zararlı yazılım özellikleri 88
Çizelge 4.10 Özellik vektörü 88
Çizelge 4.11 Veri seti ve özellikler (Liste kısaltılarak verilmiştir) 92
Çizelge 5.1 Karışıklık matrisi 105
Çizelge 6.1 Sınıflandırıcıların EMDM kullanılarak oluşturulan veri seti üzerindeki performansları 107

Çizelge 6.2 EMDM özellik seçimi ve mevcut özellik seçim algoritmasının karşılaştırılması 109
Çizelge 6.3 Karar ağaçlarının performans karşılaştırması 111
Çizelge 6.4 EMDM sonuçlarının *n*-gram ile karşılaştırılması 111
Çizelge 6.5 ClaMP, NSL-KDD ve EMDM veri setlerinin performans karşılaştırması 112
Çizelge 6.6 Sınıflandırıcıların farklı çalışmalar üzerindeki performansları..... 113
Çizelge 6.7 Farklı veri seti oluşturma tekniklerinin karşılaştırılması 113
Çizelge 6.8 EMDM ve 4-gram kullanıldığında elde edilen özellik sayıları 114
Çizelge 6.9 *n*-gram ve EMDM ortalama çalışma zamanları..... 115

1. GİRİŞ

1.1 Genel Bilgi

Bilgisayar ve İnternet kullanımının yaygınlaşması bu sistemlere karşı yapılan siber saldırıların sayısını artırmıştır. Daha önceleri basit amaçlar için gerçekleştirilen saldırılar yerini geniş çaplı ve küresel boyutta olan hedef odaklı saldırılara bırakmıştır. Gerçekleştirilen saldırıların çoğu zararlı yazılım (ZY) kullanılarak yapılmaktadır. Genelde kötü amaçlı yazılımlar olarak bilinen bu yazılımlar sistem ve uygulama yazılımlarındaki açıklardan ve zafiyetlerden (arabellek taşması, giriş verilerinin kontrol edilmemesi, hassas verilerin şifrelenmemesi, vb.) yararlanarak saldırı başlatmaktadırlar. Yaygın olarak bilinen ZY'ler ve bu yazılımların temel özellikleri (Aslan vd. 2020) çizelge 1.1'de görülmektedir.

Çizelge 1.1 Zararlı yazılım türleri ve temel özellikleri

ZY türü	Temel özellikler
Virüs	En yaygın ve iyi bilinen ZY'dir
	Çoğalmak için başka programlara ihtiyaç duyar
	Bulaştığı sistemleri yavaşlatır
Solucan	Bilgisayar ağını kullanarak yayılır
	Yetkisiz erişimlere izin verir
	Genellikle girdiği sistemlerde arka kapılar açar
Truva atı	Normal programlar içine gömülmüş ZY'dir
	Arka kapılar açar
	Yetkisiz erişimlere neden olur
	Gizli bilgileri üçüncü kişilere gönderir
Arka kapı	Geleneksel güvenlik mekanizmalarını atlatır
	Sistemi uzak erişimlere açar
	Genellikle Truva atları ve solucanları kullanarak sisteme yerleşir
	Virüsler ve solucanlar tarafından karmaşık saldırılar için kullanılır
Rootkit	Yönetici düzeyinde erişim sağlar
	Dosyalarını işletim sisteminden gizler
	Başka ZY'lerle birlikte çalışır
Fidye yazılımı	Bulaştığı sistemlerdeki verileri şifreler
	Verilerin tekrar görüntülenmesi için fidye ödenmesi gerekir
Casus yazılım	Bulaştığı sistemin gizli bilgilerini üçüncü taraflara gönderir
	Genellikle kredi kartı bilgilerini ve kullanıcı alışkanlıklarını belirlemek için kullanılır

1.2 Tezin Amacı ve Önemi

ZY'lerin sayısı, karmaşıklığı ve dünya ekonomisine verdiği zararlar her geçen gün artmaktadır. Bilimsel raporlara göre her gün yaklaşık 1 milyon ZY oluşturulmaktadır. Siber kaynaklı saldırıların dünya ekonomisine maliyeti 2019 yılında yaklaşık 3 trilyon dolar olarak

ölçülmüş ve bu maliyetin 2021 yılında yaklaşık 6 trilyon dolar olacağı öngörülmektedir (Morgan 2019). Son çalışmalar mobil ZY'lerin sayısında da hızlı bir artış olduğunu göstermiştir. McAfee mobil tehdit raporuna göre mobil cihazlar için arka kapılar, sahte uygulamalar ve bankacılık Truva atlarında büyük bir artış görülmüştür (Samani ve Davis 2019). Ayrıca sosyal medya, sağlık sektörü, bulut bilişim, nesnelerin interneti (Internet of Things-IoT) ve kripto para birimleri ile ilgili ZY saldırıları da artmaktadır. Verilen örneklerden de anlaşılacağı üzere ZY'ler dünya ekonomisine büyük zararlar vererek birçok kişisel ve kurumsal verinin risk altında olmasına neden olmaktadır. Bu sebepler ZY'lere karşı geniş ölçekli önlem almanın gerekliliğini ortaya koymaktadır.

Güvenlik duvarı, saldırı tespit ve koruma sistemleri (IDPS), antivirüs tarayıcıları (Antivirüs Scanner-AVS) gibi yazılımların tam koruma sağlayamaması (Thompson ve Flynn 2007, Aslan ve Samet 2017a), araştırmacıları bu probleme farklı açılardan bakmaya zorunlu bırakmıştır. ZY'leri engellemek için iki temel yöntemin birlikte kullanılması gerekmektedir:

1. Normal programlardaki savunmasızlıkları ve hataları daha önceden tespit edip, ZY'lerin bu açıklardan faydalanarak saldırı başlatmalarını engellemek;
2. ZY'ler bilgisayar sistemine bulaştıktan sonra, bu yazılımları kısa sürede tespit etmek ve sistemden temizlemek.

Bu tez çalışmasında 2. yöntem üzerinde durulmuştur. Bir sistemde var olan ZY'leri tespit etmek için kullanılan en yaygın yöntem AVS'lerdir (Ye vd. 2007, Fukushima vd. 2010). Birçok AVS imza tabanlı tespit yaklaşımı kullanmaktadır. Bu yaklaşım her ZY için farklı bir karakter dizisi (bit diziliminden oluşan bu özellik ilgili ZY programın karakteristik özelliğini ortaya koymaktadır) belirleyerek veri tabanında saklamakta ve sistemdeki ZY'leri tespit ederken bu imzaları kullanmaktadır. Avast, AVG, Avira, AVware, Bitdefender, ClamAV, Karspersky ve McAfee günümüzde yaygın olarak kullanılan imza tabanlı programlardır. Bu yaklaşımın en büyük dezavantajı daha önce bilinmeyen ZY'leri tespit etmede yetersiz kalmasıdır.

İlk zamanlarda, imza tabanlı tespit yaklaşımı yaygın olarak kullanılmıştır. Fakat son yıllarda ZY'lerin karmaşıklığının artmasından dolayı yeni nesil ZY'leri tespit etmede yetersiz kalmıştır. Bundan dolayı zamanla araştırmacılar davranış (behavior), bulgusal (heuristic) ve model denetleme (model checking) tabanlı tespit yaklaşımları geliştirmişlerdir. Bu

yaklaşımlarda ZY'lerin göstermiş oldukları davranışlar ya da statik özellikler belirlenerek modeller oluşturulmakta ve bu modeller kullanılarak ZY'ler tespit edilmektedir. Bu yaklaşımlar daha önce görülmemiş bazı ZY'leri tespit edebilmektedir. Ayrıca, özellikle son bir kaç yılda derin öğrenme (deep learning-DL), bulut (cloud), mobil cihazlar ve IoT tabanlı tespit yaklaşımları da kullanılmaya başlanmıştır.

Mevcut sistemlerde en çok kullanılan işletim sistemi Windows tabanlı olduğu için birçok ZY bu işletim sistemine yönelik olarak hazırlanmıştır. Bundan dolayı bu çalışma Windows işletim sistemi (WİS) referans alınarak yapılmıştır. Etkili bir koruma sisteminin oluşturulabilmesi için WİS yapısı iyi bilinmelidir. Çünkü son zamanlarda görülmeye başlayan ZY'ler kendilerini WİS işlemlerine enjekte ederek kendi varlıklarını gizlemektedir. Örneğin, WİS'in düzgün çalışabilmesi için arka planda Windows işlemleri diye bilinen "explorer.exe", "winlogon.exe", "svchost.exe" ve "services.exe", vb. işlemler çalışmaktadır ve bu işlemler kritik bilgilerin bulunduğu klasör, dosya ve veri tabanlarına erişmektedir. ZY'ler bu işlemleri kullanarak hem kendi varlıklarını gizlemekte hem de daha fazla yetki kullanarak kritik dosyalara erişebilmektedir. Bu farklılıkları bilmeden oluşturulan bir ZY tespit sistemi daha önce görülmemiş ve yeni nesil ZY'leri tespit etmede yetersiz kalacaktır.

Bu tez çalışmasında ZY problemi tamamen çözülmemiş, var olan bu problemin çözümüne yönelik veri madenciliği teknikleri ve makine öğrenmesi (machine learning-ML) algoritmaları kullanılarak yeni modeller, yöntemler ve algoritmalar önerilmiş ve uygulanmıştır. Öncelikle masaüstü ve dizüstü bilgisayarlardaki ZY'lerin gösterdikleri davranışlar analiz edilerek belirli veri setleri oluşturulmuştur. Davranışlar oluşturulurken yapılan davranışlar ve bu davranışların hangi sistem yollarında gerçekleştirildiği analiz edilerek özellikler ve özelliklerin önem dereceleri hesaplanmıştır. Daha sonra önem derecesi fazla olan özelliklerin frekanslarına bakılarak zararlı davranışlar normal davranışlardan ayırt edilerek ZY'ler tespit edilmiştir. Tez kapsamında elde edilen test sonuçları *n*-gram ve literatürdeki diğer öncü yöntemlerle karşılaştırıldığında daha iyi sonuçlar alınmıştır. Ayrıca, önerilen yöntemle hem geleneksel hem de yeni nesil ZY'ler başarılı bir şekilde tespit edilmiştir.

1.3 Tezin Katkısı

Mevcut ZY tespit yöntemlerinin büyük çoğunluğu *n*-gram ve diyagram (graph) benzeri teknikler kullandığı için özellik uzayı hızlı büyümekte ve bu durumda sınıflandırmayı zorlaştırmaktadır. Ayrıca, bazı yöntemler atlatma saldırılarına ve gizlenme tekniklerine karşı dayanıklı olmayıp yanlış alarmlar üretmektedir. Bu durum tespit oranı (detection rate-DR) ve doğruluk oranının düşmesine neden olmaktadır. Tez kapsamında yapılan çalışmada bu eksiklikler göz önünde bulundurulmuş ve her adımda gerekli katkılar yapılarak performans büyük bir oranda artırılmıştır. Tez kapsamında yapılan katkılar şöyle sıralanabilir:

EMDM (Eksiltici Merkezi Davranış Modeli) önerilmiştir: Önerilen model analiz edilen programın işletim sistemiyle olan etkileşimini inceleyerek davranışlar belirlemektedir. Bu etkileşimler değerlendirilirken yapılan etkileşimler ve hangi sistem yollarında yapıldığına bakılarak davranışlar ve davranışların önem dereceleri hesaplanmıştır. Zararlı davranış kalıpları elde edilirken ZY'lerde görülen ancak normal yazılımlarda görülmeyen veya nadiren görülen davranışlar elde edilmiştir. Bu amaçla davranışlar gerçekleştirildikleri yere göre 3 farklı grupta 120 farklı dosya yoluna ayrılarak her davranış için dosya yolu risk skorları hesaplanmıştır. Bu sayede ZY ve normal yazılım örneği aynı etkileşimleri gösterse bile elde edilen davranışlar ve özellikler farklı olmaktadır. Bu şekilde ZY'ler normal yazılımlardan ayırt edilmiştir. EMDM hem bilinen hem de bilinmeyen ZY'leri tespit etmede yüksek başarı göstermiştir. Bundan dolayı önerilen model hem birbiriyle ilişkili davranışları daha net belirleyebilmiş hem de özellik sayısını sınırlı tutarak iyi performans göstermiştir. Test sonuçlarına göre CreateFileMapping, CreateRemoteThread, CreateService, FindFirstFile, FindNextFile, MapviewOfFile, QueryDirectoryWriteFile, ReadFileWriteFile, RegSetInfoKey, RegDeleteValue, RegQueryKeyRegSetInfoKey ve WriteFile gibi özelliklerin ZY'ler tarafından sıkça kullanıldığı gözlemlenmiştir. Bu modelin mevcut AVS'lere eklenmesi ZY'lerin tespit edilmesinde büyük bir başarı sağlayacaktır. EMDM kullanılarak elde edilen sonuçlar yayınlanmıştır (Aslan vd. 2020).

EMDM ve *n*-gram kullanılarak veri setleri oluşturulmuştur: EMDM kullanılarak 2 *n*-gram kullanarak 1 veri seti oluşturulmuştur. Veri setleri oluşturulurken zararlı ve normal yazılım örnekleri farklı kaynaklardan toplanarak homejen bir dağılım oluşturacak şekilde seçilmiştir. 6700 zararlı ve 3000 normal yazılım olmak üzere toplamda 9700 program analiz edilmiştir. Analiz edilen programlar için EMDM kullanıldığında ortalama 44 özellik elde

edilirken 4-gram kullanıldığında bu sayı 1332'ye çıkmıştır. Ayrıca, makine öğrenmesi algoritmaları kullanılarak sınıflandırma yapmak için özellik vektörü oluşturulmuştur. Özellik vektöründe EMDM için yaklaşık 700 özellik ve 4-gram için yaklaşık 20.000 özellik elde edilmiştir.

Detaylı bir literatür araştırması yapılmıştır: Böyle bir detaylı araştırma yapılmasının nedeni, çözüme yönelik çalışmalara geçmeden önce problemin zorluğunu ve sınırlarını belirlemektir. Ayrıca, daha önce yapılmış çalışmaların ne derece başarılı olup olmadığını görerek ve değerlendirerek daha iyi çalışan bir model öne sürebilmektir. Bu aşamada ZY tespit yaklaşımları 7 farklı kategoriye ayrılarak açıklanmış ve bu yaklaşımlarda kullanılan farklı yöntemler karşılaştırılmıştır. Bu yaklaşımlar şöyle sıralanabilir:

- İmza tabanlı tespit yaklaşımı;
- Davranış tabanlı tespit yaklaşımı;
- Bulgusal tabanlı tespit yaklaşımı;
- Model denetim tabanlı tespit yaklaşımı;
- Derin öğrenme tabanlı tespit yaklaşımı;
- Bulut tabanlı tespit yaklaşımı;
- Mobil cihaz ve IoT tabanlı tespit yaklaşımı.

Yapılan literatür araştırmasının sonuçları yayınlanmıştır (Aslan ve Samet 2020).

ZY'lerin daha doğru analiz edilebilmeleri için BSTY (Birleştirmeli Sıralı Tespit Yöntemi) önerilmiştir: Sadece bir araç kullanarak şüpheli dosyanın zararlı olup olmadığını elle analiz ederek anlamak hem zor hem de yetersizdir. Bundan dolayı analizin doğru bir şekilde yapılabilmesi için şüpheli dosyanın birkaç farklı araç kullanılarak, belirli sıralama ve hesaplamaya göre analiz edilmesi gerekmektedir. BSTY'de her araç çıktısına belli puanlar verilerek önem dereceleri hesaplanmış ve bu değerler kullanılarak analiz edilen program hakkında bir sonuca gidilmiştir. BSTY kullanılarak elde edilen sonuçlar yayınlanmıştır (Aslan ve Samet 2017a).

Oluşturulan veri setlerine mevcut makine öğrenmesi algoritmaları uygulanarak sınıflandırma yapılmıştır: ML algoritmaları uzun yıllardır birçok farklı alanda kullanılmasına rağmen ZY analizinde ve tespitinde yeterince kullanılmamıştır. Bundan dolayı bu çalışmada ZY tespitinde bu algoritmalardan uygun olanlar kullanılmıştır. Bu algoritmalara örnek olarak Bayesian ağı (Bayesian Network-BN), C4.5/J48 karar ağacı

(Decision Tree Variant), k-en yakın komşular algoritması (K-Nearest Neighbor-KNN), basit logistik regresyon (Simple Logistic Regression-SLR), çok katmanlı algılayıcı (Multilayer Perceptron-MLP), naive bayes (NB), rastgele orman karar ağacı (Random Forest Tree-RF), lojistik model ağaçları (Logistic Model Trees-LMT), destek vektör makinesi (Support Vector Machine-SVM) ve sıralı en küçük optimizasyon algoritması (Sequential Minimal Optimization-SMO) gösterilebilir (Shabtai vd. 2009, Santos vd. 2013). Farklı özelliklerde algoritma kullanılmasının sebebi oluşturulan veri seti performanslarını daha doğru değerlendirmek ve literatürdeki mevcut sonuçlarla daha doğru bir performans karşılaştırması yapmaktır. Test sonuçlarına göre önerilen model uygun bir makine öğrenmesi sınıflandırıcı ile birleştirildiğinde tespit oranı, yanlış pozitif oranı ve doğruluk oranı sırasıyla %99.9, %0.2 ve %99.8 olarak ölçülmüştür.

Karar ağaçlarında özellik yerleştirme ve budamayla ilgili iyileştirmeler yapılarak sınıflandırma performansı artırılmıştır: Özellikler ağaca yerleştirilirken çok dallanmalı ve farklı ağırlıklı dağıtılmış olarak yerleştirilmektedir. Bu aşamada her özellik için edinilen bilgi ve ayırma kriteri kullanılmaktadır. Özelliğin tekrar sayısı, önem derecesi, zararlı ve normal yazılımlarda bulunma sıklığı ve aktifliği ayırma kriterine örnek gösterilebilir. Daha sonra ağaç performansını artırmayan dallar budanarak ağaçtan çıkartılmıştır.

Analiz sonuçlarına göre yeni bulgular elde edilmiştir: Yapılan analizler sonucunda ZY'nin türüne ve ait olduğu aileye bakarak farklı davranışlar elde edilmiş olup bu davranışlar incelenerek ZY'lere özgü öznitelikler belirlenmiştir.

1.4 Tezin Yapısı

Çalışmanın geri kalan kısmı şu şekildedir. Bölüm 2'de siber saldırılar ve ZY temel kavramları açıklanmıştır. Öncelikle siber saldırıların artma sebepleri ele alınmıştır. Daha sonra ZY türleri ve çoğalma şekilleri detaylı bir şekilde açıklanarak her ZY'nin karakteristik özellikleri sıralanmıştır. Ayrıca, ZY'lerin tespit edilme olasılıkları teorik ve pratikte olmak üzere tartışılmış olup ZY gizlenme teknikleri anlatılmıştır. Bölüm 3'te ZY tespitiyle ilgili yapılan çalışmalarının detaylı bir özeti verilmiştir. Her adımda önerilen yöntemler, metotlar ve algoritmalar detaylı bir şekilde anlatılarak avantaj ve dezavantajları sıralanmıştır. ZY analiz türleri, özellik çıkarma yöntemleri ve sınıflandırma detaylı bir şekilde sunulmuştur. Ayrıca, ZY tespit yaklaşımları gruplandırılarak detaylandırılmış ve kendi aralarında karşılaştırılarak önerilerde bulunulmuştur. Literatürde yapılan çalışmalar tespit yaklaşımlarına ve yıllara göre

gruplandırılarak açıklanmış ve araştırma eğiliminin hangi tespit yaklaşımı yönünde ilerlediği tartışılmıştır. Bölüm 4'te EMDM önerilerek detaylı bir şekilde açıklanmıştır. Bu model kullanılarak çeşitli veri setleri oluşturulmuştur. Ayrıca, önerilen BSTY analiz yöntemi ve karar ağacı özellik yerleştirme ve budama optimizasyonu açıklanmıştır. Bölüm 5'te önerilen model ve yöntem uygulamaları aşama aşama verilmiştir. Bölüm 6'da önerilen model ve yöntemlerin sonuçları mevcut öncü yöntemlerle karşılaştırılarak tartışılmıştır. Bölüm 7'de tez araştırmasının sonuçları ve öneriler sıralanmıştır.

2. SİBER SALDIRILAR VE ZARARLI YAZILIM TEMEL KAVRAMLARI

2.1 Siber Saldırılar ve Artma Sebepleri

Yapılan son araştırmalara göre siber kaynaklı saldırıların dünya ekonomisine maliyeti yıllık yaklaşık olarak 3 trilyon dolar olduğu tahmin edilmektedir (Morgan 2019). Siber saldırılarda kullanılan yöntemlerin zamanla değişmesi, bu saldırılarla mücadele etmeyi zorlaştırmaktadır. Siber saldırıları yapanlar genelde bilgisayar korsanı (computer hacker) olarak adlandırılan kişi yada gruplardır. Bilgisayar korsanları genelde işletim sistemleri hakkında geniş çaplı bilgi sahibi olan, hızlı bir şekilde program yazabilen, program ve sistem açıklarını hızlı bir şekilde fark eden kişilerdir. Normalin üstünde bilgiye sahip olan bu kişiler izinsiz olarak başka sistemlere girmekte, girdikleri sistemlerden legal olmayan yollarla veri çalmakta, veriler üzerinde değişikler yapmakta ve girdikleri sistemi kullanılmaz hale getirerek büyük zararlar vermektedirler. Genellikle yazdıkları ZY'ler sayesinde ya da var olan hazır araçları kullanarak bu amaçlarını gerçekleştirmektedirler. Daha önceleri basit amaçlar için oluşturulan ZY'ler zamanla geliştirilerek kredi kart bilgilerini çalan ve başka sistemlere zarar veren programlara dönüştürülmüşlerdir (Jang-Jaccard ve Nepal 2014). Siber saldırılara neden olan başlıca sebepler şöyle sıralanabilir:

- Var olan sistem hatalarını kullanarak yapılan siber saldırılar;
- Yeni gelişen teknolojilerden kaynaklanan hataları kullanarak yapılan siber saldırılar;
- Kritik altyapı sistemleri sayısındaki artıştan kaynaklanan hataları kullanarak yapılan siber saldırılar.

2.1.1 Var olan sistem hatalarını kullanarak yapılan siber saldırılar

Bu gruba giren siber saldırılar donanım, yazılım ve iletişim ağındaki eksiklik, savunmasızlık ve hatalardan kaynaklanmaktadır. Kötü amaçlı kişiler var olan bu hatalardan faydalanarak ZY oluşturmaktadır. Özellikle yazılan programlardaki açıkların fazla olması ve bilginin bilgisayar ağlarında taşınırken saldırıya açık olması, yapılan saldırıların başarılı olmasına neden olmaktadır. Bugün kullanılan donanım birimleri, yazılımlar ve iletişim ağı birçok yönden saldırılara karşı savunmasızdır. Yazılımlar ve iletişim ağı oluşturulurken bugünkü gibi ileri seviyedeki saldırılar hesaba katılmadan oluşturulduğundan, korumaya çalıştığımız

sistem yapısı gereği siber saldırılara açık bir sistemdir. Saldırılar genel olarak üç gruba ayrılabilir:

- Donanım eksikliklerinden kaynaklanan saldırılar;
- Yazılım tabanlı hatalardan kaynaklanan saldırılar;
- Bilgisayar ağlarındaki hatalardan ve eksikliklerden kaynaklanan saldırılar.

Donanım eksikliklerinden kaynaklanan saldırılar: Donanımsal hatalardan ve eksikliklerden faydalanarak başlatılan saldırıları önlemek zordur çünkü yazılım hatalarını yakalamak için geliştirilen antivirüs tarayıcıları ve saldırı tespit sistemleri gibi yazılımlar çoğu zaman donanım kaynaklı saldırılar karşısında etkisiz kalmaktadır (Potlapally 2011, Bright 2015). Donanıma yönelik yapılan en tehlikeli saldırılardan biri donanım Truvalarıdır. Bu ZY'ler bilgisayar kaynaklarının verimsiz bir şekilde kullanılmasına neden olabilir, bazılarının performansını yavaşlatabilir ya da durdurabilir, hatta güç kaynağının kısa sürede bitmesine neden olabilirler (Chakraborty vd. 2009, Karri vd. 2010). Ayrıca, yasadışı donanım birimlerinin kopyalanması sistemde arka kapıların oluşmasına neden olmaktadır. Donanım eksikliklerinden kaynaklanan saldırıları önlemek için bazı teknikler önerilmiştir. Örneğin, kurcalamaya karşı korumalı donanım cihazları bu tip saldırıları önlemede büyük öneme sahiptir. Diğer taraftan donanım damgalama, üretici firmanın bilgilerini sistem içine gömerek, donanım parçalarının kopyalanmasını önler. Ayrıca, donanım gizleme de kötü amaçlı kişilerden korumak amacıyla kullanılan etkin yöntemlerden biridir (Tehranipoor ve Wang 2011).

Yazılım tabanlı hatalardan kaynaklanan saldırılar: Siber saldırıların büyük bir kısmı hala uygulama yazılımlarındaki hata, zayıflık ve eksikliklerden kaynaklanmakta ve bu açıklar hızla artmaktadır (Fernandez 2004, Meland ve Jensen 2008, Aslan 2016). Yazılım kaynaklı savunmasızlıkların başlıca sebepleri şöyle sıralanabilir:

- Giriş verilerinin başarılı bir şekilde denetlenmemesi;
- Ara bellek aşımı;
- Kullanıcı erişim kontrolü ile ilgili sorunlar;
- Yapılandırılmış sorgu dili (Structured query language-SQL) saldırıları;
- Siteler arası betik çalıştırma.

Örneğin ara belleğe, taşıyacağı boyuttan fazla veri eklemeye çalışmak, veri alanının taşmasına neden olacak bu da sistemin çökmesine, bazı verilerinin kaybolmasına ya da yetkisiz kullanıcılara sistemde sınırsız erişim hakkı vererek sorunlara neden olacaktır (Aslan ve Samet 2017b). SQL saldırıları sonucunda veri tabanı aşırı yüklenebilir, kullanıcı adı, şifre ve kredi kartı gibi bilgiler çalınabilir. Yazılım güncellemeleri her zaman çözüm olmamakta, hatta bazen yeni güncellemeler yeni sorunlara da neden olmaktadır (Aslan 2016). Yazılım kaynaklı saldırılardan korunmanın en etkili yolu, kod yazımına geçilmeden önce program tasarımının eksiksiz olarak yapılmasıyla mümkündür. Kod yazan kişilerin eğitilmesi ve güvenlik hakkında yeterli bilgiye sahip olmaları gerekmektedir. Yazılımları üretip, daha sonra güvenlik açıklarını gidermeye çalışmak doğru bir yaklaşım değildir. Bunun için bir yazılım üretilirken güvenlik problemleri ve saldırılar göz önünde bulundurularak oluşturulmalı, yazılan kod blokları elle ve otomatik olarak belli aşamalarda test edilmelidir (McGraw 2002, Jang-Jaccard ve Nepal 2014). Microsoft Firması güvenli kod geliştirme yaşam döngüsü platformunu (Security Development Lifecycle-SDL) kendi kurum içinde kullanımını zorunlu hale getirdikten sonra yazılım sürecindeki hatalar büyük oranda azalmıştır. Örneğin, Windows Vista işletim sistemi SDL yayınlandıktan sonra yazılmış olup Windows XP'ye göre %45 (Maurya 2010) oranda daha az hata içermektedir. Diğer taraftan SDL uygulanarak yazılan SQL veri tabanı 2005, uygulanmadan yazılan SQL veri tabanı yönetim sistemi 2000'e göre %9 oranda daha az hata içermektedir.

Bilgisayar ağlarındaki hatalardan ve eksikliklerden kaynaklanan saldırılar: Bilgi ve veriler İnternet ortamında taşınırken kötü amaçlı kişiler ya da programlar verileri dinleyebilir, üzerinde değişiklik yapabilir ya da tamamen kullanılmaz hale getirebilirler. Bu tür problemlerin oluşmasının temel sebebi daha önce oluşturulan ağ protokollerinin güvenlik endişesi duyulmadan oluşturulmasından kaynaklanmaktadır (Stallings 2006, Sheldon ve Vishik 2010). Ağa karşı yapılan saldırıların çoğu İnternet Protokolü (Internet Protocol-IP), İletişim Kontrol Protokolü (Transmission Control Protocol-TCP) ve Alan Adı Sistemi (Domain Name System-DNS) gibi günümüzde kullanılan protokol savunmasızlıklarından kaynaklanmaktadır. Örneğin, IP protokolü ağ üzerinde paketleri taşırken bu paketlerin doğruluğunu ve gizliliğini kontrol edecek bir yapısı olmadığı için taşınma sırasında paketlerdeki bilgiler açığa çıkabilir ya da değiştirilebilir (Jang-Jaccard ve Nepal 2014). Diğer taraftan DNS yanıtları doğrulanmadığı için kullanıcı normal sunucu yerine kötü amaçlı kullanılan sunuculara yönlendirilebilir, ayrıca saldırganlar DNS'e aşırı istek göndererek bir

süreliğine kullanılmaz hale getirebilirler. Ağ saldırıları karşısında yaygın olarak kullanılan güvenlik önlemleri şöyle sıralanabilir:

- Şifreleme kullanımı;
- Güvenlik duvarı kullanımı;
- Saldırı tespit, önleme ve koruma sistemleri kullanımı;
- Güvenli İnternet Protokolü (Internet Protocol Security-IPsec) kullanımı;
- Özel Sanal Ağ (Virtual Private Network-VPN) kullanımı;
- Taşıma Katmanı Güvenliği (Transport Layer Security-TLS) ve Güvenli Soket Katmanı (Secure Socket Layer-SSL) protokolleri kullanımı.

Bu teknikler verilerin korunması noktasında birçok saldırıyı önlemede başarılı olsalar da, gerek bu tekniklerin oluşturulması sırasında kullanılan algoritmalardaki açıkların zamanla keşfedilmesi gerekse saldırı çeşitlerinin ve yöntemlerin gittikçe artması, bu tekniklerin saldırılara karşı önleme ve koruma noktasında tam başarı gösterdikleri söylenemez. Bundan dolayı var olan tekniklerin geliştirilmesi ve yeni tekniklerin önerilmesi gerekmektedir.

2.1.2 Yeni gelişen teknolojilerden kaynaklanan hataları kullanarak yapılan siber saldırılar

Teknolojinin hızlı gelişmesi sonucunda İnternet kullanımı artmış ve yeni sistemler ve cihazlar İnternet ortamına ilave edilmiştir. Son dönemde popülerlik kazanan sosyal medya, bulut teknolojisi, akıllı telefon kullanımının hızla artması yeni gelişen teknolojilere ve sistemlere örnek olarak gösterilebilir. Çizelge 2.1'de yeni gelişen teknolojilerin ortak özellikleri ve yaygın saldırı türleri görülmektedir.

Çizelge 2.1 Yeni gelişen teknolojilerin ortak özellikleri ve yaygın saldırı türleri

Teknoloji	Ortak özellik	Yaygın saldırı türü
Sosyal medya	Milyonlarca aktif kullanıcı	Web tarayıcıları ve uygulama yazılımları aracılığıyla artan saldırılar
Bulut bilişim	Coğrafi olarak sınırların olmaması ve 24/7 her zaman her yerden erişilmesi	Sosyal mühendislik teknikleri kullanarak artan saldırılar
Akıllı telefon	Servislere uygulama yazılımları kullanılarak erişilmesi	PC tabanlı olmayan saldırı miktarında artış
IoT	Bilgisayar ağı yönetiminin zorlaşması	Botnet kullanarak daha organize edilmiş saldırılar

Sosyal medya kullanımının artması bazı güvenlik sorunlarını da beraberinde getirmektedir, çünkü birçok kullanıcı özel bilgilerini bu ortamlarda paylaşmakta ve kötü amaçlı kişiler bu

bilgileri kullanarak çeşitli yasadışı olaylara karışmaktadırlar. Ayrıca bulut teknoloji kullanımının artması sonucunda da yeni güvenlik sorunları ortaya çıkmıştır. Bu sorunların başında, farklı kişi ve kurumlara ait bilgilerin aynı fiziksel makineler üzerinde depolanması ve bu veri merkezlerinin yönetim ve bakımının üçüncü şahıslara ait olması gelmektedir. Öte yandan, akıllı telefon kullanımının hızla artması, mevcutta kullanılan iletişim ağının akıllı telefonlar içinde kullanılması ve çok sayıda uygulamanın kısa sürede yazılması, siber saldırıların sayısını artırmıştır. Yeni gelişen teknolojilerden kaynaklanan sebepler şöyle sıralanabilir:

- Sosyal medya kullanımının artması;
- Akıllı telefon kullanımının artması;
- Bulut bilişim;
- IoT ile birlikte birçok yeni cihazın İnternete bağlanması.

Sosyal medya kullanımının artması: Sosyal medya (Twitter, Facebook, vb.) kullanımı dünya çapında hızla artmaktadır. Son yıllarda yapılan araştırma sonuçlarına göre Twitter ve Facebook kullanıcı sayıları milyarlarla ifade edilmektedir (Stout 2020). Çoğu sosyal medya ortamı, kullanıcılarına kişisel bilgilerini (ad, soy ad, doğum tarihi, adres, yaşadığı yer, vb.) depolama imkanı vermekte hatta son dönemlerde görüntü paylaşımına dahi imkan vermektedir. Kullanıcı verileri büyük veri merkezlerindeki sunucularda tutulmaktadır. Hem bu sunuculara karşı yapılacak saldırılar sonucu veriler sızabilir hem de kullanıcıların bilgilerini (konum bilgileri, resim, video, vb.) bu ortamlarda paylaşmaları sonucu veriler üçüncü kişiler tarafından erişilebilir ve değiştirilebilirler. Ayrıca, saldırganlar aktif olmayan bazı Facebook ve Twitter hesaplarını kullanarak ya da sosyal mühendislik teknikleri kullanarak bilgilerinize erişebilmektedir. Yapılan son araştırmalara göre sosyal medya kullanıcıların çoğu istemeyen elektronik posta (spam email) almaktadır. Ayrıca, benzer raporlara göre büyük şirketlerin %50'sinden fazlası, çalışanlarının sosyal medya ortamlarındaki bilgi paylaşmalarından dolayı şirketlerine yönelik yapılan saldırıların artığını belirtmişlerdir. Aynı zamanda, sosyal medya ortamlarında yayılan ZY'lerin büyük bir kısmı, bu ortamlarda popüler olan olay ya da haberlere tıklanması sonucu diğer sunuculara yayılmaktadır (Hogben 2007, Luo vd. 2009, Liu vd. 2019, Singh ve Tomar 2019).

Akıllı telefon kullanımının artması: Günümüzde kullanılan akıllı telefon 3 milyarın üstündedir ve bu sayı hızla artmaya devam etmektedir. Kişisel verilerin bu cihazlarda saklanması, akıllı telefonlara yönelik oluşturulan uygulama yazılımlarının giderek artması ve internete erişim için kablosuz ağlara ihtiyaç duymaları güvenlik endişelerini artırmaktadır. Symantec (2016) ve McAfee (Samani ve Davis 2019) siber güvenlik tehdit raporuna göre akıllı telefonlara yönelik oluşturulan ZY sayısında büyük bir artış görülmüştür. Normal bilgisayarlar için tehdit oluşturan ZY'ler değiştirilerek bu cihazlar için kullanılmaya başlanmıştır.

Bulut bilişim: Son yıllarda gelişen bir teknoloji olup talep ve kullanma isteğine göre hizmet veren bir mimaridir. Amazon, Google, Microsoft ve Salesforce.com (Padhy vd. 2011) yaygın olarak kullanılan bulut bilişim servis sağlayıcılarıdır. Kullanımının kolay olması, birçok farklı cihaz kullanarak dünyanın her yerinde erişilebilir olması, talebe göre verilen hizmetin esnek ve ölçeklenebilir olması bu teknolojiye duyulan talebi artırmıştır. Ayrıca bakım, onarım, güncelleme gibi ilave işlerin bulut servis sağlayıcıya ait olması da bu teknolojiye duyulan talebi artırmaktadır. Fakat bulut bilişim bazı güvenlik endişelerini artırmaktadır. Bulut ortamlarında güvenlik sorunları genel olarak aşağıdaki nedenlerden dolayı ileri gelmektedir:

- Servisi talep edenin veri üzerinde kontrolünü kaybetmesi;
- Aynı fiziksel kaynakların farklı şirketler için kullanılması;
- Sanal makine kullanımından kaynaklanan güvenlik endişeleri;
- İletişim ağında veriler taşınırken ağda meydana gelen saldırılar.

IoT ile birlikte birçok yeni cihazın İnternete bağlanması: Nesnelerin İnterneti olarak kabul edilen akıllı cihazlar (akıllı gözlükler, akıllı saatler, ev otomasyon sistemleri, akıllı park, vb.) İnternete dahil edilmeye başlanmıştır ve yakın gelecekte bu şekilde İnternete bağlı cihaz sayısının milyarlara ulaşacağı ifade edilmektedir. Bu cihazların oluşturacağı büyük veri trafiği bilgisayar ağının denetimini zorlaştıracaktır. Ayrıca, bu aletler için oluşturulacak ZY'ler tahmin edilemeyecek riskler barındıracaktır.

2.1.3 Kritik altyapı sistemleri sayısındaki artıştan kaynaklanan hataları kullanarak yapılan siber saldırılar

Kritik altyapı sistemleri modern toplumun düzenli bir şekilde işlemesi için gerekli olan sistemlerin tümüdür. Finansal servisler, sağlık servisleri, haberleşme sektörü, vb. kritik alyapılara örnek gösterilebilir (Alcaraz ve Zeadally 2015, Wood 2016). Bu sistemlerin çoğu 7/24 hizmet vermektedir. Bu sistemlerde yaşanacak ciddi bir sorun bütün toplumu ve günlük yaşamı etkileyecektir. Yapılan son araştırmalara göre kritik altyapı sistemlerine karşı yapılan saldırıların hem sayısında hem de bıraktığı etkide ciddi bir artış olmuştur (Zimba vd. 2018, Lewis 2019). Bu sistemlerin yapısal olarak karmaşık oluşu ve coğrafi olarak farklı yerlerde olması ve sistemin verimli çalışması için İnternete ihtiyaç duyulması gibi sebeplerden dolayı güvenliğinin sağlanması ciddi zorluklar içermektedir. Kritik altyapı ve endüstriyel kontrol sistemi üç ana yapıdan oluşur: Kurumsal ağlar, merkezi denetim birimi ve veri toplama merkezi (Supervisory control and data acquisition-SCADA) ve uzak alt istasyonlar (Alcaraz ve Zeadally 2015, Cazorla vd. 2015). SCADA, sistemin genel kontrolünden ve yönetiminden sorumlu olup alt istasyonlardan gelen bilgi ve alarmları kendi sunucularında depolamaktadır. Uzak alt istasyonlar otomatikleştirilmiş sistemler olup uzak terminal birimleri, programlanabilir mantık denetleyicileri, alıcılar ve algılayıcılar gibi endüstriyel aygıtlardan oluşmaktadır.

Kurumsal ağlar, SCADA ve alt istasyonlar arasındaki veri alış verişi İnternet kullanılarak yapıldığı için birçok saldırının hedefi konumuna gelmiştir (Alcaraz ve Zeadally 2013). Bu saldırıların çoğu Truva Atları, virüsler ve solucanlar kullanarak yapılmaktadır. Bu sistemlere karşı yapılan saldırılar temel olarak iki sebepten dolayı başarılı olmaktadır.

İletişim ağı olarak İnternetin kullanılması sonucu genel veri iletiminden dolayı oluşan tehditler: İletişim ağındaki savunmasızlıklardan dolayı saldırgan, iletilen verinin içeriğini görüntüleyebilmekte, veri üstünde değişiklik yapabilmekte ve sistemi kullanan kişilerin kullanıcı adı ve şifre bilgilerine erişebilmektedir. Ayrıca, sistem kaynaklarına aşırı veri göndererek sistemin geçici olarak servis dışı kalmasına neden olabilmekte ya da kötü amaçlı komutlar göndererek sistem ayarlarında değişiklik yapabilmektedir.

Kritik altyapı sistemlerinin karmaşıklığından ve yapısından dolayı öne çıkan tehditler: Kritik altyapı sistemleri birçok birimden oluştuğu için karmaşık bir yapıya sahiptir. Sistemin

karmaşık olması ve başka sistemlerde kullanılmayan birçok bileşenin bu sistemlerde kullanılması, bu sistemlere karşı yapılan saldırı sayısını artırmaktadır. Siber saldırılar kurum ağına, SCADA merkezine ya da uzak alt istasyonlara yapılabilmektedir. Bu sistemlerden herhangi birine yapılacak saldırı bütün sistemi riske sokacaktır. Örneğin alt istasyonlarda bulunan sensörlere yapılacak başarılı bir saldırı uzak terminal birimlerinin etkisiz kalmasına sebep olup gerekli kontrol bilgilerin kontrol merkezine erişimini engelleyecektir. Bunun sonucunda izlenemeyen alt istasyonlar tamamen kontrolden çıkacaktır. Öte yandan, SCADA merkezine yapılacak başarılı bir saldırı sonucunda kötü niyetli kişiler sistemin bütün kontrolünü elde edebileceklerdir. SCADA sistemlerinde kullanılan iletişim protokolleri Modbus/TCP ve DNP3 (Distributed network protocol version 3) bazı ağ saldırıları karşısında savunmasızdırlar. Örneğin, Modbus/TCP ve DNP3 protokolü üzerinde veriler şifrelenmeden taşındığı için bu veriler görüntülenebilmekte ve içerikleri değiştirilebilmektedir.

2.2 Zararlı Yazılımlar ve Çoğalma Şekilleri

2.2.1 Zararlı yazılımların tanımı, tarihi ve tespiti

Yazılım dünyasında görülen ilk ZY'ler virüslerdir. Spencer'e göre (2011) ilk virüs düşüncesi 1950'li yıllarda John von Neumann tarafından kendi kendini kopyalayan "automata" olarak ileri sürülmesine rağmen gerçekte ilk virüs 1971 yılında laboratuvar ortamında geliştirilmiştir (Love 2018). 1980 ve 1990'lı yıllarda virüs yazımında hızlı bir artış görülmüştür. Teorik olarak ilk virüs tanımı 1980'li yıllarda Cohen (1986, 1987a) tarafından yapılmıştır. Daha sonraki yıllarda ZY'leri teorik açıdan ele alan araştırmalar pek yapılmamıştır. Bu alanda yazılan önemli bilimsel araştırmalar ve makaleler çizelge 2.2'de özetlenmiştir.

Çizelge 2.2 Zararlı yazılımlarla ilgili yapılan teorik yayınların listesi

Yazarlar	Eser İsmi	Yıl
Cohen	Bilgisayar virüsleri	1986
Cohen	Bilgisayar virüsleri: Teori ve deneysel	1987a
Adleman	Bilgisayar virüslerinin teorisi	1988
Cohen	Bilgisayar solucanlarının tanımı ve sonuçlar	1992
Cohen	Bilgisayar virüsleri hakkında kısa dersler	1994
Chess ve White	Algılanamayan bilgisayar virüsü	2000
Spinellis	Sınırlı uzunluktaki virüslerin tespiti NP-complete	2003
Zuo vd.	Bilgisayar virüslerinin zaman karmaşıklığı	2005

Virüslerin tanımı ve tespiti: Cohen'e (1992) göre virüs bilgisayar hafızasında ve diskte farklı formlarda bulunan sembol dizilimlerdir ve normal yazılımları virüslerden ayırmak neredeyse imkansızdır. Virüsler üç ana evreden oluşmaktadırlar: Çoğalma, tetiklenme ve hasar verme. Virüsler genellikle farklı programları kullanarak çoğalırlar ve kendilerini kopyalayacakları programlara daha önceden bulaşıp bulaşmadıklarını kontrol ederler. Eğer daha önce bulaşmışsa aynı programa bir daha bulaşmazlar. Virüslerin sürekli kendilerini çoğaltmaları ve yeni oluşan kopyalarında belirli şartlar altında çalışmaları belirli bir zaman sonra kurban sistemin hizmet verememesine ya da beklenilenin dışında davranışlar sergilemesine neden olmaktadır.

Cohen (1992) virüsü *V :=* [*F=RASTGELE- DOSYA; V'yi F'e kopyala*] olarak formüle etmiştir. Bilgisayar solucanını (computer worm) ise virüsün alt sınıfı olarak tanımlamaktadır öyle ki *W :=* [*F=RASTGELE- DOSYA; W'i F'e kopyala; F'i çalıştır*]. Cohen virüs ve bilgisayar solucanı tespitinin karar verilemez (undecidable) olduğunu ve bu yazılımların işletim sistemi kaynaklarından temizlenmesinin zor olduğu sonucuna varmıştır.

Cohen'e (1987a) göre program *P*'nin virüs olup olmadığını anlamak teoride çelişki içerdiği için mümkün değildir. Probleme karar verme problemi olarak bakacak olursak, *D* (karar verici) *P*'nin virüs olup olmadığına karar verecektir. Cohen'e göre *P*'nin virüs olup olmadığına karar verilemez çünkü eğer *P* virüs ise *D* tarafından virüs olarak işaretlenecek ve diğer programlar üzerinde değişiklik yapamayacak bu durumda virüs gibi davranmayacağından virüs olmamış olacaktır. Eğer *D* karar verici *P*'yi virüs olarak belirlemediyse *P* diğer programlarla etkileşime girerek yayılacaktır ve virüs olmuş olacaktır. Bu karar süreci çelişki (contradiction) içermektedir ve bu yüzden *P*'nin virüs olarak belirlenmesi mümkün değildir.

Chess ve White'a (2000) göre virüs bir virüssel kümedir (viral set-*V*) öyle ki *P* programının da virüs olabilmesi için *V*'nin örneği ya da *V* tarafından enfekte olmuş olması gerekmektedir. Basit anlamda virüs bir *V*'dir ve bu *V* tek bir programdan oluşmaktadır ve kendinin bir kopyasını oluşturmaktadır. Virüs örneği *P* ise *V*'nin tekrar çoğalmasıyla oluşan programdır. Karmaşık virüsler ise kendi *V*'sinin farklı formlarını oluşturarak çoğalmaktadır. Diğer bir ifadeyle kendi *V*'sini başka bir programa enjekte ederek çoğalan programlar virüs olarak tanımlanır. Orijinal program çalıştırılırken hem programa enjekte edilen kodlar hem de

programın kendi orijinal kodları çalıştırılır. Solucan ise kendi başına çoğalan *V*'dir. Virüslerin tespit edilmesini ise şöyle tanımlamaktadır (Chess ve White 2000): Algoritma *A*, *V* içeren program *P*'yi tespit eder şöyle ki, *A(P)* doğru sonuç dönderir ancak ve ancak *P* programı sonlu ve *V* tarafından enfekte olmuş ise. Yapılan araştırmalara göre virüsün bütün farklı kümelerini tespit edebilen bir algoritma bulunmamaktadır (Cohen 1987a-b, Chess ve White 2000, Spinellis 2003, Zuo vd. 2005).

Zararlı yazılımların tanımı ve tespiti: 1970'li yılların başında çıkan bilgisayar virüsleri yazılım dünyasında görülen ilk ZY'dir. Zamanla gerçek dünya gereksinimlerinin dijitalleşmesi, yerini basit virüslerden karmaşık ZY'lere bırakmış ve büyük kâr getiren bir sektöre dönüşmüştür. Her geçen gün binlerce yeni ZY dijital ortamlarda oluşturulmakta ve bu ZY'ler devletler ve şirketler için büyük riskler barındırmaktadır. Basit anlamda ZY bir dizi sembolün bilgisayar donanım ve işletim sistemi üzerinde yaptığı genelde istenmeyen değişiklikler olarak tanımlanabilir ve aşağıdaki gibi formüle edilebilir.

$$ZY := S\,[N] \tag{2.1}$$

$(s_1, s_2, \ldots, s_a) \in S$ *ve* $(n_1, n_2, \ldots, n_a) \in N$ olmak üzere

$$ZY := \{\, s_1[n_1],\ s_2[n_2], \ldots, s_a[n_a] \,\} \tag{2.2}$$

S sembolleri (teknik anlamda komutları), *N* ise nesneleri (bilgisayar kaynaklarını), *a* ise sonlu bir sayıyı ifade etmektedir.

Statik yöntemle ZY analizinde program yapısını ortaya çıkaran imzalar kullanılmaktadır. Program yapısının değişmesi ZY'ye ait imzayı değiştireceğinden dolayı bu analiz yöntemiyle bütün ZY'leri tespit etmek mümkün değildir. Statik analizi kullanan ticari yazılımlara örnek olarak antivirüs tarayıcıları gösterilebilir. Bu programlar yeni çıkan ZY'lerin çoğunu tespit edememektedir. Bu durum statik analiz yöntemiyle bütün ZY'leri tespit etmenin mümkün olmadığını göstermektedir. Diğer taraftan yazılım davranışları benzerlik gösterdiği için bu davranışları gruplayarak bütün ZY'leri tespit etmekte mümkün görünmemektedir. Başka bir deyişle bir yazılımın işletim sistemi ve diğer yazılımlarla etkileşimine bakıp bu yazılımın amacı hakkında bir yorumda bulunmak pekte mümkün görünmemektedir çünkü bütün yazılımların bu etkileşimleri az çok benzerlikler göstermektedir. Son yıllarda geleneksel

ZY'lerin aksine yeni nesil olarak tanımladığımız daha karmaşık ZY'ler oluşturulmaktadır. Yeni nesil ve geleneksel ZY'lerin karşılaştırılması çizelge 2.3'te görülmektedir.

Çizelge 2.3 Geleneksel ve yeni nesil zararlı yazılımların karşılaştırılması

Karşılaştırma parametresi	**Geleneksel**	**Yeni nesil**
Uygulama seviyesi	Basit kodlanmış	Gömülü kodlanmış
Davranış durumu	Statik	Dinamik
Çoğalma	Her kopya benzer	Her kopya farklı
Yayılma yolu	.exe uzantısı kullanma	Farklı uzantılarda kullanma
Sistemde kalıcılık	Geçici	Kalıcı
Proseslerle etkileşimi	Birkaç proses	Çoklu proses
Gizlenme tekniklerini kullanma	Hayır	Evet
Saldırı türü	Genel	Hedeflenmiş
Savunma/Karşı koyma	Kolay	Zor
Hedeflenen cihazlar	Genel bilgisayarlar	Bilgisayarlar ve farklı cihazlar

2.2.2 Zararlı yazılım türleri ve çoğalma şekilleri

Birçok siber saldırının kökeninde ZY'ler bulunmaktadır. Son yıllarda ZY kullanılarak hedef odaklı saldırılar yapılmaktadır. Bu tehdit ve saldırılar şöyle sıralanabilir (Samet ve Aslan 2018):

- Küresel Tehditler (Global Threats);
- Gelişmiş Kalıcı Tehditler (Advanced Persistent Threats-APTs);
- Dağıtık Hizmeti Engelleme Saldırıları (Distributed Denial of Service-DDoS Attacks).

İleri düzey ve önlenmesi zor olan saldırılar olarak bilinen küresel tehditler, APTs ve DDoS saldırıları genelde birden fazla tür ZY kullanılarak gerçekleştirilmektedir (Samet ve Aslan 2018). Küresel boyuttaki tehditler sosyal mühendislik teknikleri kullanılarak elektronik posta (e-posta) yoluyla yayılmakta ve tüm dünyayı etkilemektedir. Bu tip saldırılarda çok sayıda bilgisayar kullanılarak sistemlere saldırı yapılmaktadır. Bu durumda hem asıl saldırı kaynağını tespit etmek zorlaşmakta hem de farklı sistemler saldırıda kullanıldığı için saldırının başarılı olma olasılığı artmaktadır. Cutwail ve Pushdo (Pilling 2013) bilinen en büyük botnet guruplarından biri olup başka sistemlere istenmeyen e-posta göndermektedir. Bu e-postalar tanınmış çevrimiçi siteler, bankalar, sosyal medya siteleri ve telefon şirketlerinden geliyor gibi gösterilmektedir. APT'ler son yıllarda öne çıkan tehditlerin başında gelmektedir. Bu tehditler hedef odaklı saldırılar başlatmakta olup belirli kurum ve kuruşlara yönelik yapılmaktadır (Samet ve Aslan 2018). Bu saldırılar sonrası saldırıyı sistemde uzaklaştırmak zor olup sisteme büyük zararlar vermektedir. Saldırıların hedefinde

kurumların finansal kaynaklarına ve popülerliklerine zarar vermek yatmaktadır. DDoS saldırılarında, saldırı yapılan sisteme sürekli aşırı miktarda büyük boyutlu paketler gönderilerek ya da çok farklı kaynaktan benzer paketler gönderilerek sistem cevap veremez hale getirilir. DDoS saldırıları farklı şekillerde başlatılabilmekte ve çoğunlukla zombi bilgisayarlar (savunmasız bilgisayarlar) kullanılarak yapılmaktadır. Saldırı sırasında zombi bilgisayarların ve farklı IP adreslerinin kullanılması (Lachow 2013), saldırıyı başlatan ana kaynağı bulmayı zorlaştırmaktadır (Samet ve Aslan 2018). Saldırılarda yaygın olarak kullanılan ZY'ler ve türleri şöyle sıralanabilir: Virüs, solucan, Truva atı, arka kapı, fidye yazılımı, rootkit, robotlar, casus yazılım ve yazılım bombaları.

Virüs (Virus): Çoğalmak için başka programlara ihtiyaç duyan ve içinde zararlı kodlar bulunduran programdır. Virüsler girdikleri sistemlerin dosyalarında değişiklik yaparak çalışmaz hale getirebilmektedir. Ayrıca, başka programlara zararlı kod bölümlerini ekleyerek bu programların da virüs haline gelmelerine neden olmaktadır. Gelişmiş virüsler varlıklarını gizlemek için şifreleme ve paketleme gibi bazı gizlenme teknikleri kullanmaktadır. Varlıkları ancak antivirüs tarayıcıları ve ZY tespit sistemleri kullanılarak belirlenebilmektedir. Code Red, ILOVEYOU, Melissa, Nimda ve Stuxnet yaygın olarak bilinen virüslerdir (Engel 2015, Team 2016). Virüsler yazılma amaçlarına göre yedi farklı kategoriye ayrılabilir (Norman 2004, Anonymous 2013, Anonymous 2016a). Bunlar önyükleme sektör virüsleri, makro virüsleri, yerleşik virüsler, dosya bulaşma virüsleri, çok biçimli virüsler, tarayıcı ele geçiren virüsler ve çok bölümlü virüsler.

a) Önyükleme sektör virüsleri (boot sector viruses): Bu grup virüsler disket (floppy disk) ve sabit disklerin (hard disk) önyükleme sektörlerine (boot sector) yerleşmektedir. Bilgisayar ilk başladığında aktif hale gelen bu virüsleri sistemden temizlemek oldukça zordur. İnternetin evrimi sayesinde bu virüslerin etkisi azalmıştır.

b) Makro virüsleri (macro viruses): Genelde excel ve word gibi Microsoft belgelerini kullanarak yayılan virüs türleridir. Virüsün yayılması ve tetiklenmesi için makronun aktifleştirilmesi gerekmektedir. Kodun algılanması ve devre dışı bırakılması kolaydır.

c) Yerleşik virüsler (resident viruses): Yerleşik virüsler öncelikle kendilerini sisteme yükledikleri için orijinal virüs sistemden temizlense bile yerleşik kod çalışmaya devam etmektedir. Tespit edilmeleri ve sistemden temizlenmeleri zordur.

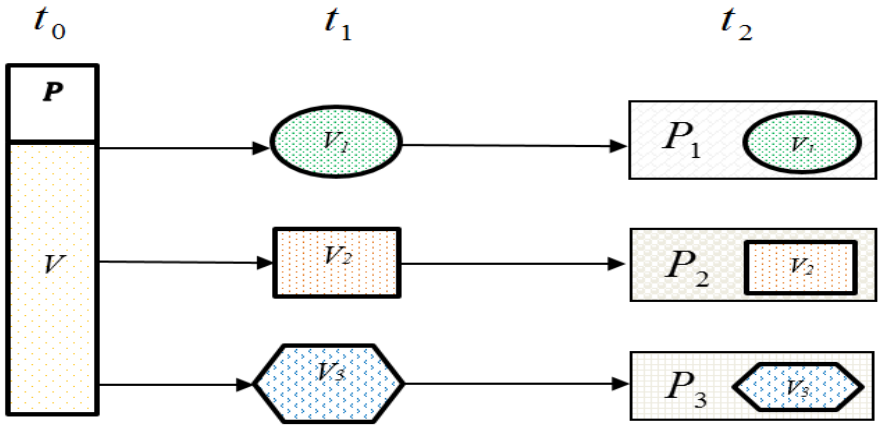
d) Dosya bulaşma virüsleri (file infector viruses): Bu tür virüsler başka dosyalara bulaşarak yayılırlar. Virüs, etkilediği dosyanın kodlarında değişiklik yaparak ya da zararlı kodları dosya kodlarına enjekte ederek yayılmaktadır.

e) Çok biçimli virüsler (polymorphic viruses): Her çalıştırıldığında kendilerini değiştiren virüs türüdür. Antivirüs tarayıcıları kullanılarak tespit edilmeleri çok zordur.

f) Tarayıcı ele geçiren virüsler (browser hijacker): Bu tür virüsler bazı web sitelerin popülerliğini ve ziyaretçi sayılarını artırmak amacıyla web tarayıcılara yerleşerek başlangıç ve arama sayfalarını değiştirirler.

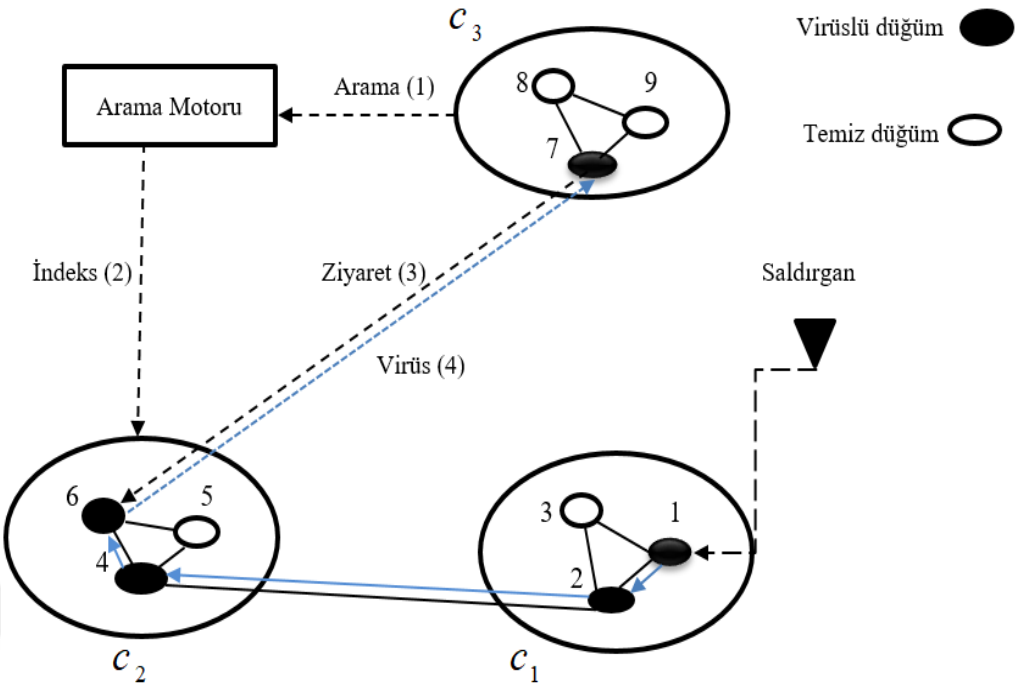
g) Çok bölümlü virüsler (multipartite virus): Bu tür virüsler hem önyükleme sektörüne hem de yürütülebilir dosyalara aynı anda bulaşabilmektedir. Bu virüsler öncelikle program dosyalarına bulaşmakta ve program açıldığında önyükleme (boot) kayıtlarına yerleşmektedir. Daha sonra bilgisayar yeniden başlatıldığında hafızaya yerleşerek diğer programlara bulaşmaktadır. Bundan dolayı verecekleri zarar diğer virüslere göre daha fazladır.

Virüsler diğer programlarla etkileşime girerek yayılırlar. E-posta eklentileri, ücretsiz yazılım platformları, sosyal medya ortamları ve USB bellekler başlıca yayılma ortamlarıdır. Virüs yayılırken bulaşacağı dosyaları rast gele belirleyebildiği gibi belli format ve özellikteki dosyaları da seçip bulaşabilmektedir. Şekil 2.1 virüsün her defasında farklı formunu üreterek çoğalmasını göstermektedir (Samet ve Aslan 2018). Her seferinde kendini değiştirdiği için virüs imzası değişecek ve antivirüs tarayıcıları tarafından tespit edilmesi zorlaşacaktır.



Şekil 2.1 Virüsün şekil değiştirerek yayılması

Şekil 2.2 virüsün arama motoru kullanarak sosyal medya ortamlarında yayılımını göstermektedir (Fu vd. 2017).



Şekil 2.2 Virüsün arama motoru kullanarak yayılması (Fu vd. 2017)

C_1, C_2, C_3 sosyal ağlarda farklı grupları göstermektedir. Öncellikle virüs saldırgan tarafından düğüm 1'e bulaştırılmaktadır. Düğüm 1'den diğer bağlantılı düğümlere bulaşmaktadır ($1 \rightarrow 2 \rightarrow 4 \rightarrow 6$). Fakat düğüm 6'dan düğüm 7'ye bağlantı olmadığı halde virüs bulaşabilmektedir çünkü düğüm 7 popüler bir dizi anahtarı arama motoru yardımıyla aramaktadır. Aranan popüler kelime grupları C_2'nin bulunduğu grupta indekslenmiştir. Eğer ilgili site düğüm 6'da ise, düğüm 6 ile düğüm 7 arasında sanal bağlantı (virtual connection) oluşmaktadır ve virüs düğüm 6'dan düğüm 7'ye ($6 \rightarrow 7$) geçmektedir.

Solucan (Worm): Bilgisayar ağlarını kullanarak yayılan ve bulaştığı sistemlerde izinsiz girişlere imkan veren programdır (Samet ve Aslan 2018). Genelde ağ üzerinde önce savunmasız makineleri belirleyip daha sonra bu makinelere izinsiz giriş yaparak kendini sisteme kopyalamaktadır (Nazario 2004). Solucanlar çoğalmak için başka programlara ihtiyaç duymazlar ve gizlenebilmek için kendi dosyalarını silerler. Ayrıca, sistemlerde arka kapılar açarak bu sistemleri başka saldırılarda kullanırlar. Son dönemlerde virüs-solucan özellikleri gösteren programlar görülmeye başlanmıştır (Samet ve Aslan 2018). Solucanlar dört farklı bölümden oluşmaktadır. Bu bölümler şöyle sıralanabilir: Hedef keşfi, yayılma-

dağıtım mekanizması, aktivasyon-tetiklenme ve payload (zararlı eylemler gerçekleştiren kod parçacıkları) (Nazario 2004, Christoffersen ve Mauland 2006, Pratama ve Rafrastara 2012).

a) Hedef keşfi: Bilgisayar solucanı bir dizi arama yaparak savunmasız sistemleri tespit etmekte ve kendini bu sistemlere kopyalayarak yayılmaktadır. Hedef kitlesi önceden belirlenebildiği gibi aktif-pasif tarama yapılarakta belirlenebilmektedir.

b) Yayılma ve çoğalma: Bilgisayar solucanı bir sistemden diğerine üç farklı yolla bulaşarak yayılmaktadır: Kendi kendine taşınarak (yayılma sürecinin bir parçası olarak), ikincil kanal açıp kullanarak ya da iletişim kanalına gömülü olarak (normal iletişimin bir parçası olarak) karşı sistemlere bulaşmaktadır.

c) Aktivasyon ve tetiklenme: Aktivasyon, solucan kodunun hedef bilgisayarda çalışmaya başlamasını ifade etmektedir. Tetiklenme üç farklı şekilde olmaktadır: Enfeksiyon süreci biter bitmez kendini aktif hale getirerek (en hızlı yayılım bu yolla olmaktadır), kullanıcı etkileşimi sonucu aktif hale gelerek (bu şekilde yayılım yavaş olmaktadır) ya da bazı durumların gerçekleşmesi sonucu aktifleşerek çalışmaktadır.

d) Payload: Yayılma ve çoğalma kodları dışında kalan zararlı kodlardır. Genelde verilmek istenen zarar ya da diğer kötü amaçlı hedefler bu kısımda bulunmaktadır. Bu kod bölümünde uzak bağlantı için solucan sistemde arka kapı açabilir, DoS başlatabilir, kredi kart bilgilerini toplayarak karşı tarafa yollayabilir ya da ilgili sisteme zarar verebilir. 1995 ve 2010 yılları arasında bilgisayar solucanı sayısı ve çeşidinde hızlı bir artış görülmüştür. Morris, Code Red, Nimda, Duqu ve Storm worm en iyi bilinen solucanlardır (Çizelge 2.4).

Çizelge 2.4 Yıllara göre bilgisayar solucanları ve karakteristik özellikleri

Sıra	Solucan ismi	Çoğalma şekli	Protokol	Payload	Exploit	Çıkış tarihi
1.	**Morris**	E-posta arka kapısı	TCP	Yok	Arabellek taşması	1988
2.	**Code Red I**	Rastgele tarama	TCP	DDoS	Arabellek taşması	2001
3.	**Code Red II**	Rastgele tarama	TCP	Arka kapı oluşturma	Arabellek taşması	2001
4.	**Nimda**	Rastgele tarama	TCP	Truva atı oluşturma, DoS	Güvenlik açığı	2001
5.	**SQL Slammer**	Rastgele tarama	UDP	Yok	Arabellek taşması	2003
6.	**Blaster**	Sıralı tarama	TCP	DDoS	Arabellek taşması	2003
7.	**Storm**	Tarama	TCP	Truva atı oluşturma, DDoS	Güvenlik açığı	2007
8.	**Conficker**	Sıralı tarama	TCP	Zarar verme	Arabellek taşması	2009
9.	**Duqu**	Hedef listesi	TCP	Arka kapı oluşturma	Zero-day güvenlik açığı	2011
10.	**NGRBot**	Sıralı tarama	UDP	Arka kapı oluşturma, DoS	Güvenlik açığı	2012

Truva Atı (Trojan Horse): Görünürde normal bir program gibi görünen gerçekte ise zararlı kodlar barındıran programdır. Sistemleri etkileyebilmeleri için iliştirilen programın çalıştırılması gerekmektedir. Genelde e-posta içine gömülerek ya da tarayıcılardaki savunmasızlıklardan dolayı İnternetten program indirme sırasında (indirilen normal program içine gömülüdürler) sistemlere bulaşmaktadır (Stallings vd. 2012). Truva atları başka dosyaları kullanarak çoğalmazlar ve kendi kendilerini kopyalamazlar. Bu tür yazılımlar sistemlerde arka kapılar açabilmekte, sistemleri izinsiz uzak erişimlere açabilmekte ve sistemlerde depolanan kritik verileri üçüncü kişilere gönderebilmektedir. Bulundukları sistemlerde fark edilmeleri çok zordur fakat girdikleri sistemleri yavaşlatmalarından dolayı varlıkları ancak uzun uğraşlar sonucunda anlaşılmaktadır.

Arka Kapı (Backdoor): Geleneksel güvenlik mekanizmalarını atlatarak sistemleri kullanıcı bilgisi dışında uzak erişimlere açan programdır. Arka kapılar çoğunlukla Truva atları kullanılarak ilgili sistemlere yüklenmekte ve yalnızca arka kapıyı oluşturan kişiler tarafından bilinmektedir (Samet ve Aslan 2018). Arka kapılar büyük ölçekli karmaşık saldırılarda virüs ve solucanlar tarafından kullanılmaktadır.

Fidye Yazılımı (Ransomware): Bulaştığı sistemde sistemin bir kısmını ya da tamamını şifreleyerek, verilerin kullanıcılar tarafından görüntülenmesini engelleyen programdır (Samet ve Aslan 2018). Kullanıcılar, verilerini görüntüleyebilmek için saldırganın talep ettiği fidyeyi ödemesi gerekmektedir (Fossi 2011) fakat fidyenin ödenmesi verilere erişimi garanti etmez. Fidye yazılımları son yıllarda çok görülmeye başlanmasına rağmen (2010) konsepti ilk olarak 1989 yılında AIDS Truva atı virüsünde görülmüştür (Gazet 2010). WinLock (2010), CryptoLocker (2013-2014), TB-Locker (2014), SimpleLocker (2015-2016), NotPetya (2016) ve WannaCry (2017) iyi bilinen fidye yazılımlarıdır. Symantec (O'Brien 2017) raporuna göre son yıllarda fidye yazılımların hem sayısında hem de dünya ekonomisine verdiği zararda hızlı bir artış olduğu görülmüştür. McAfee 2016 raporuna göre 2015 ve 2016 yılları arasında ransomware saldırı sayısında %100'den fazla artış görülmüş (Beek vd. 2016) ve bu artış sonraki yıllarda da devam etmiştir. Trend Micro güvenlik raporuna göre WannaCry fidye yazılımının 2017 yılında dünya ekonomisine verdiği zarar yaklaşık 4 milyar dolar olarak ölçülmüştür.

Rootkit: Bilgisayar sistemine yönetici düzeyinde erişim hakkı sunan ve dosyalarını işletim sisteminden gizlemek suretiyle varlığını sürdüren programlar grubudur (Samet ve Aslan 2018). En tehlikeli ZY'lerin başında gelen rootkitlerin amacı çoğalmak değil, bulundukları sistemlerde varlıklarını gizlemektir. Çoğunlukla tek başlarına saldırılarda kullanılmayan bu yazılımlar virüs, solucan ve Truva atlarıyla birlikte kullanıldıklarında çok yıkıcı saldırılar başlatabilmektedirler. İşletim sistemi seviyesinde çalışabildikleri için hem çok tehlikelidirler hem de antivirüs tarayıcıları tarafından yakalanmaları oldukça zordur (Davis vd. 2009).

Robotlar (Robots): Botnetler bir dizi robotlardan oluşmakta olup savunmasız bilgisayar sistemlerini ele geçirerek daha sonra yapacakları saldırılarda kullanılırlar. Robotlar üç kısımdan oluşmaktadır: Saldırgan, komuta-kontrol mekanizması ve zombi bilgisayarlar. Komuta-kontrol mekanizması kullanılarak savunmasız bilgisayarlar tespit edilmekte ve bu bilgisayarlar kullanılarak saldırılar gerçekleştirilmektedir. Symantec (2010, 2015) raporuna göre istenmeyen e-postaların yaklaşık %80'i 10 botnet tarafından gönderilmektedir. Botnetler kullanılarak kurban sistemler çalışmaz hale getirilebilmekte, zombi olarak kullanılan sistemlerden veriler sızdırılabilmekte veya zombi sistemler üçüncü kişilere kiraya verilebilmektedir.

Casus Yazılım (Spyware): Kullanıcı ya da kurumların bilgilerine gizlice erişip üçüncü kişilere gönderen yazılımdır. Genellikle kişisel bilgilere erişmek ya da kullanıcı alışkanlıklarını saptamak amacıyla kullanılmaktadır. Keylogger (tuş kaydedici) en iyi bilinen casus yazılımdır. Keylogger kullanıcı bilgisi dışında çalışarak klavyeden yapılan her vuruşu kaydetmekte ve bu kayıtları üçüncü kişilere göndermektedir. Casus yazılımlar web siteleri aracılığıyla ya da ücretsiz paylaşılan programlar aracılığıyla yayılmaktadır (Eilam 2011). Bu siteler ziyaret edildiğinde veya bu programlar sistemlere indirilip kullanıldıklarında sistemde yayılımları gerçekleşmektedir.

Yazılım Bombaları (Time/Logic Bombs): Belirli bir koşul gerçekleştiğinde ya da belirlenen tarih geldiğinde çalışan (Stallings ve Brown 2012) ve sistem üzerinde yıkıcı etkileri bulunan programlardır. Genelde saldırıyı planlayanlar dışardan değil de içerden biri olmaktadır. Fark edilmeleri çok zor olup ancak sistem zarar gördükten sonra varlıkları tespit edilebilmektedir. Sistem yedeğinin belirli aralıklarla alınması bu yazılımların yıkıcı etkilerini azaltmada büyük önem taşımaktadır.

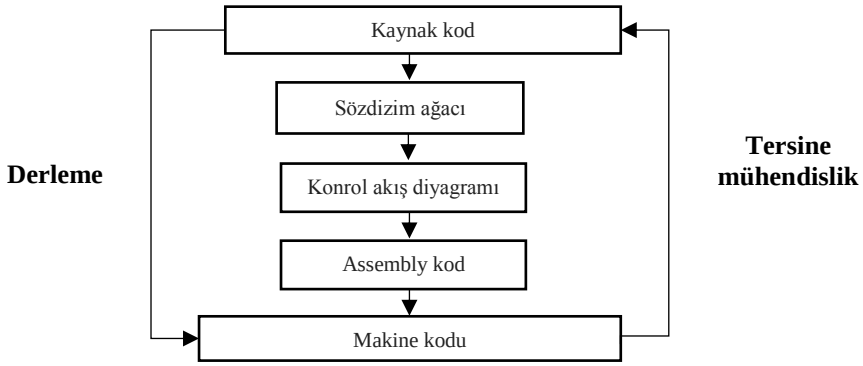
ZY'lerin yayılmalarını engellemek ve bu yazılımlardan tam olarak korunmak mümkün görünmemekle birlikte bazı proaktif önlemler alınarak bu yazılımların yayılması ve sistemlere verecekleri zararlar azaltılabilmektedir. Bu önlemler şöyle sıralanabilir:

- Kritik sistemlerde yalnızca güvenirliği test edilmiş yazılımlar kullanılmalıdır;
- Sistemler belirli aralıklarla kontrol edilerek herhangi bir anormallik olup olmadığı saptanmalıdır;
- Sık sık sistem yedeği alınmalıdır;
- Sistemdeki zafiyetlerin belirlenmesi amacıyla sızma testleri yapılmalıdır;
- İşletim sistemi ve uygulama programları belli aralıklarla güncellenmelidir;
- ZY'leri tespit eden programlar kullanılmalıdır;
- Güvenli İnternet kullanımı konusunda kullanıcılar eğitilmelidir.

2.3 Zararlı Yazılım Gizlenme Teknikleri

Program içeriğinin üçüncü kişiler tarafından görüntülenmesinin engellenmesi adına birçok programın kaynak kodu gizli tutulmaktadır. Fakat tersine mühendislik (reverse engineering) kullanılarak derlenmiş program dosyasından kaynak kodun bir kısmı ya da tamamı çıkartılabilmektedir (Eilam 2011, şekil 2.3). Kod gizleme (obfuscation) tekniği tersine

mühendisliği engelleyerek kaynak kodların üçüncü şahıslar tarafından görülmesini ve anlaşılmasını engellemeyi amaçlamaktadır (Krister 2009, Hahn 2014).



Şekil 2.3 Derleme ve tersine mühendislik işlemleri

Obfuscation tekniği yazılım üzerinde kurcalamayı önleme, yazılımın güvenliğini artırma ve fikri mülkiyet hakkını koruma gibi haklı nedenlerden dolayı normal programlar tarafından kullanılabilmektedir (Linn ve Debray 2003). Ayrıca, obfuscation, ZY'ler tarafından girdikleri sistemlerde uzun süre tespit edilmeden kalabilmek için de sıkça kullanılmaktadır (Collberg ve Thomborson 2002, Aslan ve Samet 2020). Obfuscation, hem manuel hem de otomatik denetlemeyi engelleyerek program kodlarının analiz edilmesini zorlaştıran bir dizi program dönüşümleridir. Diğer bir değişle obfuscation bir dönüşüm fonksiyonu f (x) kullanılarak program P'nin P' diye yeni bir programa dönüştürülmesidir öyle ki P' programı P programıyla aynı davranışlara sahiptir fakat P' kurcalamaya ve tersine mühendislik tekniklerine karşı korumalıdır (Collberg vd. 1997, 1998).

$P' = f(P)$;

$P \rightarrow$ Kurcalamaya karşı korumasız

$P' \rightarrow$ Kurcalamaya karşı korumalı

İlk yıllarda kulanılan obfuscation teknikleri basit olduklarından imza tabanlı yöntemle kolayca tespit edilebilmekteydi, zamanla bu tekniklerin karmaşık hale gelmesi tespit edilmelerini zorlaştırmıştır (Shahid 2014, çizelge 2.5-2.6). Obfuscation farklı yöntemlerle yapılabilmektedir: Şifreleme (encryption), oligomorfik (oligomorphic), çokbiçimlilik (polymorphic), başkalaşım (metamorphic), gizli hareket (stealth) ve paketleme (packing) (Siddiqui 2008, Alzarooni 2012, Shahid 2014).

Çizelge 2.5 Chernobyl/CIH virüsünün orjinal ve obfuscated kod bloğu (Dalla Preda 2007)

Orjinal kod		**Obfuscate edilmiş kod**	
E8 00000000	call 0h	E8 00000000	call 0h
5B	pop ebx	5B	pop ebx
8D 4B 42	lead ecx, [ebx + 42h]	8D 4B 42	lead ecx, [ebx + 42h]
51	push ecx	90	nop (∗)
50	push eax	51	push ecx
50	push eax	50	push eax
0F01 4C 24 FE	sidt [esp - 02h]	50	push eax
5B	pop ebx	90	nop (∗)
83 C3 1C	add ebx, 1Ch	0F01 4C 24 FE	sidt [esp - 02h]
FA	cli	5B	pop ebx
8B 2B	mov ebp, [ebx]	83 C3 1C	add ebx, 1Ch
		90	nop (∗)
		FA	cli
		8B 2B	mov ebp, [ebx]

Çizelge 2.6 Chernobyl virüs imzaları (Dalla Preda 2007)

Orijinal kod imzası	**Obfuscate edilmiş kod imzası**
E800 0000 005B 8D4B 4251 5050 0F01 4C24 FE5B 83C3 1CFA 8B2B	E800 0000 005B 8D4B 4290 5150 5090 0F01 4C24 FE5B 83C3 1C90 FA8B 2B

2.3.1 Gizleme yöntemleri

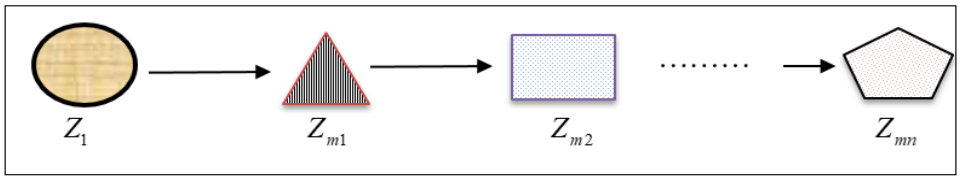
Encryption: ZY'lerin varlıklarını gizlemek için zararlı kodları barındıran bölümlerinin şifrelenerek görünmez hale getirilmesi işlemidir (Alzarooni 2012). Bu tür ZY'ler genellikle sabit bir şifre çözücü (decryptor), şifreleme anahtarı ve şifrelenmiş payload kısımlarından oluşur. ZY davranışları ve işlevsellikleri şifrelendiğinden dolayı tespit sistemleri kullanılarak tespit edilmeleri zordur. Her seferinde ZY kendi kopyasını oluştururken farklı bir şifreleme anahtarı kullanarak payload kısmını şifrelediğinden dolayı her kopya birbirinden farklı görünmektedir. Fakat aynı şifre çözücü kullanıldığı için şifre çözülürken tespit edilmeleri mümkündür çünkü bu durumda şifre çözücü bilgileri imza olarak kullanılabilmektedir.

Oligomorphic: Bu yöntemde hem ZY'nin payload kısmı şifrelenirken farklı anahtar hem de şifre çözülürken farklı şifre çözücü (decryptor) kullanıldığı için tespit edilmeleri encryption tekniği kullanılarak oluşturulmuş ZY'lere göre daha zordur (Aslan vd. 2020). Fakat kısıtlı sayıda farklı decryptor kullanabildikleri için detaylı inceleme sonucunda tespit edilebilmektedirler (Szor ve Ferrie 2001).

Polymorphic: Bu yöntemde zararlı kod kısmı encryption ve oligomorphic yöntemlerinde olduğu gibi her seferinde farklı anahtar kullanılarak şifrelenmekte fakat ZY'nin şifreli payload kısmı şifre çözücünün birkaç kopyasını barındırmaktadır (Aslan vd. 2020). Sınırsız

sayıda decryptor kullanılabilmekle birlikte bazı polymorphic ZY'lerin payload kısmı katmanlı olarak da şifrelenebilmektedir (Szor ve Ferrie 2001). Polymorphic ZY'ler üç kısımdan oluşmaktadır: Payload, şifre çözme motoru (decryption engine) ve değişim motoru (mutation engine). Her seferinde aynı ZY'nin farklı formları üretilmekte ve her biri birbirinden farklı olmaktadır. Bu durumda ZY'nin farklı formları benzer davranışlar göstermesine ragmen, program imzaları değişmektedir (Dalla Preda 2007). Program imzaları değiştiği için antivirüs tarayıcıları kullanılarak tespit edilmeleri zordur (Ahmadi vd. 2013).

Metamorphic: Şifreleme kullanmayan bu yöntemde ZY her çalıştırıldığında işlem kodu (Opcode) değiştirilmektedir. Ayrıca, bu yöntem dinamik kod gizleme olarak da adlandırılmaktadır. Bu tür ZY'leri tespit etmek çok zordur çünkü her yeni kopya tamamen farklı bir imzaya sahip olmaktadır (Shahid 2014, şekil 2.4, çizelge 2.7).



Şekil 2.4 Metamorphic zararlı yazılımın çoğalması

Çizelge 2.7 Win32/Evol virüsünün metamorphic teknik kullanarak çoğalması (Szor ve Ferrie 2001)

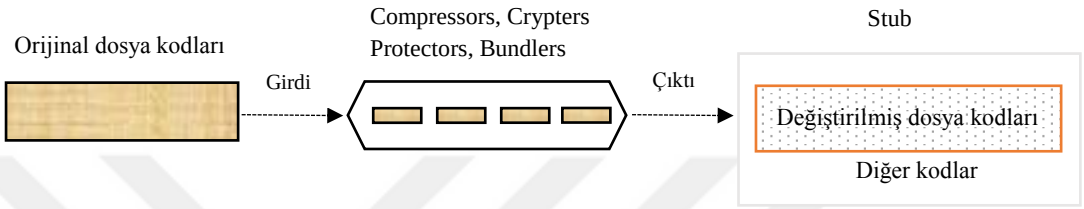
Versiyon 1: C7060F000055 mov dword ptr [esi], 5500000Fh C746048BEC5151 mov dword ptr [esi+0004], 5151EC8Bh
Versiyon 2: BF0F000055 mov edi, 5500000Fh 893E mov [esi], edi 5F pop edi 52 push edx B640 mov dh, 40 BA8BEC5151 mov edx, 5151EC8Bh 53 push ebx 8BDA mov ebx, edx 895E04 mov [esi+0004], ebx
Versiyon n: BB0F000055 mov ebx, 5500000Fh 891E mov [esi], ebx 5B pop ebx 51 push ecx B9CB00C05F mov ecx, 5FC000CBh 81C1C0EB91F1 add ecx, F191EBC0h; ecx=5151EC8Bh 894E04 mov [esi+0004], ecx

Stealth: Kod koruması olarak da adlandırılan bu yöntem doğru bir şekilde analiz edilmeyi önlemek için bir dizi karşı teknik uygulamaktadır (Szor 2005). Örneğin, sistem üzerinde değişiklik yapıp hem de bu değişikliği tespit sistemlerinden gizli tutabilmektedir (Alzarooni 2012). Anti-disassembly, anti-debugging, anti-emulation, anti-heuristic ve anti-monitoring kullandığı bazı tekniklere örnek gösterilebilir (Siddiqui 2008).

Packing: Paketleme, bir ZY'nin tespit edilmemesi için sıkıştırılarak ya da şifrelenerek gerçek kodlarının gizlenmesidir (Şekil 2.5). Bu yöntem sayesinde ZY'ler güvenlik duvarı ve antivirüs tarayıcılarını rahatça atlatabilmektedir (Yan vd. 2008, Aslan ve Samet 2020). Paketleme sırasında ZY sıkıştırılarak bazı durumlarda da şifrelenerek diskte saklanmakta ve çalışma esnasında hafızaya (RAM) yüklenirken eski haline dönüştürülerek çalışmaktadır. Genel olarak çalıştırılabilir Windows dosyaları PE formatta bulunmaktadır. Paketleme sırasında packer paketlenmek istenen programı iç yapılarına ayrıştırmaktadır. Daha sonra, PE başlık bilgileri, bölümler ve kullanılan metotların tutulduğu bilgiler belirlenerek yeni bir formatta program çalışma noktası (original entry point) değiştirilerek paketleme işlemi yapılmaktadır. Yeni oluşturulan program blogu stub olarak adlandırılmaktadır. Paketlenen programların antivirüs tarayıcıları tarafından analiz edilebilmeleri için paketlerden çıkartılmaları (unpacking) gerekmektedir. Uygulanış biçimine göre paketleyiciler dört farklı gruba ayrılabilir (Yan vd. 2008, Aslan ve Samet 2020): Sıkıştırıcılar (compressors), krypterler (crypters), koruyucular (protectors) ve paketçiler (bundlers).

- **Compressors:** Dosya boyutunun sıkıştırılarak azaltılmasıdır. Genellikle bu amaç için UPX (ultimate packer for executables) ve ASPack gibi hazır paket programlar kullanılmaktadır. Bu yöntemle paketlenen programların tersine mühendislik kullanılarak çözülmeleri nerdeyse imkansızdır.
- **Crypters:** Çalıştırılabilir dosyanın şifrelenerek sıkıştırılması işlemidir. Önce zararlı dosya şifrelenir, bu aşamada şifrelenen dosyanın içeriği görüntülenemediği için geleneksel antivirüs tarayıcıları tarafından normal program olarak algılanır. Daha sonra şifrelenmiş dosya paketten çıkartılır ve şifresi çözülerek hafızada çalıştırılır. Şifresi çözülen dosya hafızada çalışmaya başlarken şifresi çözüldüğü için bulgusal (heuristic) tabanlı algılayıcılar kullanılarak da tespit edilememektedirler. Yoda's ve PolyCrypt en yaygın kullanılan krypter örnekleridir.

- **Protectors:** Compressors ve crypters yöntemlerinin iyi yönlerini birleştiren yöntemdir. Armadillo ve Themida araçları protector oluşturmak için kullanılan yazılımlara örnek gösterilebilir.
- **Bundlers:** Birden fazla çalıştırılabilir dosyanın paketlenerek tek bir çalıştırılabilir dosya şekline getirilmesidir (Şekil 2.5). Paketlenen dosyalar paketten çıkartılırken diske değilde kendi paketi içerisine çıkartılarak çalıştırılmaktadır. PEBundle ve MoleBox araçları yaygın olarak kullanılan bundler yazılımlarıdır.



Şekil 2.5 Kod paketleme işlemi

2.3.2 Gizleme sırasında kullanılan teknikler

Yaygın olarak kullanılan obfuscation teknikleri şöyle sıralanabilir (Siddiqui 2008, Alzarooni 2012, Shahid 2014, Canell 2016): Yazmaçlara yeniden atama yapmak (register reassignment), komut değiştirme (instruction substitution), ölü kod ekleme (dead-code-insertion), kod aktarımı (code transposition), kod entegrasyonu (code integration), alt yordam sıra değişikliği (subroutine reordering) ve kodlama (encoding). Obfuscation sırasında bu tekniklerden biri ya da birkaçı bir arada kullanılarak aynı kodun farklı biçimleri oluşturulmaktadır.

Register ataması (register reassignment/variable renaming): ZY'nin kullandığı registerlerin veya değişkenlerin bir kısmının ya da tamamının kullanılmayan register ve değişkenlerle yer değiştirmesidir (Çizelge 2.8). Bu durumda program kodları değişmesine rağmen program davranışı aynı kalmaktadır.

Çizelge 2.8 Registerlerin yer değiştirmesi (Shahid 2014)

lea eax, [RIP+0x203768] add eax, 0x10 jmp eax	lea edx, [RIP+0x203768] add edx, 0x10 jmp edx

Komut yer değiştirme (instruction substitution/equivalent code replacement): Bazı kod bloklarının eş değeri ile yer değiştirmesidir (Çizelge 2.9). Program komut dizilimi değişmesine rağmen program davranışları aynı kalmaktadır.

Çizelge 2.9 Eşdeğeriyle kod değiştirme

Orjinal kod	Yeni kod

static void Main(string[] args)
{
int a = 20; int b = 80;
int sonuc = a * b;
}
	static void Main(string[] args)
	{
	int a = 40/2; int b = 80/1;
	int sonuc = (2 * a * b *2)/4;
	}

Kod ekleme (dead-code-insertion/garbage insertion): Program davranışını etkilemeyecek şekilde gerekli olmayan kod ya da kod bloklarının programa eklenmesidir (Çizelge 2.10). İşlem yapmayan kod (NOP) ve koşulsuz dallanma (unconditional jump) bu teknikte sıkça kullanılmaktadır. Genellikle aynı programın farklı formunu elde etmek için kullanılır. Program dizilimi değiştiği için statik analizi zorlaştırmakta ve program imzası değiştiği için antivirüs tarayıcılarını atlatabilmektedir.

Çizelge 2.10 Programa boş kod ekleme (Alzarooni 2012)

1:	**push eax**
2:	nop
3:	**dec esi**
4:	**add [eax], al**
5:	nop
6:	**or al, [eax]**
7:	jmp 12
8:	mov [ebp-0x18], esp
9:	mov [ebp-0x14], 0x4010e0
10:	and eax,0x1
11:	mov [ebp-0x10], eax
12:	**add [eax],al**
13:	**push 0x0**
14:	**cmp al, [eax]**
15:	nop
16:	**pop esp**
17:	**add bl, dh**
18:	**xchg edi, eax**

Kod aktarımı (code transposition/instruction reordering/code reordering): Programın davranışını değiştirmeyecek şekilde birbirinden bağımsız komutların yer değiştirmesi işlemidir. Bu tekniğin uygulanışı zor olmakla birlikte uygulandığında ZY'nin tespit edilmesi zorlaşmaktadır. Çizelge 2.11'de program komutlarının nasıl yer değiştirdiği gösterilmektedir.

Çizelge 2.11 Program komutlarının yer değiştirmesi (Shahid 2014)

Orjinal kod	**Sırası deşiltirilmiş kod**
a = 10; b = 20; x = a * b;	a = 10; b = 20; x = b * a
Assembly kod	**Değiştirilmiş assembly kodu**

movl [rbp-0xc], 0xa # a = 10
movl [rbp-0x8], 0x14 # b = 20
mov eax, [rbp-0xc];
imul eax, [rbp-0x8] # a * b
mov [rbp-0x4], eax # x = a * b
	movl [rbp-0xc], 0xa # a = 10
	movl [rbp-0x8], 0x14 # b = 20
	mov eax, [rbp-0x8] (yeniden sıralanmış)
	imul eax, [rbp-0xc] # b * a (yeniden sıralanmış)
	mov [rbp-0x4], eax # x = b * a

Makine kodu	**Değiştirilmiş makine kodu**
c7 45 f4 0a 00 00 00 c7 45 f8 14 00 00 00 8b 45 f4 0f af 45 f8 89 45 fc	c7 45 f4 0a 00 00 00 c7 45 f8 14 00 00 00 8b 45 f8 0f af 45 f4 89 45 fc

Kod entegrasyonu (code integration): Normal yazılım kodlarının program davranışını çok değiştirmeyecek şekilde zararlı kodlarla birleştirilmesidir.

Altyordam sıralaması (subroutine reordering): Program alt kodlarının rastgele değiştirilmesidir. Bu metot uygulandığında programın kod dizilimi değişmesine (Çizelge 2.12) rağmen davranışı benzer kalmaktadır.

Çizelge 2.12 Alt program kodlarını rastgele yer değiştirme

Orjinal kod	**Değiştirilmiş kod**

static void Main(string[] args)
{
int a = 10; int b = 45;
enBuyukBul(a, b);
enKucukBul(a,b);
}
static void enBuyukBul(int a, int b)
{
if (a > b) {Console.WriteLine(a);}
if (a > b) {Console.WriteLine(b);}
}
static void enKucukBul(int a, int b)
{
if (a < b) {Console.WriteLine(a);}
if (b < a) {Console.WriteLine(b);}
}
	static void Main(string[] args)
	{
	int a = 10; int b = 45;
	enBuyukBul(a, b);
	enKucukBul(a,b);
	}
	static void enKucukBul(int a, int b)
	{
	if (b < a) {Console.WriteLine(b);}
	if (a < b) {Console.WriteLine(a);}
	}
	static void enBuyukBul(int a, int b)
	{
	if (a > b) {Console.WriteLine(b);}
	if (a > b) {Console.WriteLine(a);}
	}

Kodlama (encoding): Base64 gibi kodlama teknikleri kullanarak kodun bir formattan başka bir formata çevrilmesidir (Hada 2000, Canell 2016). Çizelge 2.13 "merhaba dunya" programının orijinal hali ve obfuscate edilmiş halini göstermektedir. "Merhaba dunya" programı ASCII onaltılık ve Base64 kodlama teknikleri kullanılarak bir formdan başka bir

forma dönüştürülmüştür. Çizelge 2.13'te görüldüğü gibi orijinal kod insan tarafından rahatça anlaşılırken değiştirilmiş kod anlaşılmaz hale getirilmiştir.

Çizelge 2.13 Orjinal "merhaba dunya" programı ve ASCII onaltılık-Base64 kodlama kullanılarak formatı değiştirilmiş kod

Orjinal program kodu	Obfuscate edilmiş kod

class Program
{
static void Main(string[] args)
{
Console.WriteLine("merhaba dunya");
Console.Read();
}
}
	Y2xhc3MgUHJvZ3JhbQ= =
	IHsK
	c3RhdGljIHZvaWQgCg= =
	IHsK
	Q29uc29sZS5Xcml0ZUxpbmU=
	KCI=6D6572686162612064756E7961Iik7
	Q29uc29sZS5SZWFkKCk7
	fQo=
	fQo=

2.4 Değerlendirme

Siber saldırılar yapısı gereği dinamik olup belirli aralıklarla saldırı biçimini ve hedef kitlesini değiştirerek tüm dünyayı etkilemektedir. İnternetin aşırı yaygınlaşmasından dolayı siber saldırılar ve etkileri her geçen gün artmaktadır. Yazılan uygulama programlarındaki hatalar, İnternette kullanılan protokollerin günümüz ihtiyaçlarını karşılamadaki yetersizliği ve karmaşıklığı, ağa her gün yeni cihazların eklenmesi (IoT ve akıllı telefonlar) ve kritik öneme sahip sistemlerin yapısı gereği karmaşık oluşu ve var olan sistemlerle tam uyum göstermemesi siber saldırıların artma sebepleri arasında gösterilebilir. Bu saldırıların çoğu ZY'ler kullanılarak gerçekleştirilmektedir. ZY'ler zamanla şekil ve yöntem değiştirerek hem saldırı şiddetini artırmakta hem de gizlenme teknikleri kullanarak doğru analiz ve tespit edilmeyi engellemektedir. Artık birçok siber saldırı farklı ZY türleri virüs, solucan, Truva atı, vb. birlikte kullanılarak yapılmaktadır. Bundan dolayı siber saldırılar incelenirken saldırının kaynağı, saldırıda kullanılan ZY türleri ve özellikleri detaylı olarak incelenerek karşı önlemler alınmalıdır. Bu konuda kullanılan ZY türleri ve aileleri, genel özellikleri ve kullandıkları gizlenme tekniklerini belirlemek, ZY davranışlarını ve özelliklerini tespit ederken büyük önem taşımaktadır. Bundan dolayı veri setleri ve tespit sistemi oluşturulurken bu bölümde elde edilen bilgiler kullanılmaktadır.

3. KAYNAK ÖZETLERİ VE LİTERATÜR DEĞERLENDİRMESİ

Bu bölümde ZY tespiti ve bu alanda yapılan çalışmalar incelenmiştir. ZY tespiti, program içeriğininin analiz edilerek ilgili programın zararlı ya da normal olduğuna karar verilmesidir. Tespit aşaması üçe ayrılmaktadır:

1. Zararlı yazılım analizi;
2. Zararlı yazılım özellik çıkarma yöntemleri;
3. Zararlı yazılımlar ve sınıflandırma.

Bölüm 3.1'de ZY analizi ve türleri detaylı bir şekilde anlatılmıştır. ZY'ler analiz edilirken yazılımın ihtiva ettiği kod blokları incelenerek program yapıları ya da davranışları elde edilmektedir. Bölüm 3.2'de ZY özellik çıkarma yöntemleri anlatılmıştır. Bu bölümde veri madenciliği teknikleri, bir önceki bölümde (3.1) elde edilen program yapıları ve davranışlar kullanılarak programa ait özellikler oluşturulmaktadır. Bölüm 3.3'te elde edilen özellikler kullanılarak sınıflandırma yapılmaktadır. Sınıflandırma yapılırken ML algoritmaları kullanılmaktadır. Bölüm 4.4'te ZY tespit yaklaşımları ve bu alanda yapılan literatür çalışmaları sınıflandırılarak anlatılmıştır. Çalışmaların özgün değerleri, avantaj ve dezavantajları karşılaştırmalı olarak verilmiştir.

3.1 Zararlı Yazılım Analizi

ZY'lerin nasıl çalıştığını anlamak, yayılmasını engellemek ve ZY'leri tespit etmek amacıyla yapılan çalışmaları kapsamaktadır (Samet ve Aslan 2018). ZY analizi sadece antivirüs üreticileri tarafından değil, aynı zamanda büyük şirketlerin siber güvenlikten sorumlu birimleri tarafından da yapılmaktadır. Bunlar siber güvenlik uzmanları, sistem mühendisleri ya da ağ yöneticileri olabilmektedir. Bu analistler ZY analizi yaparak aşağıdaki durumları tespit etmeye çalışırlar:

1. Sistemde hangi güvenlik zafiyetinin kullanılarak saldırının başlatıldığını;
2. Hangi verilerin zarar gördüğünü ve çalındığını;
3. Hangi makine ve programların etkilendiğini;
4. Etkilenen makinelerin ve programların eski haline nasıl döndürüleceğini;
5. Yapılabilecek potonsiyel saldırıları.

ZY analizi temel statik analiz, ileri düzey statik analiz, temel dinamik analiz ve ileri düzey dinamik analiz olmak üzere dört gruba ayrılmaktadır (Aslan ve Samet 2017a). Statik analizde program kodları çalıştırılmadan ZY'ler incelenmektedir. Zararlı kodlar çalıştırılmadığından sistemde herhangi bir zarara neden olmamaktadır (Aslan 2017). Dinamik analizde ise ZY kodları çalıştırılarak inceleme yapılmaktadır. Sistemin zarar görmemesi için analizler çoğunlukla sandbox ya da sanal makine ortamlarında gerçekleştirilmektedir.

3.1.1 Temel (basit) statik analiz

Temel statik analiz ZY'nin karakteristik özellikleri hakkında genel bir değerlendirmede bulunmak amacıyla yapılmaktadır. Bu analiz türünde ZY'ye dışardan bakılarak genel bir değerlendirme yapılmakta ve analiz detaylandırılmamaktadır. Analiz sırasında antivirüs tarayıcıları (ClamAV, Karspersky, McAfee, Norton, vb.) ve statik analiz araçları (BinText, Md5deep, PEiD, PEview, vb.) kullanılmaktadır (Aslan ve Samet 2017a). Bu araçlar kullanılarak hash değerleri, kullanılan string dizilimler, metotlar ve dosya başlıkları belirlenerek analiz gerçekleştirilmektedir (Hahn 2014). Online olarak hizmet veren VirusTotal birçok antivirüs tarayıcısı barındırmaktadır ve temel analiz için yaygın olarak kullanılabilmektedir. MD5 ve SHA-256 hashleri ise ZY'leri etiketlemek ve tanımlamak için yaygın olarak kulanılan ZY imzalarıdır.

3.1.2 İleri düzey (gelişmiş) statik analiz

Bu analiz yönteminde program komutları detaylı olarak incelenerek analiz gerçekleştirilmektedir. Bilgisayar komutları işlemci tarafından sırasıyla yürütülen adımlar olarak tanımlanmaktadır ve farklı soyutlama seviyelerinde bulunabilmektedir. Bu soyutlama seviyeleri şöyle sıralanabilir: Makine kodu, assembly dili ve yüksek seviyeli diller (Samet ve Aslan 2018). Program kodlarının çalışabilmesi için yüksek seviyeli dillerden makine seviyesindeki dillere dönüştürülmesi gerekmektedir (Eilam 2011). Bu dönüşüm derleme olarak adlandırılmaktadır (Aslan ve Samet 2020). Diğer yandan program komutlarının analiz edilebilmesi için ise makine seviyesindeki kodların daha üst seviyeli dillere dönüştürülmesi gerekmektedir. Makine kodlarının assembly diline çevrilmesi ayırma (disassembly) olarak adlandırılırken, daha düşük seviyeli kodların yüksek seviyeli dillere dönüştürülmesi ters yönde derleme (decompilation) olarak adlandırılmaktadır. Tersine derleme sonucunda elde

edilen kodlar yorumlanarak gelişmiş statik analiz yapılmaktadır. Tersine derleme sırasında bilgi kayıpları olduğundan (C, C#, Java gibi üst seviye dillerin kaynak kodlarını elde etmek çok zordur) genelde assemly seviyesindeki diller gelişmiş statik analiz için kullanılmaktadır (Sikorski ve Honig 2012, Aslan ve Samet 2017a). Bu amaçla IDA Pro paket ayırıcı ve eklentisi olan Hex-Rays decompiler ileri düzey statik analiz için yaygın olarak kullanılmaktadır.

3.1.3 Temel (basit) dinamik analiz

ZY davranışlarının izleme programları kullanarak incelenmesidir. Process Eplorer, Process Monitor, API Monitor, Regshot, ApateDNS, Wireshark ve Sandbox'lar temel dinamik analizde yaygın olarak kullanılan araçlardır.

3.1.4 İleri düzey (gelişmiş) dinamik analiz

İleri düzey dinamik analizde hata ayıklama (debugger) araçları OllyDbg, WinDbg, vb. kullanılır. Hata ayıklama araçları değişkenlerin, paremetlerin ve hafıza alanlarının hem içeriklerini görüntüleme hem de değiştirmek için her komutu tek tek çalıştırmaya imkan vermektedir (Samet ve Aslan 2018). Bu araçlar hem kullanıcı hem de kernel seviyesinde çalışmaktadır. Gelişmiş dinamik analiz kullanılarak ZY'lerin büyük bir kısmı tespit edilebilmektedir fakat debugger kullanımı uzmanlık gerektirdiği için bu araçların kullanımı zordur.

Statik analiz kullanarak programlar hakkında genel bir bakış elde etmek ve daha önce benzerlerine sıkça rastlanan ZY'leri analiz etmek kolay ve hızlıdır, fakat gizlenme tekniği (obfuscation, packed, vb.) kullanan ZY'leri analiz etmek neredeyse imkansızdır (Çizelge 3.1). Dinamik analiz kullanılarak gizlenme tekniği kullanan ZY'ler tespit edilebilmektedir fakat sandbox ve sanal makine ortamlarında bazı ZY'ler bütün gerçek davranışlarını göstermemektedir (Egele vd. 2012). Daha önce bilinen ZY'ler için statik analiz daha hızlı ve yüksek performans göstermesine karşın daha önce bilinmeyen ve yeni nesil ZY'ler için dinamik analiz daha iyi bir performans göstermektedir.

Çizelge 3.1 Statik-dinamik analiz avantaj ve dezavantajları

ZY analizi	Avantajlar	Dezavantajlar
Statik Analiz	Genel bir bakış açısı sunar	Tersine mühendislik bazı kısıtlamar içerir
	Zaman ve kaynak tüketimi azdır	Obfuscation ve packed tekniklerine karşı savunmasızdır
	Kararlı ve tekrarlanabilir	Zor ve karmaşıktır
	Gerçek makine zarar görmez	Karmaşık ZY'leri analizde etkisizdir
Dinamik Analiz	Basit ve kesin sonuçlar üretir	ZY'nin sınırlı görünümünü (single path execution) sunar
	Gerçek zamanlı davranış bilgileri elde edilir	Çalıştırma ortamına göre farklı davranışlar gösterebilirler
	Kod obfuscation tekniklerine karşı güçlüdür	Analiz otomatikleştirildiğinde etkileşimli davranışlar eksik kalır
	Yeni nesil ZY'leri tespit edebilir	Zaman ve kaynak tüketimi fazladır
		Sanal ortamlarda ZY'ler bütün davranışlarını göstermezler

3.2 Zararlı Yazılım Özellik Çıkarma Yöntemleri

Son yıllarda veri madenciliği farklı alanlarda sıkça kullanılmaktadır. Veri madenciliği örtük, önceden bilinmeyen yararlı bilgilerin büyük veri kümelerinden veya veri tabanlarından çıkartılarak anlamlandırılması işlemidir. Bu aşamada veri içindeki belirli desenler ve daha önceden bilinmeyen değerler çıkartılarak anlamlandırılır. Son yıllarda veri madenciliği kullanarak yeni modeller oluşturulmakta ve bu modellere göre veri setleri oluşturulmaktadır. ZY özellikleri belirlenirken *n*-gram, torba (*m*-bags) ve değişkenler grubu (*k*-tuples) gibi veri madenciliği teknikleri ve diyagram modeli yaygın olarak kullanılmaktadır.

3.2.1 *n*-gram modeli

n-gram, birçok alanda ve ZY tespitinde yaygın olarak kullanılan bir özellik çıkarma tekniğidir. *n*-gram, özellik oluşturmak için hem statik hem de dinamik nitelikleri kullanabilmektedir. *n*-gram kullanılarak davranışlardan özellikler çıkarılırken sistem çağrıları veya API'ler (uygulama programlama arabirimleri) belirlenen *n* değerlerine göre (*n=2, n=3, n=4, n=6, vb.*) sıralı olarak art arda gelecek şekilde gruplanarak özellikler çıkarılmaktadır. Örneğin Program *P := <a, b, c, d, e>* sistem çağrısından oluşsun. *2*-gram ve *3*-gram sırasıyla: *2*-gram := {*<a, b>, <b, c>, < c, d>, <d, e>*} ve *3*-gram := {*<a, b, c>, <b, c, d>, < c, d, e>*} olacaktır. Tubles ve bags modelleri *n*-gram benzeri olup tubles yöntemi belirlenen *n* sayısına göre sıra önemli olmak şartıyla her uzaklıkta olabilirken bags yönteminde sıra değil frekanslara bakılır. *n*-gram etkin bir özellik oluşturma modeli olmasına karşın, özellik sayısının hızla artması ve birbiriyle ilişkili özellikleri belirlemedeki zorluklar bu modelin dezavantajları arasında gösterilebilir.

3.2.2 Diyagram (Graph) modeli

Diyagram modelinde yapılan sistem çağrıları diyagrama dönüştürülerek ifade edilmektedir. Diyagram G (V, E), öyle ki V düğümleri (sistem çağrılarını) ve E kenarları (sistem çağrıları arasındaki ilişkiyi) ifade etmektedir. Diyagramın büyüklüğü zamanla artacağı için diyagramı yaklaşık olarak ifade edebilecek alt diyagramlar bulunur. Alt diyagramın belirlenmesi birçok çalışmada NP-complete olarak ifade edilmektedir. Tüm diyagram daha az düğüm ve kenarla ifade edildikten sonra ML algoritmaları kullanılarak yazılımlar normal ya da zararlı olarak belirlenmektedir.

3.2.3 Mevcut veri setleri

Diğer araştırma alanlarında olduğu gibi ZY analizi için daha önce oluşturulmuş ve herkes tarafından kabul görmüş, kolayca erişilebilen ve yaygın olarak kullanılan veri setleri yok denecek kadar azdır. Ayrıca, var olan veri setlerinin çoğu araştırmalar için erişilebilir değillerdir, erişilen veri setleri de çoğu durumda veri madenciliği ve ML algoritmaları için uygun biçim ve formatta değillerdir. Kullanılabilir durumda olan veri setlerinin çoğu aşağıda kısaca açıklanmıştır.

NSL-KDD veri seti (2009): NSL-KDD veri seti KDD'99 veri setinin güncellenmiş hali olup yaklaşık 125.000 kayıt ve 41 özellikten oluşmaktadır (Tavallaee vd. 2009). Genellikle saldırı tespit sistemlerinin performansını ölçmek için kullanılan bu veri seti ağda gerçekleşmesi muhtemel olan saldırıları içermektedir.

CSDMC2010 API dizilim corpus veri seti (2010): Bu veri seti Windows API/sistem çağrıları kullanılarak oluşturulmuş olup 834 kayıttan oluşmaktadır (Anonymous 2010).

Drebin veri seti (2014): Bu veri seti akıllı telefonlardaki mevcut antivirüs tarayıcıların performanslarını ölçmek için oluşturulmuş olup (Arp vd. 2014), 20 aileden 5560 ZY ve 123.453 normal yazılım içermektedir.

APIMDS veri seti (2015): Dinamik API analizi kullanmakta olup 23.146 kayıttan oluşmaktadır (Ki vd. 2015).

Microsoft ZY sınıflandırma veri seti (2015): Microsoft 2015 yılında 20.000 ZY'den oluşan "Microsoft malware classification challenge" adlı veri setini yayınlamıştır (Ronen vd. 2018).

ZY'ler IDA paket ayırıcı kullanılarak analiz edilmiş ve elde edilen çıktılar ML öncesi veri madenciliği kullanılarak işlenmelidir.

ClaMP (Classification of Malware with PE headers) veri seti (2016): ClaMP veri seti 5184 kayıttan oluşmakta ve 55 özellik bulundurmaktadır (Anonymous 2016c). API dizilimleri kullanılarak oluşturulan bu veri seti zararlı ve normal yazılım örneklerini ve bunlardan elde edilen özellikleri içermektedir.

Herkes için ML veri seti (2016): Sandbox ve statik analiz araçları kullanılarak oluşturulmuştur (Ramilli 2016). Veri seti 292 APT1, 2024 Crypto, 434 Locker ve 2014 Zeus örnekleri içermektedir.

AAGM veri seti (2017): Veri seti ağ tabanlı olup ANDROID işletim sistemini kullanan akıllı telefonlardaki ZY'leri tespit etmek için oluşturulmuştur (Lashkari vd. 2017). Veri seti 400 ZY ve 1500 normal yazılımdan oluşmaktadır.

EMBER veri seti (2018): EMBER veri seti 1 milyon kayıttan oluşmakta olup zararlı ve normal yazılım özelliklerini içermektedir (Anderson ve Roth 2018).

3.3 Zararlı Yazılımlar ve Sınıflandırma

Makine öğrenmesi (Machine Learning-ML) yazılım uygulamalarının açık bir şekilde programlanmadan çıktıların doğru bir şekilde tahmin edilmesi işlemleridir. Makine öğreniminde amaç istatistiksel analizler kullanarak oluşturulan algoritmanın girdi verilerini kabul edilebilir aralıklarda çıktı değerlerine dönüştürmektir. ML kullanılarak veriler üzerinde sınıflandırma (classification), regresyon (regression) ve kümele (clustering) yapılabilmektedir. Sınıflandırmada veri öğeleri önceden tanımlanmış iki veya daha fazla kategoriye ayrılmaktadır, regresyonda değerleri bilinen öğeler yardımıyla değeri bilinmeyen öğenin ortalama değeri tahmin edilmeye çalışılmaktadır, kümelemede ise veriler için sonlu bir kategori kümesi belirlenmektedir.

Farklı problemlerin çözümünde uzun yıllardır kullanılan ML algoritmaları son yıllarda ZY sınıflandırmasında da kullanılmaya başlanmıştır. Daha önce belirlenen özellikler ML algoritmaları kullanılarak sınıflandırılmaktadır. BN, C4.5/J48, KNN, LMT, MLP, NB, RF, SLR, SVM ve SMO yaygın olarak kullanılan sınıflandırma algoritmalarıdır (Shabtai vd. 2009, Santos vd. 2013). Genel olarak bir algoritmanın diğerinden daha iyi olduğu söylenememekle birlikte her algoritmanın kendine göre avantajları ve dezavantajları

bulunmaktadır (Çizelge 3.2). Verinin dağılışına, özellik sayısına, özellikler arasındaki bağımlılıklara göre bir algoritma diğer algoritmalara göre daha iyi performans gösterebilmektedir.

Çizelge 3.2 Makine öğrenmesi algoritmalarının karşılaştırılması

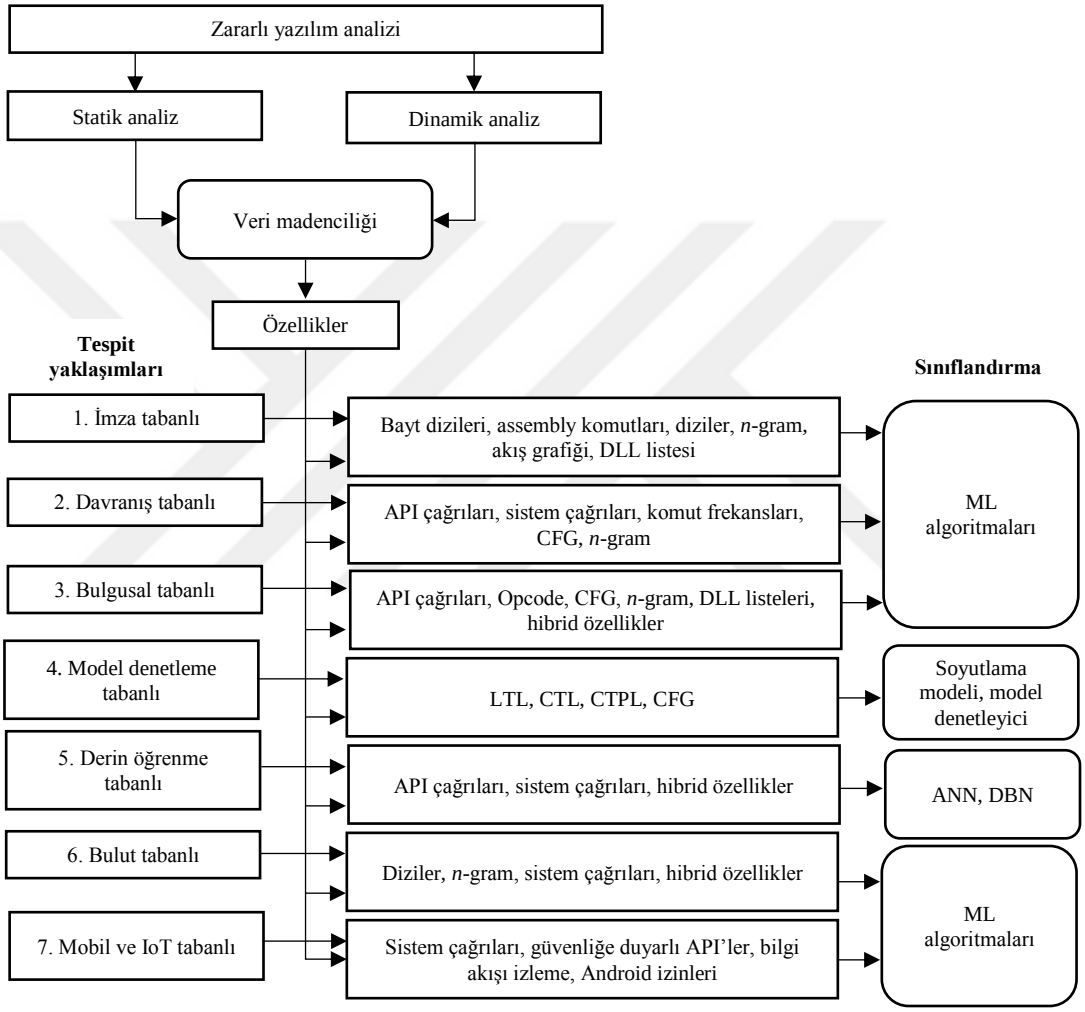
Sınıflandırıcı	Avantaj	Dezavantaj
NB	Uygulaması kolay, fazla hesaplama zamanı gerektirmeyen yüksek boyutlu (high dimensions) veriler de iyi sonuçlar üreten bir sınıflandırıcıdır.	Birbirleriyle çok ilişkili olmayan varsayımlar üzerine hesaplama yaptığı için iyi sonuçlar döndürmeyebilmektedir.
BN	İstatistiksel model bir sınıflandırıcıdır ve genel olarak hızlı ve etkili sonuç döndürmektedir.	Birçok özelliği olan veri kümeleri için uygulaması pratik değildir.
MLP	Çok katmanlı algılayıcı kullanan bu sınıflandırıcıda öğrenme süreci parelel olmakla birlikte genellikle doğru tahmin üretmektedir.	Öğrenme sırasında uzun hesaplama zamanı gerektirmektedir.
KNN	İstatistiksel model bir sınıflandırıcı olan KNN örnek tabanlı bir öğrenme kullanmaktadır. Veri dağılımı hakkında ön bilgi olmadığı durumlarda iyi sonuç üreten bir sınıflandırıcıdır. Genellikle küçük k değerleri daha iyi sonuç döndürmektedir.	Sınıflandırma için çok depolama alanı ve zaman gerektirmektedir.
SVM SMO	Kullanılan çekirdeğe (kernel) bağlı olarak hem doğrusal ayrım hem de doğrusal olmayan sınır durumlarında iyi çalışırlar. Ayrıca, yüksek boyutlu (high dimensions) verileri iyi bir şekilde işlemektedirler.	Kullanılan çekirdeğe bağlı olarak aşırı eğitim (overfitting) sorunlarına karşı hassastır.
SLR	Tahmine dayalı çalışan, uygulaması kolay ve öğrenme süreci fazla zaman gerektirmeyen bir sınıflandırıcıdır.	Doğrusal olmayan problemleri çözmede yetersiz kalmaktadır ve yüksek bias içermektedir.
RF	Mantık tabanlı çalışan bu sınıflandırıcı yüksek doğruluk oranına sahip sonuç üretmekte, verilerdeki aykırılık ve anormallikleri tespit edebilmekte ve sınıflandırmadaki önemli özniteliklerin bir tahminini verebilmektedir. Ayrıca, varyansı düşük birbirleriyle ilişkili özelliklerin çok olduğu veri setlerinde iyi sonuç döndürmektedir.	Analizi zor olan çıktılar üretmektedir.
LMT	Lojistik regresyon (LR) ve karar ağacı öğrenimini birleştiren denetimli öğrenme algoritmasına sahip bir sınıflandırıcıdır ve yüksek doğruluk oranına sahip sonuç üretmektedir.	Analizi zor olan çıktılar üretmektedir.
C4.5/J48	Tahmine dayalı olarak çalışan J48 hızlı ve ölçeklendirilebilir bir sınıflandırıcıdır.	Sürekli sınıf özelliklerinin tahmininde iyi sonuçlar döndürmez ve yanlış pozitif sonuç üretme ihtimali yüksektir.

Ayrıca, sınıflandırma yapılırken öncesinde ya da sınıflandırma sırasında özellik seçimi yapılarak performans artırılmaya çalışılır. Özellik seçimi yapılırken eğitim verisinin genel karakteristik özelliklerini ifade eden metrikler ve metotlar kullanılır (Tang vd. 2014). Doğruluk, mesafe, bilgi, bağımlılık/korelasyon ve tutarlılık bunlara örnek gösterilebilir.

3.4 Zararlı Yazılım Tespit Yaklaşımları

Son yıllarda ZY tespitiyle ilgili yapılan akademik çalışmaların sayısında hızlı bir artış görülmüştür. İlk zamanlarda, imza tabanlı tespit yaklaşımı yaygın olarak kullanılmıştır. Fakat son yıllarda ZY'lerin karmaşıklığının artmasından dolayı yeni nesil ZY'leri yakalamada

yetersiz kalmıştır (Aslan vd. 2020). Bundan dolayı zamanla araştırmacılar davranış, bulgusal/sezgisel ve model denetleme/kontrol tabanlı tespit yaklaşımları geliştirmişlerdir. Ayrıca, özellikle son bir kaç yılda DL, bulut, mobil cihazlar ve IoT tabanlı tespit yaklaşımları da kullanılmaya başlanmıştır. ZY tespit yaklaşımları, kullanılan özellikler ve tekniklerin özeti şekil 3.1'de görülmektedir.



Şekil 3.1 Zararlı yazılım tespit yaklaşımları ve özelliklerinin akış şeması

Şekil 3.1'de ANN (Artificial Neural Network), CFG (Control Flow Graph), CTL (Computation Tree Logic), CTPL (Computation Tree Predicate Logic), DBN (Deep Belief Network), LTL (Linear Temporal Logic), ML (Machine Learning) ve Opcode (Operational Code) sırasıyla yapay sinir ağı, kontrol akış şeması, hesaplama ağacı mantığı, hesaplama ağacı yüklem mantığı, derin inanç ağı, doğrusal zamansal mantık, makine öğrenmesi ve işlem komutunu göstermektedir.

Her yaklaşımda, özellik çıkarma yöntemi birbirinden farklıdır. Her bir yaklaşımın kendine özgü avantajları ve dezavantajları olduğu için bir tespit yaklaşımının kesinlikle diğerinden daha iyi çalıştığı söylenemez. Yaklaşımın kullanış biçimi, özellik çıkarma ve sınıflandırma yöntemi başarısını etkilemektedir. İmza tabanlı yaklaşım kullanılarak bilinen ZY'ler yüksek doğruluk oranıyla tespit edilebilmektedir, fakat yeni ZY'leri tespitte aynı başarıyı gösterememektedir. Davranış-, bulgusal- ve model denetleme-tabanlı tespit yaklaşımları kullanarak çok sayıda bilinmeyen ZY tespit edilebilmektedir. Fakat bazı yeni nesil ZY'leri tespitte yetersiz kalmaktadırlar. Benzer şekilde derin öğrenme-, bulut- ve mobil-IoT tabanlı yaklaşımlarda kendini sürekli değiştiren ZY'leri tespitte yetersiz kalmaktadırlar. Daha karmaşık ve bilinmeyen ZY'leri tespit edebilmek için farklı yaklaşım ve yöntemlere ihtiyaç vardır. Her tespit yaklaşımını ayrıntılı olarak açıklamadan önce, her tespit yaklaşımında iyi bilinen bazı yöntemler ve bunların literatür taramaları çizelge 3.3'te özetlenmiştir.

Çizelge 3.3 Zararlı yazılım tespitinde kullanılan yaklaşımlar ve bu yaklaşımlarda kullanılan yöntemlerin özeti

Çalışma	Özellik çıkarma yöntemi/ Özellik gösterimi	Amaç/ Performans	Yıl
	İmza tabanlı tespit		
Griffin vd.	Dizi imzaları, bulgusal tarama	ZY'lerin farklı formlarını tespit	2009
Tang vd.	Exploit tabanlı imzalar	Çok çeşitli solucanları tespit	2009
Borojerdi ve Abadi	Dizi kümeleme, benzer örüntüler	Çok biçimli ZY'leri %90.83 ile tespit	2013
Liu ve Sandhu	Kriptografik hash tabanlı imzalar	Donanım içine yerleştirilmiş ZY'leri tespit	2015
	Davranış tabanlı tespit		
Wagener vd.	Sistem çağrıları, hellinger mesafesi, filogenetik ağaç	Yeni ZY'leri ve farklı formlarını tespit	2008
Park vd.	Çekirdek nesneleri, yaygın davranışlar	ZY'lerin farklı formlarını tespit	2013
Shan ve wang	Davranış tabanlı kümeleme	Bilinmeyen ZY'leri %71.1 ile tespit	2014
Das vd.	Sistem çağrı frekansları, *n*-gram	Yeni ZY'leri ve farklı formlarını tespit	2016
Sing vd.	Çoklu API, *n*-gram	ZY'lerin farklı formlarını tespit	2017
	Bulgusal tabanlı tespit		
Schultz vd.	Yazdırılabilir diziler, DLL ve API sistem çağrıları	Yeni ZY'leri tespit	2001
Ye vd.	API dizilimi, FP-growth algoritması	Çok biçimli ve yeni ZY'leri tespit	2007
Anderson vd.	*n*-gram, Markov zinciri ve diagram	ZY'lerin farklı formlarını %96.41 ile tespit	2011
Islam vd.	Yazdırılabilir diziler, API metot frekansları	ZY'lerin farklı formlarını %97 ile tespit	2013
	Model Denetleme tabanlı tespit		
Holzer vd.	CTPL dili, sonlu durum modeli	Benzer saldırıları kullanan ZY'leri tespit	2007
Kinder vd.	Açıklamalı kontrol akışı, hesaplama ağacı doğrulama mantığı	ZY'lerin farklı formlarını tespit	2008
Song ve Touili	Pushdown sistemler, yığınsal davranış	Yeni ZY'leri tespit	2014
Battista vd.	Java bytecode, payload	DroidKungFu ve Opfake ZY ailelerini tespit	2016
	Derin öğrenme tabanlı tespit		
Saxe ve Berlin	Bağlamsal bayt özellikleri, PE özellikleri, skor kalibrasyon modeli	ZY'leri %95 ile tespit	2015
Huang ve Stokes	API çağrısı, çok görevli öğrenme, ReLU	ZY'leri tespit	2016
Ye vd.	API çağrısı, PE özellikleri, Boltzmann makinesi	Yeni ZY'leri tespit	2018
	Bulut tabanlı tespit		
Martignoni vd.	Sistem çağrıları, dinamik özellikler	ZY'nin çoklu ortamda çalıştırılması	2009
Xiao vd.	Dyna mimarisi, Q-öğrenme stratejisi	Yüksek tespit oranı	2017
Yadav	Kümeleme ve sınıflandırma modülü kullanımı	Yüksek tespit oranı	2019
	Mobil ve IoT tabanlı tespit		
Saracino	Çekirdek, uygulama, kullanıcı ve paket seviyesi özellikler	ZY'leri %96 ile tespit	2018
Li vd.	Android izinleri, 3-seviyeli madencilik	ZY'leri %93.62 ile tespit	2018
Azmoodeh vd.	IoT enerji tüketim desenleri	ZY'leri %95.65 ile tespit	2018
Alazab vd.	API çağrısı, izin istekleri	ZY'leri %94.3 f-ölçümüyle ile tespit	2020

3.4.1 İmza tabanlı tespit (Signature-based detection) yaklaşımı

Program imzaları, programdaki belirli kod bölümlerinden program yapısı çıkartılarak oluşturulan ve belirli uzunlukta olan karakter dizilimlerdir (Bazrafshan vd. 2013). Bu özellik her program için benzersiz olduğu için ZY tespitinde sıkça kullanılmaktadır (Zolkipli ve

Jantan 2010). Bu yaklaşım diğer yaklaşımlara göre hem hızlı hem de düşük hata oranına sahiptir. Fakat daha önce bilinmeyen ve imzası bulunmayan ZY'leri tespit etmede yetersiz kalmaktadır (Aslan ve Samet 2020). Ayrıca, aynı aileye ait ZY'ler az bir değişiklikle imza tabanlı tespit yaklaşımından kolayca kaçabilmektedir.

3.4.1.1 İmza belirleme süreci ve kullanılan teknikler

İmza oluşturma sırasında, ilk olarak statik özellikler yürütülebilir dosyalardan çıkartılır. Daha sonra, imza oluşturma algoritması özellikleri kullanarak imzayı oluşturur ve veri tabanında saklar. Örnek yazılımın zararlı veya normal olarak işaretlenmesi gerektiğinde, ilgili örneğin imzası daha önce olduğu gibi çıkarılır ve veri tabanındaki imzalarla karşılaştırılır. Karşılaştırma sonucunda örnek yazılım zararlı ya da normal olarak işaretlenir. İmza oluşturmak için dizi taraması, üst ve kuyruk taraması, giriş noktası taraması ve bütünlük denetimi gibi farklı teknikler kullanılır.

Dizi taraması (string scanning): Analiz edilen dosyadaki bayt (byte) dizilimiyle daha önce veri tabanına kaydedilen bayt dizilimleri karşılaştırılır. Bayt imzaları uzun yıllardır antivirüs tarayıcıları tarafından yaygın olarak kullanılmaktadır. Şekil 3.2 ClamAV bayt imzasını (Anonymous 2011) göstermektedir. Bayt imzaları aynı aileye ait farklı imzalara sahip ZY'leri tespit etmede sıkça kullanılmaktadır (Hahn 2014).

```
90FF1683EE0483EB0175F6
```

Şekil 3.2 ClamAV bayt imzası

"90FF1683EE0483EB0175F6" ilgili kod bölümünün onaltılık formatta gösterimidir ve assembly dilinde şekil 3.3'teki gibi gösterilmektedir. Aynı bayt imzası Yara formatında şekil 3.4'teki gibi gösterilmektedir (Anonymous 2011).

```
Başlama: 0x401A2E length: 0xC
90 nop
FF 16 call dword ptr [esi]
83 EE 04 sub esi, 4
83 EB 01 sub ebx, 1
75 F6 jnz short loc_401A30
```

Şekil 3.3 "90FF1683EE0483EB0175F6" imzasının assembly bayt dizilimi

```
Kural
{
        Bayt dizilim:
        imza = { 66 90 FF 16 83 EE 04 83 EB 01 75 F6 }
        durum:
        imza
}
```

Şekil 3.4 Bayt imzalarının Yara formatında gösterimi

Üst ve kuyruk taraması (top-and-tail scanning): İmzalar oluşturulurken dosyanın bütünü yerine sadece üst ve bitiş noktalarındaki ilgili bölümler kullanılmaktadır. Kendini dosya başlarına ve sonlarına ekleyen ZY'leri tespit etmek için kullanılan oldukça uygun bir imza belirleme tekniğidir.

Giriş noktası taraması (entry point scanning): Bir dosyanın ya da programın giriş noktası o dosya veya program çalıştırıldığında çalışmanın ilk olarak nereden başlayacağını gösterir. Genelde ZY'ler programların giriş noktalarını ZY'lerin başlangıç noktalarını işaret edecek şekilde değiştirerek program kodlarından önce zararlı kodların çalıştırılmasına neden olurlar (Szor 2005). Böyle durumlarda tüm dosya yerine sadece program giriş noktası taranarak ZY'ler tespit edilebilmektedir.

Bütünlük denetimi (integrity checking): Bütünlük denetimi, bir sistemdeki her dosya için ve düzenli aralıklarla bir kriptografik sağlama toplamı MD5 ve SHA-256 gibi oluşturulur ve ZY'lerin neden olabileceği olası değişiklikleri saptamak amacıyla kullanılır.

İmza oluşturma süreci ve farklı imza oluşturma teknikleri özetlenmiştir. Bu teknikler kullanılarak hızlı bir şekilde imza oluşturulabilmesine rağmen ZY gizleme tekniklerine karşı dayanıklı değillerdir. Örneğin, ZY komut dizisini ve program giriş noktasını kolayca değiştirebilir. Bu durumda oluşturulan imzalar geçersiz olacaktır. Daha dayanaklı ve genel imzalar çıkarmak için farklı tekniklerin bir arada kullanılması faydalı olabilir. Aşağıda imza tabanlı ZY tespit yaklaşımı ve bu yaklaşımda kullanılan yöntemler özetlenmiştir.

3.4.1.2 Literatürde yapılan imza tabanlı çalışmaların özeti

Otomatik imza çıkarma yöntemi Griffin vd. (2009) tarafından önerilmiştir. Makaleye göre önerilen yöntem otomatik olarak yüksek kaliteli dizi imzaları çıkarmaktadır. Önerilen yöntem, bir dizi kütüphane tanımlama tekniği ve çeşitliliğe dayalı sezgisel tarama tekniği

kullanarak ZY'ler için benzer olan imzalar belirlemektedir. Bu imzalar ZY'lerde çok görülürken normal yazılımlarda az görülmektedir. Bundan dolayı yanlış işaretleme oranı düşmektedir. Ancak, normal yazılım kümesi çok çeşitlidir ve sürekli büyümektedir bundan dolayı bu yöntem bütün normal yazılımlar için genelleştirilemez.

Tang vd. (2009) polimorfik solucanları tespit etmek için bioinformatik tekniği kullanan basitleştirilmiş düzenli ifade imzası önermişlerdir. Önerilen teknik istismar tabanlı (exploit-based) imzalar üretmekte olup üç adımdan oluşmaktadır: Dizi hizalama tekniği kullanılarak en belirgin alt dizileri belirlemek, gürültülü alt dizileri elimine etmek ve basitleştirilmiş düzenli ifade imzasını mevcut IDS'lerle uyumlu hale getirmek. Çalışmaya göre önerilen metot gürültüye toleranslı ve diğer bazı istismar tabanlı imza oluşturma metotlarına göre daha doğru ve kesin sonuçlar üretmektedir. Bunun nedeni polimorfik özelliği yansıtan değişmez baytları ve bu baytlar arasındaki mesafe kısıtlamalarını belirleyebilmesidir. Önerilen metodun polimorfik solucanlarıyla sınırlı kalması ve diğer ZY türleri için çalışmaması metodun dezavantajı arasında gösterilebilir.

Borojerdi ve Abadi (2013) dizi kümeleme ve hizalamaya dayalı çok biçimli (polymorphic) ZY'leri tespit eden ve dinamik özellikleri kullanan imza yöntemini önermişlerdir. Öncelikle sistem çağrıları aldığı parametreye (sys, self, vb.) göre gruplandırılmış, daha sonra gruplanan sistem çağrıları kullanılarak dizilimler oluşturulmuştur. Oluşturulan dizilimler kullanılarak kümeler (clusters) elde edilmiştir. Bu aşamada benzer dizilimler aynı grupta yer alacak şekilde gruplandırılmış ve gruplara göre örüntüler (pattern) elde edilmiştir. Oluşturulan örüntüler deterministik olmayan sonlu otomatlara dönüştürülmüş ve bu otomatlar kullanılarak imzalar oluşturulmuştur. İşaretlenmek istenen programın imzaları daha önce belirlenen imzalarla karşılaştırılarak ZY olup olmadığına karar verilmiştir. Önerilen yöntemin DR'si %90.83 olurken FPR'si %0.80 olarak ölçülmüştür.

Liu ve Sandhu (2015) parmak izine dayalı (fingerprint-based method) bir metot kullanarak donanımdaki ZY'leri tespit eden bir yöntem önermişlerdir. Kurcalanmaya açık mimariye (tamper-evident architecture) göre çalıştırılan program sinyal olarak farklı olan kriptografik hash tabanlı imzalar üretmekte ve bu imzalar kullanılarak donanım içine gömülü Truva atları tespit edilmektedir.

3.4.1.3 İmza tabanlı tespit yaklaşımı ve yöntemlerinin değerlendirilmesi

Literatür çalışmasında imza tabanlı tespit yaklaşımı ve bu yaklaşımda kullanılan yöntemler özetlenmiştir. İmza tabanlı tespit yaklaşımı antivirüs tarayıcıları tarafından uzun yıllardır kullanılmakta ve bilinen ZY'leri tespit etmede oldukça hızlı ve etkilidir. Bu yaklaşım genellikle iyi bilinen ya da aynı aileye ait olan ZY'leri tespit etmek için kullanılmaktadır. Ancak, gizleme teknikleri kullanan yeni nesil ZY'leri ve daha önce bilinmeyen ZY'leri tespit etmede başarı oranı düşüktür. Ayrıca, imza çıkarmak çok fazla iş gücü gerektirmekte olup uzun zaman almaktadır. Verilen kaynak özetlerinden de anlaşılacağı gibi önerilen yöntemler genellikle belli türdeki ZY'leri tespit etmek için önerilmiş olup bütün ZY'leri kapsayacak şekilde genişletilememiştir. Bundan dolayı ZY imzaları belirlenirken bazı noktalara dikkat etmek gerekmektedir:

- Etkili otomatik imza oluşturma mekanizması geliştirilmelidir;
- İmza oluşturma sırasında veri madenciliği ve ML teknikleri daha fazla kullanılmalıdır;
- İmza mümkün olduğunca kısa olmalı ve tek imzayla birçok ZY tespit edilebilmelidir;
- Belirlenen imza, paketleme ve gizleme tekniklerine karşı dayanıklı olmalıdır.

3.4.2 Davranış tabanlı tespit (Behavior-based detection) yaklaşımı

Davranış tabanlı ZY tespit yaklaşımında program davranışları izlenerek ilgili programın zararlı ya da normal olduğu belirlenir. Program koduna ve kod dizilişine değil, programın davranışına bakılır. Program kodları değişse bile programın göstereceği davranışlar çok değişmeyeceğinden dolayı bu yaklaşım kullanılarak birçok ZY tespit edilebilmektedir. Ayrıca, daha önce bilinmeyen yeni ZY'ler de bu yöntemle tespit edilebilmektedir. Davranış tabanlı tespitin en büyük dezavantajı, ZY'nin sanal makine ve sandbox ortamlarında çalıştırıldığında bütün davranışlarını göstermeme ihtimalinin olmasından dolayı normal olarak belirlenmesidir (Alosefer 2012). Genel olarak bu yöntem üç kısımdan oluşmaktadır: Davranışların çıkarılması, ilgili davranışların belirlenmesiyle özelliklerin oluşturulması ve ML algoritmaları kullanarak programın zararlı ya da normal yazılım olarak işaretlenmesi.

3.4.2.1 Davranış belirleme süreci

Program davranışları belirlenirken sistem çağrıları (system calls) ya da dosya, registry ve ağ üzerinde meydana gelen değişiklikler kullanılmaktadır. Yani, sistem çağrılarının veya yapılan dosya-registry işlemlerinin hangi sırada ve ne sıklıkla yapıldıklarına bakılarak davranışlar oluşturulmaktadır. Davranışlar gruplanarak dizilimler oluşturulmakta ve bu dizilimler kullanılarak özellikler elde edilmektedir. Davranış tabanlı tespit sistemi oluşturulurken aşağıdaki yöntemler kullanılarak davranışlar elde edilmektedir:

- Sandbox kullanarak otomatik analiz;
- Sistem çağrılarının izlenmesi;
- Dosya değişikliklerinin izlenmesi;
- Kayıt defteri anlık görüntülerinin karşılaştırılması;
- Ağ aktivitelerinin izlenmesi;
- İşlemlerin izlenmesi.

3.4.2.2 Literatürde yapılan davranış tabanlı çalışmaların özeti

Wagener vd. (2008) sistem çağrılarını program davranışları olarak kabul edip farklı programlar arasındaki davranış benzerlikleri ve farklılıklarını belirleyerek programların ZY olup olmadıklarına karar vermişlerdir. Dizi hizalama tekniği kullanılarak benzerlikler, Hellinger uzaklığı hesaplanarak davranış ilişkileri belirlenmiştir. Ayrıca, filogenetik (phylogenic) ağaçlar kullanılarak ZY'lerin ait olduğu aileler belirlenmiştir. Diyagram kullanılarak ZY tespit sistemi Kolbitsch vd. (2009) tarafından önerilmiştir. Sistem çağrıları diyagrama dönüştürülmekte öyle ki her düğüm bir sistem çağrısını kenar ise sistem çağrıları arasındaki geçişi göstermektedir. Ayrıca, bir sistem çağrısının sonucu diğer sisteme girdi olarak verilerek sistem çağrıları arasındaki bağlantılar belirlenmiştir. İşaretlenmek istenen programın diyagramı çıkartılmış ve var olan diyagramlarla karşılaştırılarak ZY olup olmadığına karar verilmiştir. Ayrıca, analiz sırasında gözlemlenen yeni davranışlar da diyagrama dinamik olarak eklenmiştir.

Trinius vd. (2009) sistem çağrılarını (120 farklı sistem çağrısı olmak üzere) ve frekanslarını belirleyerek programları grafiksel olarak ifade etmişlerdir. Bu makalede sistem çağrıları 20

farklı bölüme (dosya, sistem dosyaları, registry, thread, mutex, vb.) ayrılmış olup ilgili bölümdeki aynı sistem çağrılarının frekansları en önemli argümanlar başta olmak üzere gruplanarak yazılımlar görselleştirilmiştir. Çalışmaya göre aynı ZY ailesine ait olan yazılımlar benzer görüntüler oluşturmuştur. Fukushima vd. (2010) API çağrılarını kullanarak hiç yanlış pozitif işaretleme yapmayan ve öğrenme gerektirmeyen bir yöntem önermişlerdir. Makaleye göre WİS için önemli olan birçok veri ve dosyalar sistem, Windows ve geçici gibi özel dizinlerde tutulmaktadır. ZY'ler kendi amaçlarını gerçekleştirmek için bu dizinlerde çeşitli işlemler yapmaktadır. Bu dizinlerdeki belli davranış grupları analiz edilerek programların ZY olup olmadıkları belirlenebilmektedir. Bu yönteme göre başarı oranı %60 olmakta ve yanlış pozitif işaretleme olmamaktadır.

Fredrikson vd. (2010) sistem çağrılarını diyagrama dönüştürmüş ve en uygun özellikleri belirleyerek programın ait olduğu sınıfı zararlı ya da normal olarak belirlemişlerdir. Zararlı ve normal yazılımlar ayrı ayrı grup olarak çalıştırılmış ve diyagramlar oluşturulmuştur. Zararlı davranışlar belirlenirken ZY'ler içinde bu davranışın ne kadar fazla sıklıkla tekrarlandığına ve normal yazılımlar içinde ne kadar az tekrarlandığına bakılarak alt diyagramlar oluşturulmuştur. İstenmeyen ve tekrar eden davranış grupları budanarak diyagramdan silinmiştir. İşaretlenmek istenen program davranışları daha önce oluşturulan bu davranış gruplarıyla karşılaştırılarak ZY olup olmadığına karar verilmiştir.

Lanzi vd. (2010) sistem merkezli (system-centric) davranış modeli önermişlerdir. Çalışmaya göre ZY'lerin sistem kaynaklarıyla (dizin, dosya, registry, vb.) etkileşim biçimleri normal yazılımların etkileşiminden farklıdır. Etkileşim farklılıkları kullanılarak sistem çağrılarından davranış dizileri oluşturulmuştur. Daha sonra bu dizilimler kullanılarak ZY ve normal yazılım grupları elde edilmiştir. Önerilen yöntemde *n*-gram tekniği kullanılmıştır, fakat *n*-gram tekniği ile çok fazla davranış dizilimi oluşturulduğundan ZY'leri normal yazılımlardan ayırmak zorlaşmaktadır. Davranışa göre ZY'leri normal yazılımlardan ayırt eden yöntem Tian vd. (2010) tarafından önerilmiştir. Bu yöntemde sistem çağrıları parametreleriyle birlikte her program için ayrı ayrı belirlenmiştir. Daha sonra özellik vektörü oluşturulmuştur öyle ki analiz edilen her bir program için belirtilen API sistem çağrı grubu varsa 1 yoksa 0 olarak belirlenmiştir. Test sonuçlarına göre sistem çağrılarının çağrılma sıklığı değil çağrılıp çağrılmamaları etkili olmuştur. En son ML algoritmaları kullanılarak programların ZY olup olmadığına karar verilmiştir.

Ye vd. (2010) ilişkilendirilmiş sınıflandırma tekniği kullanarak ZY'leri normal yazılımlardan ayıran bir yöntem önermişlerdir. Önerilen yöntemde her program için API sistem çağrılarının dizilimleri ve sayıları belirlenerek özellikler çıkartılmıştır. Daha sonra API dizilimleri gruplanarak zararlı ve normal davranış grupları elde edilmiştir. Makaleye göre aşağıdaki fonksiyonellikleri içeren bir program ZY olarak işaretlenebilir:

- Nesneye erişmek için kullanılabilecek bir tanıtıcı döndürme;
- Var olan bir dosyayı dosya sisteminden silme;
- Çağıran proses adres alanı içinde çalışan bir iş parçacığı oluşturma;
- Bir uygulamanın WinInet metotlarını kullanılarak başlatılması.

Kwon ve Su (2011) yazılım benzerliklerini belirlemek için API davranış modelini önermişlerdir. Bu yöntemde API sistem çağrıları yerine API kullanım kalıpları kullanılmıştır. Kullanım kalıpları oluşturulurken davranış sırası ve tekrarları kullanılarak davranışlar düzenli ifadelere dönüştürülerek davranışsal imzalar elde edilmiştir. Daha sonra dizi kümeleme algoritması (sequence clustering algorithm) CLUSEQ (Yang vd. 2003) kullanılarak benzerlik gösteren programlar aynı kümede alacak şekilde gruplandırılmıştır. Bu şekilde bir imzayla benzer özellik gösteren birçok ZY tespit edilmiştir.

MapReduce kullanılarak sistem çağrılarını gruplayan ve bu gruplamaya göre ZY'leri tespit eden yöntem Liu vd. (2011) tarafından önerilmiştir. Bu çalışmada sistemde kalıcı hale gelmeye çalışan ZY'ler tespit edilmeye çalışılmıştır. Öncelikle, sistem çağrıları arasındaki bağımlılıklar belirlenmiştir. Bağımlılıkların hızlı bulunması adına eşleyici (mapper) ve indirgeyici (reducer) kullanılmıştır. Eşleyici, sistem çağrılarını toplayarak indirgeyiciye göndermiş, indirgeyici ise sistem çağrıları arasındaki bağımlılıkları bulmuştur. Çalışmaya göre üç çeşit davranış zararlı olarak kabul edilmiştir:

- Programın kendini çoğaltarak yayılması;
- Programın başka programlara kod enjekte etmesi;
- Programın sistemde kalıcı hale gelmesi.

Kasama vd. (2012) ZY örneklerini birden fazla kez çalıştırarak oluşacak davranış farklılıklarının ilgili programın ZY olup olmadığı hakkında bilgi vereceğini varsaymışlardır.

Çünkü bu araştırmacılara göre normal yazılımlar birden fazla kez çalıştırıldığında aynı davranışları gösterecektir. Önerilen yaklaşımın DR'si %67'lerde kalmasına rağmen FPR %1'lerde kalmıştır. Davranış farklılıklarına başka davranışlarında eklenmesi DR'yi artırabilir. Mansour vd. (2013) davranışa dayalı ZY tespit sistemi önermişlerdir. Önerilen sistemde API sistem çağrıları sıralı olacak şekilde belirlenip her biri bir rakama denk gelecek şekilde dönüşüm yapılmıştır. Sık tekrarlanan örüntüler Lo vd. (2007-2009) tarafından önerilen algoritma kullanılarak belirlenmiş ve buna göre özellikler çıkartılmıştır. Bu algoritmaya göre belli bir eşik değerinin üstünde destek değerine sahip olan tekrarlar özellik vektöründe yer almaktadır. Fisher score algoritması kullanılarak en uygun özellikler seçilerek sınıflandırma yapılmıştır.

Chandramohan vd. (2013) özellik sayısını sınırlı tutan BOFM (Bounded Feature Space Behavior Modeling) tespit yöntemini önermişlerdir. Bu modelde birbiriyle ilişkili sistem çağrıları anlamlı bir bütünlük oluşturacak şekilde davranışlara dönüştürülmüş ve daha sonra bu davranışlar kullanılarak özellikler belirlenmiştir. Bu aşamada az tekrarlanan özellikler elimine edilmiştir. Özellikler kullanılarak özellik vektörü oluşturulmuş ve özellik vektörüne ML algoritmaları uygulanarak sınıflandırılmıştır. Park vd. (2013) ZY'lerin gösterdikleri yaygın davranışları diyagramla belirleyen bir yöntem önermişlerdir. Bu yöntemde sistem çağrıları yardımıyla çekirdek nesneleri (kernel object) belirlenmiş ve bu nesnelere göre davranışlar oluşturulmuştur. ZY ailelerinin çok tekrarlanan davranışları kullanılarak diyagramlar ve bu diyagramları temsil eden alt diyagramlar oluşturulmuştur. Bu alt diyagramlar ilgili ZY ailesine ait bütün genel davranışları bulunduran özellikleri barındırmaktadır. İşaretlenmek istenen yazılımın alt diyagramı bu alt diyagramlarla karşılaştırılarak ZY olup olmadığına karar verilmiştir.

Shan ve Wang (2014) davranış tabanlı kümeleme (clustering) yöntemi kullanarak ZY'leri sınıflandırmışlardır. Sistem çağrıları kullanılarak davranışlar oluşturulmuş ve benzer davranış gösteren yazılımlar aynı kümede olmak şartıyla özellikler çıkartılmıştır. Muhtevası belirlenmek istenen program davranışları her kümeyle ayrı ayrı karşılaştırılarak en yakın kümeye dahil edilmiştir. Naval vd. (2015) program davranışlarını anlamlı hale getiren diyagram tabanlı bir yaklaşım önermişlerdir. Davranışlar belirlenirken sistem çağrıları anlamlı bir bütünlük gösterecek şekilde davranışlara dönüştürülmektedir. WİS'de 160'ı çok kullanılan olmak üzere toplamda 284 farklı sistem çağrısı (Anonymous 2016b) belirlenmiştir.

Belirlenen davranışlar sıralı olacak şekilde bir diyagrama dönüştürülmekte ve diyagramı oluşturan en anlamlı yollar AEP (Asimptotik eşdağılım özelliği) kullanılarak tespit edilmektedir (Shannon ve Weaver 1949, Cui 2011). Yazara göre bu anlamlı yollar ilgili programın en önemli ve gerekli davranışlarını içermekte olup ve bu yollar kullanılarak daha önce görülmeyen bazı ZY'ler tespit edilebilmektedir.

Das vd. (2016) semantik tabanlı gerçek zamanlı ZY tespit sistemi önermişlerdir. Önerilen sistem ZY'lerin yaklaşık %30'luk bir kısmını daha program çalışırken tespit etmektedir. Tespit sistemi donanım tabanlı olup yazılım tabanlı tespit sistemlerine göre bazı avantajlar sağlamaktadır. Önerilen yaklaşım önce sistem çağrılarını ayıklamakta ve zararlı davranışları modelleyerek üst düzey anlamsal özellikler oluşturmaktadır. Elde edilen anlamsal özelliklere ML uygulanarak sınıflandırma yapılmaktadır. Singh vd. (2017) davranış tabanlı çoklu API sistem çağrısı kullanarak ZY'leri tespit etmişlerdir. Bu yöntemde derinlik öncelikli arama ve *n*-gram kullanılarak çoklu API dizilimleri oluşturulmuştur. Çoklu API dizilimleri belirlenirken yazılımlar arasındaki benzerlikleri belirlemek amacıyla Dice coefficient, Cosine Coefficient ve Tversky index kullanılmıştır. Oluşturulan dizilimler ML algoritmaları kullanılarak sınıflandırılmıştır.

3.4.2.3 Davranış tabanlı tespit yaklaşımı ve yöntemlerinin değerlendirilmesi

Davranış tabanlı literatür özetinde mevcutta kullanılan ZY tespit yöntem ve metotları ana fikir, algoritma türleri ve özellik çıkarma yöntemleri göz önünde bulundurularak özetlenmiştir. Davranışlara dayalı tespit yaklaşımı 3 adımdan oluşmaktadır:

- Davranışları belirlemek (veri madenciliği kullanılabilir);
- Davranışları kullanarak özellik oluşturmak (veri madenciliği kullanılır);
- Özellikleri kullanarak sınıflandırma yapmak (makine öğrenmesi kullanılır).

Davranış tabanlı tespit yaklaşımında *n*-gram, *n*-tuple, bag ve diyagram gibi modeller kullanılarak davranışlar ve özellikler elde edilmektedir. Özellikler arasındaki benzerlik ve farklılık ilişkilerini belirlemek için olasılık ve istatistiksel yöntemler (Hellinger mesafesi, kosinüs katsayısı, ki-kare, vb.) kullanılmaktadır. Doğru davranış tanımlamanın getirdiği zorluklar, çıkarılan özellik sayısının fazla olması (*n*-gram, vb. kullanılırken) ve çıkarılan özellikler arasındaki benzerlik ve farklılıkları tespit etmede yaşanan zorluklar etkili bir tespit

sisteminin oluşturulmasını engellemiştir. Bazı ZY'ler koruma ortamlarında (sanal makine ve sandbox ortamları) gerçek davranışlarını gizlemektedirler bu durum normal yazılım olarak işaretlenmelerine sebep olmaktadır. Ayrıca, bazı ZY'ler birçok prosesten oluşmakta, yayılmak için sürücü (driver) ya da DLL uzantılı dosyaları kullanmakta ve gelişmiş kod gizleme teknikleri kullanarak tespit edilmeyi zorlaştırmaktadır. Mevcut davranış tabanlı metot ve yöntemler incelendiğinde istenilen performans değerleri elde edilememiştir. Bu nedenle bu tez çalışmasında mevcut davranış tabanlı yöntemlerin eksik yönleri giderilerek daha yüksek performansla çalışan model ve yöntemler önerilmiştir.

3.4.3 Bulgusal tabanlı tespit (Heuristic-based detection) yaklaşımı

Son yıllarda ZY tespitinde sıkça kullanılan bulgusal tabanlı tespit farklı teknikleri bir arada kullanan karmaşık bir tespit yaklaşımıdır. Deneyime dayalı olarak çalışan bu yaklaşım belirli kuralları ve ML tekniklerini kullanmaktadır (Alzarooni 2012). Genelde ZY'lerin farklı biçimlerini ve daha önce görülmemiş ZY'leri tespit etmek amacıyla kullanılmaktadır. Öncelikle sistem belirli özellikler kullanılarak eğitilmektedir. Daha sonra test amaçlı veriler kullanılarak anormallikler tespit edilmektedir. Yeni ZY'leri tespit etmede başarı oranı yüksek olmasına rağmen optimize çalışmaması neticesinde yanlış pozitif (False Pozitive-FP) ve yanlış negatif (False Negative-FN) oranları yüksek çıkmaktadır.

3.4.3.1 Literatürde yapılan bulgusal tabanlı çalışmaların özeti

Schultz vd. (2001) program davranışlarını üç tür özellik kullanarak belirlemişlerdir:

- Program tarafından kullanılan dinamik link kütüphaneleri (DLL);
- Her DLL tarafından çağrılan metotların listesi;
- Her DLL'de çağrılan farklı metotların toplam sayıları.

Örneğin, ($\neg$*advapi32* $\wedge$ *avicap32* $\wedge$ ... $\wedge$ *winmm* $\wedge$ $\neg$*wsock32*), program tarafından çağrılan ve hiç çağrılmayan DLL'lerin listesini gösterirken. (*Advapi32.AdjustTokenPrivileges*() $\wedge$ *advapi32.GetFileSecurityA*() $\wedge$ ... *wsock32.recv*() $\wedge$ *wsock32.send*()), hangi DLL'lerden hangi metotların çağrıldığı göstermektedir. Benzer şekilde her DLL'den çağrılan farklı metotların toplam sayıları aynı şekilde belirlenmiştir. Belirlenen özellikler kullanılarak kurallar oluşturulmuş ve bu kurallara göre sınıflandırma yapılmıştır. Örneğin,

- zararlı := ¬user32.EndDialog() ∧ kernel32.EnumCalendarInfoA();
- zararlı := ¬user32.LoadIconA() ∧ ¬kernel32. GetTemp() ∧¬advapi32;
- zararlı := shell32.ExtractAssocciatedIconA();
- zararlı := msvbvm;
- normal := aksi halde.

Ye vd. (2007) akıllı ZY tespit sistemi önermişlerdir. Sistemin amacı antivirüs tarayıcıları tarafından algılanamayan çok biçimli ve daha önce görülmemiş ZY türlerini tespit etmektir. Sistem, verilen programın API çalışma dizilimini alıp veri madenciliği algoritmalarından biri olan "FP-Growth" algoritmasını kullanarak uygun kurallar çıkarmakta ve daha sonra sınıflandırma algoritmalarını kullanarak ilgili programın zararlı ya da normal olduğuna karar vermektedir. Önerilen sistem çalıştırılabilir Windows dosya formatı üzerinde çalışmaktadır. Makalede belirtilen sonuçlara göre önerilen yöntem bazı antivirüs tarayıcılarına göre daha iyi bir performans göstermiş olmasına rağmen ileri düzey ZY'leri yakalamada istenilen performansı gösterememiştir.

Dinamik analiz kullanılarak diyagram tabanlı tespit yöntemi Anderson vd. (2011) tarafından önerilmiştir. Önerilen sistemde diyagram Markov zinciri şeklinde temsil edilmiştir. Diyagramda düğümler komutları kenarlar ise bir düğümden diğer düğüme geçiş olasılığını göstermektedir. Toplamda 160 farklı komut gözlemlenmiştir. Diyagram çekirdekleri (graph kernels) kullanılarak komutlar arasındaki benzerlikler benzerlik matrisine (yerel ve global anlamdaki benzerlikler) dönüştürülmüştür. Benzerlik matrisine SVM uygulanarak sınıflandırma yapılmıştır. Santos vd. (2011) bilinmeyen ZY'leri tespit etmek için yarı denetimli öğrenme tekniği olan toplu öğrenmeyi kullanmışlardır. Önerilen yöntemde *n*-gram tekniği kullanılarak bayt dizileri ve bu dizilerin bulunma sıklıkları çıkartılmıştır. Daha sonra WEKA programı kullanılarak belirlenen dizilere göre sınıflandırma yapılmıştır. Önerilen yöntem bazı yeni ZY'leri yakalamada belli bir oranda başarılı olsa da, paketlenmiş ZY'leri yakalamada yetersiz kalmıştır. Dinamik olarak analiz edilen ZY'ler "PEiD" benzeri araçlar yardımıyla paketlerden arındırılarak tespit edilebilir.

Canali vd. (2012) tarafından davranış tabanlı imza yöntemi önerilmiştir. Birden fazla sistem çağrısının oluşturduğu anlamlı dizilim davranış olarak tanımlanmıştır. Bu yöntemde imzalar atomlardan oluşmaktadır. Atomlar: Sistem çağrıları, argümanlar ile birlikte sistem çağrıları,

davranışlar ve argümanlarıyla birlikte davranış gruplarından oluşmaktadır. Atomlar belli modellerle gruplandırılarak imzalar oluşturulmuştur. Bu amaç için *n*-gram, tubles ve bag modelleri kullanılmıştır. Islam vd. (2013) statik ve dinamik özellikleri birleştiren bir tespit sistemi önermişlerdir. Bu sistem üç farklı çeşit özellik içermektedir: Kullanılan metot uzunluklarının (bayt olarak) frekansları, yazdırılabilir dizi (string) bilgileri, sistem çağrıları ve parametreleri. Bu özellikler birleştirilerek özellik vektörü oluşturulmuş ve sınıflandırma algoritmaları kullanılarak sınıflandırılmıştır.

3.4.3.2 Bulgusal tabanlı tespit yaklaşımı ve yöntemlerinin değerlendirilmesi

Bulgusal tabanlı ZY tespit yaklaşımı ve literatürde yapılan çalışmalar detaylı bir şekilde açıklanmıştır. Bu yaklaşımda statik ve dinamik özellikler belirli kurallara göre gruplandırılarak imzalar oluşturulmaktadır. İmza oluşturulurken API çağrıları, CFG, *n*-gram, Opcode ve hibrit özellikler kullanılmaktadır. Bulgusal tabanlı tespit, bilinen ve bilinmeyen ZY'lerin çeşitli biçimlerini algılasa da bütün yeni nesil ZY'leri tespit etmede yetersiz kalmaktadır. Ayrıca, FP oranı yüksek ölçülmektedir. Özelliklerin tam ve doğru belirlenememesi, oluşturulan kuralların eksik ve karmaşık oluşu ve önerilen yöntemlerin belli ZY türleri için etkili olmaları, yapılan çalışmaların sınırlı kalmasına neden olmuştur.

3.4.4 Model denetleme tabanlı tespit (Model checking-based detection) yaklaşımı

Model kontrolü ya da denetimi, bir sisteme ait modelin belirli şartları taşıyıp taşımadığının test edilmesidir ve farklı alanlarda uzun yıllardır kullanılmaktadır. Son yıllarda ZY tespitinde de kullanılmaya başlanmıştır. Model denetleme tabanlı tespit yaklaşımında zararlı ve normal yazılım özellikleri belirlenmekte ve belirli bir niteliği gösterecek şekilde doğrusal mantık formülleri kullanarak kodlanmaktadır. İşaretlenmek istenen yazılım ve bu yazılımdan elde edilmiş nitelikler daha önce belirlenen niteliklerle karşılaştırılarak zararlı olup olmadığı kontrol edilmektedir. Bu yaklaşım gizleme ve paketleme tekniklerine karşı dirençli olup bazı yeni ZY'leri tespit edebilmektedir.

3.4.4.1 Literatürde yapılan model denetleme tabanlı çalışmaların özeti

Holzer vd. (2007) ZY'leri tespit etmek için doğrulama sistemi önermişlerdir. Bu sistemde, anlamlı spesifikasyon dili CTPL kullanılarak zararlı davranışlar formüle edilmiş ve

yürütülebilir dosyadan sonlu durum modeli çıkartılmıştır. Eğer model denetleyici spesifikasyonu doğru olarak belirlerse, analiz edilen yazılım zararlı; aksi takdirde normal olarak işaretlenmiştir. Bu sistemde aynı saldırı tipini kullanan ailelerdeki ZY'ler tespit edilmiştir. Ayrıca, bilinen ZY'lerle benzer davranışlar gösteren yeni ZY'ler de tespit edilmiştir. Çalışmaya göre model denetleme tabanlı bir yaklaşım ZY anlamsal özelliklerini geleneksel tespit yaklaşımlara göre daha doğru bir şekilde belirlemekte ve bu durum DR'yi artırmaktadır. Önerilen yöntem ve test sonuçları hakkında makalede yeterli bilgi verilmemiştir. Daha güvenilir sonuçlar elde etmek için daha fazla zararlı ve normal yazılım analiz edilmeli, daha doğru spesifikasyonlar elde etmek içinse davranışsal bağımlılıklar net olarak belirlenmelidir.

Kinder vd. (2008) model denetleme tabanlı çalışan ve imza güncellemesi olmadan solucanın değişik biçimlerini tespit edebilen proaktif bir ZY tespit yöntemi önermişlerdir. Önerilen yöntem, kontrol akış diyagramlarını çalıştırılabilir dosyalardan çıkartmakta ve daha önce belirtilen spesifikasyonlar kullanılarak otomatik olarak doğrulamaktadır. Bunun için yeni bir spesifikasyon dili CTPL kullanmışlardır. Test sonuçlarına göre önerilen yöntem solucanların değişik biçimlerini düşük FPR ile tespit edebilmektedir. Önerilen yöntem bazı kısıtlamalar içermektedir. Bunlar:

- Model çıkarma işlemi sözdizimsel olup veri akış analizini içermemektedir;
- Zararlı davranışlar farklı prosedürlerde dağıtılmış olarak bulunmakta ve bunların birleştirilerek değerlendirilmesi performansı düşürmektedir;
- Makro, X86 mimarisinin tüm komutlarını kapsamamaktadır.

Bu eksikliklerin giderilmesi veya azaltılması performansı artıracaktır.

Song ve Touili (2014) ZY tespiti için pushdown model denetleme metodunu önermişlerdir. Önerilen metot, model denetleme problemini sembolük değişkenli boşluk Büchi pushdown sisteminin kontrolüne indirgemektedir. Öncelikle, çalıştırılabilir yazılım kodları pushdown sistemlere (PDS) dönüştürülmektedir. Daha sonra, yığın hesaplama ağaç yüklem mantığı (SCTPL) kullanılarak zararlı davranışlar belirlenmektedir. En sonunda, PDS'ler, SCTPL spesifikasyonlarıyla karşılaştırılarak yazılımlar tespit edilmektedir. Çalışmaya göre önerilen metot gizlenme tekniklerine karşı dirençli olup ZY'leri yüksek doğruluk oranıyla tespit

etmektedir. Fakat önerilen metot ancak yığındaki verilerin doğrudan bellek erişimiyle değiştirilemediği durumlarda etkili bir şekilde çalışmaktadır.

Model denetleme ile Android ZY ailelerinin belirlenmesi Battista vd. (2016) tarafından önerilmiştir. Önerilen modelin etkinliğini göstermek için en yaygın ZY ailelerinden olan DroidKungFu ve Opfake analiz edilmiştir. Önerilen algoritma, kaynak kodu derlendiğinde üretilen java bayt kodunu (bytecode) analiz ederek doğrulayabilmektedir. Çalışmaya göre önerilen model ZY payloadlarını yüksek doğruluk oranıyla hızlı bir şekilde tespit edebilmektedir. Makalede sadece birkaç Android ZY ailesi analiz edilmiştir. Analizin genişletilerek daha fazla Android ailesinin analiz edilmesi önerilen modeli değerlendirmek adına daha güvenilir sonuçlar üretecektir. Ayrıca, payload aile ağacının her ZY için çıkarılması benzer ZY'leri tespitte yararlı olacaktır.

3.4.4.2 Model denetleme tabanlı tespit yaklaşımı ve yöntemlerinin değerlendirilmesi

Model denetleme tabanlı tespit yaklaşımı ve bu yaklaşımı kullanan çalışmalar özetlenmiştir. Bu yaklaşım genellikle program doğrulaması için kullanılmış ve ZY tespiti için yeterince kullanılmamıştır. Bazı yeni ZY'leri tespit etmede etkili olsa da karmaşık ZY'leri tespit etmede yetersiz kalmaktadır. Ayrıca, karmaşık ve kaynak yoğunluklu bir yaklaşımdır. Daha doğru tanımlayıcı özellikler belirlemek, davranışsal bağımlılıklar tanımlamak ve daha etkili LTL, CTL, CTPL formülleri kullanmak performansı artırabilir. Bununla birlikte, model denetleme tabanlı yaklaşımın farklı yaklaşımlarla kullanılması performansı artırabilir.

3.4.5 Derin öğrenme (Deep Learning (DL)-based detection) tabanlı tespit yaklaşımı

Örneklerden öğrenen ve yapay sinir ağlarını (YSA) miras alan ML'nin bir alt alanıdır. Yeni bir yaklaşımdır ve görüntü işleme, sürücüsüz arabalar ve ses kontrolü gibi alanlarda yaygın olarak kullanılmasına rağmen ZY tespitinde yeterince kullanılmamıştır. DL tabanlı tespit yaklaşımı yüksek performansla çalışmakta ve özellik alanını büyük ölçüde azaltmaktadır ama atlatma saldırılarına karşı dayanıklı değildir. Ayrıca, gizli katman oluşturmak zaman almakta ve ilave katmanlar eklemek model performansını çok az artırmaktadır. DL tabanlı ZY tespit yaklaşımı oldukça erken aşamalardadır, bu nedenle bu yaklaşımı daha doğru bir şekilde değerlendirmek için daha fazla çalışmaya ihtiyaç duyulmaktadır.

3.4.5.1 Literatürde yapılan derin öğrenme tabanlı çalışmaların özeti

DL tabanlı tespit yaklaşımı kullanan Droid-Sec Yuan vd. (2014) tarafından önerilmiştir. Droid-Sec hem statik hem de dinamik analiz kullanmakta olup 200'den fazla özellik belirlemektedir. Denetimsiz eğitim öncesi ve denetimli geri yayılma safhası olmak üzere iki aşamadan oluşmaktadır. Eğitim öncesi safhada, Android uygulamalarını karakterize etmek için derin inanç ağı (DBN) (Bengio 2009) kullanılmıştır. DBN kısıtlı Boltzmann makinelerini kullanmaktadır. Geri yayılma safhasında, önceden eğitilmiş sinir ağı denetimli bir şekilde etiketlenmiştir. Bu şekilde DL modeli oluşturulmuştur. Test sonuçlarına göre doğruluk oranı %96 olarak ölçülmüş olup SVM algoritması diğer ML algoritmaları olan C4.5, LR ve NB'den daha iyi performans göstermiştir. Daha fazla uygulama analiz etmek ve analiz süreçlerini otomatikleştirmek sistem performansını ve güvenirliğini artıracaktır.

İki boyutlu yazılım özelliklerini kullanan derin sinir ağı tabanlı ZY tespit sistemi Saxe ve Berlin (2015) tarafından açıklanmıştır. Önerilen sistem üç ana bölümden oluşmaktadır:

- İlk bölümde, zararlı ve normal yazılım örneklerinden dört farklı tamamlayıcı özellik çıkartılmıştır;
- İkinci bölümde, bir giriş katmanı, iki gizli katman ve bir çıkış katmanından oluşan derin sinir ağı inşa edilmiştir;
- Üçüncü bölümde, skor kalibratörü kullanılarak sinir ağının çıktıları yorumlanmıştır. Bu aşamada dosyanın zararlı olup olmadığının tahmini yapılmıştır.

Çalışmaya göre önerilen sistem %95 DR ile çalışmakta olup FPR'si düşüktür. Önerilen sistemin performansı standart çapraz doğrulama kullanıldığında yüksek doğruluk oranına ulaşmış olsa da bölünmüş (split) doğrulama kullanıldığında performans hızlı bir şekilde düşmüştür. Bu durum deobfuscation kullanılarak ortadan kaldırılabilir. Ayrıca, analiz sırasında kullanılan ZY sayısı normal yazılım sayısına göre çok fazladır. Doğru tahmin elde etmek için daha fazla normal yazılım analiz edilmelidir.

Huang ve Stokes (2016) ZY sınıflandırması için çok görevli öğrenme olan MtNet mimarisini önermişlerdir. Önerilen mimaride, zararlı ve normal yazılımlar dinamik analizle elde edilen verilerle eğitilmiştir. Çok görevli öğrenme, gizli katmanların düşük seviyelerde bile daha iyi

öğrenmelerini sağlamıştır. Ayrıca, MtNet mimarisi ReLU (düzeltilmiş doğrusal birim) aktivasyon fonksiyonu kullanarak epoch sayısını yarıya indirmekte ve hata oranını düşürmektedir. Makale, MtNet'in normal bir sinir ağı mimarisine kıyasla büyük bir performans gösterdiğini iddia etmektedir. Fakat önerilen mimaride, ilave katman ekleyerek model performansı artırılamamış olup atlatma saldırılarına karşı direnç gösterilememiştir.

Ye vd. (2018) yeni ZY'leri tespit için heterojen DL sistemi önermişlerdir. Önerilen sistem, çok katmanlı kısıtlı Boltzmann makineleri ve ilişkili hafıza kullanılarak çalışmaktadır ve iki safhadan oluşmaktadır: Önce öğrenme ve sonra öğrenme. Önce öğrenme safhasında, etiketlenmiş ve etiketlenmemiş dosyalardan çok katmanlı özellikler öğrenilerek ön öğrenme yapılmaktadır. Bu aşamada her dosyanın karakteristik özellikleri belirlenmektedir. Sonra öğrenme safhasında, denetimli öğrenme gerçekleştirilerek ZY'ler normal yazılımlardan ayrıştırılmaktadır. Çalışmaya göre önerilen sistem geleneksel öğrenme yöntemleriyle karşılaştırıldığında performans artışı gözlemlenmiştir.

DL tabanlı ZY tespit yöntemleri performansı artırmalarına rağmen atlatma saldırılarına karşı dirençli değillerdir. Grosse vd. (2016) derin sinir ağlarına karşı yapılan saldırıları incelemişlerdir. Makaleye göre iyi hazırlanmış hileli girdiler DL modellerini yanıltarak yanlış sınıflandırmaya neden olmaktadır. Değerlendirme için DREBIN veri seti kullanılmıştır. Test sonuçlarına göre %80'e varan yanlış işaretlemeler olmuştur. Bu durum atlatma saldırılarının güvenlik açısından kritik alanlarda büyük bir tehdit olduğunu göstermektedir. Kolosnjaji vd. (2018) derin ağları kullanarak ham baytlardan öğrenen tespit yöntemlerinin güvenlik açıklarını araştırmışlardır. Makalede gradyan tabanlı saldırılar kullanılarak yakın zamanda önerilen derin ağların performansları test edilmiştir. Test sonuçlarına göre ZY'lerin %1'den daha az bir bölümü değiştirilerek derin ağlar yüksek oranda atlatılmaktadır.

3.4.5.2 Derin öğrenme tabanlı tespit yaklaşımı ve yöntemlerinin değerlendirilmesi

DL tabanlı ZY tespit yaklaşımı ve bu yaklaşımda kullanılan yöntemlerin literatür özeti verilmiştir. DL tabanlı yöntemler güçlü, etkili ve özellik alanını büyük ölçüde azaltmalarına rağmen atlatma saldırılarına karşı dayanıklı değillerdir. Ayrıca, bu yöntemlerde gizli katman eklemek zaman almakta ve performansı çok artırmamaktadır. DL tabanlı ZY tespit yaklaşımı

oldukça erken aşamalardadır. Bu nedenle bu yaklaşımı daha doğru bir şekilde değerlendirmek için daha fazla çalışmaya ihtiyaç duyulmaktadır.

3.4.6 Bulut tabanlı tespit (Cloud-based detection) yaklaşımı

Bulut bilişim kolay erişilebilirlik, istek üzerine depolama ve maliyetleri düşürme gibi birçok avantaj sağladığı için hızla gelişmektedir. Sağladığı bu avantajlardan dolayı farklı alanlarda olduğu gibi ZY tespitinde de kullanılmaktadır. Bulut tabanlı ZY tespit yaklaşımı, bilgisayarlar ile mobil cihazlar için tespit performansını artırmakta ve daha büyük veri tabanları ve hesaplama alanı sağlamaktadır. Kullanıcı her türlü dosyayı bulut ortamına yükleyebilmekte ve yüklenen dosyanın ZY olup olmadığını içeren bir rapor kullanıcıya iletilmektedir. Bulut tabanlı tespit yaklaşımının birçok avantajı olmasına rağmen bazı dezavantajları da vardır. Bunlar şöyle sıralanabilir:

- Kullanıcının konum, şifre ve kredi kart bilgileri gibi hassas verileri ifşa edebilecek dosyaların buluta yüklemesinin gerekliliği;
- Diğer tespit yaklaşımlarına göre daha fazla gecikmenin yaşanması, bu nedenle istemci ve sunucu arasındaki iletişim özellikle IoT ve mobil cihazlar için optimize edilmelidir;
- Tüm konumlardaki dosyalar için gerçek zamanlı izlemenin eksikliği.

3.4.6.1 Literatürde yapılan bulut tabanlı çalışmaların özeti

Martignoni vd. (2009) davranış tabanlı tespit yaklaşımın performansını artıran sistemi bulut ortamında tasarlamışlardır. Önerilen sistemde ZY'ler güvenlik laboratuvarı ve potansiyel kurban makineleri de dahil olmak üzere dağıtılmış bir ortamda çalıştırılarak ZY'lerin karmaşık davranışları analiz edilmiştir. Test sonuçlarına göre aynı ZY'nin farklı ortamlarda çalıştırılması performansı artırmıştır. Fakat önerilen yöntem atlatma saldırılarına karşı dirençli olmayıp ZY'lerin bir kısmını normal yazılım olarak işaretlemiştir. Sistemde gerekli iyileştirmeler yapılarak atlatma saldırılarına karşı daha dirençli bir sistem geliştirilebilir.

Cha vd. (2011) SplitScreen adlı yeni bir ZY koruma sistemi tasarlamışlardır. SplitScreen, imza eşleştirmesinden önce ek bir tarama yapan dağıtılmış bir ZY tespit sistemidir. İstemci-sunucu tabanlı çalışan bu sistem hem bellek tasarrufu sağlamakta hem de depolama miktarını

azaltmaktadır. ClamAV'nin bir uzantısı olarak geliştirilen SplitScreen, tarama performansını hafızanın yarısını kullanarak iki katından fazla artırmaktadır. Önerilen sistem ölçeklendirilebilir olup birçok farklı cihaz için kullanılabilmektedir. Bulutta sınırlı sayıda sunucu kullanılmıştır, fazla sunucu kullanmak ve sunucu performansını optimize etmek, ayrıca bazı işlemleri istemci tarafına yüklemek sistem performansını daha fazla artırabilir.

Kaynak kısıtlı IoT cihazları için verimli ve güvenilir güvenlik hizmetleri sunan CloudEyes adlı bulut tabanlı ZY tespit sistemi Sun vd. (2017) tarafından önerilmiştir. CloudEyes'ta, istemci tarafın doğru eşleme aralığını önemli ölçüde azaltmak için imza parçalarının özetini kullanan bir tarama aracı geliştirilmiştir. Sunucu tarafında ise zararlı imza parçalarının geriye dönük yönlendirmelerini sağlayan tersine çevrilebilir yeni bir imza algılama mekanizması olan şüpheli kova çapraz filtrelemesi geliştirilmiştir. Çalışmaya göre CloudEyes veri gizliliğini ve düşük maliyetli iletişimi garanti etmektedir. Ayrıca, önerilen sistemin benzer sistemlerden daha iyi performans göstereceği öne sürülmüştür. Diğer yandan, sistemin tespit ve doğruluk oranı daha da artırılabilir. Örneğin, imza başlatma sırasında depolama ve eşleştirme performanslarını optimize etmek amacıyla Winnowing Block Shingling ve Winnowing Multi-Hashing gibi yöntemler kullanılarak verilerin boyutları daha da azaltılabilir.

Xiao vd. (2017) mobil cihazların uygulama izlerini temel istasyonlar veya dinamik ağlardaki erişim noktaları aracılığıyla güvenlik sunucularına yüklediği bulut tabanlı ZY tespit oyununu önermişlerdir. Mobil cihazlar için en iyi boşaltma oranını elde etmek için Q-öğrenme şeması tasarlamışlardır. Performansı artırmak için Dyna mimarisi kullanılmış, öğrenme sürecini hızlandırmak için ise karar sonrası durum öğrenme şeması kullanılmıştır. Çalışmaya göre önerilen sistem performansı artırmış, gecikmeyi azaltmış ve mobil cihaz kullanım verimliliğini artırmıştır.

Yadav (2019) bulut ortamında çalışan ZY tespit sistemi önermiştir. Önerilen sistem sınıflandırma ve kümeleme olmak üzere iki modülden oluşmaktadır. Kümeleme modülünde, giriş verileri ağırlıklı bulanık c-ortalama algoritması (MFCM) kullanılarak kümelere dönüştürülmektedir. Sınıflandırma modülünde, kümelerdeki kütle merkezi otomatik ilişkili sinir ağına verilerek bilginin izinsiz olup olmadığını belirlenmektedir. Çalışmaya göre

önerilen yaklaşım ZY'leri yüksek DR ile işaretleyerek mevcut sınıflandırıcılardan daha iyi performans göstermiştir.

3.4.6.2 Bulut tabanlı tespit yaklaşımı ve yöntemlerinin değerlendirilmesi

Bulut tabanlı ZY tespit yaklaşımı ve bu yaklaşımı kullanan çalışmalar özetlenmiştir. Son zamanlarda, bulut tabanlı tespit yaklaşımı popülerlik kazanmaya başlamıştır. Bulut ortamı bilgisayarlar ve mobil cihazlar için daha büyük veri tabanları ve hesaplama kaynakları sunarak performansı artırmaktadır. Ayrıca, bazı yeni ZY'leri de tespit edebilmektedir. Fakat kullanıcının veri üzerindeki denetimini kaybetmesi sonucu gizlilik ve güvenlikle ilgili kaygıların artması, ayrıca bu yaklaşımın atlatma saldırılarına karşı dirençli olmaması dezavantajları arasında gösterilebilir.

3.4.7 Mobil ve IoT tabanlı tespit (Mobil and IoT-based detection) yaklaşımı

IoT mimarisi genellikle ev aletleri, ağ kameraları ve sensörler gibi çok çeşitli İnternet bağlantılı akıllı cihazlardan oluşmaktadır. Mobil ve IoT cihazlar giderek İnternet ortamında normal bilgisayarlara göre daha fazla alan kaplamaya başlamıştır ve bu nedenle saldırganlar için de daha fazla hedef haline gelmeye başlamıştır. Mobil ve IoT tabanlı tespit yaklaşımı, ZY tespiti için veri madenciliği ve ML algoritmalarını kullanmaktadır. Özellikler oluşturulurken sistem çağrıları, güvenliğe duyarlı API'ler, bilgi akışları ve kontrol akış yapıları kullanılmaktadır. Mevcut çalışmalarda mobil ve IoT tabanlı yaklaşımların geleneksel ve yeni nesil ZY'lerin tespitinde iyileştirme yaptıkları gösterilmesine rağmen karmaşık ZY'leri tespit etmede yetersiz kaldıkları gösterilmiştir.

3.4.7.1 Literatürde yapılan mobil ve IoT tabanlı çalışmaların özeti

Saracino vd. (2018) aynı anda dört düzeydeki özellikleri analiz eden ve ilişkilendiren yeni bir ZY tespit sistemi olan MADAM'ı önermişlerdir. Çalışmada, bilinen zararlı davranışlar kullanılarak kötü amaçlı davranış grupları oluşturulmuştur. Çalışmaya göre MADAM 2800 uygulama arasında zararlı uygulamaların %96'sından fazlasını tespit etmektedir fakat atlatma saldırılarına karşı savunmasızdır. Çalışmada önerilen yöntemin yeni ZY'ler karşındaki performansından bahsedilmemiştir.

Li vd. (2018) Android ZY'lerini tespit etmek için önemli izin belirleme yöntemi olan SigPID'i önermişlerdir. Tüm Android izinlerini çıkarmak ve analiz etmek yerine 3-seviyeli madencilik kullanılarak budama yapılmış ve böylece ZY'leri normal yazılımlardan ayırt eden izinler belirlenmiştir. Daha sonra ML algoritmaları kullanılarak farklı zararlı ve normal yazılım uygulama aileleri sınıflandırılmıştır. Test bulgularına göre 2000'den fazla ZY analiz edildiğinde, 135 izinden 22 iznin önemli olduğu görülmüş ve bu izinler kullanılarak sınıflandırma yapılmıştır. Diğer modern yöntemlerle karşılaştırıldığında SigPID daha iyi performans göstermiştir. SigPID kullanılarak tüm yazılımların %93.62'si ve yeni nesil ZY'lerin %91.4'ü tespit edilmiştir.

IoT ağlarda fidye yazılımların enerji tüketimine bağlı olarak tespit edilmesi Azmoodeh vd. (2018) tarafından önerilmiştir. Önerilen sistem, farklı işlemlerin enerji tüketim desenlerine ML algoritmaları uygulayarak fidye yazılımları diğer ZY'lerden ayırmıştır. Çalışmaya göre en iyi sonuçlar KNN, sinir ağları, SVM ve RF kullanılarak elde edilmiştir. Çalışmada, hangi fidye yazılım ailelerinin analiz edildiği ve bilinmeyen fidye yazılımlarıyla nasıl başa çıkıldığı anlatılmamıştır.

IoT ortamlarında DDoS ZY'lerini tespit etmek için önerilen yöntem Su vd. (2018) tarafından açıklanmıştır. Öncelikle zararlı imgeler çalıştırılabilir PE'lerden elde edilmiş ve "light-weight convolutional neural network" kullanılarak sınıflandırılmıştır. Makaleye göre önerilen yaklaşım normal ve DDoS ZY'lerini %94 doğruluk oranıyla tespit etmiştir. Önerilen yöntem hızlı ve etkili olmasına karşın, kod gizleme tekniklerine karşı savunmasızdır. Opcode dizileri ve API çağrıları gibi daha karmaşık statik özellikler kullanılarak bu sorun kısmen azaltılabilir.

Alazab vd. (2020) izin isteklerini ve API çağrılarını birleştiren etkili bir sınıflandırma modeli önermişlerdir. Android ZY uygulamalarını tanımlama olasılığı en yüksek API çağrılarını belirlemek için üç farklı gruplandırma stratejisi kullanmışlardır: Belirsiz grup, riskli grup ve tehlikeli grup. Çalışmaya göre zararlı uygulamalar hassas verilere erişmek için daha fazla tehlikeli izin isteğinde bulunmaktadır. Önerilen modelin performansını değerlendirmek için 27.891 Android uygulaması içeren veri seti kullanılmıştır. Test sonuçlarına göre önerilen model %94.3'lük bir F-ölçümü ile zararlı uygulamaları tespit etmiştir.

3.4.7.2 Mobil ve IoT tabanlı tespit yaklaşımları ve yöntemlerinin değerlendirilmesi

Mobil ve IoT tabanlı tespit yaklaşımları ve literatürde bu yaklaşımları kullanan çalışmalar özetlenmiştir. Bu yaklaşımlar hem statik hem de dinamik özellikleri kullanmakta olup sistem çağrıları, güvenliğe duyarlı API'leri, Android izinleri ve bilgi akış izleme yapılarını kullanmaktadır. Önerilen yöntemler geleneksel ZY'leri tespit ederken etkili olsa da daha önce bilinmeyen ZY'leri tespit etmede yetersiz kalmaktadır. Ayrıca, geniş uygulama paketleri için ölçeklenebilir değillerdir. Mobil ve IoT tabanlı yöntemler son birkaç yılda yaygınlaşmışlardır ve bu yöntemlerin eksik yönlerini gidermek adına bu alanda daha fazla çalışma yapılmalıdır.

3.5 Zararlı Yazılım Tespit Yaklaşımları ve Bu Alanda Yapılan Çalışmaların Değerlendirilmesi

Literatür özetinde genel olarak mevcutta kullanılan ZY tespit yaklaşımları ve ilgili yöntemleri ana fikir, algoritma türleri ve özellik çıkarma metotları göz önünde bulundurularak özetlenmiştir.

3.5.1 Yapılan çalışmaların değerlendirilmesi

Her bir tespit yönteminin kendi avantajları olmasına ve farklı veri kümeleri için daha iyi çalışmasına rağmen hiçbir tespit yöntemi bütün ZY'leri tespit edememiştir. Etkin imza çıkarmanın zorlukları, davranış tanımlamanın zorlukları, çıkarılan özellik sayısının çok fazla olması (*n*-gram benzeri modeller kullanıldığında) ve çıkarılan özellikler arasındaki benzerlikler ve farklılıkların belirlenmesindeki zorluklar, etkin bir tespit sisteminin oluşturulmasını engellemiştir. Yeni yaklaşım ve yöntemlerle birlikte veri madenciliği ve ML algoritmalarının ZY tespitinde kullanılması, çıkarılan özellikleri anlamlı hale getirmede büyük bir rol üstlenmektedir. Genel olarak yukarıda açıklanan çalışmaların içerdiği kısıtlamalar şöyle sıralanabilir:

- Birçok tespit yöntemi karmaşık bir yapıya sahiptir ve fazla donanım gerektirir;
- Zararlı davranışları belirlemek zordur ve zaman almaktadır;
- Özellik çıkarma yöntemleri etkin olmadığından özelliklerin boyutu zamanla artmaktadır;
- Tespit ve doğruluk oranları düşüktür;
- Yanlış işaretlemeye eğilimlidirler;

- Belirli bir ZY türünü ya da ailesini tespit etmektedirler;
- Yeni ve karmaşık ZY'leri etkin bir şekilde tespit edememektedirler;
- Atlatma saldırılarına karşı eğilimlidirler.

Gerek var olan yöntem ve modellerin eksikliklerini giderme adına, gerekse yeni yöntemler geliştirmek adına daha fazla sayıda bilimsel çalışmaya ihtiyaç duyulmaktadır. Bu tez çalışmasında yukarıda belirtilen kısıtlamalar göz önünde bulundurularak her adımda gerekli katkılar yapılmıştır.

3.5.2 Zararlı yazılım tespit yaklaşımlarının değerlendirilmesi

İmza tabanlı, davranış tabanlı, bulgusal tabanlı ve model denetleme tabanlı yaklaşımlar iyi bilinmektedir ve 10 yıldan fazla bir süredir ZY tespiti için kullanılmaktadır. Bu yaklaşımlar ZY'leri tespit etmek için tersine mühendislik, veri madenciliği ve ML tekniklerini kullanmaktadır. ZY tespit yaklaşımlarının karşılaştırılması ve her yaklaşımın artıları ve eksileri çizelge 3.4 ve çizelge 3.5'te görülmektedir.

İmza tabanlı tespit yaklaşımı ve ilgili yöntemleri bilinen ZY'leri tespitte hızlı ve etkili olmalarına rağmen yeni ZY'leri yakalamada yetersiz kalmaktadır (Çizelge 3.4 - 3.5). Etkili bir imza tabanlı tespit sistemi oluşturmak için:

- Gizlenme tekniklerini etkisizleştirmek için bazı dinamik özellikler kullanılabilir;
- Özellik seçim aşaması eklenebilir;
- DR'yi artırmak için DL, aktif öğrenme ve ML gibi yeni teknolojiler kullanılabilir.

ZY'nin işlevselliğini belirlemek için davranışa dayalı tespit yaklaşımı kullanılmaktadır. Bu yaklaşımda ZY imzası değişse bile program davranışları aşağı yukarı aynı kalmaktadır. Bu nedenle, yeni nesil ZY'leri ve aynı ZY'lerin farklı biçimlerini tespit edebilmektedir (Karbab ve Debbabi 2019). Ayrıca, kod gizleme tekniklerine karşı daha etkilidir (Çizelge 3.4). Bazı davranışlar zararlı ve normal yazılımlarda benzerlik gösterdiği için normal yazılımlar zararlı, zararlı yazılımlar normal olarak işaretlenebilmektedir. Davranışları daha az hata oranıyla belirlemek için bulutlardaki farklı makineler kullanılarak programlar birden fazla ortamda çalıştırılabilir. Bu durum yanlışlıkla normal olarak işaretlenmiş yazılımların sayısını azaltacaktır.

Çizelge 3.4 Zararlı yazılım tespit yaklaşımlarının karşılaştırılması

ZY tespit yaklaşımları	Sıfır gün saldırılarını tespit	Gizlenme tekniklerine dayanıklı	İyi bilinen yaklaşım	Yeni yaklaşım
İmza tabanlı	☒	☒	☑	☒
Davranış tabanlı	☑	☑	☑	☒
Bulgusal tabanlı	☑	☒	☑	☒
Model denetleme tabanlı	☑	☑	☑	☒
Derin öğrenme tabanlı	☑	☒	☒	☑
Bulut tabanlı	☑	☒	☒	☑
Mobil ve IoT tabanlı	☑	☒	☒	☑

Çizelge 3.5 Zararlı yazılım tespit yaklaşımlarının avantaj ve dezavantajları

ZY tespit yaklaşımları	Avantaj	Dezavantaj
İmza tabanlı	Bilinen ZY'ler için hızlı ve etkili	Yeni nesil ZY'leri tespit etmede yetersiz
	Uzun yıllar kullanılmakta	İmza çıkarmak zaman almakta
	Aynı aileye ait ZY'leri tespit etmede etkili	Gizlenme tekniklerine karşı hassas
Davranış tabanlı	ZY işlevselliğini belirleyebilmekte	ZY'nin sınırlı bir görünümünü sunmakta
	Yeni nesil ZY'leri tespit edebilmekte	Bazı davranışlar zararlı ve normal yazılımlarda benzerlik göstermekte
	Aynı ZY'nin farklı görünümlerini tespit edebilmekte	Bütün davranışları belirlemek mümkün olmamakta
	Gizlenme tekniklerine karşı dirençli	Çok sayıda FP
Bulgusal tabanlı	Yeni nesil ZY'leri tespit edebilmekte	Çok sayıda kural ve eğitim aşaması uzun
	Statik ve dinamik özellikleri kullanabilmekte	Metamorfik tekniklere karşı savunmasız
Model denetleme tabanlı	Aynı aileye ait ZY'leri tespit etmede etkili	Karmaşık ve kaynak yoğunluklu
	Gizlenme tekniklerine karşı dirençli	ZY'nin sınırlı bir görünümünü sunmakta
	Yeni nesil ZY'leri tespit edebilmekte	Spesifikasyon belirlemek zaman almakta
Derin öğrenme tabanlı	Güçlü ve etkili	Atlatma saldırılarına karşı hassas
	Özellik alanını önemli ölçüde azaltmakta	Ara katman oluşturmak zaman almakta
Bulut tabanlı	Bilgisayarlar ve mobil cihazlar için tespit performansını artırmakta	Gerçek zamanlı izleme eksikliği
	Büyük veri tabanları ve hesaplama alanı sağlamakta	Gizli verileri ifşa edebilmekte
	Kolay erişilebilir, yönetilebilir ve güncellenebilir	İstemci ve sunucu arasında zaman gecikmesi
Mobil ve IoT tabanlı	Yeni nesil ZY'leri tespit edebilmekte	Gizlenme tekniklerine karşı hassas
	Statik ve dinamik özellikleri kullanabilmekte	Birçok uygulamayı ölçeklendirmede yetersiz

Bulgusal tabanlı tespit yaklaşımı ve ilgili yöntemleri API çağrıları, Opcode, CFG, *n*-gram, DLL listesi gibi hem statik hem de dinamik özellikler kullanabilmektedir. Önceden bilinmeyen bazı ZY'leri tespit edebilmesine rağmen metamorfik tekniklere karşı savunmasızdır. Ayrıca, çok sayıda kural ve eğitim aşaması (Choi vd. 2019) bu tespit yaklaşımını karmaşık hale getirmektedir (Çizelge 3.5). Kural sayısının azaltılması ve daha verimli bir öğrenme aşamasının oluşturulması, bu yaklaşımın performansını artırabilir.

Model denetleme tabanlı yaklaşım güçlüdür, bilinmeyen ZY'leri tespit edebilir ve gizlenme tekniklerine karşı dirençlidir (Çizelge 3.5). Fakat bu yaklaşımda ZY'lerin sınırlı bir

görünümü elde edilebilmekte ve spesifikasyon oluşturmak zaman almaktadır. Ayrıca, çok prosesli bazı yeni nesil ZY'leri tespit etmede yetersiz kalmaktadır. Daha doğru formüller tanımlamak ve etkili model denetleyici kullanmak performansı artırabilir.

Son zamanlarda DL tabanlı yaklaşım, bulut tabanlı yaklaşım, mobil cihazlar ve IoT tabanlı yaklaşımlar ZY tespitinde kullanılmaktadır (Çizelge 3.5). DL'ye dayalı tespit yaklaşımı ve ilgili yöntemleri, yeni nesil ZY'leri tespit etmede ve özellik uzayını azaltmada etkilidirler (Zhang vd. 2019, Wang vd. 2019) ancak atlatma saldırılarına (evasion attaks) karşı dayanıklı değildirler. Öte yandan, bulut tabanlı tespit yaklaşımı ve ilgili yöntemleri DR'yi artırmakta, FPR'yi azaltmakta ve daha büyük ZY veri tabanları ve güçlü hesaplama kaynakları sağlamaktadır (Mirza vd. 2018). İstemci ve sunucular arasında zaman gecikmesinin yaşanması, istemci tarafında gerçek izlemenin olmaması ve istemci verilerinin başkaları tarafında görülme ihtimalinin olması bu yaklaşımın dezavantajları arasında gösterilebilir. Mobil cihazlar ve IoT tabanlı tespit yaklaşımları hem statik hem de dinamik özellikleri kullanabilmekte ve geleneksel ve yeni nesil ZY'leri tespit etmede bir iyileşme sağlamaktadır (Ma vd. 2019). Ancak, bazı yeni nesil karmaşık ZY'leri tespitte yetersiz kalıp çok sayıda uygulama için ölçeklendirilememektedir.

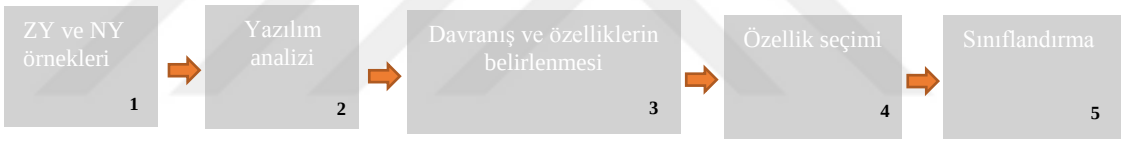
Her bir tespit yaklaşımının kendine has avantajları olmasına ve farklı veri kümeleri için daha iyi performans göstermesine rağmen hiçbir yaklaşımın bütün ZY'leri tespit ettiği gösterilmemiştir. ZY'lerin karmaşıklığı arttıkça, tüm tespit yaklaşımlarının performansları genel olarak azalmaktadır. İmza, bulgusal ve çoğu zaman mobil ve IoT tabanlı tespit yaklaşımlarının performasları diğer yaklaşımlara (davranış, model denetleme, bulut ve DL) göre daha düşük ölçülmüştür (Aslan ve Samet 2020). Bunun nedeni, daha sonraki yaklaşımların bilinmeyen ve gizlenmiş ZY'leri tespit etmede daha etkili olmasıdır. Davranış tabanlı tespit yaklaşımı oldukça iyi performans gösterirken, imza tabanlı tespit yaklaşımı en düşük performansı göstermektedir (Aslan ve Samet 2020). Model denetleme ve bulut tabanlı tespit yaklaşımları; DL, bulgusal, mobil cihazlar ve IoT tabanlı tespit yaklaşımlarına göre daha iyi performans göstermektedir. Bu tespit yaklaşımlarının güçlü yönlerini birleştirmek daha iyi tespit sistemi sağlayabilir. Örneğin, davranış tabanlı yaklaşımı model denetleme ile birleştirmek ve aynı zamanda DL ve bulut kullanmak daha iyi tespit mekanizması oluşturacaktır. Ayrıca, blok zinciri (block chain) ve büyük veri (big data) gibi yeni teknolojilerin kullanılması, daha etkili bir tespit yaklaşımı oluşturmaya yardımcı olacaktır.

ZY tespit yaklaşımları her geçen gün geliştirilse de aşağıdaki zorluklar açık bir sorun olmaya devam etmektedir:

- Yeni nesil ZY'ler, kendilerini gizlemek için bazı gizlenme ve paketleme teknikleri kullanmaktadır. Bu teknikleri kullanan ZY'ler doğru analiz edilmeyi engelleyerek tespit edilmekten kaçınmaktadır. İmza tabanlı tespit yaklaşımı, bu tekniklere karşı dirençli değildir. Davranış ve model denetleme tabanlı yaklaşımlar, birçok atlatma tekniğine karşı etkili olsalar da, tüm atlatma tekniklerine karşı dirençli değillerdir.
- Gerçek zamanlı izleme ve tespit birçok zorluk barındırmaktadır. ZY'leri tespit için yapılan akademik çalışmalarda genelde testler veri setleri üzerinde yapılmaktadır ve gerçek zamanlı değillerdir.
- ZY tespit yaklaşımlarının çoğu yanlış işaretlemeye eğilimlidir. Bazı özellikler ve imzalar ZY'lerin normal ve normal yazılımların da zararlı olarak işaretlenmesine neden olmaktadır.
- Genel olarak öğrenme algoritmaları önyargıya ve aşırı öğrenmeye karşı hassastırlar. Bu durum DR'yi düşürmekte ve FPR'yi artırmaktadır.
- ZY tespit yaklaşımlarının performanslarını değerlendirmek için kullanılabilecek iyi bilinen ve kabul görmüş veri setleri bulunmamaktadır.

4. DAVRANIŞA GÖRE ZARARLI YAZILIM ANALİZ VE TESPİT SİSTEMİNİN GELİŞTİRİLMESİ

Bu tezde yeni bir davranışa dayalı ZY tespit sistemi geliştirilmiştir. Mevcut çalışmalarda genel olarak sistem çağrıları davranış kabul edilerek özellikler oluşturulmaktadır. Özellikler oluşturulurken *n*-gram ve benzeri modeller kullanıldığı için özellik sayısı hızla artmaktadır. Ayrıca, birbiriyle ilişkili olmayan davranışlar kullanılarak özellik oluşturulması performansı da düşürmektedir. Bu tez çalışmasında birkaç sistem çağrısı anlamlı bütünlük oluşturmak şartıyla davranış olarak kabul edilmiştir. Davranışlardan özellikler oluşturulurken ancak anlamlı ve ilişkili davranışlar özellik oluşturabilmektedir. Ayrıca, davranışların hangi sistem yolunda gerçekleştirildiği de analiz edilerek her özellik için risk skoru hesaplanmaktadır. Daha sonra bu skorlar dikkate alınarak özellik seçimi yapılmaktadır. Özellik seçimi bittikten sonra mevcut ML algoritmaları kullanılarak özellikler sınıflandırılmaktadır. Ayrıca, karar ağaçları için özellik yerleştirme kriterleri iyileştirilerek sınıflandırma performansı artırılmaktadır. Önerilen sistemin mimarisi şekil 4.1'de gösterilmektedir.



Şekil 4.1 Önerilen sistemin genel mimarisi

Şekil 4.1'deki sayılar blok numaralarını göstermektedir. Tez kapsamında yapılan katkılar aşağıdaki gibi özetlenebilir:

1. ZY'lerin analizi elle yapılırken mevcut araçların nasıl kullanılacağıyla ilgili bir yöntem önerilerek test edilmiştir (Blok 2).
2. Davranışa dayalı ZY kalıplarını belirlemek için yeni bir model olan EMDM geliştirilmiştir. Bu model kullanılarak hem veri setleri oluşturulmuş hem de özellik seçimi yapılmıştır (Blok 3).
3. Özellik seçimi sırasında hem mevcut özellik seçim ölçütleri kullanılmış hem de önerilen EMDM özellik seçim algoritması kullanılarak sonuçlar karşılaştırılmıştır (Blok 4).
4. Sınıflandırma için mevcut sınıflandırma algoritmaları kullanılmıştır (Blok 5).

5. Karar ağaçları için özellik seçimi ve budamayla ilgili iyileştirmeler yapılarak yazılımlar sınıflandırılmıştır (Blok 5).

4.1 Birleştirmeli Sıralı Tespit Yönteminin (BSTY) Geliştirilmesi

ZY analizi, ZY'lerin nasıl çalıştığını anlamak, hangi zafiyetleri kullanılarak saldırı başlatıldığını saptamak ve ZY'leri tespit etmek amacıyla yapılmaktadır. ZY analizi sadece antivirüs üreticileri tarafından değil, aynı zamanda büyük kurumların siber güvenlikten sorumlu birimleri tarafından da yapılmaktadır. Analizinin temel amacı antivirüs tarayıcılarının tespit edemediği ZY'leri tespit etmek ve saldırı hakkında daha fazla bilgi toplayarak saldırılan sistemi daha güçlü hale getirmektir. ZY analizi elle ya da bazı durumlarda otomatik yapılarak şu durumlar tespit edilmeye çalışılır:

- Sistemde hangi güvenlik zafiyetlerinin kullanılarak saldırının başladığını tespit etmek;
- Hangi makine ve programların etkilendiğini tespit etmek;
- Hangi verilerin zarar gördüğünü ve çalındığını tespit etmek;
- Etkilenen programların eski haline nasıl döndürüleceğini araştırmak;
- Yapılabilecek potonsiyel saldırıları önceden tespit etmektir.

Bu amaçlar için statik ve dinamik analiz araçları kullanılmaktadır.

4.1.1 Statik analiz ve kullanılan araçlar

Statik analiz ZY kodlarının çalıştırılmadan ZY'nin işlevselliği hakkında bilgi çıkarma işlemidir. Analiz sırasında dosya adı, dosya tipi, dosya boyutu, MD5 checksum ve ZY imzası gibi bilgiler elde edilmektedir. Analiz temel ve ileri düzey statik analiz olmak üzere ikiye ayrılmaktadır. Temel analizde genel bir değerlendirme yapılmakta ve analiz detaylandırılmamaktadır. Bu analiz sırasında hash değerleri, string dizilimler, kullanılan metotlar ve dosya başlıklarından çıkartılan bilgiler kullanılmaktadır (Hahn 2014). İleri düzey statik analizde ise program komutları detaylı incelenerek analiz gerçekleştirilmektedir. Bu amaç için tersine derleme yapılarak ZY'deki komutlar ve desenler çıkartılmaktadır. Bu komut ve desenler yorumlanarak tespit işlemi gerçekleştirilmektedir. Statik analiz için yaygın olarak kullanılan açık kaynak kodlu araçlar çizelge 4.1'de gösterilmektedir.

Çizelge 4.1 Statik zararlı yazılım analiz araçları

Araç ismi	Açıklama
PEiD	Paketlenmiş dosyaları saptayan program
PEview	PE formattaki dosyaların yapısını ve içeriklerini görüntüleyen program
PE Explorer	PE formattaki dosyaların içeriğini görüntüleyen ve paketlenmiş dosyaları belirleyen program
PEBrowser Professional	Win32 ve Win64 yürütülebilir dosyalar için paket ayırıcı olarak çalışan program
BinText	İkili dosyalarda gömülü olarak bulunan karakter dizileri çıkaran program
UPX	ZY örneklerini sıkıştırmak için kullanılan paketleyici program
Md5deep	Dosyalar üzerinde MD5, SHA-1, SHA-256 gibi hash değeri hesaplayan program
Dependency Walker	ZY tarafından içe aktarılan DLL'leri ve metotları gösteren program
Resource Hacker	PE'lere gömülmüş şekilde bulunan kaynakları görüntüleme ve değiştirme amacıyla kullanılan program
IDA Pro	ZY analistleri tarafından tersine mühendislik işlemleri için kullanılmakta olan etkileşimli paket ayırıcı
Hex Editors	İkili veri içeren dosyaları görüntülememize ve düzenlememize izin veren program
Hex-Rays Decompiler	Assembly kodunu anlaşılabilir C benzeri yüksek seviyeli dillere dönüştüren IDA Pro eklentisi

4.1.2 Dinamik analiz ve kullanılan araçlar

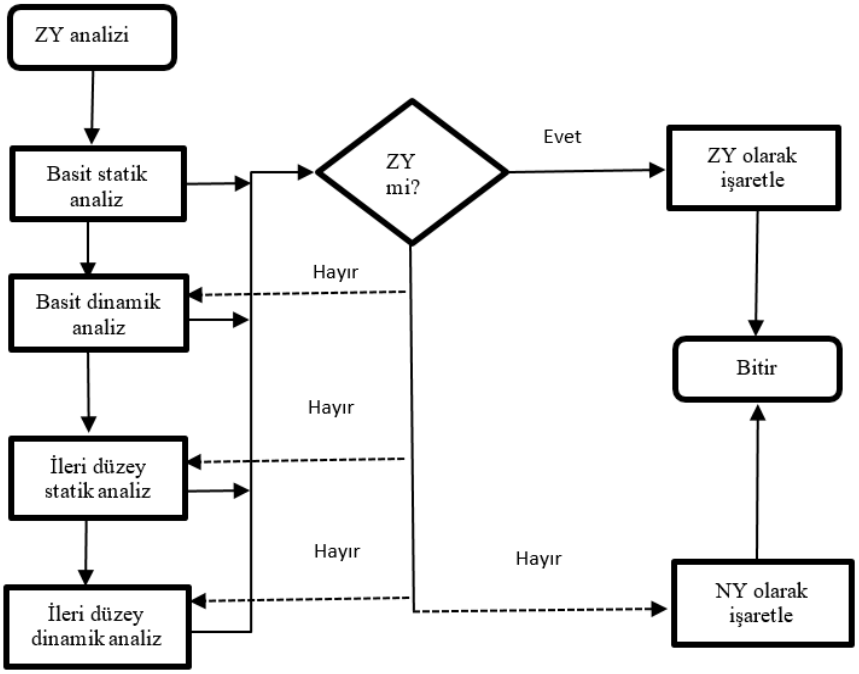
ZY'nin çalıştırılarak sistemle olan etkileşiminin izlenmesidir. Temel ve ileri düzey olmak üzere ikiye ayrılmaktadır. Temel analizde ZY davranışları izleme programları kullanarak incelenmektedir. İleri düzey analizde debugger olarak adlandırılan hata ayıklama araçları kullanılarak değişkenlerin, paremetlerin ve hafıza alanlarının hem içerikleri görüntülenebilmekte hem de bu içerikler değiştirilebilmektedir. Dinamik analiz için yaygın olarak kullanılan araçlar çizelge 4.2'de gösterilmektedir.

Çizelge 4.2 Dinamik zararlı yazılım analiz araçları

Araç ismi	Açıklama
Process Explorer	Bilgisayar sistem yolunda çalışan işlemleri gösteren program
Process Monitor	Gerçek zamanlı olarak sistemde gerçekleşen olayları izlemeye ve görüntüleye imkan sağlayan program
API Monitor	Uygulamalar ve hizmetler tarafından gerçekleştirilen API çağrılarını izlemeye ve denetlemeye imkan sağlayan program
Regshot	Windows kayıt defteri değişikliklerini gösteren program
ApateDNS	DNS yanıtlarını kontrol eden program
Netcat	Gelen ve giden ağ bağlantılarını görüntüleyen program
INetSim	Yaygın olarak kullanılan İnternet servislerini simüle eden program
Wireshark	Ağ protokolleri için paket analiz programı
Capture BAT	Sistem durumunu görüntüleyen program
WinDbg	Kullanıcı ve kernel modunda çalışabilen hata ayıklama programı
OllyDbg	Hata ayıklama ve tersine mühendislik işlemleri için kullanılan program
Burp Suite	Web uygulamalarındaki zafiyetleri saptayan yazılım platformu
Sandboxes (Cuckoo, CWSandbox, Norman Sandbox ve DroidBox)	Programların üzerinde çalıştıkları sistemi etkilemeden yürütülmelerini sağlayan yalıtılmış test ortamları

4.1.3 BSTY kullanılarak zararlı yazılım analizi

ZY'ler statik ve dinamik araçlar kullanılarak analiz edilmektedir. Analiz elle yapılabileceği gibi otomatik olarakta yapılmaktadır. Şekil 4.1 blok 2'de ZY analizi için BSTY önerilmiştir. BSTY insan müdahalesi gerektirmekte olup şüpheli dosyanın analiz edilerek tespit edilmesini sağlamaktadır. BSTY basit statik analizle başlamakta ve ileri düzey dinamik analizle bitmektedir (Aslan ve Samet 2017a). BSTY'nin akış diyagramı şekil 4.2'de gösterilmektedir.



Şekil 4.2 Zararlı yazılım analiz akış diyagramı

İleri düzey statik ve dinamik analiz yapmak daha fazla uzmanlık ve zaman gerektirdiğinden dolayı BSTY basit statik ve dinamik analizle başlamaktadır. Bu aşamada, zararlı ve normal yazılım örnekleri analiz aracına girdi olarak verilip program yapısı ve davranışlar elde edilmektedir. Elde edilen sonuçlar analizci tarafından yorumlanmaktadır. Eğer elde edilen bilgiler örnek programı işaretlemek için yeterliyse analizci ilgili programı ZY olarak işaretlemekte; aksi takdirde ileri düzey statik ve dinamik analizle işleme devam etmektedir. İleri düzey statik ve dinamik analizle elde edilen bilgiler örnek programın işaretlenmesi için yeterliyse ZY olarak işaretlenmekte; aksi takdirde normal olarak işaretlenmekte ve analiz sonlandırılmaktadır (Şekil 4.2). BSTY'nin ZY tespit algoritması şekil 4.3'te gösterilmektedir.

Girdi: analiz edilecek dosya: x′; Çıktı: ZY ya da NY

```
1.  x′ ← analiz edilecek dosya
2.  esikDegeri ← α
3.  tespitDegeri ← 0
4.  toplamSayı← 0
5.  ψ ← aracSonuc = 0
6.  x′← supheliDosya
7.  for i← 1 to n
8.    for j← 1 to m
9.        f_j(x) ← f_j( P(x′ [j]))
10.       tespitDegeri ← tespitDegeri + f_j(x)
11.   end for
12.   if (tespitDegeri > α)
13.     ψ ← ψ + tespitDegeri
14.     toplamSayı++
15.     döngüden çık
16.   else
17.     ψ ← ψ + tespitDegeri
18.     toplamSayı ++
19.   end if
20. end for
21. if ( ψ/ toplamSayı) > α)
22.   zararlı (skor)
23. else
24.   normal (skor)
25. end if
```

Şekil 4.3 Zararlı yazılım tespit algoritması

Önerilen yöntem matematiksel olarak ifade edilmiş beş formüle göre çalışmaktadır:

$$F(x) = (\sum_{i=1}^{n} \psi_i(x)) / n \quad (4.1)$$

$$\psi_i(x) = f_i(P(x')), \text{ öyle ki } i = 1 \ldots. n \quad (4.2)$$

$$P(x') = (T_1 \cup T_2 \cup \ldots. \cup T_m) \quad (4.3)$$

$$f(P(x')) = ((T_1 = 0 \; or \; 1) \cup (T_2 = 0 \; or \; 1) \cup \ldots. \cup (T_m = 0 \; or \; 1)) / m \quad (4.4)$$

$$F(x) = (\sum_{i=1}^{n} \psi_i(((T_1 = 0 \; or \; 1) \cup (T_2 = 0 \; or \; 1) \cup \ldots. \cup (T_m = 0 \; or \; 1)) /m) / n \quad (4.5)$$

$F(x)$ ZY tespit edici, örnek programı girdi olarak almakta ve sonuç olarak zararlı ya da normal yazılım olarak çıktı üretmektedir. X'şüpheli dosya örneğini, $\psi_i(x)$ şüpheli dosya skorlarını göstermektedir ve 1'den n'e kadar her sonuç farklı bir analiz aracı tarafından hesaplanmaktadır. Her şüpheli dosya ($T_1 \cup T_2 \cup \ldots. \cup T_m$) gibi parçalara bölünmekte (bu formüldeki T'ler davranışları ya da string bloklarını temsil etmektedir) ve her parça 0 ya da 1 olarak işaretlenmektedir. Her farklı araç kullanılarak oluşturulmuş 0 ve 1'lerin kombinasyonu ilgili araç için analiz edilen dosyanın ZY olup olmadığını göstermektedir.

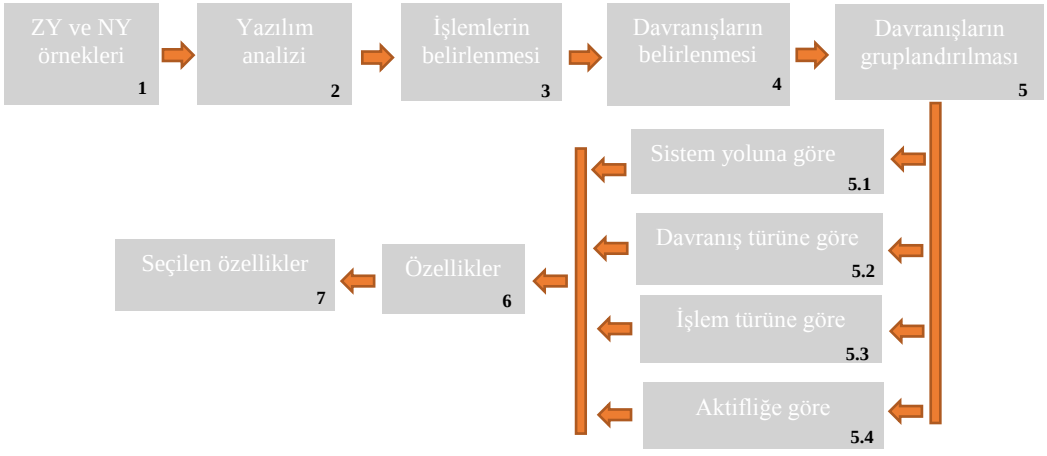
Elde edilen bütün araç sonuçları neticesinde programın zararlı olup olmadığına karar verilmektedir. Karar verme sırasında analistin alan bilgisi büyük önem arz etmektedir.

4.2 Eksiltici Merkezi Davranış Modelinin (EMDM) Geliştirilmesi

ZY'lerin elle analiz edilmesi çok zaman almakta ve alan uzmanlığı gerektirmektedir. Bundan dolayı ZY'leri otomatik analiz eden sistemlere ihtiyaç vardır. ZY'ler genel olarak birbiriyle ilişkili işlemler yapmakla birlikte zaman zaman göstermiş oldukları davranışları gizlemek için birbiriyle ilişkili olmayan işlemler de yapmaktadır. Bundan dolayı veri seti oluştururken birbiriyle ilişkili işlemlerin belirlenerek özelliklerin çıkarılması büyük önem taşımaktadır. *n*-gram ve benzeri otomatik veri seti oluşturma modelleri bu konuda yetersiz kalmaktadır. Çünkü sıralı olarak yapılan her işlem birbiriyle ilişkili olmamakla birlikte ayrıca çok fazla özelliğin oluşturulmasına neden olmaktadır. Bu durum daha sonra yapılacak işlemleri özellik seçimi ve sınıflandırma gibi zorlaştırmaktadır. Bu nedenlerden dolayı *n*-gramdan farklı ve daha iyi performans gösteren modellere duyulan ihtiyaç her geçen gün artmaktadır. Bundan dolayı bu çalışmada EMDM önerilmiştir. EMDM kullanılarak ZY'ler otomatik analiz edilmektedir (Şekil 4.1 Blok 2-3-4). EMDM hem birbiriyle ilişkili davranışları belirlemekte hem de özellik sayısını sınırlı tuttuğundan dolayı *n*-gram benzeri yöntemlere göre daha etkili çalışmaktadır.

4.2.1 EMDM sistem mimarisi

Önerilen EMDM'nin sistem mimarisi şekil 4.4'te verilmiştir.



Şekil 4.4 EMDM sistem mimarisi

Bundan sonraki açıklamalarda EMDM mimarisine ait bölümlerden bahsedilirken şekil 4.4'teki blok numarası (Blok n) verilerek anlatılacaktır. EMDM işlemleri otomatik yapılmakta olup insan müdahalesi gerektirmemektedir. EMDM'ye göre yazılım örnekleri ilgili dinamik araçlar kullanılarak analiz edilmektedir (Blok 2). Sonra, yapılan işlemlere göre davranışlar elde edilmektedir (Blok 3-4). Davranışlar belirlenen kurallara göre gruplandırılmakta ve her davranış için risk skorları atanmaktadır. Daha sonra davranışlar kullanılarak özellikler ve her özellik için de önem dereceleri hesaplanmaktadır (Blok 5.1, 5.2, 5.3, 5.4). Son olarak, özelliklerin skorları kullanılarak özellik seçimi yapılmaktadır (Blok 6). EMDM sisteminin çalışma şeklini adım adım gösteren mimariye ait özet algoritma 1 şekil 4.5'te görülmektedir.

EMDM Algoritma 1 ZY tespiti

Girdi: şüpheli dosya ← x′, **Çıktı:** ZY ya da NY
Tanımlamalar:
A: analiz aracı, **n:** analiz edilecek şüpheli dosya sayısı

1. for i ← 1 to n
2. Analiz aracı kullanarak aktivite listesi elde et $A_{liste(i)} \leftarrow A[x'(\boldsymbol{i})]$
3. $A_{liste(i)}$ kullanarak davranışları oluştur $D_{liste(i)} \leftarrow$ DavranısOlusturAlg[$A_{liste(i)}$]
4. $D_{liste(i)}$ için risk skorları hesapla $D_{skorluListe(i)} \leftarrow$ SkorAlg[$D_{liste(i)}$]
5. $D_{skorluListe(i)}$ için davranış ilişkileri hesapla $D_{İliskiSkorluListe(i)} \leftarrow$ İliskiHesapAlg[$D_{skorluListe(i)}$]
6. $D_{İliskiSkorluListe(i)}$ kullanarak özellikleri oluştur $O_{liste(i)} \leftarrow$ OzelikOlusturAlg [$D_{İliskiSkorluListe(i)}$]
7. $O_{liste(i)}$ için özellik seçimi yap $O_{SecListe(i)} \leftarrow$ OzelikSecAlg[$O_{liste(i)}$]
8. end for
9. for i ← 1 to n
10. $O_{SecListe(i)}$ için birleştirme yap $O_{BirlesikListe} \leftarrow O_{BirlesikListe}$ + OzelikBirlesAlg [$O_{SecListe(i)}$]
11. $O_{BirlesikListe}$ için sınıflandırma yaparak x′ leri ZY ya da NY olarak belirle
12. end for

Şekil 4.5 EMDM zararlı yazılım tespit özet algoritması

Tespit işlemi sırasında EMDM, ZY'lerde görülen ancak normal yazılım örneklerinde görülmeyen ya da nadiren görülen zararlı davranış kalıplarını belirlemektedir. Davranış kalıpları belirlenirken her davranış için skor ataması yapılmaktadır. Skor atamaları yapılırken yapılan davranışlar, davranışların türleri, davranışların aktif olup olmaması ve hangi sistem yollarında gerçekleştirildikleri dikkate alınarak hesaplamalar yapılmaktadır. Burada amaç ZY'ler tarafından sıkça tekrarlanan ve normal yazılımlar tarafından nadiren ya da hiç tekrarlamayan davranışları elde edebilmektir. Skor atamaları yapılırken en tehlikesiz grup 0 olarak belirlenmiş ve davranışın zararlı olma ihtimali arttıkça tehlike seviyeleri her seferinde 1 artırılmıştır. Davranışlar yapıldıkları yere göre 3 farklı grupta 120 farklı dosya yolunu gösterecek şekilde gruplanarak skor atamaları yapılmıştır. Örneğin, zararlı (Z) ve normal (N)

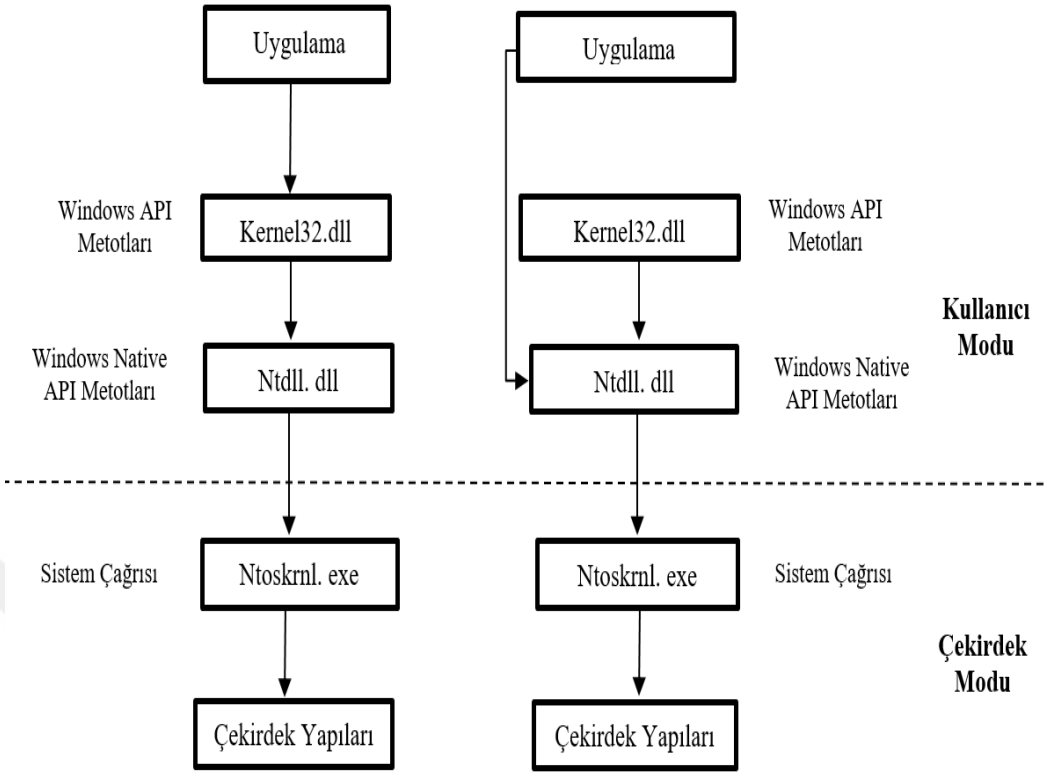
yazılım örneklerinin sistem çağrıları aynı olsa bile (gerçek örneklerde durum böyle değildir) $Z = N = \{a, b, c, d, e\}$, davranış kalıpları farklı olacaktır. $Z_{örüntü}$ $(aday) = \{ab, ac, ce\}$ burada $ab_{skor} = 4$, $ac_{skor} = 1$, $ce_{skor} = 3$; $N_{örüntü}$ $(aday) = \{ab, ac, ce, de\}$ burada $ab_{skor} = 1$, $ac_{skor} = 1$, $ce_{skor} = 1$, $de_{skor} = 3$. Eşik değeri 3 seçildiğinde $Z_{örüntü} = \{ab, ce\}$, $N_{örüntü} = \{de\}$ olmaktadır. Bu şekilde zararlı yazılımlar normal yazılımlardan ayrılmaktadır.

4.2.2 EMDM kuralları

EMDM birbiriyle ilişkili davranış ve özellikleri belirleyebilmek için belirli kurallar kullanmaktadır. Bu kurallar şöyle sıralanabilir:

1. Programın göstermiş olduğu işlemlerden davranışlar belirlenmekte;
2. Davranışlar gerçekleştirildikleri yollara göre gruplanmakta ve tehlike skorları atanmakta;
3. Davranış türüne göre tehlike skorları atanmakta;
4. Davranışlar kullanılan kaynak çeşidine göre gruplanmakta ve özellikler oluşturulurken dikkate alınmakta;
5. Aynı kaynak grubu üzerinde farklı kaynak örneği üzerinde oluşturulan davranışlardan özellik çıkarma;
6. Tekrarlanan davranışlardan özellik çıkarma;
7. Farklı kaynak üzerinde olup ilişkili olan verilerden özellik çıkarma;
8. Aktif/Pasif davranışlardan özellik çıkarma.

Kural 1. Programın göstermiş olduğu işlemlerden davranışlar belirlenmekte: Bu aşamada Windows API ve Windows Native API sistem çağrıları daha yüksek seviyeli işlemlere dönüştürülüp ilişkilendirilerek davranışlar oluşturulmaktadır (Şekil 4.4 Blok 3-4). Şekil 4.6 Windows API ve direk Windows Native API sistem çağrılarının işletim sistemiyle olan etkileşimini göstermektedir (Russinovich ve Solomon 2004).



Şekil 4.6 Windows API ve direk Windows Native API kullanımı

Örneğin, programın göstermiş olduğu işlem sırası **NtCreateFile**, **NtWriteFile** ve **NtCloseFile** olsun. Bu durumda bir dosya oluşturulmuş ve bu dosyaya bazı veriler yazılmıştır. Yapılan asıl işlem yazma olduğu için bu üç işlem grubu **WriteFile** davranışını oluşturmaktadır. Benzer şekilde sistem çağrı dizilimi **NtCreateFile**, **NtQueryFile** ve **NtCloseFile** olan işlem grubu **SearchFile** davranışını oluşturmaktadır çünkü belirli bir dizinde rastgele tarama yapılarak dosyalar aranmıştır. Bu şekilde düşük seviyeli olarak verilen sistem çağrıları daha yüksek seviyeli davranışlara dönüştürülmüştür. Çizelge 4.3 örnek bir programın sistem çağrı dizilimini ve çizelge 4.4 buna karşılık oluşturulan davranışları göstermektedir (Aslan vd. 2020). Bu şekilde hem programı tanımlayan davranışlar netleşmiş olmakta hem de özellik çıkarımı öncesi analiz edilecek veri yaklaşık olarak en az yarıya indirilmektedir.

Çizelge 4.3 Örnek bir zararlı yazılım çalıştırıldığında göstermiş olduğu aktivitelerin listesi (Liste kısaltılarak verilmiştir)

Sıra	Aktiviteler listesi
1	NtCreateFile("c:\...\kdosya$_1$.exe", zararlı.exe→ 2)
2	NtCreateFile("c:\programfiles\...\", zararlı.exe→ 3)
3	NtQueryDirectory("c:\programfiles\...\", zararlı.exe→ 4)
4	NtReadFile (4, "bdosya$_1$.txt", zararlı.exe→ 5)
5	NtReadFile (4, "bdosya$_2$.exe", zararlı.exe→ 6)
6	NtCloseFile(5,"bdosya$_1$.txt", zararlı.exe→ 7)
7	NtWriteFile (2, "kdosya$_1$.exe", zararlı.exe→ 8)
8	NtReadFile (8, "kdosya$_1$.exe", zararlı.exe→ 9)
9	NtWriteFile (6, "bdosya$_2$.exe", kdosya$_1$.exe→10)
10	NtCreateKey("hklm\software\...\...\anahtar$_1$", bdosya$_2$.exe→ 11)
11	NtSetValue(11, "anahtar$_1$", bdosya$_2$.exe→ 12)
12	NtRegCloseKey(12, "anahtar$_1$", bdosya$_2$.exe →13)
13	NtCreateFile("c:\windows\...\sdosya$_1$.dll", bdosya$_2$.exe→ 14)
14	NtCreateFile("c:\windows\...\sdosya$_2$.dll", bdosya$_2$.exe→15)
15	NtCloseFile(8,"kdosya$_1$.exe", zararlı.exe→16)
16	NtDeleteFile(8,"kdosya$_1$.exe", zararlı.exe→17)
17	NtDeleteFile("zararlı.exe", bdosya$_2$.exe →18)
18	NtReadFile(14, "sdosya$_1$.dll", bdosya$_2$.exe→19)
19	NtReadFile(14, "sdosya$_1$.dll", bdosya$_2$.exe→20)
20	NtReadFile(15, "sdosya$_2$.dll", bdosya$_2$.exe→21)
21	NtCloseFile(14,"sdosya$_1$.dll", bdosya$_2$.exe →22)
22	NtCloseFile(15,"sdosya$_2$.dll", bdosya$_2$.exe →23)

Çizelge 4.4 Çıkarılan zararlı yazılım davranışları (Liste kısaltılarak verilmiştir)

Sıra	Davranışlar listesi
1	CreateFile("c:\...\kdosya$_1$.exe", zararlı.exe→ 2)
2	SearchDirectory("c:\programfiles\...\", zararlı.exe→ 4)
3	ReadFile (4, "bdosya$_1$.txt", zararlı.exe→ 5)
4	ReadFile (4, "bdosya$_2$.exe", zararlı.exe→ 6)
5	WriteFile (2, "kdosya$_1$.exe", zararlı.exe→ 8)
6	ReadFile (8, "kdosya$_1$.exe", zararlı.exe→ 9)
7	WriteFile (6, "bdosya$_2$.exe", kdosya$_1$.exe→10)
8	SetValue(11, "anahtar$_1$", bdosya$_2$.exe→ 12)
9	DeleteFile(8,"kdosya$_1$.exe", zararlı.exe→17)
10	DeleteFile("zararlı.exe", bdosya$_2$.exe →18)
11	ReadFile(14, "sdosya$_1$.dll", bdosya$_2$.exe→19)
12	ReadFile(14, "sdosya$_1$.dll", bdosya$_2$.exe→20)
13	ReadFile(15, sdosya$_2$.dll", bdosya$_2$.exe→21)

Davranışlar oluşturulurken kullanılan EMDM algoritma 2 şekil 4.7'de gösterilmektedir.

EMDM Algoritma 2 Davranış Oluşturma

Girdi: dosya$_1$: aktiviteler dizisi; **Çıktı**: dosya$_2$: davranışlar dizisi

Tanımlamalar:

d$_1$:verilerin okunduğu dosya; **d$_2$**:verilerin yazıldığı dosya; **u**:uzunluk; **AE**:aktif aktiviteler:SetBasicInformationFile, RegDeleteKey, WriteFile, vb.; **PE**:pasif aktiviteler:ReadFile, QueryFile, vb.; **ψ**:aktivite durumu; **A**:aktif aktivite; **P**:pasif aktivite; **p**:proses; **µ**:aktivite türü; **k**:self dosya; **ED**:aktivite tür durumu; **s**:sistem dosyası; **u**:üçüncü parti dosya; **o**:aktivite değeri; **S**:sonuç; **pT**:dosya yolu; **sy**:c,hklm, "c:\boot', 'c:\windows\system32", vb.; **uy**: "c:\program files", "c:\users, hkcu\software", vb.; **rcK:**RegCloseKey; **cF**:CloseFile; **tE**:Thread Exit; **pE**:Process Exit

```
1.   d1 ← dosya1, d2 ← dosya2, n ← u(d1)
2.   for i← 1 to n   # 1'den n'e kadar olan aktiviterin durum bilgilerini belirle
3.       if (d1[i][durum] == 'AE')  # durum bilgisini kontrol et
4.           ψ ← A  # aktivite durum bilgisini aktif olarak belirle
5.       else
6.           ψ ← P  # aktivite durum bilgisini pasif olarak belirle
7.       end if
8.       if (p.ad == d1. dosyaAd)   # aktiviteyi gerçekleştiren prosesi kontrol et
9.           µ ← k   # aktivite türünü self olarak belirle
10.      elif (ED=='sy')  # aktivite tür durumu sistem mi kontrol et
11.          µ ← s   # aktivite türünü sistem olarak belirle
12.      elif (ED=='uy')  # aktivite tür durumu üçüncü parti mi kontrol et
13.          µ ← u   # aktivite türünü üçüncü parti olarak belirle
14.      else
15.        1 ← 1
16.      end if
17.      if (d1[i-1] [o] != d1[i][o])   # hariç tutulacak aktiviteleri belirle
18.          if (d1[i][o] !=rcK && d1[i][o] !=cF && d1[i][o] !=tE &&  d1[i][o] !=pE )
19.              if(d1[i-1] [S] == d1[i][S])
20.                  yaz. d2()   # oluşturulan davranışları dosyaya yaz
21.              end if
22.          end if
23.      end if
24.      if (d1[i-1] [o]== d1[i][o])
25.          if (d1[i-1] [pT] != d1[i][pT] )  # hariç tutulacak aktiviteleri belirle
26.              if (d1[i][o] != rcK&& d1[i][o] !=cF && d1[i][o] !=tE && d1[i][o] !=pE)
27.                  yaz. d2()   # oluşturulan davranışları dosyaya yaz
28.              end if
29.          end if
30.      end if
31.    end for
```

Şekil 4.7 Davranış oluşturma algoritması

Kural 2. Davranışlar gerçekleştirildikleri yollara göre gruplanmakta ve tehlike skorları atanmakta (Şekil 4.4 Blok 5.1): Önceki aşamada belirlenen davranışlar gruplandırılarak tehlike seviyeleri belirlenmektedir. Tehlike seviyeleri belirlenirken kullanılan grup türleri ve risk skor parametreleri çizelge 4.5'te görülmektedir. Tehlike seviyeleri belirlenirken davranışların hangi sistem yollarında gerçekleştirildiği büyük önem arz etmektedir. Çünkü bazı sistem yolları ZY'ler tafından daha fazla kullanılmakta ve normal yazılımlar tarafından daha az kullanılmaktadır. Bundan dolayı skor atamaları yapılırken bu yolların ZY davranışlarında bulunma sıklıkları göz önüne alınarak gerçekleştirilmektedir. Söz konusu üç grup şu şekildedir:

- Analiz edilen programın kendi dizininde ve kendine yönelik oluşturduğu davranışlar (K);
- Üçüncü parti yazılımları üzerinde oluşturduğu davranışlar (B);
- Sistem yazılımları üzerinde oluşturduğu davranışlar (S).

K ilgili yazılımın kendi dosya yolunda, B üçüncü parti yazılım yollarında ve S sistem yolunda gerçekleştirdiği davranışları göstermektedir. Bu bölümde risk skoru her davranış yolu için ayrı ayrı hesaplanmaktadır. Windows işletim sistemi için bu yollar 120 farklı dosya yolunu gösterek şekilde ayrılmış ve hesaplamalar bu yollara göre yapılmıştır.

Çizelge 4.5 Risk skoru hesaplanırken kullanılan grup türleri ve parametreleri

Grup türü	**Risk skor parametresi**
Sistem yoluna göre	Kendi sistem yolu
	Üçüncü parti sistem yolları
	İşletim sistemi yolları
	Geçici (temp) sistem yolları
	Registry otomatik başlama konumları
Davranış türüne göre	AdjustTokenPrivileges, CreateProcess, ReadFile, ReadProcessMemory, RegSetInfoKey, RegSetValue, WriteFile, vb.
İşlem türüne göre	Ağ, dosya, hafıza, mutexs, proses, registry, thread, vb.
Aktif davranışlar	Create, delete, load, set, start, write, vb.
Pasif davranışlar	Connect, enum, open, query, read, vb.

Risk skoru 0 ile 4 arasında numaralandırılmaktadır öyle ki burada 0, ilgili davranış yolunun normal olduğu ve hem zararlı hem de normal yazılımlarda görülebileceğini anlatırken, 4 ilgili davranış yolunun riskli olduğunu ve ZY'lerde sıklıkla görüldüğünü ve normal yazılımlarda nadiren görüldüğünü anlatmaktadır. Skor atamaları yapılırken en tehlikesiz görülen davranışlar için skor 0 olarak belirlenmekte ve sistem yollarının tehlike seviyeleri arttıkça tehlike skorları her seferde 1 artırılmaktadır. Örneğin, bir programın kendi dizininde oluşturduğu davranışların dosya yolu risk skoru 0 olarak atanmaktadır. Benzer şekilde özel dosya yolları için "c:\docs" gibi risk skoru 1, geçici dosya yolları için "c:\users\kullanıcıadı\appdata\local\temp" gibi risk skoru 2 ve registry başlangıç konumları için "hklm\software\microsoft\windows\currentversion\runonce" gibi risk skoru 3 olarak atanmaktadır. Davranış türüne göre skor atamaları kural 3'te, aktif ve pasif davranışlara göre skor atamaları ise kural 8'de açıklanmıştır. Davranışlar hariç tutulurken belirli bir eşik değeri kullanılmaktadır. Örneğin, X_i özelliği y_1, y_2,...,y_n davranışlarından oluşmuş olsun. X_i özelliğinin dosya yol risk skoru (*DYRS*) şöyle hesaplanır:

$$X_i\,(DYRS) = (y_1\,(DYRS) + y_2\,(DYRS) + \ldots + y_n\,(DYRS)) \,/\, n \qquad (4.6)$$

Belirtilen eşik değeri *a* olsun, bu durumda X_i $(DYRS) \geq a$, *Xi* özellik kümesinde; aksi takdirde özellik kümesinde değildir.

Kural 2.1 Analiz edilen programın kendine yönelik oluşturduğu davranışlar (K): Program sürekli bulunduğu dizine yönelik davranışlar oluşturuyorsa bu davranışlar tehlikesiz olarak belirlenmekte ve en düşük risk skoru olan 0 atanmaktadır. Çünkü bu durumda program düzgün çalışabilmek için kendi dosyasından bazı verileri diskten okuması gerekmektedir ve bundan dolayı bazı davranışlar oluşturmaktadır. Eğer program kendi kendine bazı dosya ve registry yollarına yönelik davranışlar sergiliyorsa bu davranış grubu biraz daha tehlikeli sayılmakta ve risk skoru olan 1 atanmaktadır. Örneğin, başka bir dosya oluşturup kendi verilerini bu dosyaya kopyalaması bir önceki davranışa göre daha tehlikeli bir davranıştır. Özellik seçimi sırasında yüksek risk skoruna sahip özellikler öncelikli olarak seçilmektedir. Bu durumda risk skoru 1 olan özelliğin 0 olan özelliğe göre seçilme olasılığı çok daha yüksektir.

Kural 2.2 Üçüncü parti yazılımları üzerinde oluşturduğu davranışlar (B): Birçok program düzgün çalışabilmek için bazı üçüncü parti yazılımlara ihtiyaç duymaktadır. Örneğin, "Python" diliyle yazılmış bir programın derlenip çalıştırılabilmesi için sık sık bu dilin olduğu dosya yoluna yönelik davranışlar gerçekleştirmektedir. Bu tür davranışlar bizce zararsız kabul edildiği için risk skoru 0 olarak atanmaktadır. Diğer taraftan kendisiyle direk ilişkisi olmayan dizin ve dosya yollarına yönelik gerçekleştirilen davranışlar için tehlike seviyeleri gerçekleştirildiği yola göre 1, 2 ya da 3 olarak atanmakdır.

Kural 2.3 Sistem yazılımları üzerinde oluşturduğu davranışlar (S): Programların doğru çalışabilmeleri için işletim sistemiyle etkileşime girmeleri gerekmektedir. Genelde bu etkileşim WİS'de sistem DLL'leri, arka plan işlemleri, Windows servisleri, vb. proseslerle sağlanmaktadır. Bu etkileşimlerin çoğu normal olarak kabul edilirken bazıları zararlı kabul edilmektedir. Eğer bir program işletim sisteminde doğru çalışabilmek için bazı etkileşimlere ihtiyaç duyuyorsa bundan dolayı bazı sistem yollarına dair davranışlar oluşturuyorsa bu tür yollara ait tehlike skoru 0 olarak atanmaktadır. Windows ortamında çalışırken çağrılan DLL'lerin dosya yolları bu gruba girmektedir. Eğer program prefetch ve SysWOW64 gibi sistem dosya yolarına erişmeye çalışıyorsa tehlike skoru 1 atanmaktadır. Eğer program sistem proseslerinin ("svchost. exe", "winlogon.exe", vb.) olduğu sistem yollarına sık sık erişiyorsa, tehlike seviyesi 2 olarak atanmakta ve farklı sistem yollarında sistem dosyalarıyla aynı isimde

“svchost. exe”, “winlogon. exe”, “smss. exe”, vb. dosya oluşturulmuşsa tehlike skoru 3 olarak atanmaktadır. Eğer pogram bir süre çalıştıktan sonra kendini sistem proseslerinin yer aldığı dosya yolunda her hangi bir prosese enjekte etmişse bu durumda tehlike skoru 4 olarak atanmaktadır. "Process Monitor" gibi araçlar kullanılarak başka prosesler altında çalışan programlar görüntülenebilmektedir.

Ayrıca, aşağıda gösterilen dosya yolları ZY’ler tarafından sıkça kullanıldığından dolayı bu ve benzeri yollar için tehlike skoru 2 olarak atanmaktadır:

- "c:\users\kullanıcıadı\appdata\local\temp";
- "c:\documents\settings\kullanıcıadı\localsettings\temp";
- "c:\users\kullanıcıadı\appdata\roaming\...\startmenu\programs\startup";
- "c:\programdata\microsoft\windows\startmenu\programs\startup"; vb.

Windows kayıt defteri otomatik konumlarına yerleşerek sistem her başlatıldığında kendini otomatik başlatıyorsa, bu durumda tehlike seviyesi 3 olarak atanmaktadır (Aslan vd. 2020). Windows otomatik başlama noktaları şöyle sıralanabilir:

- "hklm\software\microsoft\windows\currentversion\run";
- "hklm\software\microsoft\windows\currentversion\runonce";
- "hkey_local_machıne\software\microsoft\windows\currentversion\run"; vb.

Kural 3. Davranış türüne göre tehlike skorları atanmakta (Şekil 4.4 Blok 5.2): Sistem çağrıları sırasında kullanılan DLL ve metotların kullanım türü dikkate alınarak davranışlar için tehlike skorları belirlenmektedir. Windows tarafından sıkça kullanılan DLL ve metotların listesi çizelge 4.6 ve çizelge 4.7’de gösterilmektedir. Program hem zararlı hem de normal davranışlarda kullanılabilen “Gdı32.dll” ve “Shell32.dll” (Skorski ve Honig 2012) gibi DLL’leri kullanıyorsa tehlike skoru 0 olarak atanmaktadır. Program “User32.dll” ve “kernel32.dll” gibi DLL’leri kullanıyorsa tehlike skoru 1 olarak atanmaktadır çünkü bu iki DLL ZY’ler tarafında daha sık normal yazılımlar tarafından ise daha az kullanılmaktadır. Eğer program sık sık “Wininet.dll” ve “Advapi32.dll” kullanıyorsa tehlike skoru 2, “kernel32.dll” yerine direk “Ntdll.dll” kullanıyorsa tehlike skoru 3 ve “ReadProcessMemory” ve “AdjustTokenPrivileges” (Çizelge 4.7) gibi risk seviyesi yüksek metotlar kullanıyorsa tehlike skoru 3 olarak belirlenmektedir. Ayrıca, program sistem

proseslerine müdahale ediyorsa, içeriklerine erişmeye çalışıyorsa ya da bu prosesleri kullanarak kritik bilgilerin olduğu sistem veri tabanlarına erişmeye çalışıyorsa bu davranışlar da zararlı kabul edilmekte ve tehlike skoru olarak 4 atanmaktadır.

Çizelge 4.6 Çok kullanılan Windows DLL'leri

DLL	Açıklama
Advapi32.dll	Gelişmiş çekirdek Windows bileşenlerine hizmet yöneticisi ve kayıt defteri gibi erişim sağlar.
Gdi32.dll	Grafiklerin görüntülenmesi ve değiştirilmesiyle ilgili işlevler içerir.
Kernel32.dll	En çok kullanılan Dll'lerden biri olup dosya ve donanıma erişim ve değiştirmeyle ilgili metotlar sunar.
Ntdll.dll	Windows çekirdeğinin arabirimidir. Yürütülebilir dosyalar genellikle direk olarak bu DLL'i ithal (import) etmek yerine Kernel32'yi kullanarak import ederler. Bu sayede normalde Windows kullanıcılarının erişemeyeceği çekirdek (kernel) bölgelerine bu DLL kullanılarak erişilir.
Shell32.dll	Windows Shell API metotlarını içerir. Bu metotlar web sayfaları ve dosyalar açılırken kullanılır.
User32.dll	Tüm kullanıcı arabirim bileşenlerini içerir.
Wininet.dll	Üst düzey ağ işlevlerini gerçekleştiren metotların protokollerini içerir FTP, HTTP, NTP, vb.
WSock32.dll ve Ws2_32.dll	Bunlar ağ DLL'leridir. Bunlardan birine erişen bir program büyük olasılıkla bir ağa bağlanır veya ağla ilgili işlemler yapar.

Çizelge 4.7 Windows tarafından yaygın olarak kullanılan metotlar

Metot	Açıklama
AdjustTokenPrivileges	Belirli erişim ayrıcalıklarını etkinleştirmek veya devre dışı bırakmak için kullanılır. İşlemlere kendini gizleyen ZY'ler bu metodu kullanarak daha fazla erişim hakkı elde eder (Sikorski ve Honig 2012).
CertOpenSystemStore	Yerel sistemde saklanan sertifikalara erişmek için kullanılır.
CreateFile	Yeni bir dosya oluşturur veya mevcut bir dosyayı açar.
CreateProcess	Yeni bir proses oluşturur.
CreateProcessWithTokenW	Yeni bir proses ve işlemin ana parçacığını oluşturur. Yeni proses, belirtilen belirteç tarafından temsil edilen kullanıcının güvenlik bağlamında çalışır. İsteğe bağlı olarak belirtilen kullanıcı için kullanıcı profilini yükleyebilir (Anonymous 2018).
CreateService	Önyükleme zamanında başlatılabilecek bir hizmet oluşturur. ZY CreateService kullanarak kernel sürücülerini yükler, kendini gizler ve sistem başlatılması sırasında otomatik çalışır.
DeviceIoControl	Kullanıcı alanından (user space) bir aygıt sürücüsüne (device driver) bir kontrol mesajı gönderir. DeviceIoControl kernel ZY'leriyle ünlüdür çünkü kullanıcı alanından kernel alanına bilgi göndermenin kolay yoludur.
DeleteUmsCompletionList	Belirtilen kullanıcı modu zamanlama (scheduling) listesini siler.
FindWindow	Masaüstünde açık bir pencere arar ve döndürür.
FindFirstFile/FindNextFile	Bir dizinde arama yapmak ve dosya sistemini belirlemek için kullanılır.
GetDC	Bir pencere veya tüm ekran için bir aygıt bağlamına bir tanıtıcı döndürür. Ekran görüntüsü alan casus yazılımlar genellikle bu metodu kullanır.
GetWindowsDirectory	Windows dizininin dosya yolunu döndürür (genellikle "C:\ Windows"). ZY bazen bu dizin yolunu kullanarak başka ZY'leri belirtilen dizine indirir.
ReadProcessMemory	Uzak bir işlem belleğini okumak için kullanılır.
İnet_addr	27.0.0.1 gibi bir IP adresini diziye dönüştürür. Dönüştürülen dizi bazen ağ imzası (network-based signature) olarak kullanılır.
InternetOpen	WinINet kullanarak üst düzey İnternet erişim fonksiyonlarını başlatır, örneğin InternetOpenUrl ve InternetReadFile, vb. InternetOpen'ı aramak, İnternet erişim işlevselliğinin başlangıcını bulmak için iyi bir yoldur. InternetOpen parametrelerinden biri bazen iyi bir ağ tabanlı imza oluşturabilir (Anonymous 2006).

Kural 4. Davranışlar kullanılan kaynak çeşidine göre gruplanmakta ve özellikler oluşturulurken dikkate alınmaktadır (Şekil 4.4 Blok 5.3): İşletim sistemi kaynakları dosya, Windows kayıt defteri, ağ, işlem, iş parçacığı, muteks ve hafıza (Çizelge 4.8) olarak gruplandırılmış olup özellik ilişkileri belirlenirken genelde aynı tür kaynaklar göz önünde bulundurularak yapılmaktadır. Davranışlardan özellikler oluşturulurken *n* sayısına göre sıra önemli olmak şartıyla her uzaklıktaki davranışlar birbirleriyle bağlantılı olmak şartıyla özellik oluşturabilmektedir. Optimum *n* sayısı 10 olarak seçilmiştir. Örneğin, Çizelge 4.4'te **ReadFile** (8, "kdosya$_1$.exe", zararlı.exe→ 9) ve **WriteFile** (6, "bdosya$_2$.exe", kdosya$_1$.exe→10) davranışları birbirleriyle doğrudan ilişkiliyken; **SetValue** (11, "anahtar$_1$", bdosya$_2$.exe→ 12) ve **ReadFile** (14, "sdosya$_1$.dll", bdosya$_2$.exe→19) birbirleriyle ilişkili değildir. Bundan dolayı **ReadFile** ve **WriteFile** özellik oluştururken **SetValue** ve **ReadFile** özellik oluşturmamaktadır.

Çizelge 4.8 Windows kaynakları ve yapılan davranışlar

Kaynak türü	**İşlemler listesi**
Dosya (File)	Dosya oluşturmak
	Dosya çalıştırmak
	Dosya silmek
	Dosyadan veri okumak
	Dosyaya yazma yapmak
	Dosya yolunu görüntülemek
	Dosya özelliklerini değiştirmek
	Dosya üzerinde değişiklik yapmak
Windows kayıt defteri (Registry)	Windows kayıt defteri değerini okumak
	Windows kayıt defteri değerini değiştirmek
	Windows kayıt defteri değerini silmek
Ağ (Network)	Başka sistemlere bilgi gönderip almak
	Ağ koruma programlarını etkisiz hale getirmek
	Ağ olaylarını takip etmek
İşlem (Process)	Çalışan bir işlemi yok etmek
	Başka işlemler üstünde değişiklikler yapmak
	Başka bir programa kendi kodlarını kopyalamak
İş parçacığı (Thread)	Thread oluşturmak, var olan threadleri kullanmak
Muteks (Mutex)	Mutex oluşturmak, mutex açmak
Hafıza (Memory)	Hafızada çalışan programların adreslerine ve içeriklerine erişmek
	Hafıza koruma bitlerini etkisiz hale getirmek

Kural 5. Aynı kaynak grubu üzerinde farklı kaynak örneği üzerinde oluşturulan davranışlardan özellik çıkarma (Şekil 4.4 Blok 5.3): Aynı kaynak üzerinde (dosya, registry, vb.) farklı dosya formatları üzerinde (exe, txt, sys, dll, vb.) yapılan davranışlar farklı özellik oluşturmakta, aynı dosya formatı üzerinde yapılan davranışlar aynı özelliği oluşturmaktadır. **ReadFile** (4, "bdosya$_1$.txt", zararlı.exe→ 5) ve **ReadFile** (4, "bdosya$_2$.exe",

zararlı.exe→ 6) farklı iki özellik, **ReadFile** (14, "sdosya$_1$.dll", bdosya$_2$.exe→19) ve **ReadFile** (15, "sdosya$_2$.dll", bdosya$_2$.exe→21) aynı özelliği oluşturmaktadır (Çizelge 4.4). EMDM özellik oluşturma algoritmaları I ve II şekil 4.8 ve şekil 4.9'da görülmektedir (Aslan vd. 2020).

EMDM Algoritma 3 Özellik Oluşturma I

Girdi: dosya$_2$: davranışlar dizisi, Çıktı: dosya$_3$: özellikler dizisi I
Tanımlamalar:
rD ← svchost.exe, winlogon.exe, csrss.exe, wininnit.exe, smss.exe, vb.;
tY←"c:\users\kullanıcıadı\appdata\local","c:\users\kullanıcıadı\appdata\local\temp","c:\users\kullanıcıad\appdata\roaming", vb.; **aS**←"hklm\software\microsoft\windows\currentversion\run", "hklm\software\microsoft\windows\currentversion\runonce", vb.;
sRY←"hklm\system\currentcontrolset…\", "hkcu\jsfile\shell\..", vb.; **yP** ← dosya yolu risk skoru;
pT: dosya yolu

```
1.  d2← dosya2, d3← dosya3, n← u(d2)
2.  for i← 1 to n     # 1'den n'e kadar olan davranışlar için risk skorları hesapla
3.      if (μ == 'k')  # davranış türü self mi
4.          if (p.ad == d2. dosyaAd)    # davranışı gerçekleştiren prosesi kontrol et
5.              yP ← 0    # dosya yolu risk skoruna 0 ata
6.          elif (p.ad! = d2. dosyaAd && d2. dosyaAd == rD)   # davranış işletim sistemi dosya yolunda mı
7.              yP ← 2   # dosya yolu risk skoruna 2 ata
8.          else
9.              yP ← 1   # dosya yolu risk skoruna 1 ata
10.         end if
11.     elif (μ == 'uy')   # davranış türü üçüncü parti mi
12.         if (d2[i][pT] == tY)   # dosya yolu geçici dosyalar mı
13.             yP ← 2     # dosya yolu risk skoruna 2 ata
14.         elif (d2[i][pT] == aS)    # registry otomatik başlangıç konumlarını kontrol et
15.             yP ← 3    # dosya yolu risk skoruna 3 ata
16.         else
17.             yP ← 0   # dosya yolu risk skoruna 0 ata
18.         end if
19.     elif (μ == 'sy')  # davranış türü sistem mi
20.       if(p.ad == d2. dosyaAd)   # davranışı gerçekleştiren prosesi kontrol et
21.             yP ← 0     # dosya yolu risk skoruna 0 ata
22.       elif (d2. dosyaAd == '*.exe')   # oluşturulan dosya exe mi
23.             yP ←  3     # dosya yolu risk skoruna 3 ata
24.       elif (d2[i][pT] == sRY)    # dosya yolu özel registry girdisi mi
25.             yP ← 3    # dosya yolu risk skoruna 3 ata
26.       elif (d2[i][pT] == rD)  # davranış işletim sistemi dosya yolunda mı
27.           yP ← 4   #  dosya yolu risk skoruna 4 ata
28.       else
29.           yP ← 0   # dosya yolu risk skoruna 0 ata
30.   end if
31. end if
32. end for
```

Şekil 4.8 Özellik oluşturma algoritması I

Kural 6. Tekrarlanan davranışlardan özellik çıkarma (Şekil 4.4 Blok 5.3): Aynı kaynak üzerinde ve aynı kaynak örneği üzerinde arka arkaya olan davranışlar tek özellik olarak, farklı yerlerde ve isimde gerçekleşen davranışlar da aynı özelliğe set edilmekte ve özelliğin önem derecesi artırılmaktadır.

EMDM Algoritma 4 Özellik Oluşturma II

Girdi: dosya$_2$: davranışlar dizisi; Çıktı: dosya$_3$: özellikler dizisi II

Tanımlamalar:

dr:davranış durumu; **μ**:davranış türü; **ψ**:davranış son durum ataması; **O**,**O$_1$**,**O$_2$**,:davranış değerleri:ReadFile, WriteFile, vb.; **π**:işlem değeri; **rdF,wdF**:ReadFile, WriteFile; **pT, pT$_1$, pT$_2$**:davranışların gerçekleştirildiği dosya yolları

```
1. d2← dosya2, d3← dosya3, n←u(d2)
2. for i← 1 to n  # 1'den n'e kadar olan davranışlar için özellik hesapla
3.     if (i < n-10)   # i sayısının belirtilen aralıkta olduğunu kontrol et
4.         for j ← i+1 to i+10 # art arda gelen 10 davranış için özellik hesapla
5.            pT2 ← d2[j][pT]  # dosya yolu ataması gerçekleştir
6.            if (pi. dr == pj. dr && pi. dr == 'A') # davranışların aktiflik durumunu kontrol et
7.                ψ← 'AA' # davranışları aktif aktif olarak belirle
8.            elif (pi. dr == pj. dr && pi. dr == 'P')  # davranışların pasiflik durumunu kontrol et
9.                ψ← 'PP' # davranışları pasif pasif olarak belirle
10.           else
11.               ψ← 'AP' = 'PA' # davranışları pasif aktif olarak belirle
12.           end if
13.           if (d2[j][o] == rdF && d2[j+1] [o] == wdF) # davranışları kontrol et
14.               π ← O1+' '+O2  # işlem değeri ataması gerçekleştir
15.           if (d2[j]['μ'] == 'k')  # self mi
16.               π ← π +'K'    # özellik sonuna self olduğunu belirt
17.           elif (d2[j]['μ'] == 'uy')  # üçüncü parti mi
18.               π ← π +'U'  # özellik sonuna üçüncü parti olduğunu belirt
19.           elif (d2[j]['μ'] == 'sy') # sistem mi
20.               π ← π +'S' # özellik sonuna sistem olduğunu belirt
21.           else
22.               2← 2
23.           end if
24.           yaz. d3()  # belirlenen özellikleri dosyaya yaz
25.           if (pT1 == pT2 && O1 != O2) # davranış yolu ve değerini kontrol et
26.               π ← O1+' '+O2    # işlem değeri ataması gerçekleştir
27.           if (d2[j]['μ'] == 'k')  # self  mi
28.               π ← π +'K'   # özellik sonuna self olduğunu belirt
29.           elif (d2[j]['μ'] == 'uy')  # üçüncü parti mi
30.               π ← π +'U'   # özellik sonuna üçüncü parti olduğunu belirt
31.           elif (d2[j]['μ'] == 'sy')   # sistem mi
32.               π ← π +'S'    # özellik sonuna sistem olduğunu belirt
33.           else
34.               2← 2
35.           end if
36.           yaz. d3()  # belirlenen özellikleri dosyaya yaz
37.        end for
38.    end if
39. end for
```

Şekil 4.9 Özellik oluşturma algoritması II

Kural 7. Farklı kaynak üzerinde olup ilişkili olan davranışlardan özellik çıkarma (Şekil 4.4 Blok 5.3): Farklı kaynak üzerinde olup dolaylı olarak aralarında ilişki olduğu belirlenebilen davranışlar da özellik oluşturmaktadır. Örneğin, **WriteFile** (6, "bdosya$_2$.exe", kdosya$_1$.exe→10) ve **SetValue** (11, "anahtar$_1$", bdosya$_2$.exe→ 12) davranışları farklı kaynaklar arasında gerçekleşmiş olmasına rağmen aralarında ilişki saptandığı için bir özellik oluşturmaktadır (Aslan vd. 2020). Bu tür davranışlar tekli olarak tekrardan davranış

oluşturamazlar. Örneğin, SetValue tekli olarak yeni özellik oluşturamaz. EMDM özellik frekansı hesaplama algoritması 5 şekil 4.10'da görülmektedir.

EMDM Algoritma 5 Özellik Frekansı Hesaplama

Girdi: dosya$_3$: özellikler dizisi; **Çıktı**: dosya$_4$: frekanslar dizisi
Tanımlamalar:
dY:değer yazma; **sy**:sayaç; **scr$_1$**:skor$_1$; **scr$_2$**:skor$_2$; **scr$_3$**:skor$_3$; **scr**:skor; **pT**:aktivitenin gerçekleştirildiği dosya yolu; **u**:uzunluk; **aL**:liste; **wK**:WriteFileK, **wU**:WriteFileU, **wS**:WriteFileS, **cwK**:CreateFileWriteFileK, **cwU**:CreateFileWriteFileU, **cwS**:CreateFileWriteFileS, **rwK**:ReadFileWriteFileK, **rwU**:ReadFileWriteFileU, **rwS**:ReadFileWriteFileS, vb.

```
1.  d3←dosya3, d4←dosya4, n← u(d3), aL← [], scr1← 0, scr2← 0, scr3← 0
2.  for i← 1 to n  # 1'den n'e kadar olan özelliklerin frekanslarını hesapla
1.    sY← 1, dY← 'yanlış'   # bayraklar kontrol amaçlı
3.    if (d3[i][o] == wK || d3[i][o] == wU || d3[i][o] == wS || d3[i][o] == cwK || d3[i][o] == cwU
4.      || d3[i][o] == cwS || d3[i][o] == rwK || d3[i][o] == rwU || d3[i][o] == rwS)   # özellik durumlarını kontrol et
5.           scr1← 3 # özellik skoruna 3 ata
6.    elif (ψ == 'A') scr1← 2 # özellik skoruna 2 ata
7.    elif (ψ == 'P') scr1← 0 # özellik skoruna 0 ata
8.    elif (ψ == 'AP') scr1← 2 # özellik skoruna 2 ata
9.   elif (ψ == 'PA') scr1← 2 # özellik skoruna 2 ata
10.  elif (ψ == 'AA') scr1← 3 # özellik skoruna 3 ata
11.  elif (ψ == 'PP') scr1← 0 # özellik skoruna 0 ata
12.  else          1← 1
13.  end if
14.  for j← i+1 to n # 1'den n'e kadar
15.    if (d3[i][μ] == sy) # sistem mi
16.         if (d3[i][o] == d3[j][o] && d3[j][μ] == sy) # bir sonraki özellikle kontrol et
17.            dY← 'dogru', sY← sY+1, aL. ekle(j) # yazma bayrağını set et ve özelliği listeye ekle
18.         end if
19.      elif (d3[i][μ] == k) # self mi
20.         if (d3[i][o] == d3[j][o] && d3[j][μ] == k) # bir sonraki özellikle kontrol et
21.            dY← 'dogru', Y← sY+1, aL. ekle(j) # yazma bayrağını set et ve özelliği listeye ekle
22.         end if
23.      elif (d3[i][μ] == uy) # üçüncü parti mi
24.         if (d3[i][o] == d3[j][o] && d3[j][μ] == uy) # bir sonraki özellikle kontrol et
25.            dY← 'dogru', sY← sY+1, aL. ekle(j) # yazma bayrağını set et ve özelliği listeye ekle
26.         end if
27.      else
28.           1← 1
29.      end if
30.      if (dY == 'dogru') # yazma bayrağını kontrol et
31.         scr2← d3[i][pT], scr3← dosyaS, scr← (scr1+scr2+scr3)/3, yaz. d4() # özellik frekanslarını dosyaya yaz
32.      end if
33.  end for
34. end for
```

Şekil 4.10 Özellik frekansı hesaplama algoritması

Kural 8. Aktif/Pasif davranışlar (Şekil 4.4 Blok 5.4): Yapılan aktif davranışlar pasif davranışlara göre daha tehlikeli olarak değerlendirilmekte ve bunun sonucunda risk skoru daha fazla olmaktadır. Örneğin, **ReadFile** ve **QueryFile** için risk skoru 0 olarak belirlenirken **WriteFile** ve **DeleteFile** için 3 olarak belirlenmektedir.

EMDM kullanarak çizelge 4.3'ten çizelge 4.4 ZY davranışları, çizelge 4.9 ZY özellikleri ve çizelge 4.10 özellik vektörü oluşturulmuştur (Aslan vd. 2020). Çizelge 4.9 kullanılarak çizelge 4.10 elde edilirken var olan özellikler için tekrar sayıları yazılmış, değeri olmayan özellikler için ise 0 değeri yazılmıştır. Ayrıca, davranışlar gruplarına bakılarak ve tehlike skorları göz önünde bulundurularak risk seviyeleri de özelliğin bir alt özelliği olarak atanmıştır. Risk seviyeleri hesaplanırken çizelge 4.5 ve sekiz kuraldan faydalanılmıştır.

Çizelge 4.9 Çıkarılan zararlı yazılım özellikleri

Sıra numarası	Risk numarası	Özellik	İlgili kaynak
F_1	$I_1$2, A	{CreateFileK}	"c:\...\kdosya$_1$.exe"
F_2	$I_1$2, A	{WriteFileK }	"c:\...\kdosya$_1$.exe"
F_3	$I_1$2, $I_1$2,A,P	{WriteFileK, ReadFileK }	"c:\...\kdosya$_1$.exe" "c:\...\kdosya$_1$.exe"
F_4	$I_1$2, $I_2$2,P,A	{ReadFileK, WriteFileB}	"c:\...\kdosya$_1$.exe" "c:\programfiles\...\bdosya$_2$.exe"
F_5	$I_1$2,A	{DeleteFileK}	"c:\...\kdosya$_1$.exe" "c:\...\zararlı.exe"
F_6	$I_2$2,P	{SearchDirectoryB}	"c:\programfiles\...\"
F_7	$I_2$1, $I_2$2,P,P	{SearchDirectoryB, ReadFileB}	"c:\programfiles\...\" "c:\programfiles\...\bdosya$_1$.txt"
F_8	$I_2$2, $I_2$2,P,P	{SearchDirectoryB, ReadFileB}	"c:\programfiles\...\" "c:\programfiles\...\bdosya$_2$.exe"
F_9	$I_2$2, $I_2$2,A,A	{WriteFileB, SetValueB}	"c:\programfiles\...\bdosya$_2$.exe" "hklm\software\...\...\anahtar$_1$"
F_{10}	$I_3$1,P	{ReadFileS}	"c:\windows\...\sdosya$_1$.dll" "c:\windows\...\sdosya$_1$.dll" "c:\windows\...\sdosya$_2$.dll"

Çizelge 4.10 Özellik vektörü

Özellikler	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}
ZY_1	1 2	1 2	1 2	1 3	2 3	1 2	1 2	1 3	1 5	3 3
-	-	-	-	-	-	-	-	-	-	-
ZY_n	-	-	-	-	-	-	-	-	-	-

EMDM kullanılarak hem *n*-gram modeline göre çok daha az sayıda özellik oluşturulmuş hem de birbirleriyle daha yakın ilişkili olan özellikler belirlenmiştir. Bu durum daha sonra yapılacak olan sınıflandırma için büyük avantaj sağlamaktadır.

4.2.3 Özellik seçimi

Özellik seçimi yapılırken davranışlar ve davranışların tehlike seviyelerini gösteren risk skorları kullanılmakta olup şu adımları içermektedir:

1. Özelliği oluşturan davranışlar belirlenir;
2. Her davranış için skorlar hesaplanır:

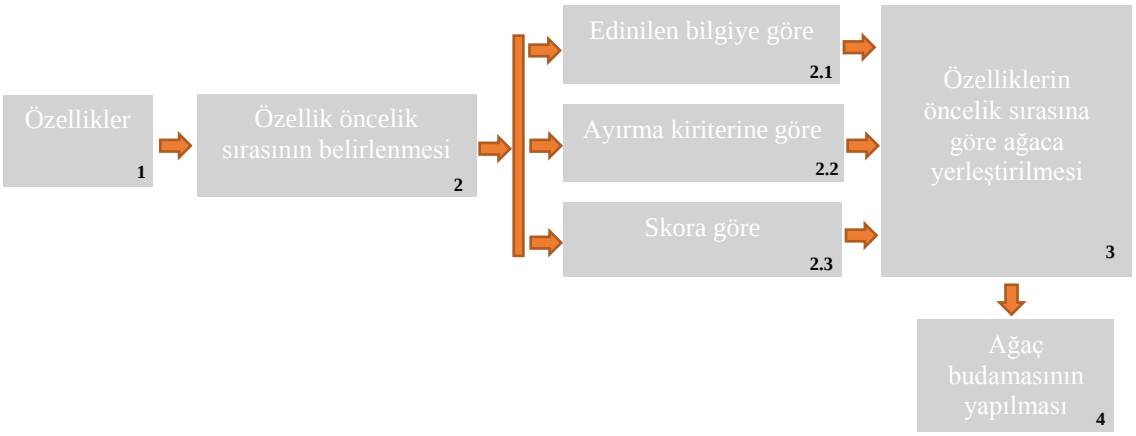
a. Dosya yolu risk skoru,
b. Davranış türü risk skoru,
c. İşlem türü risk skoru,
d. Aktif/Pasif risk skoru hesaplanır.

3. 2. adımdaki risk skorların ortalamaları alınarak son durum risk skoru elde edilir;
4. Özelliği oluşturan tüm davranışların risk skorları ve özelliğin tekrar sayısı da göz önünde bulundurularak özelliğin önem derecesi hesaplanır;
5. Özelliğin önem derecesi daha önce belirtilen eşik değeriyle karşılaştırılarak seçim yapılır.

Eşik değeri özellik sayısına ve veri setinin büyüklüğüne göre değişiklik gösterebilir. Eğer eşik değeri büyük seçilirse veri kümesi daha az özellik içerecek, daha küçük seçilirse daha fazla özellik içerecektir. Eşik değerine eşit ya da daha büyük önem derecesine sahip özellikler seçilerek ayrıştırılmaktadır. Önerilen yöntemde en iyi performans değerleri eşik değeri 3 olarak seçildiğinde elde edilmiştir. Daha sonra sınıflandırma yapılırken seçilen özelliklerin frekansları büyük önem taşıyacak olup zararlı davranış kalıpları belirlenirken belirleyici olmaktadır.

4.3 Karar Ağacı Özellik Seçim Kriterinin Çok Dallanmalı ve Farklı Ağırlıklı Dağıtılmış Olacak Şekilde İyileştirilmesi

Ağaç yapısının performansını artırmak için temelde iki noktada değişiklik yapmak gerekir: Özellik seçim kriteri ve budamanın etkili yapılması. Dallanma yapılırken hangi özellikten başlanarak bölümlemenin yapılacağı önem kazanmaktadır. Çünkü özyinelemeli olarak daha önemli görülen özelliğin daha önce seçilmesi hem ağacın dengeli dağılmasını sağlayacak hem de performansı büyük ölçüde artıracaktır. Diğer taraftan etkili bir budama daha az önemli görülebilecek özellikleri ağaçtan çıkartarak ağacın daha sade olmasını sağlayacak, arama süresini kısaltacak ve çoğu durumda performansı artıracaktır. Bundan dolayı optimizasyon karar ağacı özellik seçim kriteri ve budama üzerine temellendirilmiştir. Şekil 4.11 özellik seçimi iyileştirilmiş karar ağacı mimarisini göstermektedir. İyileştirmeler açıklenırken mimariye ait bölümler (Blok n) şeklinde belirtilmiştir.



Şekil 4.11 Özellik seçimi iyileştirilmiş karar ağacı mimarisi

4.3.1 Karar ağacı özellik seçim kriterinin iyileştirilerek ağaca yerleştirmenin yapılması

Özellikler karar ağacına yerleştirilirken özellik seçim ölçütleri kullanılmaktadır. Bilgi kazanımı (information gain), kazanç oranı (gain ratio) ve gini indeksi (gini index) en çok bilinen özellik seçim ölçütleridir. Bilgi kazanımı ve gini indeksinin çoklu değerlere sahip özellikleri seçmede taraflı (biased) davranması, kazanç oranının dengeli olmayan bölümlemeler yapmasından dolayı dengesiz ağaçlar oluşturması, performansı düşürmektedir. Bundan dolayı bu eksik yönler giderilerek ağaca yerleştirme yapılmaktadır.

İyileştirilen yöntem özellikler arasındaki ilişkiyi de kullanarak özellik öncelik sırası hesaplamakta ve bu sıralamaya göre ağaca yerleştirme yaparak sınıflandırmanın daha etkili bir biçimde yapılmasını sağlanmaktadır. Bu bölümde amaç en anlamlı özellikleri sırasıyla seçmektir. Bu aşamada izlenen adımlar şöyle sıralanabilir:

Edinilen bilgiye göre (Şekil 4.11 Blok 2.1): Veri setinin ilk özelliğinden başlanarak her özellik için bilgi kazanımı ve gini indeksleri hesaplanmaktadır.

- Bilgi kazanımı entropi kullanmakta olup karar ağacı için veri setinde en yüksek bilgi kazancına sahip özellikleri sırasıyla seçerek belirsizliği azaltmaktadır (Han vd. 2011). Bilgi kazanımı şöyle hesaplanmaktadır:

$$Kazanç\ (A) = Bilgi(D) - Bilgi_A(D) \quad (4.7)$$

$$Bilgi(D) = -\sum_{i=1}^{m} P_i\ log_2\ P_i \quad (4.8)$$

$$Bilgi_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|}\ Bilgi(D_j) \quad (4.9)$$

$Bilgi(D)$ veri setindeki sınıf etiketlerini tanımlamak için gereken ortalama bilgi miktarını, $Bilgi_A(D)$ özellikler için sınıflandırma sırasında her bölümlemeden sonra ihtiyaç duyulan bilgi miktarını ve $Kazanç\ (A)$ ilgili özellik kullanılarak dallanma yapıldığında ne kadarlık bilgi kazanımı olacağını göstermektedir. Özyinelemeli olarak bütün özellikler için bilgi kazanımı hesaplanmaktadır.

- Gini indeksi hedef özellik değerlerinin olasılık dağılımları arasındaki farklılıkları ölçmek için kullanılmakta olup karar ağacı için veri setinde minimum gini indeksine sahip özellikleri sırasıyla seçerek belirsizliği azaltmaktadır.

$$Gini(A) = Gini(D) - Gini_A(D) \quad (4.10)$$

$$Gini(D) = 1 - \sum_{i=1}^{m}(P_i)^2 \quad (4.11)$$

$$Gini_A(D) = \frac{|D_1|}{|D|} Gini(D_1) + \frac{|D_2|}{|D|} Gini(D_2) \quad (4.12)$$

Formülde $Gini(D)$ sınıf etiketlerinin olasılık dağılımlarını, $Gini_A(D)$ ise ilgili özelliğin olasılık dağılımını göstermektedir. Belirsizliği azaltmak için bilgi kazanımı ve gini indeksi değerlerinin ortalaması alınarak bilgiye dayalı özellik önem sırası hesaplanmaktadır.

Ayırma kriterine göre (Şekil 4.11 Blok 2.2): Ayırma kriterleri hesaplanırken aşağıdaki değerler ayrı ayrı hesaplanarak ortalamaları alınmaktadır.

- Her özelliğe ayırt ediciliği dikkate alınarak önem dereceleri verilmektedir. Bu aşamada özelliklerin ZY sınıfında ne kadar fazla sıklıkla ve normal yazılım sınıfında ne kadar az sıklıkla bulunduğuna bakılmaktadır. Ayrıca, özelliklerin tekrar sayılarına da bakılarak ZY sınıfında bulunan özelliklere daha yüksek değerler verilmek şartıyla özelliklere ayırt edici özellik değerleri verilmektedir.
- Aktif özellikler ZY'ler tarafından daha çok kullanıldığı için bu özellikler daha yüksek katsayılarla çarpılarak her özellik için aktiflik-pasiflik değeri elde edilmektedir.

Ayırma kriterine göre elde edilen değerlerin ortalaması alınarak her özellik için ayırma kriteri değeri elde edilmektedir.

Skora göre (Şekil 4.11 Blok 2.3): EMDM bölümünde her özellik için hesaplanan tehlike skorları bu bölümde özellik skoru olarak kullanılmaktadır.

Daha önce hesaplanan 1 (bilgiye dayalı önem sırası), 2 (ayırma kriteri değeri) ve 3 (özellik skoru) değerlerinin ortalaması alınarak özelliklerin öncelik sıraları belirlenmektedir. Öncelik sıraları karşılaştırılarak en yüksek değere sahip özellikler sırasıyla ağaca yerleştirilmektedir (Şekil 4.11 Blok 3).

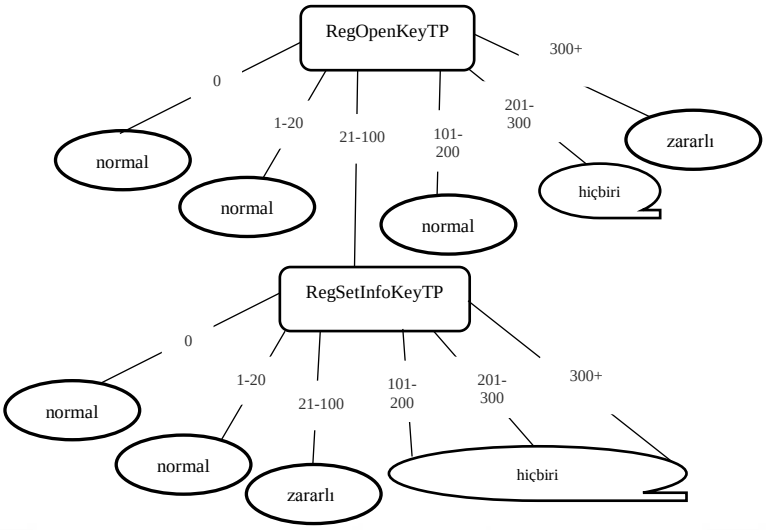
Hesaplama sırasında çok sayıda ölçüt kullanıldığı için önyargılı seçim azaltılmakta, aşırı öğrenme olmamakta ve ağaç dengeli büyümektedir. Fakat fazla hesaplama yapıldığı için ağacın zaman karmaşıklığı artmaktadır.

4.3.2 Etkin karar ağacı budamasının yapılması

Özellikler ağaca yerleştirilirken gürültü ve aykırı özelliklerden dolayı ağaçta anomaliler oluşmakta bu durumda aşırı öğrenmeye sebep vererek test verisi üzerinde hata oranını artırmakta ve DR'yi düşürmektedir. Ağaç budamasında amaç ağaçtaki anomalileri varsa tekrarlayan özellikleri ağaçtan çıkartarak yani budayarak ağacı daha anlaşılır hale getirmek ve sınıflandırmayı hızlandırmaktır. Bu iyileştirme sürecinde karar ağacı sonra budama yöntemini kullanmakta ve ağaç tamamen oluşturulduktan sonra en alt düğümlerden başlanarak yeteri derecede iyi görülmeyen dallar budanarak ağaçtan silinmekte ve alt ağaç yeniden bir sınıfa dahil edilmektedir (Şekil 4.11 Blok 4). Özellikler budanırken frekans sayısının sürekli 0 olarak tekrarlanan dallarda budama yapılarak alt ağaçlar yerine en fazla etikete sahip sınıf olan normal yazılım yazılmaktadır. Frekans sayısının yüksek tekrarlandığı yerlerde 300 ve daha yüksek değerlerde, ayrıca eğitim verisinde boş kalan dallar içinde budama yapılıp sınıf etiketi olarak ZY yazılmaktadır. Veri seti çizelge 4.11'de görülmektedir. İyileştirmeler yapıldıktan sonra oluşturulan farklı ağırlıklı dağıtılmış ağaç şekil 4.12'de gösterilmektedir.

Çizelge 4.11 Veri seti ve özellikler (Liste kısaltılarak verilmiştir)

Sınıf, isim, RegOpenKeyTP, RegQueryValueTP, RegSetInfoKeyTP
ZY, f2ec3cbe4d3840b9b11d3b4052ee2dc7.exe, 760, 0, 508
NY, cmd. exe, 15, 14, 0
ZY, f2f72360bada04cb04a148334fb9b4f0.exe, 67, 48, 48
NY, calc. exe, 62, 59, 9
ZY, f3a61848058d68097e7948cc3662963f.exe, 546, 701, 305
NY, notepad. exe,31,48,4
ZY, f3ab8adddce6730b1ee494e59ca88d70.exe, 321, 312, 213
NY, services. exe, 103,84,0
ZY, f3aa954ad390fc6be0be4c89120138e0.exe, 498, 557, 342
NY, taskhost. exe, 0, 0, 6



Şekil 4.12 Çok dallanmalı ve farklı ağırlıklı dağıtılmış karar ağacı

Verilen dört özellikten isim, RegOpenKeyTP, RegQueryValueTP, RegSetInfoKeyTP sadece ikisi kullanılarak sınıflandırma doğru bir şekilde yapılmaktadır. Burada önemli nokta hangi özelliğin önce seçileceğine karar vermek ve dallanmalar yapılırken her seferinde altı farklı gruba {0}, {1-20}, {21-100}, {101-200}, {201-300}, {300+} ayırarak ağırlıklı dallanma yapmaktır. Normal yazılımlarda dallanmalar ağırlıklı olarak ilk gruplarda olurken, ZY'lerde genelde ağırlıklar son gruplarda olmaktadır. Bunun asıl nedeni ZY'lerde belli davranışların sürekli tekrarlamasından dolayı özellik frekanslarının fazla olmasına buda belli davranış kalıplarının oluşmasına neden olmaktadır.

4.4 Değerlendirme

Bu bölümde ZY analiz ve tespit sistemi geliştirilmiştir. Önerilen sistem üç bölümden oluşmaktadır: BSTY yöntemi, EMDM modeli ve karar ağacı özellik yerleştirme optimizasyonu. BSTY, otomatik olarak çalışan tespit sistemlerinde tespit edilemeyen ZY'lerin analist tarafından elle nasıl analiz edileceğini belirtmektedir. Ayrıca, saldırılan sistemde oluşan hasarı tespit etmek için de kullanılmaktadır. EMDM'de ZY'ler otomatik olarak tespit edilmektedir. EMDM kullanılarak yazılımların işletim sistemleriyle olan etkileşimleri belirlenmektedir. Bu etkileşimler analiz edilen program davranışlarından oluşmakta olup ilgili yazılımın karakteristik özelliğini gösterecek şekilde özniteliklere dönüştürülmektedir. Bu öznitelikler kullanılarak ZY'ler normal yazılımlardan ayrılmaktadır.

Ayrıca, sınıflandırma öncesi özellik seçimi yapılarak model performansı ve hızı artırılmaktadır. Seçilen özellikler mevcut sınıflandırma algoritmaları kullanılarak sınıflandırılmaktadır. Ek olarak, karar ağaçları için özellik yerleştirme ve budamayle ilgili iyileştirmeler yapılarak karar ağaçlarının performansı artırılmaktadır. Özellikler ağaca yerleştirilirken ağaç dengeli ve özellikler çok dallara ayrılarak ağırlıklı bir şekilde yerleştirilmektedir. Gerekli görülmeyen dallar ağaçtan budanmakta ve alt ağaçlar yerine en fazla etikete sahip olan sınıf yazılmaktadır.

5. ÖNERİLEN ZARARLI YAZILIM TESPİT SİSTEMİNİN UYGULAMASI

Bu bölümde önerilen modelin uygulanması ve test ortamı açıklanmaktadır. Testler farklı Windows sürümleri üzerinde yapılmıştır. Bunlar: Windows 7 sanal makinesi, Windows 8 sanal makinesi, Windows 8 sanal sunucusu ve Windows 10 makinesidir. Bazı durumlarda Kali-Linux makinesi saldırgan olarak kullanılmıştır. Analiz sırasında program davranışlarını elde etmek için çoğunlukla Process Explorer ve Process Monitor araçları kullanılmıştır. Gerekli görülmesi halinde API monitor, Autoruns, Wireshark, Regshot ve UPX gibi araçlarından da faydalanılmıştır. EMDM otomatik çalışmakta olup uygulaması Python dili kullanarak yapılmıştır. Sınıflandırma sırasında Python dili kullanılmış olup Weka programından da yararlanılmıştır.

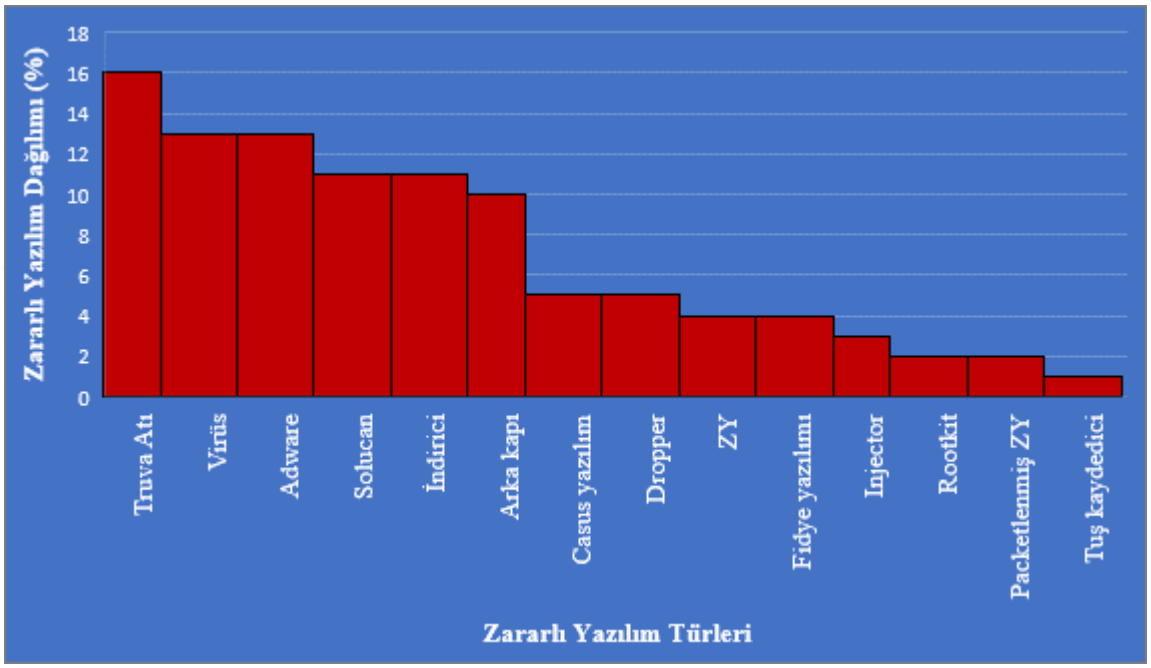
Önerilen modelin performansını değerlendirmek için EMDM kullanılarak iki farklı veri seti oluşturulmuştur. Ayrıca, *n*-gram kullanılarak da bir veri seti oluşturulmuştur. Toplamda 6700 ZY ve 3000 normal yazılım analiz edilmiştir. Bu bölüm altı alt bölümden oluşmaktadır: Veri toplama ve gösterimi; BSTY kullanılarak yazılım analizi; EMDM kullanılarak özelliklerin belirlenmesi ve zararlı davranış kalıplarının ayrıştırılması; özellik seçimi, makine öğrenmesi ve tespit; önerilen modelin performans değerlendirmesi ve bölüm değerlendirmesi.

5.1 Veri Toplama ve Gösterimi

ZY örnekleri open malware benchmark, ViruSign, malshare, tekdefense, vb. kaynaklardan toplanmıştır. Virustotal (çevrimiçi olarak yaklaşık 70 antivirüs tarayıcısı kullanmaktadır) ve 10 antivirüs tarayıcısı (Avast, AVG, ClamAV, Kaspersky, McAfee, Symantec, vb.) kullanılarak bu yazılımlar işaretlenmiştir. Analiz edilen ZY'ler farklı tür ve ailelere aittir. Analiz edilen ZY türleri virüs, Truva atı, solucan, arka kapı, rootkit, fidye yazılım, paketlenmiş yazılım, vb. şeklindedir (Şekil 5.1). Analiz edilen ZY aileleri agent, rooter, generic, ransomlock, cryptolocker, sality, snoopy, win32, CTB-Locker, vb. şeklindedir. Analiz edilen normal yazılımlar sistem araçları, oyunlar, ofis belgeleri, ses, multimedya ve diğer üçüncü parti yazılımlarıdır.

ZY'ler işaretlenirken MD5 ZY imzaları kullanılmış ve mümkün olabilecek en derin seviyeye inilerek etiketleme yapılmıştır. Örnegin, "keylogger" da casus yazılım olmasına rağmen

"keylogger" olarak işaretlenmiş, "Virüs → dropper" ya da "Trojan→ dropper" "dropper" olarak işaretlenmiştir. Az sayıda ZY tam olarak bir kategoriye dahil edilemediginden dolayı ZY olarak işaretlenmiştir. Test edilen ZY'lerin çoğunluğunu Truva atı, virüs, solucan, arka kapı ve reklam yazılımlar oluşturmaktadır (Şekil 5.1). Önerilen yöntemin performansını değerlendirmek amacıyla farklı sayıda zararlı ve normal yazılım örnekleri analiz edilmiştir.



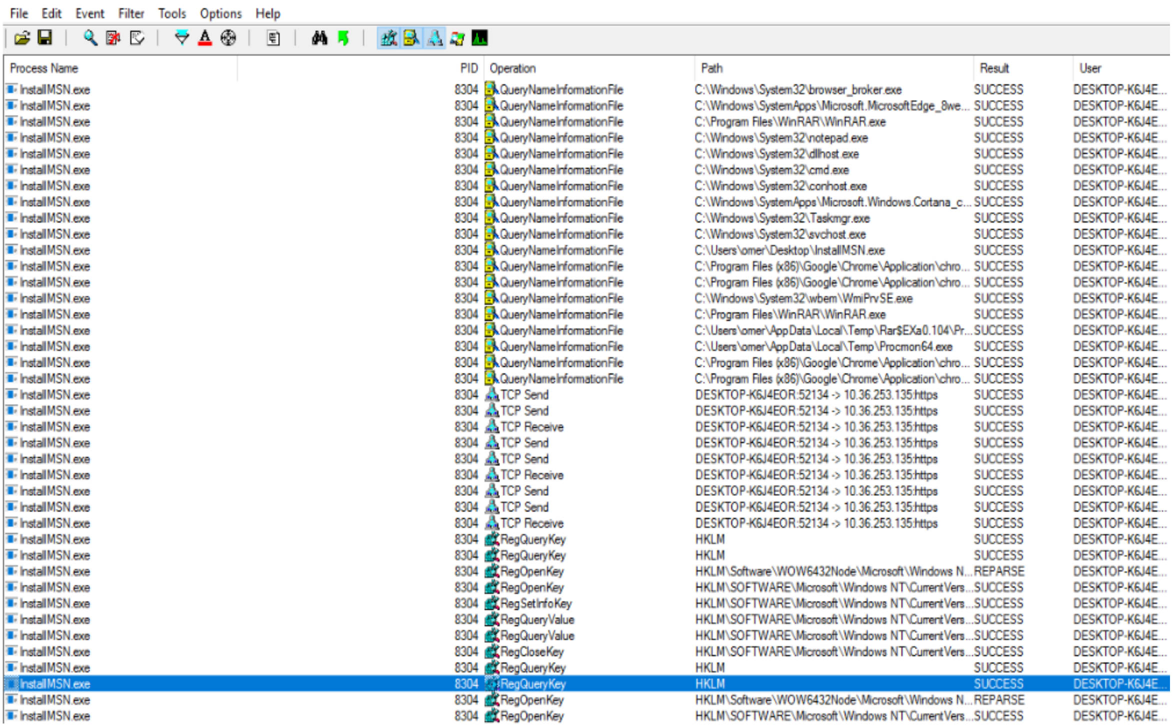
Şekil 5.1 Analiz edilen zararlı yazılımların oransal dağılımı (%)

EMDM'de çalıştırılabilir zararlı ve normal yazılım örnekleri sisteme girdi olarak verilmekte ve bir dizi özellikten oluşan bir vektör elde edilmektedir. Bu özellikler zararlı ve normal yazılımları tanımlamak için kullanılmaktadır. Her özellik, analiz edilen programın işletim sistemiyle etkileşimi sonucu oluşturduğu davranışların birleşimidir. Özellikler belirlenirken davranışlar arasındaki mantıksal bağımlılık ilişkilerine ve davranışların hangi sistem yollarında yapıldığına bakılmaktadır.

5.2 BSTY Kullanılarak Yazılım Analizi

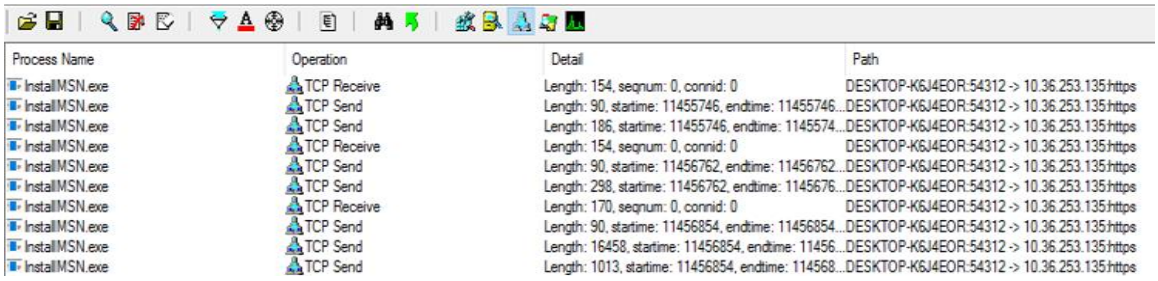
Farklı kaynaklardan toplanan yazılımlar BSTY kullanılarak analiz edilmektedir. Analiz sırasında farklı araçlar kullanılarak içeriği bilinmeyen yazılımın zararlı ya da normal olduğuna karar verilmektedir. Bu kapsamda elde edilen sonuçlar yayınlanmıştır (Aslan ve Samet 2017a).

Şekil 5.2 “Process Monitor” kullanılarak Kali Linux ortamında “Metasploit Framework” kullanılarak oluşturulan “InstallMSN.exe” arka kapısının analizini göstermektedir. “InstallMSN.exe” msfvenom –p Windows/meterpreter/reverse_tcp LHOST = 10.36.253.135 LPORT = 443 –a x86 –platform win –e x86/shikata_ga_nai –i 3 –f exe> /var/www/InstallMSN.exe komutuyla oluşturulmuştur. Analiz sırasında gerçek zamanlı olarak dosya, registry, ağ, vb. işlemler izlenmektedir.



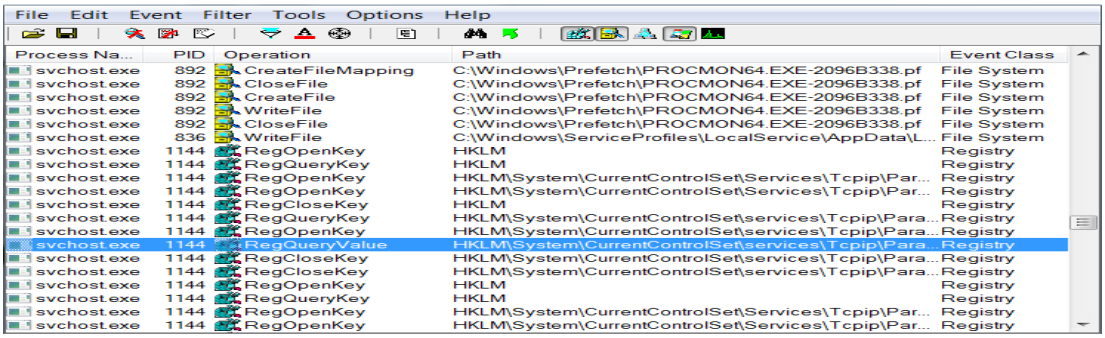
Şekil 5.2 “Process Monitor” kullanarak “InstallMSN.exe” arka kapı analizi (Aktivitelerin listesi kısaltılmıştır)

“InstallMSN.exe” çalıştırıldıktan sonra başka sisteme (10.36.253.135) uzak bağlantı açmakta ve konsol bağlantısı kullanarak saldırgan kurban sistem üzerinde komut çalıştırarak veri transferi gerçekleştirmektedir. Şekil 5.3’te her seferinde gerekli filtrelemeler yapıldıktan sonra hangi IP adresine kaç bayt veri gönderildiği görülmektedir.



Şekil 5.3 "InstallMSN.exe" arka kapısının başka sisteme (10.36.253.135) bağlanırken yaptığı ağ işlemleri (Aktivitelerin listesi kısaltılmıştır)

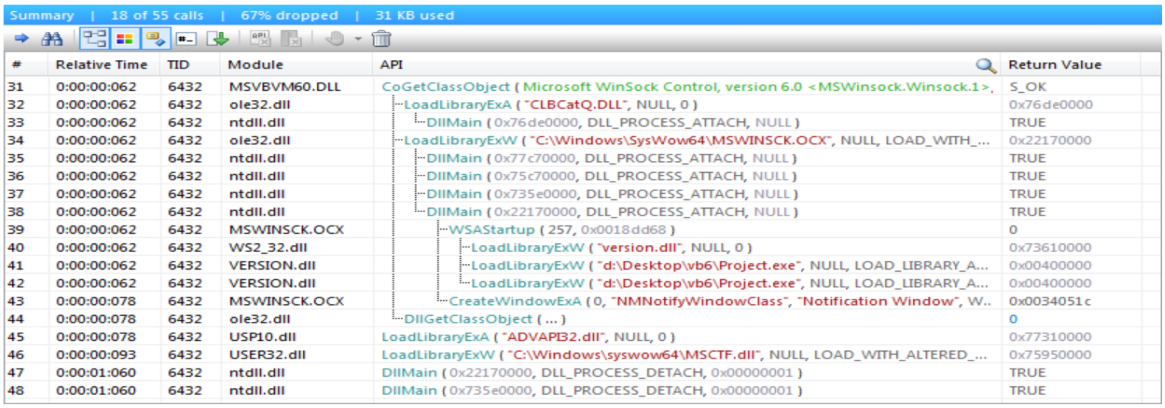
Saldırgan sisteme eriştikten sonra "meterpreter" kullanarak birçok komutu çalıştırabilmektedir. "İpconfig", "execute", "getprisv", "kill", "steal_token" ve "screenshot" kullanılabilecek komutlara örnek gösterilebilir. Saldırgan kendini gizleyebilmek ve analiz araçlarını atlatmak için "migrate PID" komutunu kullanarak kendisini "svchost.exe" sistem prosesine enjekte etmektedir. "InstallMSN.exe" kendini "svchost.exe" sistem prosesine enjekte ettikten sonra proses ismi yerinde "svchost.exe" görünmektedir (Şekil 5.4). "InstallMSN.exe" bu şekilde hem analiz araçlarından kaçmış olmakta hem de daha üst yetkiyle çalışmaktadır. Analizin doğru bir şekilde devam etmesi için proses tanımlayıcı kullanılarak (PID) hangi "svchost.exe" prosesine taşınmanın gerçekleştiği belirlenmeli ve analize ilgili "svchost.exe" ile devam edilmelidir.



Şekil 5.4 "svchost.exe" işlemleri (Aktivitelerin listesi kısaltılmıştır)

Şekil 5.5 örnek bir yazılımın gerçekleştirdiği API sistem çağrıları "API Monitor" kullanılarak gösterilmektedir. Şekilde hangi DLL ve metotların çağrıldığı görülmektedir. "API Monitor" benzeri araç çıktılarının değerlendirilebilmesi için gerekli filtrelemeler (Anonymous 2020) yapıldıktan sonra önemli görülen çıktılar ZY analisti tarafından yorumlanmalıdır. Ayrıca, önemli görülen çıktılardan özellikler belirlenip bu özelliklere

makine öğrenmesi uygulanarak otomatik çalışan tespit sistemi de oluşturulabilmektedir. Örnegin, "AdjustTokenPrivileges", "GetWindowsDirectory", "ReadProcessMemory" gibi Windows API'lerinin (Skorski 2012, Samet ve Aslan 2018) bir program tarafından çağrılması ilgili programın ZY olup olmadığı hakkında önemli ipuçları sunmaktadır. Ayrıca, araç çıktılarında bilgisayar kaynakları (dosya, registry, ağ, vb.) üzerinde yapılan değişiklikler de görüntülenebilmektedir. Tehlikeli görülen dosya ve registry girdileri silinerek veya güncellenerek ZY'nin yayılması ve zarar vermesi engellenebilmektedir.



#	Relative Time	TID	Module	API	Return Value
31	0:00:00:062	6432	MSVBVM60.DLL	CoGetClassObject (Microsoft WinSock Control, version 6.0 <MSWinsock.Winsock.1>,	S_OK
32	0:00:00:062	6432	ole32.dll	LoadLibraryExA ("CLBCatQ.DLL", NULL, 0)	0x76de0000
33	0:00:00:062	6432	ntdll.dll	DllMain (0x76de0000, DLL_PROCESS_ATTACH, NULL)	TRUE
34	0:00:00:062	6432	ole32.dll	LoadLibraryExW ("C:\Windows\SysWow64\MSWINSCK.OCX", NULL, LOAD_WITH_...	0x22170000
35	0:00:00:062	6432	ntdll.dll	DllMain (0x77c70000, DLL_PROCESS_ATTACH, NULL)	TRUE
36	0:00:00:062	6432	ntdll.dll	DllMain (0x75c70000, DLL_PROCESS_ATTACH, NULL)	TRUE
37	0:00:00:062	6432	ntdll.dll	DllMain (0x735e0000, DLL_PROCESS_ATTACH, NULL)	TRUE
38	0:00:00:062	6432	ntdll.dll	DllMain (0x22170000, DLL_PROCESS_ATTACH, NULL)	TRUE
39	0:00:00:062	6432	MSWINSCK.OCX	WSAStartup (257, 0x0018dd68)	0
40	0:00:00:062	6432	WS2_32.dll	LoadLibraryExW ("version.dll", NULL, 0)	0x73610000
41	0:00:00:062	6432	VERSION.dll	LoadLibraryExW ("d:\Desktop\vb6\Project.exe", NULL, LOAD_LIBRARY_A...	0x00400000
42	0:00:00:062	6432	VERSION.dll	LoadLibraryExW ("d:\Desktop\vb6\Project.exe", NULL, LOAD_LIBRARY_A...	0x00400000
43	0:00:00:078	6432	MSWINSCK.OCX	CreateWindowExA (0, "NMNotifyWindowClass", "Notification Window", W..	0x0034051c
44	0:00:00:078	6432	ole32.dll	DllGetClassObject (...)	0
45	0:00:00:078	6432	USP10.dll	LoadLibraryExA ("ADVAPI32.dll", NULL, 0)	0x77310000
46	0:00:00:093	6432	USER32.dll	LoadLibraryExW ("C:\Windows\syswow64\MSCTF.dll", NULL, LOAD_WITH_ALTERED_...	0x75950000
47	0:00:01:060	6432	ntdll.dll	DllMain (0x22170000, DLL_PROCESS_DETACH, 0x00000001)	TRUE
48	0:00:01:060	6432	ntdll.dll	DllMain (0x735e0000, DLL_PROCESS_DETACH, 0x00000001)	TRUE

Şekil 5.5 "API Monitor" kullanarak program analizi (API sistem çağrılarının listesi kısaltılmıştır)

5.3 EMDM Kullanılarak Özelliklerin Belirlenmesi ve Zararlı Davranış Kalıplarının Ayrıştırılması

Tespit işlemi sırasında önerilen model, ZY'lerde sıkça görülen ancak normal yazılımlarda nadiren görülen zararlı davranış kalıplarını belirlemektedir. Bunu yapmak için EMDM'deki algoritmaları kullanmaktadır. EMDM kullanılarak elde edilen sonuçlar yayınlanmıştır (Aslan vd. 2020). Zararlı davranış kalıpları belirlenirken aşağıdaki adımlar dikkate alınmaktadır:

- Örnek programın yaptığı davranışlar ve sistem yolları belirlenir;
- Her davranış için önem derecesi skoru hesaplanır;
- Belirtilen skoru geçemeyen davranışlar listeden kaldırılır;
- Davranış grupları seçilen davranışların sırasına göre belirlenir;
- Sınıflandırma seçilen davranışların frekansına göre yapılır.

Bu adımlar kullanarak zararlı davranış kalıpları belirlenmektedir öyle ki iki farklı programın işletim sistemiyle etkileşimi aynı olsa bile, davranış kalıpları farklı olmaktadır. Farklı iki

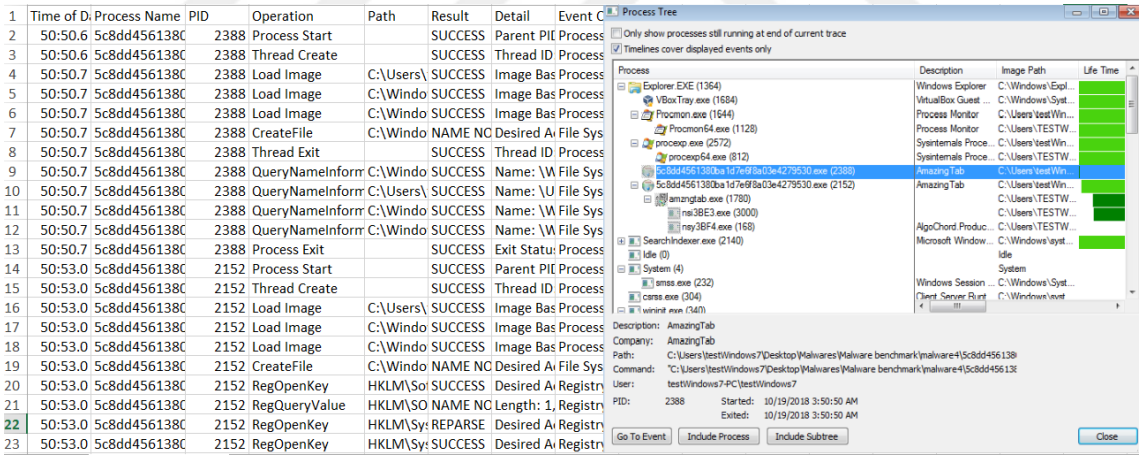
programın yapmış olduğu işlemler şöyle olsun: $P_1 = P_2 = \{a, b, c, d, e\}$. Uygun adımlar uygulandıktan sonra iki farklı program için davranış kalıpları şöyle olacaktır:

P_1 ($örüntü_{aday}$) = {ab, ac, ce} burada $ab_{skor} = 4$, $ac_{skor} = 1$, $ce_{skor} = 3$

P_2 ($örüntü_{aday}$) = {ab, ac, ce} burada $ab_{skor} = 1$, $ac_{skor} = 2$, $ce_{skor} = 1$

Eşik değeri 3 seçildiği takdirde skoru 3 ve 3'ten büyük olan aday örüntüler seçilerek davranış kalıpları oluşturulmaktadır. Bu durumda P_1 ($örüntü$) = {ab, ce}, P_2 ($örüntü$) = {}.

Şekil 5.6-5.11 MD5 imzası "5c8dd4561380ba1d7e6f8a03e4279530" olan ve Truva indiricisi (Trojan.Downloader) olarak etiketlenen ZY'nin tespit süreci aşama aşama gösterilmektedir. Şekil 5.6 Truva indiricisinin oluşturduğu proseslerin listesi (amzngtab.exe, nsi3BE3.exe ve nsy3BF4.exe) ve işletim sistemiyle etkileşimi sonucu oluşturduğu aktivitelerin listesi gösterilmektedir. İlgili ZY işletim sistemi üzerinde 15.997 aktivite oluşturmuştur. Bu aktivitelerin bir bölümünü daha sonra oluşturulan prosesler tarafından gerçekleştirilmiştir. Bundan dolayı oluşturulan proseslerin yaptıkları aktiviteler de "Process Monitor" kullanılarak analiz edilmiştir.



Şekil 5.6 "5c8dd4561380ba1d7e6f8a03e4279530" zararlı yazılım çalıştırıldığında göstermiş olduğu aktivitelerin listesi (Liste kısaltılarak verilmiştir)

Şekil 5.7 EMDM kullanarak gerçekleştirilen aktivitelerden oluşturulan davranışların listesini göstermektedir. Bu aşamada EMDM algoritmaları kullanılarak aktiviteler arasındaki ilişkiler belirlenerek davranışlar oluşturulmuştur. Şekil 5.6'deki 15.997 aktivite kullanılarak 8856 davranış elde edilmiştir.

	Process Name	Path	Operation	DirectoryFileName	Event Class	PID	TID	Parent PID	Image Pat	Result
2	5c8dd4561380ba1d7e6f8a03e4279530.exe	C:\Users\testWindows7\Desktop\Mal	ProcessStart	5c8dd4561380ba1d7e6f8a03e42	Process	2388	512	1364	C:\Users\	SUCCESS
3	5c8dd4561380ba1d7e6f8a03e4279530.exe	C:\Users\testWindows7\Desktop\Mal	ThreadCreate	5c8dd4561380ba1d7e6f8a03e42	Process	2388	512	1364	C:\Users\	SUCCESS
4	5c8dd4561380ba1d7e6f8a03e4279530.exe	C:\Users\testWindows7\Desktop\Mal	LoadImage	5c8dd4561380ba1d7e6f8a03e42	Process	2388	456	1364	C:\Users\	SUCCESS
5	5c8dd4561380ba1d7e6f8a03e4279530.exe	C:\Windows\System32\ntdll.dll	LoadImage	ntdll.dll	Process	2388	456	1364	C:\Users\	SUCCESS
6	5c8dd4561380ba1d7e6f8a03e4279530.exe	C:\Windows\SysWOW64\ntdll.dll	LoadImage	ntdll.dll	Process	2388	456	1364	C:\Users\	SUCCESS
7	5c8dd4561380ba1d7e6f8a03e4279530.exe	C:\Windows\System32\apisetschema.	QueryNameInformationFi	apisetschema.dll	File System	2388	456	1364	C:\Users\	SUCCESS
8	5c8dd4561380ba1d7e6f8a03e4279530.exe	C:\Users\testWindows7\Desktop\Mal	QueryNameInformationFi	5c8dd4561380ba1d7e6f8a03e42	File System	2388	456	1364	C:\Users\	SUCCESS
9	5c8dd4561380ba1d7e6f8a03e4279530.exe	C:\Windows\System32\ntdll.dll	QueryNameInformationFi	ntdll.dll	File System	2388	456	1364	C:\Users\	SUCCESS
10	5c8dd4561380ba1d7e6f8a03e4279530.exe	C:\Windows\SysWOW64\ntdll.dll	QueryNameInformationFi	ntdll.dll	File System	2388	456	1364	C:\Users\	SUCCESS
11	5c8dd4561380ba1d7e6f8a03e4279530.exe		ProcessStart	C:	Process	2152	2988	1364	C:\Users\	SUCCESS
12	5c8dd4561380ba1d7e6f8a03e4279530.exe		ThreadCreate	C:	Process	2152	2988	1364	C:\Users\	SUCCESS
13	5c8dd4561380ba1d7e6f8a03e4279530.exe	C:\Users\testWindows7\Desktop\Mal	LoadImage	5c8dd4561380ba1d7e6f8a03e42	Process	2152	2168	1364	C:\Users\	SUCCESS
14	5c8dd4561380ba1d7e6f8a03e4279530.exe	C:\Windows\System32\ntdll.dll	LoadImage	ntdll.dll	Process	2152	2168	1364	C:\Users\	SUCCESS
15	5c8dd4561380ba1d7e6f8a03e4279530.exe	C:\Windows\SysWOW64\ntdll.dll	LoadImage	ntdll.dll	Process	2152	2168	1364	C:\Users\	SUCCESS
16	5c8dd4561380ba1d7e6f8a03e4279530.exe	C:\Windows	CreateFile	Windows	File System	2152	2168	1364	C:\Users\	SUCCESS
17	5c8dd4561380ba1d7e6f8a03e4279530.exe	C:\Windows\System32\wow64.dll	CreateFile	wow64.dll	File System	2152	2168	1364	C:\Users\	SUCCESS
18	5c8dd4561380ba1d7e6f8a03e4279530.exe	C:\Windows\System32\wow64.dll	QueryBasicInformationFil	wow64.dll	File System	2152	2168	1364	C:\Users\	SUCCESS
19	5c8dd4561380ba1d7e6f8a03e4279530.exe	C:\Windows\System32\wow64.dll	CreateFile	wow64.dll	File System	2152	2168	1364	C:\Users\	SUCCESS
20	5c8dd4561380ba1d7e6f8a03e4279530.exe	C:\Windows\System32\wow64.dll	LoadImage	wow64.dll	Process	2152	2168	1364	C:\Users\	SUCCESS
21	5c8dd4561380ba1d7e6f8a03e4279530.exe	C:\Windows\System32\wow64win.dll	CreateFile	wow64win.dll	File System	2152	2168	1364	C:\Users\	SUCCESS

Şekil 5.7 EMDM kullanılarak çıkarılan "5c8dd4561380ba1d7e6f8a03e4279530" zararlı yazılım davranışları (Liste kısaltılarak verilmiştir)

Şekil 5.8 EMDM algoritmaları kullanılarak şekil 5.7'deki davranışlardan elde edilen özellikler ve özelliklerin skorlarını göstermektedir. Davranışlardan özellikler oluşturulurken *n* sayısına göre sıra önemli olmak şartıyla her uzaklıktaki davranışlar birbirleriyle bağlantılı olmak şartıyla özellik oluşturabilmektedir. En yüksek performans değerleri *n* = 10 seçildiğinde elde edilmiştir.

Property	Resource	Frequency	OverAll Score
ProcessStartSF	C:\Users\testWindows7\Desktop\Malwares\Ma	1	2
ProcessStartThreadCreateSF	C:\Users\testWindows7\Desktop\Malwares\Ma	1	3
ProcessStartLoadImageSF	C:\Users\testWindows7\Desktop\Malwares\Ma	1	2
LoadImageSY	C:\Windows\System32\kernel32.dll	218	1
LoadImageSY	C:\Windows\System32\user32.dll	217	1
CreateFileSY	C:\Windows	418	1
CreateFileQueryNameInformationFileSY	C:\Windows	4	1
QueryNameInformationFileSY	C:\Windows	210	1
RegOpenKeyTP	HKLM\Software\Wow6432Node\Microsoft\Win	935	1
RegSetInfoKeyTP	HKLM\SOFTWARE\MICROSOFT\WINDOWS NT\C	308	5
RegSetInfoKeySY	HKLM\System\CurrentControlSet\Control\SESSI	54	3
CreateFileTP	C:\Users\testWindows7\Desktop\Malwares\Ma	380	1
LoadImageSY	C:\Windows\SysWOW64\kernel32.dll	216	1
RegSetValueTP	HKCU\Software\Microsoft\Windows\CurrentVe	3	5
RegDeleteValueTP	HKCU\Software\Microsoft\Windows\CurrentVe	2	5
RegDeleteValueTP	HKCU\Software\Microsoft\Windows\CurrentVe	1	5
RegQueryKeySY	HKCU	308	1
RegCreateKeyTP	HKLM\SOFTWARE\Wow6432Node\Google\Chrc	1	3
CreateFileSY	C:\Windows\Microsoft.NET\Framework64\v2.0.	1	1
CreateFileTP	C:\Users\testWindows7\AppData\Roaming\Mic	1	1
RegOpenKeyTP	HKLM\Software\Microsoft\Windows NT\Curren	1	3
QueryNameInformationFileSF	C:\Users\testWindows7\AppData\Local\Temp\n	1	0
QueryNameInformationFileSY	C:\Windows\winsxs\amd64_microsoft.vc80.crt_	24	1
QueryNameInformationFileSY	C:\Windows\System32\user32.dll	23	1

Şekil 5.8 EMDM kullanılarak elde edilen "5c8dd4561380ba1d7e6f8a03e4279530" zararlı yazılım özellikleri (Liste kısaltılarak verilmiştir)

Şekil 5.9 "5c8dd4561380ba1d7e6f8a03e4279530" ZY için 4-gram kullanıldığında elde edilen özellikleri göstermektedir. 4-gram kullanılarak toplamda 1413 özellik oluşturulmuştur.

```
name,5c8dd4561380ba1d7e6f8a03e4279530.exe
ProcessStartThreadCreateLoadImageLoadImage, 6
ThreadCreateLoadImageLoadImageLoadImage, 3
LoadImageLoadImageLoadImageCreateFile, 3
LoadImageLoadImageCreateFileCreateFileThreadExit, 2
LoadImageCreateFilThreadExitQueryNameInformationFile, 1
CreateFileThreadExitQueryNameInformationFileQueryNameInformationFile, 1
ThreadExitQueryNameInformationFileQueryNameInformationFileQueryNameInformationFile, 6
QueryNameInformationFileQueryNameInformationFileQueryNameInformationFileQueryNameInformationFile, 215
QueryNameInformationFileQueryNameInformationFileQueryNameInformationFileProcessExit, 6
QueryNameInformationFileQueryNameInformationFileProcessExitProcessStart, 2
QueryNameInformationFileProcessExitProcessStartThreadCreate, 1
ProcessExitProcessStartThreadCreateLoadImage, 1
LoadImageLoadImageCreateFileRegOpenKey, 3
LoadImageCreateFileRegOpenKeyRegQueryValue, 3
CreateFileRegOpenKeyRegQueryValueRegOpenKey, 3
RegOpenKeyRegQueryValueRegOpenKeyRegOpenKey, 3
```

Şekil 5.9 4-gram kullanılarak elde edilen "5c8dd4561380ba1d7e6f8a03e4279530" zararlı yazılım özellikleri ve frekansları (Liste kısaltılarak verilmiştir)

Şekil 5.10 EMDM algoritmaları kullanılarak şekil 5.8'deki özelliklerden elde edilen özellik frekanslarını göstermektedir. MD5 imzası "5c8dd4561380ba1d7e6f8a03e4279530" olan Truva indiricisi için toplamda 250 özellik belirlenmiştir.

```
name,5c8dd4561380ba1d7e6f8a03e4279530.exe
ProcessStartSF real,1
ProcessStartThreadCreateSF real,1
ProcessStartLoadImageSF real,1
ProcessStartQueryNameInformationFileSF real,1
ThreadCreateSF real,1
ThreadCreateLoadImageSF real,1
ThreadCreateQueryNameInformationFileSF real,1
LoadImageSF real,5
LoadImageQueryNameInformationFileSF real,1
LoadImageSY real,226
LoadImageQueryNameInformationFileSY real,2
QueryNameInformationFileSY real,213
QueryNameInformationFileSF real,5
QueryNameInformationFileLoadImageSF real,1
QueryNameInformationFileLoadImageSY real,5
ProcessStartTP real,4
```

Şekil 5.10 EMDM kullanılarak elde edilen "5c8dd4561380ba1d7e6f8a03e4279530" zararlı yazılım özellikleri ve frekansları (Liste kısaltılarak verilmiştir)

5.4 Özellik Seçimi, Makine Öğrenmesi ve Tespit

Sınıflandırma yapılmadan önce önemli görülen özellikler veri setinden seçilmektedir. Bu aşamada önerilen EMDM özellik seçim algoritması kullanılmaktadır. Ayrıca, mevcut özellik seçim algoritmaları da kullanılarak önerilen algoritmanın performansı değerlendirilmektedir.

Şekil 5.11 EMDM algoritması kullanılarak seçilen özelliklerin listesini göstermektedir. EMDM kullanılarak 250 özellik arasından en önemli görülen 50 özellik seçilmektedir. Özellik seçimi sırasında en iyi performans değerleri eşik değeri 3 olarak seçildiğinde elde edilmektedir. Eşik değerinin daha büyük seçilmesi özellik sayısını çok azaltmakta ve küçük seçilmesi ise özellik sayısını artırarak performans değerlerini düşürmektedir.

```
name,5c8dd4561380ba1d7e6f8a03e4279530.exe
ProcessStartThreadCreateSF real,1
ProcessStartTP real,4
ProcessStartThreadCreateTP real,4
ThreadCreateTP real,49
RegSetInfoKeyTP real,308
RegSetInfoKeySY real,54
RegOpenKeyRegSetInfoKeySY real,12
RegSetInfoKeyRegEnumKeySY real,2
RegSetInfoKeyRegQueryKeySY real,14
RegOpenKeyRegSetInfoKeyTP real,178
RegSetInfoKeyRegOpenKeyTP real,59
RegSetInfoKeyRegQueryKeyTP real,217
RegQueryKeyRegOpenKeyTP real,50
RegQueryKeyRegSetInfoKeyTP real,21
ReadFileWriteFileSF real,29
WriteFileTP real,25
```

Şekil 5.11 EMDM kullanılarak seçilen özellikler ve frekansları (Liste kısaltılarak verilmiştir)

Özellik seçimi yapıldıktan sonra özellik vektörü oluşturulmaktadır. Özellik vektörü oluşturulurken olan davranışlar için tekrar sayıları olmayan davranışlar için 0 değeri yazılmaktadır. Özellik vektörü oluşturulduktan sonra sınıflandırma yapılmaktadır. Zararlı ve normal yazılım örneklerini sınıflandırmak için mevcut ML algoritmaları kullanılmaktadır. Bu algoritmalar BN, C4.5/J48, KNN, LMT, MLP, NB, RF, SLR ve SMO'dur. Bu kadar çok algoritma kullanılmasının sebebi önerilen EMDM performansını daha doğru değerlendirebilmektir. ML algoritmaları kullanılarak hem EMDM performansı diğer model performaslarıyla karşılaştırılmış hem de sınıflandırma algoritmalarının performansları kendi aralarında değerlendirilmiştir. Ayrıca, karar ağacı özellik seçim kriterleri iyileştirilerek karar ağaçlarının performansları kendi içinde de değerlendirilmiştir. Şekil 5.12 ilk 14 özelliği ve şekil 5.13 bu özelliklerin tekrar sayılarını göstermektedir.

```
class
name
ProcessStartThreadCreateSF
CreateFileSetBasicInformationFileSY
SetBasicInformationFileSY
SetBasicInformationFileQueryFileInternalInformationFileSY
SetBasicInformationFileFileSystemControlSY
SetBasicInformationFileQueryAttributeTagFileSY
SetBasicInformationFileCreateFileMappingSY
RegOpenKeyTP
RegQueryValueTP
RegSetInfoKeyTP
RegSetInfoKeyRegQueryKeyTP
RegSetInfoKeyRegEnumKeyTP
RegQueryKeyRegEnumKeyTP
RegEnumKeyTP

```

Şekil 5.12 EMDM kullanılarak elde edilen veri seti özellikleri (Liste kısaltılarak verilmiştir)

1	Benign	calc.exe	1	44	44	44	4	37	37	62	59	9	7	1	6	4
2	Benign	cmd.exe	1	21	21	21	0	18	18	15	14	0	0	0	0	0
3	Benign	conhost.exe	1	54	54	54	4	38	38	68	92	0	0	0	0	0
4	Benign	csrss.exe	0	0	0	0	0	0	0	512	63	432	994	0	8	8
5	Benign	DllHost.exe	1	0	0	0	0	0	0	100	58	10	4	0	0	0
6	Benign	FreeCell.exe	1	0	0	0	0	0	0	0	697	649	13	21	7	0
7	Benign	hearts.exe	1	0	0	0	0	0	0	0	493	297	13	21	7	0
8	Malware	1ce293a2be7d4cc968279d23a685183e.exe	1	0	0	0	0	0	0	0	0	391	287	10	48	32
9	Malware	2d0a147c8ea7b711c44310786e055b36.exe	1	4	4	0	0	1	0	0	0	88	44	1	6	33
10	Malware	5c8dd4561380ba1d7e6f8a03e4279530.exe	1	0	0	0	0	0	0	0	0	308	217	3	0	25
11	Malware	6afd90d3b14a9a314e93f386a23b5ed3.exe	1	0	0	0	0	0	0	863	0	528	262	4	11	21
12	Benign	mspaint.exe	1	69	69	69	4	61	61	192	96	12	8	1	6	4
13	Benign	notepad++.exe	1	0	0	0	0	0	0	0	0	80	51	9	0	0
148	Benign	WINWORD.EXE	1	0	0	0	0	0	0	0	0	39	24	1	0	0
149	Benign	wuauclt.exe	0	0	0	0	0	0	0	0	0	2	2	0	0	0
150	Benign	YandexWorking.exe	1	0	0	0	0	0	0	0	10	27	2	0	0	0
151	Malware	f2fbce27272eec683ea9865f940027e0.exe	0	0	0	0	0	0	0	2206	0	1216	700	7	0	0
152	Malware	f2fe97b9014ed5799b42e3b1722695e0.exe	1	0	0	0	0	0	0	543	558	353	171	4	0	0
153	Malware	f2fea6c855e3824aa6109b0d5655a900.exe	1	0	0	0	0	0	0	76	18	43	25	0	0	0
154	Malware	f3a6ab31986c928d312699dc7208a211.exe	1	0	0	0	0	0	0	175	279	139	61	1	0	0
155	Malware	f3a8ba5f872eba950dd2f95739ccd830.exe	1	0	0	0	0	0	0	40	17	42	3	0	0	0
156	Malware	f3a9b97f92d8434f8d29a1d249ede8b3.exe	1	0	0	0	0	0	0	93	0	102	77	0	0	0
157	Malware	f3a61848058d68097e7948cc3662963f.exe	1	0	0	0	0	0	0	546	701	305	211	1	14	12
158	Malware	f3aa954ad390fc6be0be4c89120138e0.exe	1	0	0	0	0	0	0	498	557	342	134	4	0	0
159	Malware	f3ab8adddce6730b1ee494e59ca88d70.exe	1	0	0	0	0	0	0	321	312	213	81	2	0	0

Şekil 5.13 EMDM kullanılarak elde edilen veri seti özellik frekansları vektörü (Liste kısaltılarak verilmiştir)

5.5 Önerilen Modelin Performans Değerlendirmesi

Algoritmaların performanslarını karşılaştırmak için bazı metrikler kullanılmıştır: Tespit oranı (recall-DR), FP, FN, kesinlik (precision), F-ölçümü (F-measure) ve doğruluk, vb. Bu değerler karışıklık matrisi kullanılarak hesaplanmaktadır (Çizelge 5.1).

Çizelge 5.1 Karışıklık matrisi

		Tahmin edilen sınıf	
		Evet	Hayır
Gerçek sınıf	Evet	TP	FN
	Hayır	FP	TN

Bu değerler TP (ZY'nin zararlı olarak işaretlenme sayısı), TN (Normal yazılımın normal olarak işaretlenme sayısı), FP (Normal yazılımın yanlışlıkla ZY olarak işaretlenme sayısı), FN (ZY'nin yanlışlıkla normal olarak işaretlenme sayısı) olarak gösterilmektedir. Bu değerler kullanılarak tespit oranı, FPR, kesinlik, F-ölçümü ve doğruluk oranları hesaplanmıştır.

$$\textit{Tespit oranı}\ (\textit{Detection Rate-DR}) = TP/\ (TP+FN) \quad (5.1)$$

$$\textit{Yanlış pozitif oranı}\ (\textit{False Positive Rate-FPR}) = FP/\ (FP+TN) \quad (5.2)$$

$$\textit{Kesinlik}\ (\textit{Precision}) = TP/\ (TP+FP) \quad (5.3)$$

$$\textit{F-ölçümü}\ (\textit{F-Measure}) = (2 * precision * recall)\ /\ (precision + recall) \quad (5.4)$$

$$\textit{Doğruluk}\ (\textit{Accuracy}) = TP+TN/\ (TP+TN+FP+FN) \quad (5.5)$$

Ayrıca, ML model performanslarını değerlendirmek için holdout ve çapraz doğrulama (cross validation, k-fold cross validation) kullanılmıştır.

Holdout yönteminde veri seti eğitim ve test verisi olmak üzere 2'ye ayrılmaktadır. Eğitim seti kullanılarak sistem eğitilmekte, test seti kullanılarak eğitimin ne kadar başarılı olduğu ölçülmektedir. Veri setinin büyüklüğüne göre eğitim ve test verisi ayrımı değişmektedir. Veri setinin bir defaya mahsus olarak ayrılması ve gözlem sayısının az olması durumunda ayrımın yeterince yapılamaması bu yöntemin dezavantajları arasında gösterilebilir. Çapraz doğrulama yönteminde veri seti belirlenen 'k' değerine göre eşit gruplara ayrılmaktadır. Her seferinde gruplardan biri test seti olarak kullanılırken diğerleri eğitim seti olarak kullanılmaktadır. Böylece tüm gruplar test setinde bir sefer olmak üzere kullanılmakta ve sistem bu şekilde eğitilmektedir. K değeri genelde 10 olarak seçilmektedir. Çapraz doğrulamada test grubu her seferinde değiştiğinden daha fazla hesaplama zamanı gerekmektedir. Veri setinin küçük olduğu durumlarda genellikle çapraz doğrulama tercih edilmektedir çünkü model her seferinde eğitim-test grubu ile eğitim olanağı verdiğinden dolayı daha önce bilinmeyen veriler üzerinde daha iyi performans göstermektedir. Diğer taraftan, büyük veri setlerinde

holdout yöntemi kullanışlıdır çünkü bu durumda yeterince örnek bulunduğundan dolayı sistem eğitilebilmektedir.

Bu çalışmada önerilen yöntemin performasını değerlendirmek amacıyla holdout ve çapraz doğrulama yöntemleri kullanılmıştır. Öncelikle veri seti üzerinde çapraz doğrulama uygulanmış daha sonra holdout yöntemi uygulanmıştır. Çapraz doğrulama sırasında en iyi performans değerleri $k = 10$ seçildiğinde elde edilmiştir. İlk zamanlar analiz edilen program sayısı az olduğundan dolayı çapraz doğrulamayla daha iyi sonuçlar elde edilse bile analiz edilen program sayısı arttıkça holdout yöntemiyle de iyi sonuçlar elde edilmiştir. Holdout yöntemi kullanıldığında en iyi performans değerleri verinin %75 (eğitim) ve %25 (test) olarak ayrıldığı durumda elde edilmiştir.

5.6 Değerlendirme

Bu bölümde önerilen ZY tespit sisteminin uygulaması anlatılmıştır. Zararlı ve normal yazılım örnekleri Process Explorer ve Process Monitor gibi dinamik araçlar kullanarak analiz edilmiş ve analiz çıktıları Python programlama dili kullanılarak işlenmiştir. Analizler için farklı ZY türleri ve aileleri malware benchmark, ViruSign, malshare, kernelMode, tekdefense, vb. kaynaklardan toplanmış ve Virustotal kullanılarak bu yazılımlar işaretlenmiştir. İşaretleme sırasında MD5 ZY imzaları kullanılmış ve mümkün olabilecek en derin seviyeye inilerek etiketleme yapılmıştır. Test edilen ZY'lerin çoğunluğunu Truva atı, virüs, solucan, arka kapı ve reklam yazılımları oluşturmaktadır. EMDM kullanılarak her aşamada veri setinin nasıl oluşturulduğu örnek bir ZY kullanılarak anlatılmıştır. Önerilen yöntemleri doğru şekilde değerlendirmek için tespit oranı, doğruluk oranı ve F-ölçümü gibi farklı metrikler; holdout ve çapraz doğrulama gibi yöntemler kullanılmıştır.

6. BULGULAR, TARTIŞMALAR VE DEĞERLENDİRMELER

Bu bölümde tez kapsamında geliştirilen ZY analiz ve tespit sisteminin uygulanması sonucu elde edilen test sonuçları ve bu sonuçların yorumlanması yer almaktadır. Elde edilen sonuçlar literatürde yapılan öncü çalışmalarla karşılaştırılarak tartışılmıştır. Ayrıca, elde edilen bulgular maddeler halinde sıralanmıştır.

6.1 EMDM Kullanarak Elde Edilen Sonuçlar

Çizelge 6.1 sınıflandırma algoritmalarının oluşturulan veri seti üzerindeki performanslarını göstermektedir. Elde edilen sonuçlar holdout (%75 eğitim ve %25 test) ve cross validation (k=10) ortalaması alınarak elde edilmiştir. Oluşturulan veri setinde az kayıt varken holdout yöntemiyle iyi sonuçlar elde edilememiştir. Fakat analiz edilen program sayısı arttıkça holdout yöntemiyle de iyi performans değerleri ölçülmüştür. Performanslar DR, FPR, F-ölçümü ve doğruluk değerleri göz önüne alınarak değerlendirilmiştir.

Çizelge 6.1 Sınıflandırıcıların EMDM kullanılarak oluşturulan veri seti üzerindeki performansları

Sınıflandırıcı	DR (%)	FPR (%)	F-ölçümü (%)	Doğruluk oranı (%)
C4.5/J48	99.9	0.2	99.9	99.8
LMT	99.8	0.2	99.8	99.7
RF	99.7	0.4	99.8	99.6
KNN	99.6	0.8	99.6	99.6
SLR	97.4	2	97.5	97.4
SMO	93.2	6.8	93.2	93.1
BN	89	8.6	89.3	89
NB	75.6	15.3	76.5	75.6

Sınıflandırıcıların performansları karşılaştırıldığında karar ağaçları (C4.5, RF, LMT), diğer sınıflandırıcılara göre daha iyi bir performans göstermiştir. Örneğin, C4.5 kullanıldığında DR, FPR ve doğruluk oranları sırasıyla %99.9, %0.2 ve %99.8 olarak ölçülürken SLR kullanıldığında bu değerler %97.4, %2 ve %97.4 olarak ölçülmüştür (Çizelge 6.1).

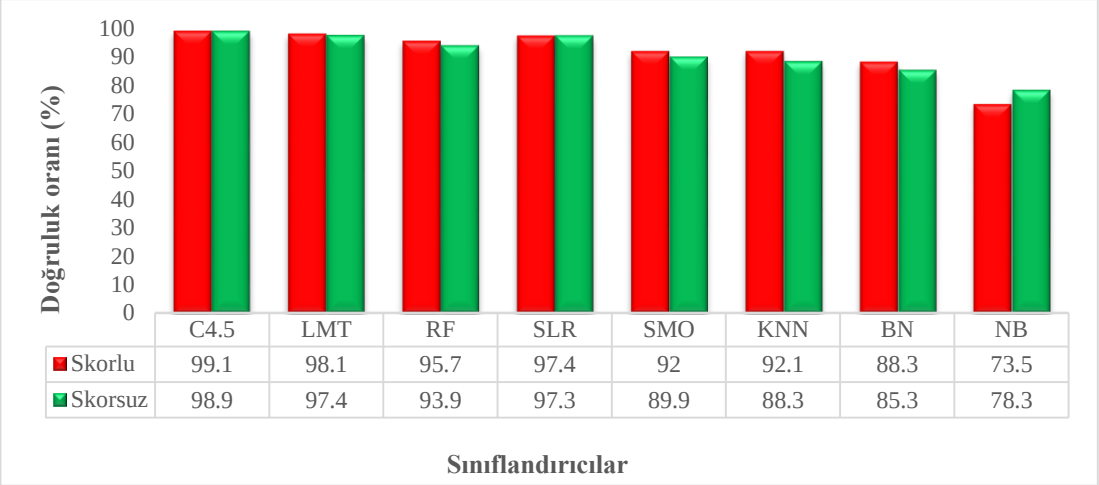
Mantık tabanlı çalışan karar ağaçlarının verilerdeki aykırılıkları (anomalileri) tespit edebilmeleri, varyansı düşük ve birbirleriyle ilişkili özelliklerin çok olduğu veri setlerinde yüksek doğruluk oranına sahip sonuçlar üretmeleri, ölçeklendirilebilir olmaları ve varsayılan olarak bir nevi özellik seçim algoritmalarını kullanmaları bu sınıflandırıcıların performanslarının yüksek çıkmasını sağlamıştır. İstatistiksel bir sınıflandırıcı olan KNN

örnek tabanlı çalışmakta ve veri dağılımı hakkında ön bilgi bulunmadığı durumlarda iyi sonuç üretmektedir. KNN kullanılarak yüksek performans değerleri elde edilmiştir. SLR algoritmasının doğrusal olmayan problemleri çözmede yetersiz kalması ve yüksek bias içermesi, algoritma verimini düşürmektedir fakat önerilen modelle oluşturulmuş veri setleri üzerinde iyi sonuçlar döndürmüştür. SVM ve SMO'nun kullanılan çekirdeğe bağlı olarak hem doğrusal ayrım (linear separation) hem de doğrusal olmayan sınır (non-linear boundary) durumlarında iyi çalışması ve yüksek boyutlu (high dimensions) verileri iyi bir şekilde işlemesi bu algoritmaları ön plana çıkarmaktadır ve elde edilen performans değerleri istenilen düzeydedir.

MLP'nin öğrenme süreci paralel olmakla birlikte genellikle doğru tahmin üretmektedir fakat öğrenme sırasında uzun hesaplama zamanı gerektirdiğinden dolayı sınıflandırma zamanı uzun sürmüştür bundan dolayı performas ölçümleri ilgili tabloya eklenmemiştir. En düşük performans BN ve NB kullanıldığında gözlemlenmiştir. İstatistiksel model bir sınıflandırıcı olan BN genel olarak hızlı ve etkili sonuç döndürmektedir fakat birçok özelliği olan veri kümeleri için uygulaması pratik olmaması sebebiyle doğruluk oranı %89 olarak ölçülmüştür. Uygulaması kolay, fazla hesaplama zamanı gerektirmeyen yüksek boyutlu (high dimensions) verilerde iyi sonuçlar üreten NB birbirleriyle çok ilişkili olmayan varsayımlar üzerine hesaplama yaptığı için iyi sonuçlar döndürmeyebilmektedir. Test sonuçlarına göre performansı en düşük ölçülen sınıflandırıcı olmuştur.

Test sonuçlarına göre NB hariç diğer tüm sınıflandırıcılar için skor kullanılarak elde edilen sonuçlar skor kullanmadan elde edilen sonuçlara göre daha yüksek çıkmıştır. Skorlu veri seti skorsuz veri setinden daha az önemli görülen özelliklerin çıkartılmasıyla elde edilen veri setidir. Bu aşamda EMDM'nin ilgili algoritması kullanılmaktadır. Her sınıflandırma için ortalama doğruluk oranları skorlu ve skorsuz veri setleri için şekil 6.1'de görülmektedir. Örneğin, C4.5/J48 kullanıldığında doğruluk oranı skorlu ve skorsuz sırasıyla %99.1, %98.9 ölçülürken; LMT kullanıldığında bu oranlar %98.1, %97.4; RF kullanıldığında %95.7, %93.9; SLR kullanıldığında %97.4, %97.3; SMO kullanıldığında %92, %89,9 ve KNN kullanıldığında %92.1, %88.3 olarak ölçülmüştür. Benzer değerler DR, FPR ve F-ölçümü için de gözlemlenmiştir. Skorsuz veri setinde elde edilen değerler skorlu veri setine göre düşük ölçüldüğü için ve zamanla skorsuz veri setindeki özellik sayısı artığından dolayı 1000 program analiz edildikten sonra sadece skorlu veri setiyle analize devam edilmiştir. Bu durum

sınıflandırma için birbiriyle ilişkili daha az özellik kullanmanın sınıflandırma için daha iyi sonuçlar döndürdüğünü doğrulamaktadır.



Şekil 6.1 Sınıflandırıcıların oluşturulan skorlu ve skorsuz veri setleri üzerindeki performansları (1000 program analiz edildiğinde)

Çizelge 6.2 özellik seçimi yapılmadan ve yapılarak elde edilen sonuçları göstermektedir. Özellik seçimi yapılırken mevcut yöntemler ve EMDM ayrı ayrı uygulanarak gerekli özellikler veri setinden seçilmiştir.

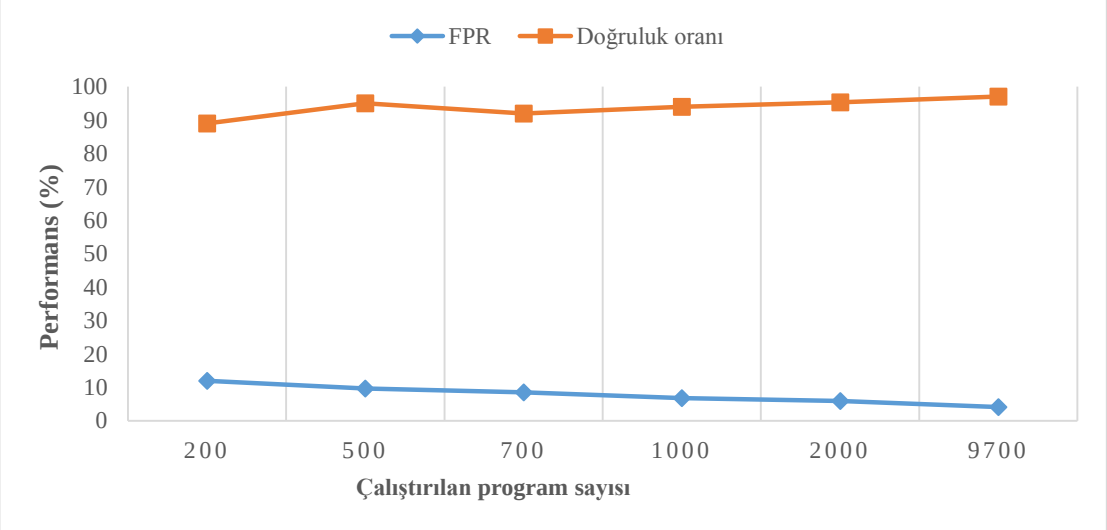
Çizelge 6.2 EMDM özellik seçimi ve mevcut özellik seçim algoritmasının karşılaştırılması

	C4.5/J48	**LMT**	**RF**	**SLR**	**SMO**	**KNN**	**BN**	**NB**
Özellik seçimi yapılmadan	98.9	97.4	93.9	97.3	89.9	88.3	85.3	78.3
Pearson korelasyon katsayısı kullanarak özellik seçimi	98.3	97.6	95.2	97.8	90.6	91.4	86.8	76.6
EMDM kullanarak özellik seçimi	99.1	98.1	95.7	97.4	92	92.1	88.3	73.5

Sınıflandırma öncesi uygun özellik seçim algoritmalarının kullanılması sınıflandırma performansını artırmaktadır (NB hariç) (Çizelge 6.2). Ağaç yapılarıyla elde edilen sonuçlar birbirine yakın çıkarken diğer sınıflandırıcılarda bu fark daha fazla çıkmıştır. Bu durum ağaç yapılarında varsayılan olarak özellik seçim algoritmalarının kullanılmasıyla açıklanabilir. Genel olarak EMDM kullanılarak yapılan özellik seçimi mevcut özellik seçim algoritmalarına göre daha iyi sonuçlar döndürmüştür (Çizelge 6.2). Bazı durumlarda özellik seçiminin uygun bir şekilde yapılmaması performansın hızlı bir şekilde düşmesine neden

olabilmektedir. Bundan dolayı özellik seçimi yapılırken dikkatli olunmalı ve veri seti için uygun olan metotlar ve arama kriterleri kullanılmalıdır.

Önerilen sistemi daha doğru bir şekilde değerlendirmek için farklı sayıda zararlı ve normal yazılım örnekleri analiz edilmiştir. Şekil 6.2 analiz edilen program sayısına karşılık sınıflandırıcıların ortalama doğruluk ve FP oranlarını göstermektedir.



Şekil 6.2 Sınıflandırıcıların analiz edilen program sayısına göre ortalama doğruluk ve FP oranları

Analiz edilen program sayısı arttıkça tüm sınıflandırma algoritmalarının doğruluk oranı artarken FPR düşmektedir (Şekil 6.2). Örneğin, 200 program analiz edildiğinde doğruluk oranı %89 olarak ölçülürken bu oran analiz edilen program sayısı arttıkça artmış ve 9700 program analiz edildiğinde %97'ye çıkmıştır. Diğer taraftan FPR başlangıçta %12 olarak ölçülürken zamanla bu oran %4'e kadar düşmüştür. Test sonuçları bize analiz edilen program sayısı arttıkça, sınıflandırma algoritmalarından bağımsız olarak performans değerlerinin iyileştiğini göstermiştir.

Çizelge 6.3 özellik yerleştirmesi optimize edilmiş karar ağacı ve diğer karar ağaçlarının EMDM veri seti üzerindeki performanslarını göstermektedir (1000 program analiz edildiğinde). Test sonuçlarına göre iyileştirilmiş ağaç performansı LMT ve RF'ye göre artmaktadır. Fakat daha fazla hesaplama yaptığından dolayı daha yavaş çalışmaktadır.

Çizelge 6.3 Karar ağaçlarının performans karşılaştırması

	DR (%)	FPR (%)	Doğruluk oranı (%)
C4.5/J48	99.3	1.4	99.1
LMT	97.9	1.6	98.1
RF	97.5	9.5	95.7
İyileştirilmiş karar ağacı	98.2	2.8	98.3

6.2 Elde Edilen Sonuçların Mevcut Veri Seti Sonuçlarıyla Karşılaştırılması

Elde edilen sonuçlar *n*-gram ve literatürdeki öncü yöntem sonuçlarıyla karşılaştırılarak tartışılmıştır. Bu amaçla *n*-gram kullanarak veri seti oluşturulmuş, daha önce mevcutta kullanılabilir veri setleri kullanılmış ve bilimsel çalışmalarda kullanılan yöntemlere göre sonuçlar değerlendirilmiştir. İstisnalar dışında her durumda EMDM *n*-gram (2-gram, 3-gram, 4-gram, vb.) ile oluşturulmuş veri setlerine göre daha iyi sonuçlar döndürmüştür. Çizelge 6.4 önerilen model sonuçlarının 4-gram ile karşılaştırmasını göstermektedir.

Çizelge 6.4 EMDM sonuçlarının *n*-gram ile karşılaştırılması

Model	Sınıflandırıcı	DR (%)	FPR (%)	Doğruluk oranı (%)
Bilinen 4-gram Modeli	**C4.5/J48**	91.4	9.1	91
	LMT	97.7	2.4	97.4
	RF	85.1	18.8	85
	SLR	94.6	6.3	94.5
	SMO	92	9.6	92.1
	KNN	87	16.2	87.3
	BN	-	-	-
	MLP	-	-	-
	NB	86.7	16.4	87
Önerilen EMDM Modeli	**C4.5/J48**	99.5	0.7	99.4
	LMT	98.6	1.5	98.4
	RF	96.1	4.9	96
	SLR	98.5	4	97.2
	SMO	97.4	2.4	97.3
	KNN	87.4	13.6	87.7
	BN	86.6	12.8	86.5
	MLP	88	17.64	87.5
	NB	75.8	20	75.5

Örneğin, SLR algoritmasının 4-gram ve önerilen model üzerindeki DR, doğruluk oranı ve FPR sırasıyla %94.6, %94.5, %6.3 (4-gram) ve %98.5, %97.2, %4 (EMDM). Benzer şekilde SMO kullanıldığında bu sonuçlar %92, %92.1, %9.6 (4-gram) ve %97.4, %97.3, %2.4 (EMDM) ve C4.5 kullanıldığında bu sonuçlar %91.4, %91, %9.1 (4-gram) ve %99.5, %99.4, %0.7 (EMDM) olarak ölçülmüştür (Çizelge 6.4). Verilen oranlardan da görüleceği üzere

önerilen modelin hem tespit hem de doğruluk oranı çok daha yüksek çıkmış, FPR ise çoğu durumda %5'in altında çıkmıştır. LMT, RF ve KNN sınıflandırıcıları üzerinde de önerilen model 4-grama göre daha iyi bir performans göstermiştir. Benzer sonuçlar 2-gram, 3-gram, 4-gram, vb. kullanıldığında da ölçülmüştür. Ayrıca, BN ve MLP ile uzun süre beklenmesine rağmen *n*-gram kullanılarak oluşturulan veri seti üzerinde herhangi bir sonuç alınamamıştır. Bunun sebebi *n*-gram ile oluşturulan veri setinde özellik uzayının çok fazla olmasından kaynaklanmaktadır. *n*-gram kullanılarak oluşturulan veri seti EMDM'ye göre yaklaşık 30 kat daha fazla özellik içermektedir.

ClaMP (Classification of Malware with PE headers) veri seti 5184 programdan oluşmaktadır. API dizilimleri kullanılarak oluşturulan bu veri seti zararlı ve normal yazılım örneklerini içermektedir. NSL-KDD veri seti KDD'99 veri setinin güncellenmiş hali olup yaklaşık 125.000 kayıttan oluşmaktadır (Tavallaee vd. 2009). Genellikle saldırı tespit sistemlerinin performansını ölçmek için kullanılan bu veri seti ağda gerçekleşmesi muhtemel olan saldırıları içermektedir. Çizelge 6.5 C4.5, SLR, SMO, BN ve NB sınıflandırıcılarının ClaMP, NSL-KDD ve EMDM veri setleri üzerindeki performanslarını göstermektedir.

Çizelge 6.5 ClaMP, NSL-KDD ve EMDM veri setlerinin performans karşılaştırması

Algoritma	**ClaMP Veri Seti (5184 program)**			**NSL-KDD Veri Seti (47.000 program)**			**EMDM Veri Seti (9700 program)**		
	DR	**FPR**	**Doğruluk**	**DR**	**FPR**	**Doğruluk**	**DR**	**FPR**	**Doğruluk**
C4.5/J48	97.8	2	97.8	81.1	15	81	99.9	0.2	99.8
SLR	96	4	96.1	75.5	20	75	97.4	2	97.4
SMO	95.7	4	95.6	75.1	21	75	93.2	6.8	93.1
BN	94	5.4	94.7	75	20	74.5	89	8.6	88.9
NB	65.2	32	65	76.6	20	76.5	75.6	15.3	75.6

Performanslar ölçülürken DR, yanlış pozitif oranı ve doğruluk oranı kullanılmıştır. Genel olarak EMDM yöntemiyle oluşturulan veri setinin sonuçları daha iyi olmakla birlikte öğrenme süreci daha az sürmektedir. Örneğin, C4.5, SLR ve NB kullanıldığında EMDM yöntemiyle elde edilen sonuçlar diğer iki veri setine (ClaMP, NSL-KDD) göre çok daha iyi çıkmıştır. Ayrıca, istisnai durumlar hariç EMDM yöntemiyle her durumda NSL-KDD veri setine göre daha iyi sonuçlar elde edilmiştir. Çizelge 6.6 sınıflandırma algoritmalarının farklı çalışmalar üzerindeki performanslarını ve çizelge 6.7 farklı veri seti oluşturma tekniklerinin karşılaştırılmasını göstermektedir.

Çizelge 6.6 Sınıflandırıcıların farklı çalışmalar üzerindeki performansları

Çalışma	Sınıflandırıcı	DR (%)	FPR (%)	Doğruluk oranı (%)	Yıl
Firdausi vd.	**KNN**	81.7	8.1	86.5	2010
	NB	58.1	12.8	65.4	
	SVM	90.4	8.4	91	
	J48	90.9	3.8	93.6	
Ye vd.	**NB**	63.25	-	50.22	2010
	SVM	84.60	-	83.45	
	J48	56.87	-	57.28	
Islam vd.	**SVM**	-	14	84.34	2013
	DT	-	11.9	86.59	
	RF	-	10.4	87.8	
Santos vd.	**DT: J48**	92	9	91.25	2013
	KNN K = 1	93	7	92.8	
	KNN K = 3	91	8	91.7	
	NB	90	31	79.64	
	SVM: RBF	67	3	81.67	
	SVM: Polinomsal	88	9	89.65	
Yousefi-Azar vd.	**SVM**	94.93	5.07	93.44	2018
	RF	93.18	6.82	90.05	
	KNN	90	10	91.28	
Aslan vd.	**NB**	75.6	15.3	75.62	2020
	BN	89	8.6	88.9	
	SMO	93.2	6.8	93.1	
	KNN	99.6	0.8	99.62	
	SLR	97.4	2	97.4	
	LMT	99.9	0.1	99.87	
	RF	99.8	0.4	99.82	
	J48	99.9	0.2	99.8	

Çizelge 6.7 Farklı veri seti oluşturma tekniklerinin karşılaştırılması

Çalışma	Yıl	Özellik çıkarma yöntemi	Başarı (%)
Bailey vd.	2007	Sistem durumu değişikliklerine göre davranışsal özellik çıkarma	91.6
Ye vd.	2010	API çağrıları – CIMDS	88.09
Shan ve wang	2014	Davranış tabanlı kümeleme	71
Narayanan vd.	2017	İçerik duyarlı, uyarlanabilir ve ölçeklendirilebilir kurallara göre özellik çıkarma	89.92
Saracino	2018	Çekirdek, uygulama, kullanıcı ve paket seviyesi özellikler	96
EMDM yöntemi	2020	API sistem çağrılarını gruplayarak ve özellik sayısını sınırlı tutarak özellik çıkarma	99.8

Sınıflandırma algoritmaları önerilen yöntem ile oluşturulmuş veri seti üzerinde genel olarak daha iyi bir performans göstermiştir. Örneğin, C4.5/J48 algoritması kullanılarak DR %90.9 (Firdausi vd. 2010), %56.87 (Ye vd. 2010), %92 (Santos vd. 2013) ve %99.9 (Aslan vd. 2020) (Çizelge 6.6). Farklı yöntemler kullanılarak oluşturulmuş veri setlerinin başarı oranları sırasıyla: %91.6 (Bailey vd. 2007), %88.09 (Ye vd. 2010), %71 (Shan ve wang 2014), %89.92 (Narayanan vd. 2017), %96 (Saracino 2018) ve %99.8 (EMDM 2020) (Çizelge 6.7).

Verilen değerlerden de anlaşılacağı üzeri önerilen yöntemle oluşturulan veri setinin başarı oranı daha önce yapılmış çalışmalara göre daha iyi durmakla birlikte özellik sayısında da kayda değer bir azalış görülmektedir. Bu durum önerilen yöntemin birbirleriyle ilişkili özellikleri belirlemede etkili olduğunu göstermektedir.

6.3 Eksiltici Merkezi Davranış Modeli Kullanılarak Elde Edilen Özellikler

EMDM kullanılarak *n*-gram ve benzeri veri seti oluşturma tekniklerine göre çok daha az sayıda özellik elde edilmiştir. EMDM ve 4-gram kullanılarak elde edilen özellik sayıları çizelge 6.8'de görülmektedir.

Çizelge 6.8 EMDM ve 4-gram kullanıldığında elde edilen özellik sayıları

	EMDM veri seti	**4-gram veri seti**
Ortalama ZY özellik sayısı (Özellik vektörü öncesi)	52	1420
Ortalama NY özellik sayısı (Özellik vektörü öncesi)	36	1245
Toplam ZY ve NY özellik sayısı (Özellik vektörü sonrası)	700	20.000

Oluşturulan veri setlerinde ZY'lerin ortalama özellik sayıları normal yazılımlara göre daha fazla çıkmıştır. Özellik vektörü oluşturulmadan önce ZY'ler için ortalama özellik sayısı önerilen sistem için 52 olurken bu sayı 4-gram kullanıldığında 1420'ye çıkmıştır (Çizelge 6.8). Benzer şekilde normal yazılımlar için önerilen sistemde özellik sayısı 36 olurken 4-gram kullanıldığında bu sayı 1245'e çıkmıştır. CreateFileMapping, CreateRemoteThread, CreateService, FindFirstFile, FindNextFile, MapviewOfFile, QueryDirectoryWriteFile, ReadFileWriteFile, RegSetInfoKey, RegDeleteValue, RegQueryKeyRegSetInfoKey, RegOpenKeyRegSetInfoKey ve WriteFile gibi özelliklerin ZY'lerde sıkça tekrarlandığı gözlemlenmiştir.

Belirli kurallar kullanılarak özellik vektörü oluşturulmadan da zararlı ve normal yazılım örnekleri ayrıştırılabilmektedir. Özellikler makine öğrenmesi öncesi özellik vektörü oluşturacak şekilde birleştirilmektedir. Özellik vektörü oluşturulurken var olan özellikler için tekrar sayıları olmayan özellikler için ise 0 yazılmaktadır. Programların göstermiş oldukları davranışlar farklılık gösterebildiğinden dolayı zamanla özellik vektöründeki özellik sayısı artmaktadır. Bu artış belirli sayıda program analiz edildikten sonra EMDM için neredeyse

sabit kalırken *n*-gram için artış devam etmektedir ve belirli bir noktadan sonra sınıflandırmayı zorlaştırmaktadır.

6.4 Eksiltici Merkezi Davranış Modelinin Çalışma Zamanı Performans Analizi

Bu alt bölümde geliştirilen sistemin çalışma zamanı performansı Intel (R) Core i5-3470 3.20 GHz işlemcili ve 8 GB bellekli bilgisayar üzerinde ölçülmüştür. EMDM ile analizler hızlı bir şekilde yapılmış olup fazla gecikmeler yaşanmamıştır. Ayrıca, sonuçlar *n*-grama göre çok hızlı elde edilmiştir. Sınıflandırıcıların eğitim ve test aşamasındaki performansları da kabul edilebilir zaman aralığında çıkmıştır. Bazı sınıflandırıcıların *n*-gram kullanılarak oluşturulan veri setleri üzerindeki çalışma zamanları çok yüksek çıkmıştır. Örneğin, BN ve MLP kullanılarak *n*-gram üzerinde birkaç saat beklenmesine rağmen herhangi bir sonuç alınamamıştır. Diğer taraftan EMDM kullanılarak oluşturulan veri setleri üzerinde birkaç saniye içinde gerekli sonuçlar alınmıştır. Çizelge 6.9 EMDM, *n*-gram ve sınıflandırıcıların ortalama çalışma zamanlarını göstermektedir.

Çizelge 6.9 *n*-gram ve EMDM ortalama çalışma zamanları

	Veri seti oluşturma zamanı (1 program için)	**Sınıflandırma zamanı**	**Özellik sayısı yaklaşık**
***n*-gram**	10-15 saniye	60-100 saniye	20.000
EMDM	1-5 saniye	5-10 saniye	700

6.5 Zararlı Yazılım Analiz Sonucunda Elde Edilen Bulgular

Zararlı yazılım analizi sonucunda tespit yöntemleri ve ZY'lerin karakteristik özelliklerine dair bazı bulgular elde edilmiştir. Bu bulgular önemli olup daha sonra bu alanda çalışacak araştırmacılar için bir kaynak oluşturacaktır. Ayrıca, ZY analistleri için de önemli olup ZY analizi yaparken bu bulgulardan faydalanabileceklerdir. Elde edilen bulgular iki farklı başlık altında değerlendirilmiştir: ZY tespit yöntemleriyle ve ZY karakteristik özellikleriyle ilgili elde edilen bulgular.

6.5.1 Zararlı yazılım tespit yöntemleriyle ilgili elde edilen bulgular

Her çalışmada kullanılan ZY tespit yöntemi ve özellik çıkarma tekniği birbirinden farklıdır. Tespit yöntemlerinin başarısı birçok bileşene bağlı olmakla birlikte genel olarak analiz edilen yazılımların türüne, ailesine, hangi yıllara ait olduklarına ve kullanılan özelliklerin sayısına

göre değişmektedir. Yapılan araştırmalara ve analizlere göre mevcut ZY tespit yöntemlerine dair elde edilen bulgular şöyle sıralanabilir:

- Mevcut yöntemler anlamlı birbirleriyle ilişkili davranışları belirlemede yetersiz kalmaktadırlar;
- Özellik sayısı hızla artmaktadır bu durum performansı düşürmektedir;
- Özellikler arasındaki benzerlikleri ve farklılıkları belirlemede yetersiz kalmaktadırlar;
- Başarıları sınırlı kalmakta olup sadece belirli ZY türleri ya da aileleri için yüksek performans göstermektedirler;
- Daha önce görülmemiş ZY'leri tespit etmede yetersiz kalmaktadırlar;
- Gizlenme tekniklerine karşı dirençli değillerdir;
- Atlatma saldırılarına karşı dirençli değillerdir.

6.5.2 Zararlı yazılım karakteristik özellikleriyle ilgili elde edilen bulgular

ZY'ler kullanıcılar tarafından istenmeyen amaçlar için oluşturulduklarından dolayı sistemle etkileşimleri normal programlara göre farklılıklar göstermektedir. Bundan dolayı sistemde kendilerini gizlemeye çalışmakta, varolan koruma yazılımlarını etkisizleştirmeye çalışmakta ve rastgele dosyalar oluşturarak kendilerini bu dosyalara kopyalamaktadırlar. Ayrıca, sistemde kalıcı hale gelmek için kendilerini Windows otomatik başlangıç konumlarına yerleştirmekte, kendilerine servisler oluşturmakta ve ancak belli koşullar gerçekleştiğinde gerçek davranışlarını sergilemektedirler. Yapılan analizler sonucunda elde edilen ZY bulguları şöyle sıralanabilir:

- ZY'lerin çoğu var olan prosesleri kullanarak ya da yeni prosesler oluşturarak zararlı davranışları bu proseslere yaptırmaktadır.
- Yeni nesil ZY'ler var olan sistem ve üçüncü parti yazılım dosyalarının benzerlerini oluşturarak kendilerini gizlemeye çalışmaktadır. Örneğin, "regebox.exe" "c:\recycler\regebox.exe" yolunda kendini oluşturmaktadır bu da çoğu kullanıcı tarafından "\Recycle Bin" çöp kutusu yolu olarak algılanmaktadır.
- ZY'lerin büyük bir bölümü zararlı davranışlarını geçici dosya yollarında oluşturmaktadır: "c:\documents and settings\kullanıcı adı\local settings\temp", "c:\users\ kullanıcı adı\appdata\local\temp", vb.

- ZY'ler genelde Windows otomatik başlangıç konumlarına yerleşerek sistemde kalıcı olmaya çalışırlar. Bu konumlardan bazıları şöyle sıralanabilir:
  "shell startup" dosya yolları;
  "%appdata%\microsoft\windows\startmenu\programs\startup";
  "%programdata%\microsoft\windows\start";
  "hkcu\software\microsoft\windows\currentversion\run";
  "hkcu\software\wow6432node\microsoft\windows\currentversion\run";
  "hkcu\software\microsoft\windows\currentversion\runonce"; vb.
- Bazı ZY'ler ancak "administrator" yetkisiyle çalıştırıldıklarında gerçek davranışlarını sergilemektedir.
- ZY'lerin büyük bir kısmı rastgele dosyalar oluşturmakta ve bu dosyalar için anlamsız isimler kullanmaktadır.
- Bazı ZY'ler belirli bir işletim sistemi ve sürümü için yazıldıklarından dolayı farklı işletim sistemi ve sürümlerinde çalışmamaktadır.
- Yeni nesil ZY'lerin büyük bir bölümü "svchost. exe", "winlogon.exe", "conhost.exe", vb. Windows dosyalarına kendilerini enjekte etmekte ya da bu isimlerle kendilerini farklı dosya yollarına kopyalamaktadır.
- ZY'lerin bir bölümü çalışır çalışmaz sistemde var olan güvenlik yazılımlarını (güvenlik duvarı, antivirüs tarayıcısı, vb.) bulup etkisiz hale getirmeye çalışmaktadır.
- Bazı ZY'ler sanal ortamda analiz edildiklerini anladıkları zaman gerçek davranışlarını sergilememekte ve çalışmalarını durdurmaktadır.
- Yeni nesil ZY'lerin bir kısmı kendilerine servis oluşturarak Windows başlatıldığında sistemde otomatik olarak çalışmaktadır.

6.6 Değerlendirme

Bu bölümde önerilen sistemin performansı ve literatürdeki öncü yöntemlerle karşılaştırması anlatılmıştır. Önerilen sistem hem ZY'lerin farklı türlerini ve ailelerini hem de yeni nesil ve daha önce bilinmeyen ZY'leri başarılı bir şekilde tespit etmektedir. Ölçülen performans değerleri analiz edilen program sayısı arttıkça artmaktadır. Ayrıca, önerilen sistemde belirli sayıda program analiz edildikten sonra özellik sayısı artmamaktadır. Bu durum sınıflandırmanın daha başarılı bir şekilde gerçekleştirilmesini sağlamaktadır. Sınıflandırma algoritmalarının EMDM veri setleri üzerindeki performansları yüksek ölçülmüş olup en iyi

sonuçlar sırasıyla C4.5, RF, LMT, KNN, SLR ve SMO kullanıldığında elde edilmiştir. Ayrıca, özellik seçimi ve budaması optimize edilmiş karar ağacı kullanılarak da yüksek performans değerleri ölçülmüştür.

Elde edilen sonuçlar literatürdeki öncü yöntem sonuçlarına oranla daha yüksek çıkmıştır. Mevcut çalışmalarda davranışlarının net belirlenememesi, çıkarılan özellik sayısının fazla olması, sadece belirli ZY türü ve ailesi için çalışmaları, yeni nesil ZY'leri tespit etmede yetersiz kalmaları, atlatma saldırıları ve gizlenme tekniklerine karşı dirençli olmamaları performansın düşmesine yol açmaktadır. Diğer taraftan önerilen sistemde bu eksiklikler belirlenerek gerekli katkılar yapılmıştır. Bu durum performans değerlerinin iyileşmesini sağlamıştır.

.

7. SONUÇ

7.1 Sonuçlar

Bilgisayar, mobil ve İnternet teknoloji kullanımının hızla artması birçok siber güvenlik sorununu da beraberinde getirmiştir. Yapılan araştırmalar güvenlik saldırılarının büyük çoğunluğunun ZY'ler tarafından yapıldığını göstermektedir. Daha önce basit amaçlar için gerçekleştirilen bu saldırılar yerini geniş çaplı, küresel boyutta ve hedef odaklı saldırılara bırakmıştır. Bu nedenle, her gün yeni ZY'ler yazılmakta ve bu yazılımlar zamanla şekil ve yöntem değiştirmektedir. ZY üretiminin hızla artması ve bu yazılımların sürekli şekil değiştirmesi, güncel olarak kullanılan antivirüs tarayıcılarının bu yazılımları tespit etmede yetersiz kalmasına neden olmaktadır. Ayrıca, bu yazılımların dünya ekonomisine verdiği zararlar da her geçen gün artmaktadır. Bu sebepler bu yazılımlara karşı alınacak tedbirleri zorunlu kılmakta ve daha fazla bilimsel araştırma yapmanın gerekliliğini ortaya koymaktadır.

Bu tez çalışmasında ZY tespit etme yöntemleri incelenmiş, var olan eksiklikler belirtilmiş ve bu eksikliklerin giderilmesi konusunda neler yapılabileceği tartışılmıştır. Var olan eksiklikler göz önünde bulundurularak yeni yaklaşımlar ve yöntemler ileri sürülmüştür. Bu amaçla tez kapsamında:

1. EMDM önerilerek ZY davranış ve özellikleri belirlenmiş;
2. BSTY önerilerek ZY'ler daha doğru analiz edilmiş;
3. Mevcut sınıflandırma algoritmaları kullanılarak ve özellik seçimi optimize edilmiş karar ağacı iyileştirilerek bu yazılımlar sınıflandırılmış;
4. Analiz sonuçlarına göre ZY karakteristik özellikleri elde edilmiştir.

EMDM masaüstü ve dizüstü bilgisayarlardaki ZY'lerin sistem içinde gösterdikleri davranışları gruplayarak bu yazılımları tespit etmektedir. Öncelikle yapılan davranışlar ve davranışların hangi sistem yollarında yapıldıkları belirlenerek özellikler ve özelliklerin önem dereceleri hesaplanmıştır. Daha sonra önem derecesi fazla olan özelliklerin frekanslarına bakılarak zararlı davranışlar normal davranışlardan ayırt edilmiştir. Bu aşamada mevcut

sınıflandırma algoritmaları ve özellik seçimi iyileştirilmiş karar ağacı kullanılarak bu yazılımlar tespit edilmiştir.

Önerilen model ve yöntemlerin performansını değerlendirmek amacıyla çeşitli veri setleri oluşturulmuş ve sonuçlar *n*-gram ve literatürdeki öncü yöntem sonuçlarıyla karşılaştırılmıştır. Test sonuçlarına göre önerilen model uygun bir makine öğrenme algoritması ile birleştirildiğinde DR, FPR ve doğruluk oranı sırasıyla %99.8, %0.2 ve %99.9 olarak ölçülmüştür. Bu sonuçlar hem *n*-gram hem de diğer model sonuçlarına göre oldukça yüksek çıkmış olup zararlı özellikleri belirlemede önerilen sistemin etkili olduğunu göstermektedir. Ayrıca oluşturulan özellik sayısının az olması ve zamanla artmaması, en önemli ve ayırt edici özelliklerin seçilmiş olması skor kullanarak oluşturulan veri setini daha ön plana çıkarmaktadır. Analiz edilen program sayısı arttıkça başarı oranı (tespit, doğruluk, F-ölçümü oranları, vb.) artmıştır. En iyi sonuçlar karar ağaçları (C4.5, LMT ve RF), KNN, SLR ve SMO'dan elde edilirken BN, MLP ve NB algoritmalarından istenilen sonuçlar elde edilememiştir. Uygun özellik seçim algoritmaları kullanıldığında sınıflandırıcıların performanslarında iyileşme görülmesine rağmen önerilen EMDM özellik seçim algoritması kadar artırmamıştır.

Literatürdeki ZY tespit yaklaşımları ve bu yaklaşımlar kullanılarak geliştirilen metotlar incelendiğinde, performansı düşüren bazı eksiklikler tespit edilmiştir. Bu eksiklikler şöyle sıralanabilir:

- Oluşturulan imzaların tespit etmede yetersiz kalması;
- ZY davranışlarının net belirlenememesi;
- Oluşturulan özelliklerin fazla olması ve birbiriyle ilişkili olmaması;
- Fazla kurallar kullanarak öğrenmeyi zorlaştırmaları;
- Yeni nesil ZY'leri tespit etmede yetersiz kalmaları;
- Atlatma saldırılarına ve gizlenme tekniklerine karşı dirençli olmamalarıdır.

Bu tez çalışmasında bu eksiklikler dikkate alınmış ve her adımda gerekli iyileştirmeler yapılmıştır. Yapılan katkılar şöyle sıralanabilir:

EMDM önerilmiştir: EMDM analiz edilen programın işletim sistemiyle olan etkileşimini inceleyerek davranışlar belirlemektedir. Bu etkileşimler değerlendirilirken yapılan

etkileşimler ve bu etkileşimlerin hangi dosya yolunda yapıldığına bakılarak davranışlar ve her davranış için önem derecesi hesaplanmaktadır. ZY davranış kalıpları elde edilirken zararlı yazılımlarda görülen ancak normal yazılımlarda görülmeyen veya nadiren görülen davranışlar elde edilmektedir. Bunun için davranışlar gerçekleştirildikleri yere göre 3 farklı grupta 120 farklı dosya yolunu gösterecek şekilde ayrılmış ve dosya yolu risk skorları bu dosya yollarına göre hesaplanmıştır. Bu sayede ZY ve normal yazılım örneği aynı etkileşimleri gösterse bile elde edilen davranışlar ve özellikler farklı olmaktadır. Bu şekilde ZY'ler normal yazılımlardan ayrılmaktadır. Test sonuçlarına göre CreateFileMapping, CreateRemoteThread, CreateService, FindFirstFile, FindNextFile, MapviewOfFile, QueryDirectoryWriteFile, ReadFileWriteFile, RegSetInfoKey, RegDeleteValue, RegQueryKeyRegSetInfoKey ve WriteFile gibi özelliklerin ZY'ler tarafından sıkça kullanıldığı gözlemlenmiştir. EMDM hem bilinen hem de bilinmeyen ZY'leri tespit etmede büyük bir başarı göstermiştir. Bu başarı önerilen modelin birbiriyle ilişkili davranışları daha net belirleyebilmesinde ve özellik sayısını sınırlı tutmasından kaynaklanmaktadır. Bu modelin mevcut AVS'lere eklenmesi ZY'lerin tespitinde büyük bir başarı sağlayacaktır.

EMDM ve *n*-gram kullanılarak veri setleri oluşturulmuştur: EMDM kullanılarak 2 *n*-gram kullanarak 1 veri seti oluşturulmuştur. Veri setleri oluşturulurken zararlı ve normal yazılım örnekleri farklı kaynaklardan seçilmiş olup toplamda 9700 program analiz edilmiştir.

Detaylı literatür araştırması yapılarak ZY tespit yaklaşımları sınıflandırılmıştır: Böyle detaylı bir araştırma yapılmasının nedeni çözüme yönelik çalışmalara geçmeden önce problemin zorluğunu ve sınırlarını belirlemektir. Ek olarak, daha önce yapılmış çalışmaların performanslarını değerlendirerek daha iyi çalışan bir model öne sürebilmektir. Ayrıca, bu aşamada ZY tespit yaklaşımları 7 farklı kategoriye ayrılarak açıklanmış ve bu yaklaşımlarda kullanılan farklı metotlar karşılaştırılmıştır. Bu yaklaşımlar şöyle sıralanabilir: İmza-, davranış-, bulgusal-, model denetleme-, derin öğrenme-, bulut-, mobil cihazlar ve IoT-tabanlı tespit yaklaşımı.

ZY'lerin daha doğru analiz edilebilmeleri için BSTY önerilmiştir: Şüpheli dosyalar analiz araçları kullanılarak elle ya da otomatik olarak analiz edilebilmektedirler. Bu araçları rastgele kullanmak hem tespiti zorlaştırmakta hem de zaman almaktadır. Bundan dolayı analizin doğru bir şekilde yapılabilmesi için şüpheli dosyaların birkaç farklı araç kullanılarak belirli bir yöntem çerçevesinde yapılması gerekmektedir. BSTY'de her araç çıktısına analist

tarafından incelenerek puanlar verilmekte ve bu puanlar kullanılarak analiz edilen program hakkında bir sonuca gidilmektedir.

Karar ağaçlarında özellik yerleştirme ve budamayla ilgili iyileştirmeler yapılarak sınıflandırma performansı artırılmıştır: Özellikler ağaca yerleştirilirken çok dallanmalı, ağırlığı fazla olan öznitelikler öncelikli olmak üzere dağıtılmış olarak ağaca yerleştirilmektedir. Her özellik için ağırlık hesaplanırken bilgi kazanımı, özniteliğin tekrar sayısı, zararlı ve normal yazılımlarda bulunma sıklığı ve aktif/pasif durumu kullanılarak öznitelik puanları hesaplanmaktadır. Bu puanlar kendi aralarında karşılaştırılarak en büyük puana sahip özellikler başta olmak şartıyla ağaca yerleştirilmektedir. Daha sonra ağaç performansını artırmayan dallar budanarak ağaç küçültülmektedir.

Oluşturulan veri setlerine mevcut makine öğrenmesi algoritmaları uygulanarak sınıflandırma yapılmıştır: Oluşturulan veri setlerine ML algoritmaları uygulanarak bu yazılımlar sınıflandırılmıştır. ML algoritmaları uzun yıllardır birçok farklı alanda kullanılmasına rağmen ZY analizinde ve tespitinde yeterince kullanılmamıştır. Bundan dolayı bu çalışmada ZY tespitinde bu algoritmalardan uygun olanlar kullanılmış ve diğer yaklaşımlarla elde edilen sonuçlarla karşılaştırılmıştır. Test sonuçlarına göre önerilen EMDM çıktıları uygun bir makine öğrenmesi algoritması ile birleştirildiğinde tespit oranı, yanlış pozitif oranı ve doğruluk oranı sırasıyla %99.9, %0.2 ve %99.8 olarak ölçülmüştür. Elde edilen sonuçlara göre önerilen yöntem literatürdeki öncü yöntemlere göre daha iyi performans göstermiştir.

Analiz sonuçlarına göre yeni bulgular elde edilmiştir: Yapılan analizler sonucunda ZY'nin türüne ve ait olduğu aileye bakarak farklı davranışlar elde edilmiş olup bu davranışlar incelenerek ZY'lere özgü bulgular elde edilmiştir. Bu bulgular şöyle sıralanabilir: Sistemde kalıcı hale gelmeye çalışmak, var olan koruma yazılımlarını etkisizleştirmek, rastgele dosyalar oluşturarak kendilerini bu dosyalara kopyalamak, işletim sisteminde otomatik başlangıç konumlarına yerleşmek, kendilerine servis oluşturup çalıştırmak ve ancak belirli koşullar altında çalışmak şeklindedir.

7.2 Öneriler ve Gelecek Çalışmalar

ZY kaynaklı saldırılar ve suçlar gelecekte de var olacak ve artan oranda devam edecektir. Bu saldırıların engellenmesi tamamen mümkün olmayabilir ama azaltılması mümkündür. ZY tespit sistemleri oluşturulurken birbiriyle bağlantılı davranış gurupları kullanılarak özellikler

oluşturulmalıdır. Buna ek olarak doğru davranışların elde edilebilmesi için program örnekleri farklı platformlarda da çalıştırılarak sonuçlar karşılaştırılmalıdır. Diğer önemli bir nokta ise belirlenen özelliklerden sonra sınıflandırmanın doğru bir şekilde yapılmasıdır. Doğru bir şekilde yapılmayacak bir sınıflandırma sonucunda birçok ZY normal olarak işaretlenecektir. Tespit yaklaşımlarının iyi taraflarını birleştirerek birbirine entegre etmek daha iyi bir tespit sisteminin oluşturulmasını sağlayacaktır. Örneğin, davranış tabanlı yaklaşımı model denetleme tabanlı yaklaşımla birleştirmek ve aynı zamanda DL ve bulut kullanmak daha iyi tespit sistemi sunabilecektir. Ayrıca, blok zinciri ve büyük veri gibi yeni teknolojilerin kullanılması daha etkin bir tespit sistemi oluşturmaya yardımcı olacaktır. Bu kapsamda yapılacak öneriler şöyle sıralanabilir:

- Program imzaları kısa ve ZY'nin farklı şekillerini de kapsayacak şekilde belirlenmelidir.
- Gizlenme tekniklerine karşı güçlü metotlar önerilmelidir. Çünkü gizlenme teknikleri programların asıl davranışlarını gizleyerek analizi zorlaştırmakta ve yanlış işaretlemelere neden olmaktadır.
- Doğru davranışları elde etmek için analiz edilen yazılımların farklı ortamlarda çalıştırılması gerekmektedir.
- Programların göstermiş oldukları davranışların çokluğu sebebiyle özellik sayısı hızla artmaktadır. Zararlı davranışlar belirlenirken sadece anlamlı, az ve birbiriyle ilişki kurulabilecek davranışlar göz önünde bulundurularak oluşturulmalıdır.
- ML algoritmaları sistem iyi eğitilmediği çoğu durumda aşırı öğrenmeye yol açmakta ve bunun sonucunda test verileri yanlış işaretlenmektedir. Yeni yaklaşımlar önerilirken aşırı öğrenmeyi azaltan teknikler önerilen modellere enteğre edilmelidir.
- Tespit yaklaşımlarının iyi yönleri birleştirilerek daha etkin çalışan ve bütüncül yaklaşan metotlar önerilmelidir.
- Sistem kaynaklarının kullanımı belirli aralıklarla ölçülmeli ve bir anomali görüldüğünde bir uyarı oluşturularak sistem yöneticisine bilgi gönderilmelidir.
- Antivirüs taracıyıcıları geliştirilerek diğer güvenlik sistemleriyle (güvenlik duvarı, saldırı tespit sistemi, vb.) entegre edilebilir duruma getirilmelidir.
- Son kullanıcılar farkındalık eğitimleriyle bilgilendirilmelidir.
- Online hizmet veren ZY tespit sistemlerinin sayısı artırılmalıdır.

KAYNAKLAR

Adleman, L.M. 1988. An abstract theory of computer viruses. In Conference on the Theory and Application of Cryptography. Springer, New York; 354-374.

Ahmadi, M., Sami, A., Rahimi, H. and Yadegari, B. 2013. Malware detection by behavioural sequential patterns. Computer Fraud & Security 2013(8); 11-19.

Alazab, M., Alazab, M., Shalaginov, A., Mesleh, A., and Awajan, A. 2020. Intelligent mobile malware detection using permission requests and api calls. Future Generation Computer Systems, 107; 509-521.

Alcaraz, C. and Zeadally, S. 2013. Critical control system protection in the 21st century. Computer 46(10); 74-83.

Alcaraz, C. and Zeadally, S. 2015. Critical infrastructure protection: Requirements and challenges for the 21st century. International journal of critical infrastructure protection 8; 53-66.

Alosefer, Y. 2012. Analysing Web-based Malware Behaviourthrough Client Honeypots. Doctoral dissertation, Cardiff University School of Computer Science & Informatics.

Alzarooni, K.M.A. 2012. Malware variant detection. Doctoral dissertation, University College London.

Anderson, B., Quist, D., Neil, J., Storlie, C. and Lane, T. 2011. Graph-based malware detection using dynamic analysis. Journal in computer Virology 7(4); 247-258.

Anderson, H.S. and Roth, P. 2018. Ember: an open dataset for training static PE malware machine learning models. arXiv preprint arXiv:1804.04637.

Anonymous. 2006. Web Sitesi: https://docs.microsoft.com/en-us/previous-versions/ms915101(v%3Dmsdn.10), Erişim Tarihi 03.02.2020.

Anonymous. 2010. Web Sitesi: http://www.csmining.org/index.php/malicious-software-datasets-.html, Erişim Tarihi: 12.02.2020.

Anonymous. 2011. An Intro to Creating Anti-Virus Signatures: http://hooked-on-mnemonics.blogspot.com.tr/2011/01/intro-to-creating-anti-virus-signatures.html, Erişim Tarihi: 04.02. 2020.

Anonymous. 2013. Web Sitesi: https://www.thocp.net/reference/virus/what_is_a_virus.htm, Erişim Tarihi: 25.01. 2020.

Anonymous. 2016a. Web Sitesi: http://www.youngupstarts.com/2016/04/14/9-types-of-computer-viruses-that-you-should-know-about-and-how-to-avoid-them/, Erişim Tarihi: 27.01. 2020.

Anonymous. 2016b. Web Sitesi: http://j00ru.vexillium.org/syscalls/nt/32/, Erişim Tarihi 28.01.2020.

Anonymous. 2016c. ClaMP Classification of malware with PE headers): https://github.com/urwithajit9/ClaMP, Erişim Tarihi: 15.02.2020.

Anonymous. 2018. Web Sitesi: https://msdn.microsoft.com/en-us/library/windows/desktop/ms724880(v=vs.85).aspx, Erişim Tarihi 27.01.2020.

Anonymous. 2020. API Monitor. Web sitesi: http://www.rohitab.com/apimonitor, Erişim Tarihi: 06.02. 2020.

Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K. and Siemens, C.E.R.T. 2014. Drebin: Effective and explainable detection of android malware in your pocket. In Ndss 14; 23–26.

Aslan, Ö. 2016. How to decrease cyber threats by reducing software vulnerabilities and bugs. 1st International Mediterranean science and engineering congress, Çukurova University 210; 639–646.

Aslan, Ö. and Samet, R. 2017a. Investigation of Possibilities to Detect Malware Using Existing Tools. IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA); 1277-1284.

Aslan, Ö. and Samet, R. 2017b. Mitigating cyber security attacks by being aware of vulnerabilities and bugs. IEEE 2017 International Conference on Cyberworlds (CW); 222-225.

Aslan, Ö. 2017. Performance Comparison of Static Malware Analysis Tools Versus Antivirus Scanners To Detect Malware. In International Multidisciplinary Studies Congress (IMSC).

Aslan, Ö. and Samet, R. 2020. A Comprehensive Review on Malware Detection Approaches. IEEE Access 8; 6249-6271.

Aslan, Ö., Samet, R. and Tanrıöver, Ö.Ö. 2020. Using a Subtractive Center Behavioral Model to Detect Malware. Security and Communication Networks, 2020.

Azmoodeh, A., Dehghantanha, A., Conti, M. and Choo, K.K.R. 2018. Detecting crypto-ransomware in IoT networks based on energy consumption footprint. Journal of Ambient Intelligence and Humanized Computing 9(4); 1141-1152.

Bailey, M., Oberheide, J., Andersen, J., Mao, Z. M., Jahanian, F. and Nazario, J. 2007. Automated classification and analysis of internet malware. In International Workshop on Recent Advances in Intrusion Detection Springer, Berlin, Heidelberg; 178-197.

Battista, P., Mercaldo, F., Nardone, V., Santone, A., and Visaggio, C. A. 2016. Identification of Android Malware Families with Model Checking. In Proc. 2nd Int. Conf. Inf. Syst. Secur. Privacy (ICISSP); 542-547.

Bazrafshan, Z., Hashemi, H., Fard, S.M.H. and Hamzeh, A. 2013. A survey on heuristic malware detection techniques. IEEE 5th Conference on Information and Knowledge Technology (IKT); 113-120.

Beek, C., Fiorella, J., Fralick, C., Frosst, D., Greve, P., Marwan, A., Paget, F., Pan, T., Peterson, E., Schmugar, C., Simon, R., Sommer, D. and Su, B. 2016. McAfee Labs Threats Report: https://www.mcafee.com/enterprise/en-us/assets/reports/rp-quarterly-threats-sep-2016.pdf, Erişim Tarihi: 18.01.2020.

Bengio, Y. 2009. Learning deep architectures for AL. Foundations and trends in Machine Learning, 2(1); 1–127.

Borojerdi, H.R. and Abadi, M. 2013. MalHunter: Automatic generation of multiple behavioral signatures for polymorphic malware detection. IEEE 3th International Conference on Computer and Knowledge Engineering (ICCKE); 430-436.

Bright, P. 2015. It's time to change your password ... again: http:/ arstechnica.com /security/2014/10/7-million-dropbox-usernamepassword-pairs-apparently-leaked/, Erişim Tarihi: 08.01.2020.

Canali, D., Lanzi, A., Balzarotti, D., Kruegel, C., Christodorescu, M. and Kirda, E. 2012. A quantitative study of accuracy in system call-based malware detection. Proceedings of the 2012 International Symposium on Software Testing and Analysis; 122-132.

Canell, J. 2016: https://blog.malwarebytes.com/threat-analysis/2013/05/nowhere-to-hide-three-methods-of-xor-obfuscation/, Erişim Tarihi: 07.02.2020.

Cazorla, L., Alcaraz, C. and Lopez, J. 2015. Awareness and reaction strategies for critical infrastructure protection. Computers & Electrical Engineering 47; 299-317.

Cha, S.K., Moraru, I., Jang, J., Truelove, J., Brumley, D., and Andersen D. G. 2011. SplitScreen: Enabling efficient, distributed malware detection. Journal of Communications and Networks 13(2); 187–200.

Chakraborty, R.S., Narasimhan, S. and Bhunia, S. 2009. Hardware Trojan: Threats and emerging solutions. IEEE International High Level Design Validation and Test Workshop; 166-171.

Chandramohan, M., Tan, H.B.K., Briand, L.C., Shar, L.K. and Padmanabhuni, B.M. 2013. A scalable approach for malware detection through bounded feature space behavior modeling. IEEE/ACM 28th International Conference on Automated Software Engineering (ASE); 312-322.

Chess, D.M. and White, S.R. 2000. White. An undetectable computer virus. In Proceedings of Virus Bulletin Conference 5; 1-4.

Choi, C., Esposito, C., Lee, M. and Choi, J. 2019. Metamorphic malicious code behavior detection using probabilistic inference methods. Cognitive Systems Research 56; 142-150.

Christoffersen, D. and Mauland, B. J. 2006. Worm Detection Using Honeypots. Master thesis, Institutt for telematikk.

Cohen, F. 1986. Computer viruses. Doctoral dissertation, University of Southern California.

Cohen, F. 1987a. Computer viruses: theory and experiments. Computers & security 6(1); 22-35.

Cohen, F. 1987b. A cryptographic checksum for integrity protection. Computers & Security 6(6); 505-510.

Cohen, F.B. 1992. A formal definition of computer worms and some related results. Computers & Security 11(7); 641-652.

Cohen, F.B. 1994. A short course on computer viruses. John Wiley & Sons.

Collberg, C., Thomborson, C. and Low, D. 1997. A taxonomy of obfuscating transformations. Department of Computer Science, The University of Auckland, New Zealand.

Collberg, C., Thomborson, C. and Low, D. 1998. Manufacturing cheap, resilient, and stealthy opaque constructs. In Proceedings of the 25th ACM SIGPLAN-SIGACT symposium on Principles of programming languages; 184-196.

Collberg, C.S. and Thomborson, C.D. 2002. Watermarking, tamper-proofing, and obfuscation - tools for software protection. IEEE Transactions on software engineering, 28(8); 735-746.

Cui, C., Dang, Z. and Fischer, T.R. 2011. Typical paths of a graph. Fundamenta Informaticae. 110, 1(4); 95–109.

Dalla Preda, M. 2007. Code obfuscation and malware detection by abstract interpretation. Doctoral dissertation.

Das, S., Liu, Y., Zhang, W. and Chandramohan, M. 2016. Semantics-based online malware detection: Towards efficient real-time protection against malware. IEEE transactions on information forensics and security 11(2); 289-302.

Davis, M., Bodmer, S. and LeMasters, A. 2009. Hacking Exposed Malware and Rootkits. McGraw-Hill, Inc.

Egele, M., Scholte, T., Kirda, E. and Kruegel, C. 2012. A survey on automated dynamic malware-analysis techniques and tools. ACM computing surveys (CSUR) 44(2); 1-42.

Eilam, E. 2011. Reversing: Secrets of Reverse Engineering. Hoboken, NJ, USA: Wiley Publishing.

Engel, K. 2015. Which Computer Viruses Caused The Most Damage Around The World?: http://www.whoishostingthis.com/blog/2015/06/01/8-worst-viruses/, Erişim Tarihi: 25.01. 2020.

Fernandez, E.B. 2004. A Methodology for Secure Software Design. Software Engineering Research and Practice; 130-136.

Firdausi, I., Erwin, A. and Nugroho, A.S. 2010. Analysis of machine learning techniques used in behavior-based malware detection. In 2010 IEEE Second international conference on advances in computing, control, and telecommunication Technologies; 201-203.

Fossi, M., Egan, G., Haley, K., Johnson, E., Mack, T., Adams, T., ... and Wood, P. 2011. Symantec internet security threat report trends for 2010.

Fredrikson, M., Jha, S., Christodorescu, M., Sailer, R. and Yan, X. 2010. Synthesizing near-optimal malware specifications from suspicious behaviors. IEEE Symposium on Security and Privacy (SP); 45-60.

Fu, C., Liu, X.Y., Yang, J., Yang, L.T., Yu, S. and Zhu, T. 2017. Wormhole: The hidden virus propagation power of a search engine in social networks. IEEE Transactions on Dependable and Secure Computing.

Fukushima, Y., Sakai, A., Hori, Y. and Sakurai, K. 2010. A behavior based malware detection scheme for avoiding false positive. 6th IEEE Workshop on Secure Network Protocols; 79-84.

Gazet, A. 2010. Comparative analysis of various ransomware virii. Journal in computer virology 6(1); 77–90.

Griffin, K., Schneider, S., Hu, X., and Chiueh, T. C. 2009. Automatic generation of string signatures for malware detection. In International workshop on recent advances in intrusion detection, Springer, Berlin, Heidelberg; 101-120.

Grosse, K., Papernot, N., Manoharan, P., Backes, M., and McDaniel, P. 2016. Adversarial perturbations against deep neural networks for malware classification. arXiv preprint arXiv:1606.04435.

Hada, S. 2000. Zero-knowledge and code obfuscation. International Conference on the Theory and Application of Cryptology and Information Security. Springer, Berlin, Heidelberg; 443-457.

Hahn, K. 2014. Robust static analysis of portable executable malware. Master thesis, HTWK Leipzig; 134.

Han, J., Pei, J. and Kamber, M. 2011. Data mining: concepts and techniques. Elsevier.

Hogben, G. 2007. Security issues and recommendations for online social networks. Position paper, European Network and Information Security Agency (ENISA) 1; 1-36.

Holzer, A., Kinder, J., and Veith, H. 2007. Using verification technology to specify and detect malware. In International Conference on Computer Aided Systems Theory, Springer, Berlin, Heidelberg; 497-504.

Huang, W., and Stokes, J. W. 2016. MtNet: a multi-task neural network for dynamic malware classification. In International conference on detection of intrusions and malware, and vulnerability assessment, Springer, Cham; 399-418.

Islam, R., Tian, R., Batten, L.M. and Versteeg, S. 2013. Classification of malware based on integrated static and dynamic features. Journal of Network and Computer Applications 36(2); 646-656.

Jang-Jaccard, J. and Nepal, S. 2014. A survey of emerging threats in cybersecurity. Journal of Computer and System Sciences 80(5); 973-993.

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

Karri, R., Rajendran, J., Rosenfeld, K. and Tehranipoor, M. 2010. Trustworthy hardware: Identifying and classifying hardware trojans. Computer 143(10); 39-46.

Kasama, T., Yoshioka, K., Inoue, D. and Matsumoto, T. 2012. Malware detection method by catching their random behavior in multiple executions. IEEE/IPSJ 12th International Symposium on Applications and the Internet; 262-266

Kassambara, A. 2018. Data clustering basics: https://www.datanovia.com/en/lessons/clustering-distance-measures/, Erişim Tarihi: 11.02.2020.

Ki, Y., Kim, E. and kim, H.K. 2015. A novel approach to detect malware based on API call sequence analysis. International Journal of Distributed Sensor Networks 11(6); 659101.

Kinder, J., Katzenbeisser, S., Schallhart, C., and Veith. 2008. Proactive detection of computer worms using model checking. IEEE transactions on dependable and secure computing, 7(4); 424-438.

Kolbitsch, C., Comparetti, P. M., Kruegel, C., Kirda, E., Zhou, X.Y. and Wang, X. 2009. Effective and Efficient Malware Detection at the End Host. USENIX security symposium 4; 351-366.

Kolosnjaji, B., Demontis, A., Biggio, B., Maiorca, D., Giacinto, G., Eckert, C., and Roli, F. 2018. Adversarial malware binaries: Evading deep learning for malware detection in executables. In IEEE 26th European Signal Processing Conference (EUSIPCO); 533-537).

Krister, K.M. 2009. Automated analyses of malicious code. Master's thesis. Institutt for datateknikk og informasjonsvitenskap.

Kwon, T. and Su, Z. 2011. Modeling high-level behavior patterns for precise similarity analysis of software. IEEE 11th International Conference on Data Mining (ICDM); 1134-1139.

Lachow, I. 2013. Active cyber defense: A framework for policymakers. Center for a New American Security.

Lanzi, A., Balzarotti, D., Kruegel, C., Christodorescu, M. and Kirda, E. 2010. Accessminer: using system-centric models for malware protection. Proceedings of the 17th ACM conference on Computer and communications security; 399-412.

Lashkari, A.H., Kadir, A.F.A., Gonzalez, H., Mbah, K.F. and Ghorbani, A.A. 2017. Towards a network-based framework for android malware detection and characterization. Proceeding of the 15th international conference on privacy, security and trust and Trust (PST); 233-23309.

Lewis, T.G. 2019. Critical infrastructure protection in homeland security: defending a networked nation. John Wiley & Sons.

Li, J., Sun, L., Yan, Q., Li, Z., Srisa-an, W. and Ye, H. 2018. Significant permission identification for machine-learning-based android malware detection. IEEE Transactions on Industrial Informatics 14(7); 3216-3225.

Ligh, M., Adair, S., Hartstein, B. and Richard, M. 2010. Malware analyst's cookbook and DVD: tools and techniques for fighting malicious code. Wiley Publishing.

Linn, C. and Debray, S. 2003. Obfuscation of executable code to improve resistance to static disassembly. In Proceedings of the 10th ACM conference on Computer and communications security; 290-299.

Liu, S.T., Huang, H.C. and Chen, Y.M. 2011. A system call analysis method with mapreduce for malware detection. IEEE 17th International Conference on Parallel and Distributed Systems (ICPADS); 631-637.

Liu, B. and Sandhu, R. 2015. Fingerprint-based detection and diagnosis of malicious programs in hardware. IEEE Transactions on Reliability, 64(3), 1068-1077.

Liu, G., Wang, C., Peng, K., Huang, H., Li, Y. and Cheng, W. 2019. Socinf: Membership inference attacks on social media health data with machine learning. IEEE Transactions on Computational Social Systems 6(5); 907-921.

Lo, D., Khoo, S.C. and Liu, C. 2007. Efficient mining of iterative patterns for software specification discovery. Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining; 460-469.

Lo, D., Cheng, H., Han, J., Khoo, S. C. and Sun, C. 2009. Classification of Software Behaviors for Failure Detection. A Discriminative Pattern Mining Approach. In Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining; 557-566.

Love, J. 2018. A Brief History of Malware—Its Evolution and Impact: https://www.lastline.com/blog/history-of-malware-its-evolution-and-impact/, Erişim Tarihi 20.01.2020.

Luo, W., Liu, J., Liu, J. and Fan, C. 2009. An analysis of security in social networks. Eighth IEEE International Conference on Dependable, Autonomic and Secure Computing (DASC); 648–651.

Ma, Z., Ge, H., Liu, Y., Zhao, M. and Ma, J. 2019. A Combination Method for Android Malware Detection Based on Control Flow Graphs and Machine Learning Algorithms. IEEE Access 7; 21235-21245.

Mal share: https://malshare.com, Erişim Tarihi: 07.02.2020.

Martignoni, L., Paleari, R., and Bruschi, D. 2009. A framework for behavior-based malware analysis in the cloud. In International Conference on Information Systems Security, Springer, Berlin, Heidelberg; 178-192.

Maurya, H. 2010. Microsoft security Development lifecycle: http://techsurface.com/2010/01/microsoft-security-development-lifecycle-sdl.html. Erişim Tarihi: 10.01.2020.

McGraw, G. 2002. Building secure software: better than protecting bad software. IEEE Software 19(6); 57-58.

Meland, P.H. and Jensen, J. 2008. Secure software design in practice. IEEE Third International Conference on Availability, Reliability and Security; 1164-1171.

Mirza, Q.A., Awan, I. and Younas, M. 2018. CloudIntell: An intelligent malware detection system. Future Generation Computer Systems 86; 1042-1053.

Morgan, S. 2019. Cybersecurity Almanac: 100 Facts, Figures, Predictions and Statistics. Cisco and Cybersecurity Ventures: https://cybersecurityventures.com/cybersecurity-almanac-2019. Erişim Tarihi: 03.01.2020.

Narayanan, A., Chandramohan, M., Chen, L. and Liu, Y. 2017. Context-aware, adaptive, and scalable android malware detection through online learning. IEEE Transactions on Emerging Topics in Computational Intelligence 1(3); 157-175.

Naval, S., Laxmi, V., Rajarajan, M., Gaur, M.S. and Conti, M. 2015. Employing program semantics for malware detection. IEEE Transactions on Information Forensics and Security 10(12); 2591-2604.

Nazario, J. 2004. Defense and Detection Strategies against Internet Worms. Artech House.

Neagu, C. 2018. What are Windows services, what do they do and how do you manage them: https://www.digitalcitizen.life/what-are-windows-services-what-they-do-how-manage-them, Erişim Tarihi 03.02.2020.

Norman, A.S.A. 2004. The norman book on computer viruses.

O'Brien, D. 2017. Internet Security Threat Report (ISTR) on Ransomware. Technical Report.Symantec: https://docs.broadcom.com/docs/istr-ransomware-2017-en, Erişim Tarihi: 18.01.2020.

Open malware benchmark: http://malwarebenchmark.org/, Erişim Tarihi: 06.02.2020.

Padhy, R.P., Patra, M. R. and Satapathy, S. C. 2011. Cloud computing: security issues and research challenges. International Journal of Computer Science and Information Technology & Security (IJCSITS) 1(2); 136-146.

Park, Y., Reeves, D.S. and Stamp, M. 2013. Deriving common malware behavior through graph clustering. Computers & Security 39; 419-430.

Pilling, R. 2013. Global threats, cyber-security nightmares and how to protect against them. Computer Fraud & Security 2013(9); 14-18.

Potlapally, N. 2011. Hardware security in practice: Challenges and opportunities. IEEE International Symposium on Hardware-Oriented Security and Trust (HOST); 93-98.

Pratama, A. and Rafrastara, F.A. 2012. Computer worm classification. International Journal of Computer Science and Information Security 10(4); 21.

Ramilli, M. 2016. Malware training sets: A machine learning dataset for everyone.

Ronen, R., Radu, M., Feuerstein, C., Yom-Tov, E. and Ahmadi, M. 2018. Microsoft malware classification challenge. arXiv preprint arXiv:1802.10135.

Russinovich, M.E. and Solomon, D.A. 2004. Microsoft Windows Internals: Microsoft Windows Server 2003, Windows XP, and Windows 2000. Microsoft Press.

Samani, R. and Davis, G. 2019. McAfee Mobile Threat Report Q: https://www.mcafee.com/enterprise/en-us/assets/reports/rp-mobile-threat-report-2019.pdf. Erişim Tarihi: 05.01.2020.

Samet, R. ve Aslan, Ö. 2018. Bölüm 8: Kötü Amaçlı Yazılımlar ve Analizi (225-251). Sağıroğlu, Ş. ve Alkan, M (Ed.) Siber Güvenlik ve Savunma Farkındalık ve Caydırıcılık, ISBN: 978-605-2233-22-1.

Santos, I., Laorden, C. and Bringas, P.G. 2011. Collective classification for unknown malware detection. In Proceedings of the International IEEE Conference on Security and Cryptography; 251-256.

Santos, I., Brezo, F., Ugarte-Pedrero, X. and Bringas, P.G. 2013. Opcode sequences as representation of executables for data-mining-based unknown malware detection. Information Sciences 231; 64-82.

Saracino, A., Sgandurra, D., Dini, G. and Martinelli, F. 2018. Madam: Effective and efficient behavior-based android malware detection and prevention. IEEE Transactions on Dependable and Secure Computing 15(1); 83-97.

Saxe, J., and Berlin, K. 2015. Deep neural network based malware detection using two dimensional binary program features. In IEEE 10th International Conference on Malicious and Unwanted Software (MALWARE); 11-20.

Schultz, M.G., Eskin, E., Zadok, F. and Stolfo, S.J. 2001. Data mining methods for detection of new malicious executables. IEEE Symposium on Security and Privacy (S&P); 38-49.

Shabtai, A., Moskovitch, R., Elovici, Y. and Glezer, C. 2009. Detection of malicious code by applying machine learning classifiers on static features: A state-of-the-art survey. information security technical report 14(1); 16-29.

Shahid, A. 2014. A framework for metamorphic malware analysis and real-time detection. Doctoral dissertation, University of Victoria Engineering and Technology Lahore.

Shan, Z. and Wang, X. 2014. Growing grapes in your computer to defend against malware. IEEE Transactions on Information Forensics and Security 9(2); 196-207.

Shannon, C.E. and Weaver, W. 1949. The Mathematical Theory of Communication. Univ. of III Press. Urbana, 20.

Sheldon, F.T. and Vishik, C. 2010. Moving toward trustworthy systems: R&D Essentials. Computer 43(9); 31-40.

Siddiqui, M.A. 2008. Data mining methods for malware detection. University of Central Florida.

Sikorski, M. and Honig, A. 2012. Practical malware analysis: the hands-on guide to dissecting malicious software. no starch press.

Singh, A., Arora, R. and Pareek, H. 2017. Malware Analysis using Multiple API Sequence Mining Control Flow Graph. arXiv preprint arXiv:1707.02691.

Singh, N.K. and Tomar, D.S. 2019. Privacy Preservation of Social Media Services: Graph Prospective of Social Media. Censorship, Surveillance, and Privacy: Concepts, Methodologies, Tools, and Applications. IGI Global; 473-501.

Song, F., and Touili, T. 2014. Pushdown model checking for malware detection. International Journal on Software Tools for Technology Transfer, 16(2), 147-173.

Spencer, S. 2011. "Timeline of Computer Viruses" https://www.mapcon.com/us-en/timeline-of-computer-viruses, Erişim Tarihi 20.01.2020.

Spinellis, D. 2003. Reliable identification of bounded-length viruses is NP-Complete. IEEE Transactions on Information Theory 49(1); 280-284.

Stallings, W. 2006. Cryptography and Network Security. Pearson Education, India.

Stallings, W. and Brown. L. 2012. Computer security: principles and practice. Upper Saddle River, Pearson Education. NJ, USA.

Stout, D.W. 2020. Social Media Statistics 2020: Top Networks by the Numbers. Web Sitesi: https://dustinstout.com/social-media-statistics/#facebook-stats. Erişim Tarihi: 10.01.2020.

Su, J., Vasconcellos, V.D., Prasad, S., Daniele, S., Feng, Y. and Sakurai, K. 2018. Lightweight classification of IoT malware based on image recognition. IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC). 2; 664-669.

Sun, H., Wang, X., Buyya, R., and Su, J. 2017. CloudEyes: Cloud- based malware detection with reversible sketch for resource- constrained internet of things (IoT) devices. Software: Practice and Experience, 47(3); 421-441.

Szor, P. and Ferrie, P. 2001. Hunting for metamorphic. In Virus Bulletin Conference.

Szor, P. 2005. The art of computer virus research and defense. Pearson Education.

Tang, Y., Xiao, B., & Lu, X. 2009. Using a bioinformatics approach to generate accurate exploit-based signatures for polymorphic worms. Computers & security, 28(8); 827-842.

Tavallaee, M., Bagheri, E., Lu, W. and Ghorbani, A.A. 2009. A detailed analysis of the KDD CUP 99 data set. IEEE symposium on computational intelligence for security and defense applications; 1-6.

Team, N. 2016. The 8 Most Famous Computer Viruses of All Time. Nort UK Blog: https://uk.norton.com/norton-blog/2016/02/the_8_most_famousco.html, Erişim Tarihi: 25.01. 2020.

Tehranipoor, M. and Wang, C. 2011. Introduction to hardware security and trust. Springer Science & Business Media.

Tekdefense: http://www.tekdefense.com/downloads/, Erişim Tarihi: 08.02.2020.

Thompson, G.R. and Flynn, L.A. 2007. Polymorphic malware detection and identification via context-free grammar homomorphism. Bell Labs Technical Journal 12(3); 139-147.

Tian, R., Islam, R., Batten, L. and Versteeg, S. 2010. Differentiating malware from cleanware using behavioural analysis. IEEE 5th International Conference on Malicious and Unwanted Software (MALWARE); 23-30.

Trinius, P., Holz, T., Göbel, J. and Freiling, F.C. 2009. Visual analysis of malware behavior using treemaps and thread graphs. IEEE 6th International Workshop on Visualization for Cyber Security (VizSec); 33-38.

ViruSign: http://www.virusign.com/, Erişim Tarihi: 07.02.2020.

Virustotal: https://www.virustotal.com, Erişim Tarihi: 09.02.2020.

Wagener, G., State, R. and Dulaunoy, A. 2008. Malware behaviour analysis. Journal in computer virology 4(4); 279-287.

Wang, W., Zhao, M. and Wang, J. 2019. Effective android malware detection with a hybrid model based on deep autoencoder and convolutional neural network. Journal of Ambient Intelligence and Humanized Computing 10(8); 3035-3043.

Williams, M. 2016. What's really running on your PC? Part 1: Windows System Processes: https://betanews.com/2015/10/21/whats-really-running-on-your-pc-part-1-windows-system-processes/, Erişim Tarihi 28.01.2020.

Wood, P., Nahorney, B., Chandrasekar, K., Wallace, S. and Haley, K. 2016. Symantec internet security threat report. Symantec Corporation, Tech. Rep., 21: https://docs.broadcom.com/docs/istr-21-2016-en /, Erişim Tarihi: 12.01.2020.

Xiao, L., Li, Y., Huang, X., and Du, X. 2017. Cloud-based malware detection game for mobile devices with offloading. IEEE Transactions on Mobile Computing, 16(10); 2742-2750.

Yadav, R.M. 2019. Effective analysis of malware detection in cloud computing. Computers & Security 83; 14-21.

Yan, W., Zhang, Z. and Ansari, N. 2008. Revealing packed malware. IEEE security & Privacy 6(5); 65-69.

Ye, Y., Wang, D., Li, T. and Ye, D. 2007. IMDS: Intelligent malware detection system. In Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining; 1043-1047.

Ye, Y., Li, T., Jiang, Q., and Wang, Y. 2010. CIMDS: adapting postprocessing techniques of associative classification for malware detection. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 40(3); 298-307.

Ye, Y., Chen, L., Hou, S., Hardy, W., and Li, X. 2018. DeepAM: a heterogeneous deep learning framework for intelligent malware detection. Knowledge and Information Systems, 54(2); 265-285.

Yousefi-Azar, M., Hamey, L.G., Varadharajan, V. and Chen, S. 2018. Malytics: a malware detection scheme. IEEE Access 6; 49418-49431.

Yuan, Z., Lu, Y., Wang, Z., and Xue, Y. 2014. Droid-sec: deep learning in android malware detection. In Proceedings of the ACM conference on SIGCOMM; 371-372.

Zhang, H., Zhang, W., Lv, Z., Sangaiah, A.K., Huang, T. and Chilamkurti, N. 2019. MALDC: a depth detection method for malware based on behavior chains. World Wide Web; 1-20.

Zimba, A., Wang, Z. and Chen, H. 2018. Multi-stage crypto ransomware attacks: A new emerging cyber threat to critical infrastructure and industrial control systems. Ict Express 4(1); 14-18.

Zolkipli, M.F. and Jantan, A. 2010. A framework for malware detection using combination technique and signature generation. IEEE Second International Conference on Computer Research and Development; 196-199.

Zuo, Z., Zhu, Q. and Zhou, M. 2005. On the time complexity of computer viruses. IEEE Transactions on information theory 51(8); 2962-296.

ÖZGEÇMİŞ

Adı Soyadı : Ömer ASLAN
Doğum Yeri : Kilis
Yabancı Dil : İngilizce, Rusça

Eğitim Durumu

Lise : Kilis Yabancı Dil Ağrlıklı Lise (2003)

Lisans : Trakya Üniversitesi Mühendislik-Mimarlık Fakültesi Bilgisayar Mühendisliği (2009)

Yüksek Lisans : Teksas Üniversitesi Bilgisayar Bilimleri (08/2012-12/2014)

Doktora : Ankara Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalı (02/2015-07/2020)

Çalıştığı Kurum

Siirt Üniversitesi Mühendislik Fakültesi Bilgisayar Mühendisliği Bölümü-Araştırma Görevlisi (2015-devam ediyor)

Yayınlar

Makale (SCI/SCI-Expanded)

1. Aslan, Ö. and Samet, R. 2020. A Comprehensive Review on Malware Detection Approaches. IEEE Access 8; 6249-6271.

2. Aslan, Ö., Samet, R. and Tanrıöver, Ö.Ö. 2020. Using a Subtractive Center Behavioral Model to Detect Malware. Security and Communication Networks, 2020.

Uluslararası Kongre Sunum

1. Aslan, Ö. and Samet, R. 2017a. Investigation of Possibilities to Detect Malware Using Existing Tools. IEEE/ACS 14th International Conference on Computer Systems and Applications (AICCSA); 1277-1284.

2. Aslan, Ö. and Samet, R. 2017b. Mitigating cyber security attacks by being aware of vulnerabilities and bugs. IEEE 2017 International Conference on Cyberworlds (CW); 222-225.

3. Aslan, Ö. 2017. Performance Comparison of Static Malware Analysis Tools Versus Antivirus Scanners to Detect Malware. In International Multidisciplinary Studies Congress (IMSC).

4. Aslan, Ö. 2016. How to decrease cyber threats by reducing software vulnerabilities and bugs. 1st International Mediterranean science and engineering congress, Çukurova University 210; 639–646.

Kitap Bölümü

1. Samet, R. ve Aslan, Ö. 2018. Bölüm 8: Kötü Amaçlı Yazılımlar ve Analizi (225-251).Sağıroğlu, Ş. ve Alkan, M (Ed.) Siber Güvenlik ve Savunma Farkındalık ve Caydırıcılık, ISBN: 978-605-2233-22-1.