Melez Özelliklerin Modellenmesi ile Zararlı Yazılım Tespiti
(Malware Detectionby Using Markov Models of Hybrid Features)

Mert NAR
Gebze Teknik Üniversitesi
m.nar@gtu.edu.tr
ORCID: 0000-0002-6103-2909

İbrahim SOĞUKPINAR
Gebze Teknik Üniversitesi
ispinar@gtu.edu.tr
ORCID: 0000-0002-0408-0277

Öz

Zararlı yazılımlar sahip oldukları yeteneklerden ötürü bilgisayar ve sistemlere büyük tehlike oluşturmaktadır. Etkin tespit sistemlerinin gelişmesinden aynı şekilde etkilenerek daha tehlikeli ve donanımlı hale gelmektedirler. Otomatik bir tespit sistemi geliştirmek için, zararlı yazılımlar iyi analiz edilmeli ve gelişim meyilleri doğru tespit edilmelidir. Zararlı yazılımların çalıştığı bilgisayarda yarattığı etkiler ve kod yapısı ayrıntılı incelenmeli ve öyle önlem alınmalıdır. Bu çalışmada önerilen tespit sistemi, zararlı yazılımın hem davranış hem kod yapısı bilgisini kullanarak Markov zinciri yöntemi ile istatistiksel bir anlam çıkarmaktadır. Daha sonra derin öğrenme teknikleri ile temellendirilmiş model melez veri kaynağı ile eğitilmiş ve tespit ortamı hazırlanmıştır. Yaptığımız testler sonucunda önerilen tespit yöntemi %96,8'lik doğruluk göstermiştir.

***Anahtar Sözcükler**: Zararlı Yazılımlar, Markov Zinciri, Derin öğrenme,* Evrişimsel, *Dinamik Analiz, Statik Analiz*

Abstract

Malware poses a great danger to computers and systems due to their capabilities. They are also affected by the development of effective detection systems and become more dangerous and equipped. In order to develop an automated detection system, malware must be well analyzed, and inclination of their evolution should be accurately understood. The runtime effects of malicious software on the computer and code structure should be examined in detail and precautions should be taken. The detection system proposed in this study makes a statistical meaning with Markov chain method using both behavior and code structure knowledge of malware. Then the model based on deep learning techniques is trained with the hybrid data source and detection environment is prepared. As a result of the tests we performed, the accuracy of the detection method was 96.8%.

***Keywords:** Malware, Markov Chain, Deep learning, Convolutional Neural Network, Dynamic Analysis, Static Analysis,*

Gönderme ve kabul tarihi: 07.10.2019 - 0.12.2019
Makale türü: Araştırma

1. Giriş

Zararlı yazılımlar (malware), günümüzde sahip oldukları yeteneklerden dolayı siber güvenlik alanında en büyük güvenlik tehditlerinden biridirler. Öyleki, imza tabanlı zararlı yazılım tespit araçlarını kolaylıkla atlatabilir, kendilerini iyi huylu yazılımlara ekleyebilir, birçok farklı şekle bürünebilir, güvenli olarak bilinen programları taklit edebilir ve gerçek yüzünü tetiklenene kadar gizleyebilirler. Bir başka boyutu ise zararlı yazılımların, şekil değiştirme teknikleri ve karanlık alemlerde var olan zararlı yazılım üretme araçları sayesinde internette dolaşan zararlı yazılım çeşitliliği ve sayısı her geçen gün artmaktadır. Malware bytesLabs [1] tarafından yayınlanan rapora göre, 2018 yılında yaklaşık 750,3 milyon kötü amaçlı yazılım tespit edilmiştir. Ayrıca, bir başka rapora göre [2] her gün 350.000'in üzerinde yeni kötü amaçlı programın internete salındığı bildirilmiştir. Bu raporlar göz önünde bulundurulup genel bir değerlendirme yapacak olursak, saldırganlarla savunmacılar arasında bir silahlanma yarışı olduğu söylenebilir. Zararlı yazılım yazarlarının, zararlı yazılım yapım kitleri (malware construction kit), kaçınma teknikleri, mevcut zararlı yazılım kaynak kodları vb. gibi silahları vardır [3]. Dolayısıyla, kötü amaçlı yazılım saldırılarının yayılmasını ve yaygınlaşmasını, imza tabanlı tespit sistemleri kullanarak kovuşturmak ve engellemek mümkün değildir. İçerisinde bulunduğumuz bu yarışta, mücadeleye devam edebilmek için savunucuların zararlı yazılım tespit teknikleri için geleneksel tespit sistemi yöntemlerini tek yol olarak kullanmaktan vazgeçmesi gerekir. Zararlı

yazılımların imza eşleştirme yöntemi ile tespit edilmesinin yerine niteleyici ortak örüntü bularak tespit edilmesi gerekir. Bu nedenle, literatürde benzer örüntü arayan pek çok zararlı yazılım tespit yöntemi önerilmiştir. [3,4,5,6].

Benzer örüntü bulmak hususunda düşünülecek ilk konu zararlı yazılım analizidir [7]. Zararlı yazılım analizi ile, işlem kodları (OpCodes), uygulama programlama arabirim (API) fonksiyonlarının çağrımı, davranış bilgileri, başlık bilgileri, çalışma süresi, üzerinde çalıştığı bilgisayardaki etkiler gibi öznitelikler çıkarılır. Yazılım analizi genel olarak yazılım davranışı ve kod yapısı hakkında bilgi vermektedir. Bu bilgiyi otomatize bir şekilde çıkarmak ve öğrenmek için analiz ile öznitelikler çıkarılır. Öznitelikler zararlı yazılımın ne yaptığını ve nasıl kullanıldığını açıklar. Kötü amaçlı yazılım analizi, statik ve dinamik olmak üzere iki yaklaşımla gerçekleştirilir. Statik teknikler, ikili kodu çalıştırmadan analiz eder; dinamik teknikler ise zararlı yazılım kontrollü bir ortamda çalışırken bilgisayarda yarattığı etkiyi gözlemleyerek davranışlarını incelemeye dayanır [8]. Bu yaklaşımlardan herhangi biri tek başına uygulanması zararlı yazılımlar hakkında bilgi kaybına sebebiyet verebilir [9]. Çünkü, farklı öznitelik tipleri yazılımın farklı bir yönünü göstermektedir ve belki o bakış açısı zararlı yazılımı açık eder. Yani, her iki analiz türünde de bazı avantajlar ve dezavantajlar vardır [10].

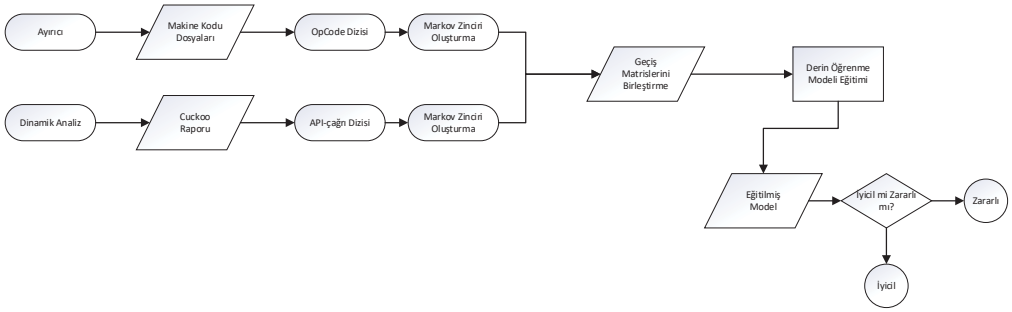
Silahlanma yarışının diğer tarafından bakıldığında, saldırganların ve zararlı yazılım yazarlarının analizden kaçınmak için kullandıkları yöntemler vardır. Durağan analizden kaçınmak için en popüler ve güçlü yöntem kendi kendini değiştirme yöntemidir. Zararlı yazılım kendi kodunda yapısal değişiklikler yaparak farklı bir imza ve kod yapısına sahip benzerlerini üretebilir [11]. Böylece durağan analiz yapan tespit sistemleri ve geleneksel imza tabanlı tespit yöntemleri atlatılabilir. Diğer taraftan, bazı yöntemler ile (hedefleme, tersine ayar testi, erteleme, tetikleme bazlı kodlama ve dosyasız (AVT) saldırı gibi [7]) dinamik analizden de kaçmak mümkündür. Analizden kaçınmak bir yana, Demetrio ve diğerleri [12] belirttiği gibi derin öğrenme teknikleri uygulanmış ileri düzey zararlı yazılım tespit sistemlerini aldatmak da mümkündür. Bu aldatma öznitelik mühendisliği yapılmadığı durumlarda gerçekleşir. İşlenmemiş ham veri ile eğitilmiş derin öğrenme modelleri, literatürde ortaya çıkarılan taktikler ile kandırılabileceği gösterilmiştir [13]. Bu yüzden zararlı yazılımlar ve aileleri dikkatlice incelenmelidir.

Bu çalışmanın amacı Windows ortamında çalışan zararlı yazılımların tespitini yapmaktır. Bunun için zararlı yazılımlar hem dinamik hem durağan analiz yöntemleriyle incelenmiştir. Böylece statik veya dinamik analiz sınırlamaları ortadan kalkmaktadır. Çalıştırılabilir dosyanın kod yapısı ve çalışma zamanı davranış bilgileri istatistiksel öznitelik olarak ele alınmaktadır. Sistem tüm resmi tam anlamıyla yakalayacak şekilde tasarlanmıştır. Yaklaşımdaki özgünlük, istatistiksel özellikleri ön plana çıkaran Markov zinciri geçiş matrisleri ile derin öğrenme tekniği olan evrişimsel sinir ağının (Convolutional Neural Network) birlikte kullanılması ve melez analiz yaparak hem davranış hem kod yapısı verilerinden faydalanılmasıdır. Dinamik analiz ile yazılım davranışlarını, diğer bir deyişle API-çağrım dizilimlerini, durağan analiz ile OpCode dizilimlerini çıkararak istatistiksel bir yöntem olan Markov zincirleri inşa edilmiş, bu Markov zincirleri birleştirilerek eğitme ve tespit yapılmaktadır. Sistem otomatik olarak Windows ortamında çalışacak şekilde geliştirilmiştir. Bu çalışmada önerilen tespit yönteminin temelini hem durağan hem dinamik analiz yaparak çıkarılan öznitelikler ile eğitilmiş derin öğrenme modeli oluşturmaktadır. Böylece derin öğrenme aldatmasından kaçınılmıştır.

Makalenin geri kalanı şu şekilde düzenlenmiştir: 2. Bölüm, önerilen yöntem hakkında arka plan bilgisi sağlarken, bu alandaki ilgili çalışmaları özetlemektedir. Sonraki 3. bölüm yaklaşımımız hakkında detaylı bilgi verirken, gelen Bölüm 4'te deneysel sonuçlar ile değerlendirme ve tartışmalar vardır. Son olarak, Bölüm 5 sonuç ve önerilerdir.

2. İlgili Çalışmalar

Zararlı yazılım analizi, zararlı yazılımın farklı bileşenlerini parçalayarak incelenmesi ve onun davranışı ve kod yapısı üstüne yapılan çalışmaların tümüdür. Zararlı yazılım tespiti ve sınıflandırma üstüne çalışan neredeyse tüm araştırmacılar zararlı yazılım analizi yapmışlardır [4,5,7,8]. Bununla birlikte, Ortaya çıkan yeni türler ile zararlı yazılım türlerini sınıflandırmak veya tespit etmek daha zor hale gelmektedir [12]. Veyahut analizden kaçınma yöntemleri, yapay zekâ teknolojileri ve benzeri yöntemler ile zararlı yazılımlar daha donanımlı hale gelmektedir.



Şekil-1: Bu çalışmada geliştirilen modelin genel gösterimi

Saldırganlar ile Savunucular arasındaki bu silahlanma yarışında, ne yazık ki, kötü niyetli davranış araştırmacıların zararlı davranışların sadece tespitin özniteliği olarak kullandığı, davranışları tespit etmedikleri sürece donanımlı ve önceden bilinmeyen bir zararlının yayılmasını engellemek mümkün değildir. Garetto ve diğerleri [14] ve Chen&Ji [15] Zararlı yazılımın bulaşma ve yayılma davranışı üstüne çalışmışlar, bu davranışı tespit etmek için Markov zinciri ile temellendirilmiş birer model geliştirmişlerdir. Karyotis [16] yine Zararlı yazılım yayılma davranışı üstüne bir model geliştirmiş, fakat onun modeli stokastik bir Teknik olan Markov Rastgele Alanları (Markov Random Fields) yönteminden faydalanmıştır. Sartea ve Farinelli [8] de Markov zincirlerini kullanarak davranış tespit yöntemi öne sürdü. Bu çalışmada yine Markov zinciri yöntemi kullanıldı. Fakat bizim çalışmamızda Markov zinciri bir davranışın istatistiksel analizini yapması yerine, davranış ve kod yapısını belirten dizilim verisini istatistiksel olarak işlemesi için kullanılmıştır. Daha sonra bu işlenmiş veriden anlam çıkarması ve öğrenmesi için derin öğrenme tekniği kullanılmıştır.

Pektaş ve Acarman [4], API çağrı dizilerini kullanarak çalışma zamanı davranış analizine dayanan özgün bir zararlı yazılım sınıflandırma mekanizması geliştirdi. Zararlı yazılımların API çağrı dizisi içinde bir örüntü yakalamak üstüne çalışan Pektaş, bu belirteç örüntüyü n-gram ve oylama uzman algoritması (Voting expert Algorithm) yardımıyla buldu ve eşleştirme yöntemi ile tespit yaptı. Zhang ve diğerleri [17] OpCode n-gramlarını ve dinamik olarak ölçülen API-çağrım sıklıklarını öznitelik olarak kullanan melez bir zararlı yazılım tespit yöntemi geliştirdi. Yöntemin temelini Temel Bileşen Analizi (principal component analysis - PCA) yöntemi ve derin öğrenme anlayışı oluşturur. Vemparala ve diğerleri [5] önerdiği yöntem ise dinamik zararlı yazılım analizi yaparak çıkarılan API-çağrım dizilerini kullanır. Geliştirilen yöntem Hidden Markov Model yöntemi ile geliştirildi. Xiao ve diğerleri [18] ile Onwuzurike ve diğerleri [19] Android güvenliği için derin öğrenme teknikleri ve Markov zinciri tekniğini kullanarak çalışan birer yöntem önerdiler. Bu çalışmada Derin öğrenme ve Markov zinciri tekniklerinden faydalanılmıştır, fakat önerilen yöntem melez öznitelik ile çalışır ve Windows zararlı yazılımlarının tespiti için geliştirilmiştir. [10]'da gösterildiği gibi, melez modeller zararlı yazılımların atlatma ve saklanma tekniklerini alt etmek üzere önemli avantajlar sağlamaktadır. Derin öğrenme tekniğini kullanan bir diğer çalışma Safa ve diğerleri tarafından[6] ile sunulmuştur. Onlar sınıflandırma modelini CNN/RNN ile geliştirmiştir Önerdikleri model melez analiz ile çıkarılan öznitelikleri kullanmaktadır. Derin öğrenmenin bir diğer kullanım yolu da Xiao ve diğerleri [20] tarafından sunulan özgün yöntemde IoT cihazlarını zararlı yazılımlardan önlemesi için kullanılmıştır. Stacked Auto encoders kullanarak geliştirilen yöntem, API-çağrı grafları ile çalışır ve eşleştirme yapılıp eşik değeri aşması ile tespit yapmıştır. [22] ve [23]'de de derin öğrenme yöntemleri kullanılarak tespit yapılmıştır. Zararlı yazılım tespitinde derin öğrenme teknikleri bilinçli ve akıllıca kullanılmalıdır. [11,12]'deki çalışmalar göstermiştir ki derin öğrenme temelli zararlı yazılım sınıflandırma yöntemleri kandırılabilir. Şu anlaşılmaktadır ki, derin öğrenme teknikleri bilgisayar savunmacıları için önemli silahlardır fakat bunların uzman bakışı ve öznitelik mühendisliğine ihtiyaçları vardır. Kolosnjaji ve diğerleri [25] de derin öğrenme tekniklerinin zararlı yazılım tespitinde kullanımı ve tespit çalışmalarındaki etkisi üstüne çalışmıştır.

Çizelge-1: En sık kullanılan API fonksiyonları

Zararlı Yazılım çağrımları		İyi Huylu Yazılım çağrımları	
GetAsyncKeyState	11031128	'NtClose'	79101
'NtReadFile'	3884730	'NtUnmapViewOfSection'	20576
'NtDelayExecution'	1829146	'LdrGetDllHandle'	20091
exception	1508285	'NtClose'	79101
'LdrGetProcedureAddress'	1506665	LdrGetProcedureAddress	18041
'FindWindowA'	1403559	NtWriteFile	15796
'NtWriteFile'	1162551	LoadStringW	13251
'Process32NextW'	1159408	NtFreeVirtualMemory	11154
'RegCloseKey'	1091177	NtTerminateProcess	8399
'ReadProcessMemory'	995228	GetSystemDirectoryW	8226
'FindWindowExA'	957958	NtAllocateVirtualMemory	8111
'NtClose'	889148	GetSystemMetrics	6519
'RegOpenKeyExA'	871264	LdrLoadDll	6338
'GetFileAttributesW'	819933	LdrUnloadDll	6253
'SetFilePointer'	779942	NtQueryValueKey	6249
'NtProtectVirtualMemory'	746158	GetFileAttributesW	6234
'DrawTextExW'	687782	SetUnhandledExceptionFilter	5562
'LdrLoadDll'	632524	UnhookWindowsHookEx	5460
'RegQueryValueExW'	631574	NtCreateFile	5107
'GetForegroundWindow'	589540	RegCloseKey	4690
'RegOpenKeyExW'	582413	RegOpenKeyExW	4486
'RegQueryValueExA'	483729	GetFileSize	4270
'LdrGetDllHandle'	419620	RegQueryValueExW	4132
'FindFirstFileExW'	365860	NtOpenKey	3653
'InternetReadFile'	353925	GetSystemTimeAsFileTime	2935
'GetCursorPos'	311370	CreateActCtxW	2785
'GetSystemMetrics'	300117	MessageBoxTimeoutW	2701
'NtCreateFile'	290325	NtQueryAttributesFile	2686
'GetKeyState'	279108	WriteConsoleA	2596
'NtAllocateVirtualMemory'	259751	NtOpenFile	2427
'EnumWindows'	250398	NtReadFile	2215
'RegSetValueExA'	227628	NtMapViewOfSection	1940
'NtQueryDirectoryFile'	223418	GetKeyState	1504
'CreateProcessInternalW'	177809	GetShortPathNameW	1231
'FindWindowW'	145172	SetFilePointer	1096
'NtFreeVirtualMemory'	121417	RegOpenKeyExA	955
'SetFileAttributesW'	119152	RegQueryValueExA	888
'NtOpenProcess'	104315	FindFirstFileExW	807
'RegEnumValueA'	101386	GetFileType	541
'LoadStringW'	101188	NtQueryDirectoryFile	519
'recv'	92812	NtQueryInformationFile	513
'NtOpenFile'	87392	NtCreateSection	388
'SHGetFolderPathW'	81316	GetSystemDirectoryA	387
'GetSystemDirectoryA'	76916	NtDuplicateObject	345
'NtQueryInformationFile'	70926	DrawTextExW	296
'NtQueryValueKey'	65889	GetSystemWindowsDirectoryA	264
'GetSystemTimeAsFileTime'	65109	DeleteFileW	233
'LoadResource'	64747	NtProtectVirtualMemory	212
'NtOpenKey'	60113	NtQuerySystemInformation	211
'SetErrorMode'	58527	FindResourceExW	199
'SHGetSpecialFolderLocation'	58473	CreateThread	189
'FindResourceExW'	57596	LoadResource	189

Onun çalışmasına göre, derin öğrenme bazı kurtulma saldırıları ile kandırılabilir. Bu yüzden onun çalışmasına göre, Lee ve diğerleri [24]'ün yaptığı gibi ham data kullanan derin öğrenme temelli tespit sistemleri zafiyet göstermektedir. Kolosnjaji ve diğerleri [26] bir başka çalışmasında ise CNN/RNN katmanlarından oluşan yapay sinir ağı kurmuş ve sistem fonksiyon çağrımlarıyla çalışan bir model geliştirmiştir.

3. Önerilen Yöntem

3.1 Genel Bakış

Aynı aileye ait olan zararlı yazılımlar benzer davranışlar göstermekte ve kod yapılarında benzerliklere rastlanmaktadır [17]. Bu çalışmada istatistiksel veri kaynağı ve derin öğrenme teknikleri kullanılarak geliştirilmiş zararlı yazılım tespit yöntemi önerilmiştir. Çalışmanın hedefi zararlı davranışa sebep olabilecek yapı ve davranış örüntüsü bulmaktır. Bunun için, davranış analizi API-çağrım dizileri ile ve yapı analizi ise Operasyon kodu (OpCode) dizileri ile incelenmiştir. Bu incelemeler sonucunda istatistiksel bir yöntem olan Markov zinciri ile anlamlı veri elde edilebileceği görülmüştür. Her iki diziden çıkarılmış zincir çıktıları birleştirilerek, derin öğrenme yöntemi ile geliştirilen model eğitilmiştir. Eğitim modeli evrişimli sinir ağları (Convolutional neural network CNN), maksimum havuzlama (max-pooling) ve yoğunluk katmanları (dense-layer) içermektedir. Bu sayede, API-çağrı çiftinin olasılık değeri kombinasyonunu kullanıp iç değişkenleri optimize etmek için hesaplama yapan bir karar modeli eğitilir. Her modülün ve ilişkinin detaylı açıklaması verilmiştir.

Yöntem 4 adımdan oluşmaktadır. İlk olarak zararlı yazılım analizleri yapılmaktadır. Güvenli bir ortamda yapılan dinamik analiz ile yazılım numunelerinin davranışları görüntülenmekte vejson formatında raporlanmaktadır. Ayırıcı (disassembler) ile de numunelerin tümleşke (assembly) kodu çıkarılmıştır. Bu analiz raporlarından API-çağrım dizisi ve OpCode dizisi çıkarılmaktadır. Her iki diziden Markov zinciri oluşturulur ve geçiş matrisleri birleştirilir. Birleştirilmiş matrisler derin öğrenme modelini eğitmek için kullanılmaktadır. Test sürecinde ise yine zararlı yazılımın analizi yapılıp birleştirilmiş geçiş matrisi oluşturulmuştur. Geçiş matrisini girdi olarak alan model, zararlı yazılım olup olmadığına karar vermektedir. Her modülün ve ilişkinin detaylı açıklaması sonraki bölümlerde verilmektedir.

3.2 Analiz

Uygulama programlama arayüzü (API) işletim sistemi tarafından sunulan işlem kümesidir. API işlem çağrımları sayesinde ağ yönetimi, bellek yönetimi, kayıt defteri operasyonları, dosya giriş/çıkışları, proses ve izlekler ile alakalı işlemler yapılabilmektedir. Her bir çağrım, sistem için ayrı bir işlemi temsil etmektedir. O çağrım sonucunda, çağrının girdileri ile bilgisayarda bazı değişiklikler meydana gelir. Bu yüzden API-çağrım dizilimi mikro boyutta yazılım davranışı hakkında bilgi vermektedir. Yazılım davranışının zararlı olması, bilgisayarda istenmeyen etkilere sebep olması anlamına gelmektedir [27]. Bunu ölçmek ve zararlı yazılımı tespit etmek için, API-çağrımları ile davranışı anlamlandırmanın doğru olacağı düşünülmüş ve bu düşünce çalışmanın temelini oluşturmuştur.

Çizelge-2: Zararlı davranışlar ve o davranış için yapılan API çağrımları

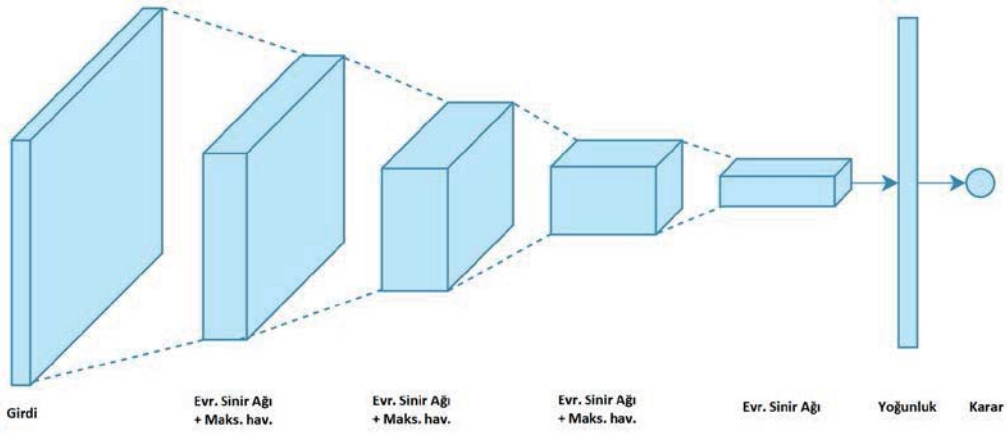
Zararlı Davranış	API çağrım Dizisi
İndirme ve Çalıştırma	URLDownloadToFile, ShellExecute
TCP Port bağlantısı	WSAStartup, socket
Ağ Trafiği izleme	Socket, bind, WSAloctl, recvfrom
Kendini Silme	GetModuleFileName, ExitProcess, DeleteFile
Açıldığında Aktif Olma	RegOpenKeyExW, RegQueryValueExW, RegCloseKey

Önerilen tespit sisteminin ilk ve en önemli ayağını dinamik analiz oluşturmaktadır. Dinamik analiz kum havuzu yardımıyla yapılmaktadır. Bu çalışmada kullanılan veri kümesindeki tüm numuneler sırayla güvenilir ortamda çalıştırılır ve çalışan bilgisayar izlenir. Bu gözetleme işlemi sonunda bilgisayarda olan tüm değişiklikler ve yazılımın çalışma anı hareketleri raporlanır. Sistem bu raporlardan API-çağrım dizilerini çıkarmaktadır.

Çizelge-3: En sık kullanılan Komutlar uzayı

'MOV', 'ADD', 'PUSH', 'DEC', 'POP', 'INC', 'DB', 'XCHG', 'CMP', 'XOR', 'OR', 'AND', 'SUB', 'ADC', 'SBB', 'CALL', 'TEST', 'JMP', 'INT', 'OUT', 'IN', 'JZ', 'OUTS', 'RET', 'IMUL', 'LEA', 'INS', 'JNZ', 'RETF', 'STD', 'NOP', 'JB', 'JAE', 'POPA', 'JO', 'ARPL', 'MOVSD', 'JP', 'JS', 'JLE', 'AAS', 'JL', 'JNS', 'CLD', 'JG', 'JBE', 'JA', 'JGE', 'CLC', 'PUSHA', 'LOOPNZ', 'AAA', 'STC', 'STI', 'JNP', 'INT1', 'CLI', 'JNO', 'CDQ', 'HLT', 'DAS', 'IRET', 'AAM', 'DAA', 'CMC', 'LEAVE', 'XLAT', 'ENTER', 'LOOPZ', 'SCASD', 'JECXZ', 'LOOP', 'LAHF', 'SALC',	'CWDE', 'AAD', 'INTO', 'PUSHF', 'WAIT', 'POPF', 'STOSD', 'SAHF', 'LODSB', 'CMPSD', 'SCASB', 'MOVSB', 'LODSD', 'STOSB', 'CMPSB', 'BOUND', 'ROL', 'SAR', 'SHL', 'SHR', 'RCL', 'ROR', 'SAL', 'RCR', 'LES', 'LDS', 'FLD', 'FSTP', 'FILD', 'NEG', 'FISTP', 'FISTTP', 'DIV', 'IDIV', 'FADD', 'NOT', 'FDIVR', 'FDIV', 'FMUL', 'MUL', 'FSUBR', 'FSUB', 'FCOMP', 'FCOM', 'FST', 'FIADD', 'MOVZX', 'FIDIVR', 'FIST', 'FIMUL', 'FICOMP', 'FISUBR', 'FIDIV', 'FICOM', 'FISUB', 'FNSTSW', 'MOVAPS',	'LOCK', 'FBSTP', 'FBLD', 'FNSTCW', 'FLDENV', 'FLDCW', 'FNSTENV', 'FRSTOR', 'FNSAVE', 'REP', 'REPNZ', 'MOVUPS', 'MOVS', 'CMPS', 'LODS', 'BSWAP', 'MULPS', 'MOVSX', 'FADDP', 'FCOMIP', 'FDIVP', 'ADDPS', 'FCMOVNU', 'FMULP', 'FCMOVU', 'FUCOMIP', 'MOVQ', 'FFREE', 'FXCH', 'FCOMI', 'FSUBP', 'FCMOVNBE', 'FCMOVNE', 'FCMOVBE', 'FUCOMI', 'FDIVRP', 'FSUBRP', 'FCMOVNB', 'FUCOMP', 'FCMOVB', 'MOVDQA', 'FUCOM', 'FCMOVE',	'MOVD', 'SUBPS', 'SETZ', 'MOVSS', 'SETNZ', 'REPZ', 'ADDSS', 'SLDT', 'MOVDQU', 'XORPS', 'PUNPCKLBW', 'CMOVZ', 'CVTDQ2PS', 'MOVLPS', 'PADDSW', 'PADDD', 'CMPXCHG', 'LAR', 'SHUFPS', 'MOVHPS', 'BT', 'XADD', 'PADDW', 'SHRD', 'CMOVNZ', 'PMADDWD', 'PUNPCKLWD', 'SETG', 'UD2', 'PSRAW', 'SHLD', 'CVTPS2PD', 'STOS', 'PSRAD', 'PSUBSW', 'PSUBW', 'SCAS', 'BTR', 'PXOR', 'PACKUSWB', 'PMULLW', 'UNPCKLPS', 'CMOVAE', 'CMOVNS', 'PAVGB', 'BTS', 'STOSW',	'FNCLEX', 'PUNPCKHWD', 'LSL', 'SYSENTER', 'COMISS', 'CVTTPS2PI', 'FNOP', 'PSUBB', 'SETB', 'CMOVG', 'CMOVA', 'CLTS', 'PAND', 'PSLLW', 'PMULHW', 'CWD', 'PACKSSDW', 'PADDB', 'PSHUFW', 'SETGE', 'POR', 'CMOVB', 'INVD', 'ANDPS', 'MULSS', 'FCOS', 'PSADBW', 'CMOVO', 'CMOVL', 'CMOVS', 'UNPCKHPS', 'PUNPCKLDQ', 'SYSRET', 'CVTPI2PS', 'CMOVLE', 'SETL', 'PSRLW', 'PMAXUB', 'MULSD', 'PUNPCKHDQ', 'BTC', 'FLDZ', 'PUNPCKHBW'

Sistemin ilk hedefi zararlı davranış örüntülerini bularak tespit yapmak olmasına rağmen, dinamik analiz ile ortaya çıkarılamayan bilgileri de gözden kaçırmamak için yardımcı olarak durağan analiz ile elde edilen işlem kodu (OpCode) dizileri de öznitelik olarak kullanılmıştır. Makine kodu CPU'nun yapması için komutlardan meydana gelmektedir. Her komut; işlem kodu, kaynak operand ve hedef operandan oluşmaktadır. İşlem kodları programın görevlerini yerine getirmek için CPU tarafından alınan talimatı belirtmektedir. Operandlar ise işlem kodlarının etkileyeni ve etkilenenidir. Önerilen sistem yazılımın ne yaptığı ile ilgilendiği için tüm komut satırı içinde sadece işlem kodlarını kullanmaktadır [11]. İkili kodları olan çalıştırılabilir dosyalar, tek tek ayırıcı (disassembler) ile çözümlenmiş ve makine kodu (assembly) dosyaları elde edilmiştir. Bu dosyalar daha sonrasında yazdığımız program ile OpCode dizilerine çevrilmişlerdir. OpCode dizileri Çizelge-3'te belirtilen işlem kodlarından oluşmaktadır**.** API-çağrımlarının yanında veri kaynağı olarak kullanılan işlem kodları, yazılım davranışlarını daha iyi analiz etmek için önemli ölçüde katkı vermektedir.



Şekil 2. Derin öğrenme modelinin görselleştirilmesi

Çizelge-4: Zararlı Yazılımların Yaptığı en uzun ortak alt diziler

Tür	Zararlı Davranış Dizisi
Worm	GetSystemDirectoryA-NtCreateFile-GetFileSize-NtClose
virus	NtOpenKey-LoadStringA-NtAllocateVirtualMemory-NtFreeVirtualMemory-LoadStringW-NtDuplicateObject-GetFileType-NtCreateMutant-CreateThread-NtAllocateVirtualMemory-NtOpenFile-NtClose-GetSystemWindowsDirectoryA-GetFileAttributesW-NtCreateFile-SetFilePointer-NtReadFile- NtClose
Rootkit	NtFreeVirtualMemory-NtUnmapViewOfSection-NtClose-GetSystemMetrics-LdrUnloadDll-NtClose-LdrGetDllHandle-LdrGetProcedureAddress-NtClose
backdoor	NtOpenKey-GetSystemTimeAsFileTime-LdrLoadDll-LdrGetProcedureAddress-NtClose-GetSystemMetrics
trojan	NtOpenSection-NtMapViewOfSection-NtUnmapViewOfSection-NtClose-NtAllocateVirtualMemory-FindResourceExW-LoadResource-FindResourceExW-LoadResource-NtClose
flooder	RegOpenKeyExW-GlobalMemoryStatusEx-LdrLoadDll-NtDuplicateObject-GetComputerNameW-NtDuplicateObject-NtCreateFile-NtSetInformationFile

Alt dizileri kullanarak bir imza tabanlı tespit sistemi oluşturmanın sakıncası Çizelge-4'te gözükmektedir. Akıllı olmayan bir örüntü bulma ve karşılaştırma yaparak geliştirilen tespit sistemini aldatmak ve atlatmak çok zor olmayacaktır. Keza, anlamsız şekilde yapılan bir API-çağrısı bile dizeyi bozabilir ve bu tip sistemi atlatabilir. Kaldı ki yeni zararlı yazılımlar oldukça gelişmiş atlatma yöntemleri kullanmaktadır. Bu çıkarım çalışmayı durumsuz (stateless) istatistiksel analiz yapmaya ve akıllı bir sistem geliştirmeye yöneltmiştir.

3.3 Markov Zinciri

Markov zincirleri stokastik süreçlerin temel bir parçasıdır.Bu, sistemin sonraki durumu önceki durumlarından bağımsız olarak ancak ve ancak o anki durumuna bağlı olmasıdır. Diyelim ki S= $\{s_1, s_2, s_3, ..., s_r\}$ durumlar kümesi olsun. Öyleyse Markov zinciri şöyle tanımlanmaktadır: S uzayında, eğer bir stokastik süreç $X = \{X_n, n \in \mathbb{N}\}$,

her bir $n \geq 0$ için, $X_n \in S$,

her $n \geq 1$ için ve her $i_0, i_1, \ldots, i_n \in S$ için

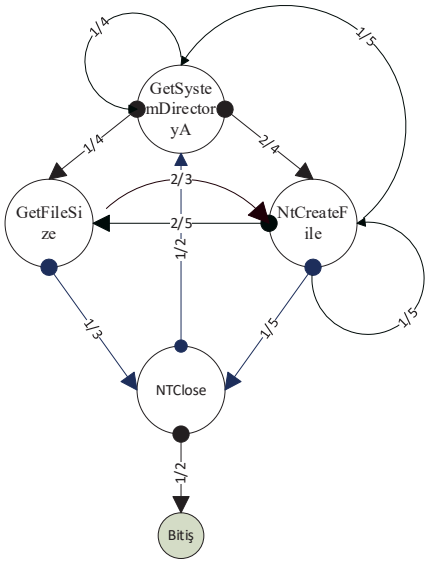
$$\mathbb{P}(X_n = i_n \mid X_{n-1} = i_{n-1}, \ , X_0 = i_0) = \mathbb{P}(X_n = i_n \mid X_0 = i_0) \quad (1)$$

ise X Markov zinciridir.

Görüleceği üzere Markov zinciri, dizinin içerdiği durumların değişimlerindeki olasılık dağılımından oluşmaktadır. Markov zinciri X için t anındaki geçiş matrisi P_t, durumlar arasındaki geçiş olasılıklarının tutulduğu bir matristir.

Bu matrisin her bir gözün matematiksel tanımlaması şöyle yapılmaktadır:

GetSystemDirectoryA-NtCreateFile-GetFileSize-NtClose-GetSystemDirectoryA-NtCreateFile-GetFileSize-NtCreateFile-NtCreateFile-GetSystemDirectoryA-GetSystemDirectoryA-GetFileSize-NtCreateFile-NtClose



Şekil 3: API-çağrım dizisi ve Markov Zinciri Dönüşümü

$$(P_t)_{i,j} = \mathbb{P}\,(X_{t+1}=j \mid X_t = i), \quad i, j \in S. \qquad (2)$$

Bu, matrisin her satırının bir olasılık vektörü olduğu ve vektördeki elemanların toplamlarının 1 olduğu anlamına gelmektedir.

Bu makalede, Markov zinciri analizi ile yapılan durum analizinde zaman faktörü 2 değişkenlidir; şimdi t=0 ve sonra t=1. Matristeki her göz (entry) şimdiki ve sonraki durum geçiş olasılığını tutmaktadır. Şimdiki ve sonraki durum olasılığı bilgisi yazılımın zararlı davranışa sahip olduğu konusunda bir bilgi içermektedir. Geliştirilen sistem, bu bilgi örüntüsünü kullanarak tespit yapmaktır.

Geçiş matrisinin en önemli özelliği durumsuz sıralama olasılıklarını içermesidir. Bu olasılıklar dizi içinde geçen her eleman için bir sonrakinin gelme olasılığıdır. Böylece her iki eleman için bir olasılık değeri belirlenir. Geçiş matrisinde her bir durum için geçiş olasılıkları vardır. Öyle ki satır kısmına denk gelen durumdan sütun kısmına denk gelen duruma geçiş olasılığı o hücrede belirtilen olasılık değeridir.

Özet olarak söyleyecek olursak Markov zincirleri sistemin bir durumdan başka bir duruma geçişini modellemek için kullanılır. Bu çalışmada analiz edilen dizi içindeki her bir eleman dizinin o sıradaki durumu olarak ele alınmış ve geçiş matrisleri bu varsayım ile hesaplanmıştır. Durumlar arasındaki geçişler, yeni bir duruma geçme olasılığı veren koşullu bir olasılık dağılımına tabidir. Bu koşullu olasılık dağılımı geçiş matrisi ile sağlanmaktadır. Her bir dizi için bir Markov zinciri kurulur ve Markov zincirleri geçiş matrisleri ile ifade edilmektedir. Geçiş matrisi Algoritma 1.'de belirtilen yöntem ile oluşturulmaktadır.

Açıktır ki geçiş matrisi NxN boyutundadır, N sayısı durum kümesi eleman sayısına denk gelir. Durum matrisini çıkarmak için yinelemeli algoritma, Algoritma 1. de gösterildiği gibi yazılmıştır. Algoritmaya göre her adımda dizi 2'ye bölünerek her iki kısmı için tekrar çağrılır, ta ki 2 ya da 3 eleman kalana kadar. Eğer dizinin eleman sayısı tek ise ortadaki 3 sayı ile, eğer çift ise ortadaki 2 sayı ile M geçiş matrisi güncellenir. 2 sayılı dizi ile güncellemede bir, 3 sayılı dizi ile güncellemede iki arttırma işlemi olur. Algoritma ilk çağrılışında M matrisinin tüm elemanları 0 olmalıdır. Karmaşıklık analizi, N dizinin eleman sayısı ve B da durum kümesinin eleman sayısı olduğu yerde, O(BlogN) olur.

Algoritma 1. Markov Zincirinin Geçiş Matrisini Bulma algoritması

Algoritma 1. MZO- Markov Zinciri Oluştur

Girdi: DurumKumesi ≠ Ø, S ≠ Ø, N = |S|, B = |DurumKumesi|, M $\in R^{B\times B}$

```
1:  Eğer N çift ise;
2:      Eğer N>2 ise;
3:          o← N/2                          //Dizinin ortasındaki elemanın indisi bulunur
4:          oSeq ← S[c: c+1]                // Dizinin ortasındaki 2 eleman alınır
5:          oSeq ve DurumKümesi ile M matrisini güncelle
6:          rseq ← S[0:N/2]
7:          lseq ← S[N/2:N]
8:          MZO(rseq, DurumKumesi, M)
9:          MZO(lseq, DurumKumesi, M)
10:     Değilse;
11:         S ve DurumKümesi ile M matrisini güncelle
12:     Eğeri Bitir.
13: Değilse;                                //Eğer çift tek ise
14:     Eğer N>3 ise;
15:         o ← N/2 +1                      //Dizinin ortasındaki elamanın indisi bulunur
16:         oSeq ← S[o-1:o+1]               //Dizinin ortasındaki 3 eleman oSeq'e alınır
17:         oSeq ve DurumKumesi ile M matrisini güncelle
18:         rSeq ← S[0:o-1]
19:         lSeq ← S[o+1:N]
20:         MZO(rseq, DurumKumesi, M)
21:         MZO(lseq, DurumKumesi, M)
22:     Değilse;
23:         S ve DurumKümesi ile M matrisi güncelle
24:     Eğeri Bitir.
25:Eğeri Bitir.
```

Oluşturulan Markov zinciri Şekil 3.'de örnek üstünde gösterilmiştir. Markov zincirleri sistemimizde geçiş matrisleri ile ifade edilmektedir. Geçiş matrisleri her bir numune için 2 tane çıkarılmaktadır; API-call dizisi için ve OpCode dizisi için. Bu matrisler birleştirilir. Birleştirilmiş matriste, her hücrede iki eleman olacaktır, ilki API fonksiyon durumu geçiş olasılığı diğeri ise işlem kodu durumu geçiş olasılığıdır. Veri kümesinde var olan her bir numune bu birleştirilmiş matris ile ifade edilmektedir ve derin öğrenme modeline bu hali ile verilmektedir.

3.4 Derin Öğrenme Modeli

Derin öğrenme özelleştirilmiş bir makine öğrenmesi alt alanıdır [17]. Buyüzden derin öğrenme konusunu incelemeden önce makine öğrenmesini incelemek gerekir. Makine öğrenmesi şu soru ile ortaya çıkmıştır: Bir bilgisayar kendi başına belirli bir görevi nasıl gerçekleştireceğini öğrenebilir mi [31]? Diğer bir deyişle, klasik programların yaptığı belirli kurallar çerçevesinde algoritma ile yazılıp verilen veri ile cevap üretirken, makine öğrenmesi algoritması veri ve o verilerin cevapları ile kuralları üretir ve gelecek yeni verilerle ona göre cevap verir.

Derin öğrenme de makine öğrenmesinin alt alanı olarak aynı kaygıyı gütmektedir. Buradaki "derin" ifadesi, öğrenme işlemini her defasında daha anlamlı olacak şekilde katman katman yapılmasına denk gelir [19]. Önceki çizilen gösterimden açıklanacak olunursa; derin öğrenmenin ilk katmanında makine öğrenmesinde olduğu gibi veri ve cevaplardan kurallar üretilir.Bir sonraki katmanında ise önceki katmanda üretilen kurallar ve ilk girdideki cevaplar kullanılarak yeni kurallar üretilir. "Derin" kavramı da bu katmanlı yapıda katman sayısının artması anlamına gelir ve anlaşılacağı gibi derin öğrenme işlemi öğrenilmiş kurallardan yeni kurallar üretme üstüne kurulmuştur. Son katmanda ise cevaplar uzayından bir elemanı karar olarak vermesi beklenir.

Derin öğrenme yöntemi en saf haliyle çok katmanlı algılayıcı (Multilayer perceptron) olarak

tanımlanabilir. Genel olarak yapay sinir ağın yegâne amacı herhangi bir *f** fonksiyonuna yakınlaşmaktır. Yapay sinir ağı y = ***f(x;θ)*** gönderimini tanımlar ve fonksiyon f sonuçlarına en yakın $\boldsymbol{\theta}$ değerlerini hesaplar [32]. Eğitim aşamasında verilen y ve x değerleri ile $\boldsymbol{\theta}$ değişkenleri optimize edilir. Test aşamasında ise verilen x ve eğitim değişkenleri ile y tahmin edilir. Eğitim aşamasında yapılan işleme loss fonksiyonu denir ve bu deneyde kullandığımız loss fonksiyonu şu şekilde verilmektedir:

$$CE = -\sum_i^C f * (x_i) log\,(f(x_i;\boldsymbol{\theta})) \quad (4)$$

CE fonksiyonu çapraz entropi (cross entropy) olarak isimlendirilmektedir. f fonksiyonu derin öğrenme modelinin aktivasyon fonksiyonunu ifade etmektedir. Bu fonksiyon her nöronda yapılan işlemdir ve her katman için farklı bir aktivasyon fonksiyonu belirlenebilmektedir. Deneylerde kullandığımız aktivasyon fonksiyonu ReLu (Recrified linear unit) fonksiyonudur ve şu şekilde ifade edilmektedir:

$$f(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases} \quad (5)$$

Her adımda f fonksiyonu gerçek değer olan *f** 'a yaklaşacak şekilde optimize edilir. Bu çalışmada Adam optimizer [33] yöntemi kullanılarak değişkenler optimize edilmiştir. Derin öğrenme modelimiz Çizelge-5'te gösterilen katmanlardan oluşmaktadır. Bu katmanlar evrişimsel sinir ağı (CNN) ve maksimum havuzlama yöntemlerini içermektedir. Bu yöntemler [32]'de anlatıldığı gibi, çok katmanlı algılayıcı 'da yapılan çoklu işlemleri daha az değişkenle yapılmasını sağlamaktadır. Maksimum havuzlama ile de nöron sayısını azaltmakta ve değerler üstünde sadeleştirmeye gidilmektedir.

Zararlı yazılım tespiti için geliştirilen derin öğrenme modeli 4 adet CNN katmanıyla her katman arasına maksimum havuzlama katmanı içermektedir. Karar katmanından bir önceki katmanda ise tüm nöronların birbirine bağlandığı yoğunluk katmanı kullanılmıştır. Model girdi olarak 264x264x2 boyutlu bir matris almaktadır. Çıktı olarak da zararlı yazılım ya da iyicil yazılım olduğuna karar vermektedir. Toplam optimize edilen değişken sayısı 85,746'dir.

Çizelge-5: Derin Öğrenme Model Özeti

Katman (tür)	Çıktı	Değişken
CNN	262, 262, 16	304
Maksimum Havuzlama	87, 87, 16	0
CNN	85, 85, 32	4640
Maksimum Havuzlama	28, 28, 32	0
CNN	26, 26, 64	18496
Maksimum Havuzlama	8, 8, 64	0
CNN	6, 6, 96	55392
Geçiş	3456	0
Yapay sinir ağı	2	6914
Eğitilen Toplam Değişken Sayısı:		85,746

4. Deney Sonuçları ve Değerlendirme

4.1 Veri Kümesi

Deneyde kullanılan veri kümesi zararlı yazılımlar Vxheaven'dan [28], iyicil yazılımlar ise Windows sistem dosyalarından ve Cygwin dosyalarından derlenmiştir. Bu çalışmada 2 veri kümesi kullanılmıştır. Bu veri kümelerinden A olarak isimlendirdiğimiz veri kümesi 2000 adet iyicil dosyadan, 2000 adet zararlı dosyadan oluşmaktadır. Bu veri kümesi çalışmanın deneylerinin yapıldığı ve diğer deneylerle karşılaştırma yapmak için kullanılan veri kümesidir. Sadece bu veri kümesi ile karşılaştırma yapılmasının sebebi, diğer çalışmaların büyük veri kümesi ile yavaş çalışması ve kaynak yetersizliğine sebebiyet vermesidir.

Çizelge-6: Deneyin veri kümesinde bulunan zararlı ve iyicil yazılım sayıları

	Numuneler	Sayılar
A	İyicil	**2000**
	Zararlı	**2000**
B	İyicil	**6857**
	Zararlı	**8936**

B ismi ile adlandırılan veri kümesi ise daha kapsamlı ve daha fazla numuneyi içermektedir. Bu veri kümesinde iyicil olarak, zararlı yazılımlara benzerliği bir nebze daha yüksek olan Download.com sitesinden indirilen iyi huylu yazılımlar da kullanılmıştır. Bu veri kümesinin amacı, işleri zorlaştırarak çalışmanın doğruluğunu kanıtlamaktır. Veri kümelerinin numune sayıları Çizelge-6'da içerikleri ise Çizelge-7'de gösterilmiştir.

Çizelge-7: Zararlı yazılım kümesi tür dağılımları

Türler	A	B
Backdoor	410	1711
Constructor	8	24
DoS	1	10
Exploit	9	60
Flooder	3	27
HackTool	6	22
Hoax	8	53
Packed	3	9
Rootkit	25	104
SpamTool	-	2
Spoofer	-	2
Trojan	844	4016
Trojan-DDoS	1	3
Trojan-GameThief	209	952
Trojan-IM	5	18
Trojan-Notifier	1	2
Trojan-PSW	152	544
Trojan-Ransom	-	4
VirTool	5	18
Virus	209	945
Worm	100	410

4.2 Değerlendirme

Geliştirilen tespit sisteminin başarımı gerçek pozitif oranı ve yanlış pozitif oranı ile ölçülmüştür. Bu oranları incelemeden önce gerçek pozitif (TP), gerçek negatif (TN), yanlış pozitif (FP) ve yanlış negatif (FN) değerlerini anlamak gerekmektedir. Bu değerler genel olarak o numunenin gerçek sınıfı ve modelin tahmin ettiği sınıfını karşılaştırmak için kullanılan ölçüm değerleridir.

Bizim deney üstünden düşünülecek olursa, zararlı ve iyi huylu olarak iki sınıf vardır. Zararlı olarak 2000 ve iyi huylu olarak 2000 numune ile deney yapılmıştır. Zararlılar için değerlendirecek olursak; TP değeri, gerçekte zararlı olan ve deney sonucunda modelin zararlı olarak tespit ettiği numune sayısını göstermektedir. TN ise gerçekte iyi huylu olup deney sonucunda modelin iyicil olarak tespit ettiği numune sayısını belirtmektedir. FP değeri gerçek sınıfı iyicil olup modelin zararlı olarak tespit ettiği ve aslında yanlış tespit edilen numunelerin sayısını vermektedir. FN ise gerçek sınıfı zararlı olup modelin iyicil dediği ve aslında yanlış tespit edilen numune sayısını göstermektedir. TPR gerçek pozitif (TP) değerlerinin gerçekte pozitif olanlara (TP + FN) oranı ile bulunur. FPR ise FP değerlerinin gerçekte negatif olanlara (FP + TN) oranı ile hesaplanmaktadır.

4.3 Deney Sonuçları

Deneyin başlangıcı olarak yapılan işlem analiz sürecidir. Durağan analiz ve dinamik analiz olmak üzere 2 tip analiz yapılmıştır. Dinamik analiz sanal makine ile güvenli ortam oluşturularak yapılmıştır. Bu işlem için Cuckoo kum havuzu kullanılmıştır. Cuckoo'yu çoklu kullanıma hazırlamak için ve tüm dinamik analiz ortamını hazırlayıp otomatize etmek için çalışma zamanı gözetleme modülü kullanılmıştır. Cuckoo verilen numuneyi güvenli sanal makinede çalıştırmak, sanal makinede olup biteni izlemek ve raporlama açısından oldukça kullanışlı ve faydalı bir araçtır. Yazılımları sanal makinede çalıştırıp izleme aşamasının sonunda json formatında rapor dosyaları oluşturulmuştur. Modül bu dosyalardan sıralı API-çağrım dizilerini çıkartmaktadır. Durağan analiz için distorm3 kütüphanesi kullanarak yazdığımız ayırıcı modülü ile gerçekleştirilmiştir. Ayırıcı modülü veri kümesindeki numunelerin tek tek makine kodu dosyalarını (assembly file) oluşturur ve sıralı OpCode dizilerini çıkartmaktadır.

Çizelge-8: Zararlı Yazılım tespit sistemi deney sonuçları

		Eğitim		Test	
		TPR	FPR	TPR	FPR
A	İyicil	0,9872	0,0726	0,9885	0,0879
	Zararlı	0,9274	0,0128	0,9121	0,0115
B	İyicil	0,9787	0,0259	0.9845	0.0303
	Zararlı	0,9741	0,0213	0.9697	0.0155

Her dizi için geçiş matrisleri oluşturmak amacıyla Markov Zinciri Oluşturma modülü yazılmıştır. Bu modül Algoritma 1. Temelli çalışmaktadır. Her dizi için Durum Kümeleri tipine göre değişmektedir. Durum Kümesi; API-çağrım için deneyde çıkarılan tüm API fonksiyonlarından, OpCode için ise en çok kullanılan OpCode'lardan oluşmaktadır. Bu modülün çıktısı olarak her zararlı yazılım için 2 adet 264x264'lik matris oluşmaktadır. Bu matrisler birleştirilerek numuneyi deneyde temsil eden birleştirilmiş matris oluşturulur. Matrislerin satır ve sütunları, en çok kullanılan durumlardan en az kullanılan durumlara göre sıralı olacak şekilde sıralanmıştır.

Derin öğrenme modelini gerçeklemek için keras [29] kütüphanesi kullanılmıştır. Python'nun 3.5 versiyonu kullanılmış ve bütün tespit sistemi bu dil ile gerçekleştirilmiştir. Test sonuçları Çizelge-9'da verilmiştir.

Yapılan çalışmada test sonuçlarının yüksek çıkmasındaki ana pay dinamik analiz ile elde edilen davranış özniteliklerindedir. Davranış analizi ile zararlı yazılım ve iyicil yazılım farkı ortaya konmuş, durağan analiz ile de bu fark daha da belirgin hale getirilmiş ve destekleyici olmuştur.

Çizelge-9: Diğer Yöntemler ile Karşılaştırma

Yöntem	TPR	FPR	ACC
MZO-DÖ	**0.950**	**0.049**	**0.968**
Rastgele Orman	0.943	0.062	0.959
J48	0.925	0.088	0.935
Destek Vektör Makinesi	0.718	0.685	0.727
İkili Ham Veri - DÖ	0.649	0.327*	0.729*

Çizelge-9'da referans yöntemler ile bu çalışmada sunulan yöntemin başarı karşılaştırması verilmiştir. Karşılaştırma ortamını hazırlamak için, A veri kümesi kullanılarak melez öznitelikler çıkarılmış ve dizilerin histogram analiz verisi oluşturulmuştur. Histogram dosyaları birleştirilip weka [30] aracı kullanılarak rastgele orman, J48 ve destek vektör makinesi algoritmaları ile test edilmiştir. Diğer karşılaştırma ise referans olarak aldığımız derin öğrenme yöntemi kullanılarak yapılmıştır. Bu karşılaştırma deneyinde ise ikili dosyalar işlenmeden [24]'de belirtilen yöntem ile resim dosyasına çevrilmiş ve her bir numune için 1024x1024'lük resimler oluşturulmuştur. Bu Resim dosyaları ile bu çalışmada kullandığımız derin öğrenme modeli ile testler yapılmıştır. Deneylerde A veri kümesi kullanılmıştır. Bunun sebebi ise veri kümesinin büyük olması, bu deneylerin çalışma zamanı ve kaynak kullanımını büyük oranda arttırmasıdır. Deneylerin başarım sonuçları Çizelge-10'da belirtilmektedir. Bu çalışmalar ile önerilen yöntem karşılaştırıldığında, başarım oranı daha yüksek olduğu net bir şekilde görülmektedir.

Çizelge- 10'da ise derin öğrenme yöntemi kullanılarak yapılan diğer çalışmalar ile karşılaştırmalar gösterilmektedir. Bu araştırmada ortaya çıkarılan yöntem, sadece evrişimsel sinir ağı kullanmasına rağmen oldukça yüksek başarım sağladığı görülmüştür. Fakat önerilen yöntem dinamik bir çalışma ile tespit yaptığı için, hızlı karar vermesi uygulanabilirliği açısından önemli bir parametredir. Daha basit bir derin öğrenme modelinin tercih edilmesi ve öne sürülen modelin farklı yöntemlerle karmaşıklaştırılmamasının sebebi, bu çalışmanın yapısında var olan statik analiz ile nitelik sadeleştirme ile kazanılan kaynak kullanım miktarındaki azalma ve test süresini kısaltma özelliklerinden vazgeçmek istenmemesidir

Çizelge-10: Derin Öğrenme ile Tespit yapan diğer Araştırmalar ile Karşılaştırma

REF	Öznitelik	Model	Amaç	Doğr.
[17]	API & Opcode	PCA + NN	Tespit	95.1
[20]	API çağrı Graf	SAE + DT	Tespit	99.1
[24]	İkili Ham Dosya-Resim	CNN +LSTM	Tespit	97.1
[26]	System-Call	CNN +LSTM	Snf.	89.4
MZO-DÖ	API & Opcode	CNN	Tespit	97.8

5. Sonuç ve Öneriler

Çalışmada veri analizi ve öznitelik seçimi yapılırken sıklık ön planda tutulmuştur. İleriki çalışmalarda statik ve dinamik veri kaynakları birbiriyle daha somut ve çalışma yapısı ile ilgili bağlantılar kurularak birleştirilebilir, böylece dizi verisi daha anlamlı hale getirilebilir.

Bir diğer husus ise önerilen yöntemin farklı platformlarda da nasıl bir başarım sağlayacağı konusudur. [18] ve [19]'da da Markov zinciri kullanılarak android zararlıları tespit edilmiştir. Fakat bu çalışmadan farklı olarak onlar sadece dinamik veri kaynağı üstüne yoğunlaşmış ve bu çalışmada önerilen modelden daha karmaşık bir derin öğrenme modeli kullanmışlardır. Mobil platformlarda önerilen modelin uygulanması için, yeni bir öznitelik mühendisliği yapmalıdır. Örnek olarak android cihazlarda API-çağrımlara ek olarak manifesto dosyasından kullanım izinleri çıkarılabilir ya da bayt kod dosyalarından yardımcı nitelikler bulunabilir. Bunun için kapsamlı incelemeye ihtiyaç vardır.

Bu çalışmada, Windows ortamında çalışan zararlı yazılımların tespiti için istatistiksel analiz yöntemi ve derin öğrenme tekniklerinden faydalanılarak bir yöntem önerilmiştir. Yöntem zararlı yazılımların davranış ve kod bilgisini kullanmaktadır. Önerilen yöntem diğer tespit yöntemleri ile karşılaştırıldığında başarılı olduğu görülmüştür.

Teşekkür

Bu çalışma Türkiye Bilimsel ve Teknik Araştırma Kurumu (TUBITAK) tarafından desteklenmiştir. Destek No: ARDEB-116E624.

Kaynakça

[1] Malware Bytes-Labs. 2019 State of Malware, 2019.https://blog.malwarebytes.com/malware bytes-news/ctnt-report (accessed April 15, 2019).

[2] AV-Test-Institute. 2019 New Malware, 2019. https://https://www.av-test.org/en/statistics/malware/(accessed April 15, 2019).

[3] Rajeswaran, D., DiTroia, F., Austin, T. H., &Stamp, M. (2018). Function call graphs versus machine learning for malware detection. In Guide to Vulnerability Analysis for Computer Networks and Systems (pp. 259-279). Springer, Cham.

[4] Pektaş, A., &Acarman, T. (2017). Malware classification based on API calls and behavior analysis. IET Information Security, 12(2), 107-117.

[5] Vemparala, S., DiTroia, F., Corrado, V. A., Austin, T. H., & Stamo, M. (2016, March). Malware detection using dynamic birthmarks. In Proceedings of the 2016 ACM on International Workshop on Security And Privacy Analytics (pp. 41-46). ACM.

[6] Safa, H., Nassar, M., & Al Orabi, W. A. R. (2019, June). Benchmarking Convolutional and Recurrent Neural Networks for Malware Classification. In 2019 15th International Wireless Communications & Mobile Computing Conference (IWCMC) (pp. 561-566). IEEE.

[7] Afianian, A., Niksefat, S., Sadeghiyan, B., &Baptiste, D. (2018). Malware Dynamic Analysis Evasion Techniques: A Survey. arXivpreprint arXiv:1811.01190.

[8] Sartea, R., & Farinelli, A. (2018, July). Detection of Intelligent Agent Behaviors Using Markov Chains. In Proceedings of the 17th International Conference on Autonomous Agents and Multi Agent Systems (pp. 2064-2066). International Foundation for Autonomous Agents and Multiagent Systems.

[9] Martín, A., Rodríguez-Fernández, V., &Camacho, D. (2018). CANDYMAN: Classifying Android malware families by modelling dynamic traces with Markov chains. Engineering Applications of Artificial Intelligence, 74, 121-133.

[10] Anderson, B., Storlie, C., &Lane, T. (2012, October). Improving malware classification: bridging the static/dynamic gap. In Proceedings of the 5th ACM workshop on Security and artificial intelligence (pp. 3-14). ACM.

[11] Nar, M., Kakisim, A. G., Yavuz, M. N., & Soğukpinar, İ. (2019, September). Analysis and Comparison of Disassemblers for OpCode Based Malware Analysis. In 2019 4th International Conference on Computer Science and Engineering (UBMK) (pp. 17-22). IEEE.

[12] Demetrio, L., Biggio, B., Lagorio, G., Roli, F., &Armando, A. (2019). Explaining Vulnerabilities of Deep Learning to Adversarial Malware Binaries. arXivpreprint arXiv:1901.03583.

[13] Biggio, B., Rieck, K., Ariu, D., Wressnegger, C., Corona, I., Giacinto, G., &Roli, F. (2014, November). Poisoning behavioral malware clustering. In Proceedings of the 2014 workshop on artificial intelligent and security workshop (pp. 27-36). ACM.

[14] Garetto, M., Gong, W., &Towsley, D. (2003, March). Modeling malware spreading dynamics. In IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No. 03CH37428) (Vol. 3, pp. 1869-1879). IEEE.

[15] Chen, Z., &Ji, C. (2005). Spatial-temporal modeling of malware propagation in networks. IEEE Transactions on Neural networks, 16(5), 1291-1303.

[16] Karyotis, V. (2010). Markov random fields for malware propagation: the case of chain networks. IEEE Communications Letters, 14(9), 875-877.

[17] Zhang, J., Qin, Z., Yin, H., Ou, L., &Zhang, K. (2019). A feature-hybrid malware variants detection using CNN based opcode embedding and BPNN based API embedding. Computers& Security, 84, 376-392.

[18] Xiao, X., Wang, Z., Li, Q., Xia, S., &Jiang, Y. (2016). Back-propagation neural network on Markov chains from system call sequences: a new approach for detecting Android malware with system call sequences. IET Information Security, 11(1), 8-15.

[19] Onwuzurike, L., Mariconti, E., Andriotis, P., Cristofaro, E. D., Ross, G., &Stringhini, G. (2019). MaMaDroid: Detecting android malware by building Markov chains of behavioral models (extended version). ACM Transactions on Privacy and Security (TOPS), 22(2), 14.

[20] Xiao, F., Lin, Z., Sun, Y., &Ma, Y. (2019). Malware Detection Based on Deep Learning of Behavior Graphs. Mathematical Problems in Engineering, 2019.

[21] Ndibanje, B., Kim, K. H., Kang, Y. J., Kim, H. H., Kim, T. Y., & Lee, H. J. (2019). Cross-Method-Based Analysis and Classification of Malicious Behavior by API Calls Extraction. Applied Sciences, 9(2), 239.

[22] Alsulami, B., &Mancoridis, S. (2018, October). Behavioral Malware Classification using Convolutional Recurrent Neural Networks. In 2018 13th International Conference on Malicious and Unwanted Software (MALWARE) (pp. 103-111). IEEE.

[23] Sun, G., &Qian, Q. (2018). Deep learning and visualization for identifying malware families. IEEE Transactions on Dependable and Secure Computing.

[24] Le, Q., Boydell, O., Mac Namee, B., &Scanlon, M. (2018). Deep learning at the shallow end: Malware classification for non-domain experts. Digital Investigation, 26, S118-S126.

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

[26] Kolosnjaji, B., Zarras, A., Webster, G., &Eckert, C. (2016, December). Deep learning for classification of malware system call sequences. In Australasian Joint Conference on Artificial Intelligence (pp. 137-149). Springer, Cham.

[27] Kakisim, A. G., Nar, M., Carkaci, N., &Sogukpinar, I. (2018, November). Analysis and Evaluation of Dynamic Feature- Based Malware Detection Methods. In International Conference on Security for Information Technology and Communications (pp. 247-258). Springer, Cham.

[28] VxHeaven. Computer virus collection, 2014. http://83.133.184.251/virensimulation.org/(accessed April 15, 2019).

[29] Chollet, F. Keras (2015) Git Hubrepository. https://github.com/fchollet/keras

[30] Eibe Frank, Mark A. Hall, and Ian H. Witten (2016). The WEKA Workbench. Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques", Morgan Kaufmann, Fourth Edition, 2016.

[31] Chollet, F. (2018). Deep Learning mit Pythonund Keras: Das Praxis-Handbuchvom Entwickler der Keras-Bibliothek. MITP-VerlagsGmbH&Co. KG.

[32] Goodfellow, I., Bengio, Y., &Courville, A. (2016). Deeplearning. MIT press.

[33] Kingma, D. P., &Ba, J. (2014). Adam: A method for stochastic optimization. ArXivpreprint arXiv:1412.6980.