ANKARA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

YÜKSEK LİSANS TEZİ

UYGULAMA PROGRAMLAMA ARAYÜZÜ (UPA) ÇAĞRILARININ KULLANILMASINA DAYALI MAKİNE ÖĞRENMESİ YÖNTEMLERİYLE ZARARLI YAZILIM TESPİTİ

Adnan Kutay YÜKSEL

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

ANKARA
2021

ÖZET

Yüksek Lisans Tezi

UYGULAMA PROGRAMLAMA ARAYÜZÜ (UPA) ÇAĞRILARININ KULLANILMASINA DAYALI MAKİNE ÖĞRENMESİ YÖNTEMLERİYLE ZARARLI YAZILIM TESPİTİ

Adnan Kutay YÜKSEL

Ankara Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

Danışman: Dr. Öğr.Üyesi Yılmaz AR

Günümüzde internet ve bilgi sistemleri her yerdedir ve insan hayatının ayrılmaz bir parçası haline gelmiştir. Hükümetler, ordular ve her türlü kuruluşlar günümüzde internete eskisinden çok daha fazla bağımlıdır. Bu durum da kolaylıkların yanı sıra beraberinde potansiyel olarak yıkıcı güvenlik açıklarını da beraberinde getirmektedir. Bu tür riskler için sayısız çözümler bulunmaktadır ve bu çözümlerin güvenliğe büyük ölçüde katkıda bulunduğu mutlak bir gerçektir, ancak Sıfırıncı Gün zararlı yazılımlarına karşı henüz etkin bir çözüm bulunmuş değildir. Birçok alanda gerçekten iyi işler çıkaran makine öğrenme yöntemleri, Sıfırıncı Gün tehlikelerine karşı da potansiyel bir çözümdür. Bu tezde, her türlü kötü amaçlı yazılımın üstesinden gelebilmek için geleneksel yöntemlerin dışında kararlı bir çözüm araştırılmıştır. Literatürdeki çeşitli kaynaklarda olduğu gibi karmaşık, zaman maliyetli ve heterojen özelliklerden oluşan çözümler yerine; basit, zaman maliyeti düşük ve homojen yapıdaki özellikler ile kararlı bir çözüm elde edilmiştir. Bu yöntemle elde edilen %98.04 doğruluk yüzdesi, yöntemin oldukça başarılı olduğunu göstermektedir.

Şubat 2021, 65 sayfa

Anahtar Kelimeler: Siber Güvenlik, Zararlı Yazılım Tespiti, Makine Öğrenmesi, Derin Öğrenme, API Çağrısı

ABSTRACT

Master Thesis

MALWARE DETECTION WITH MACHINE LEARNING METHODS BASED ON APPLICATION PROGRAMMING INTERFACE (API) CALLS

Adnan Kutay YÜKSEL

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

Supervisor: Dr. Yılmaz AR

Today, internet and information systems are everywhere and have been an inseperable part of human life. Governments, armies, all kinds of organizations now depend on Internet very much. This brings out not only easiness, but also potentially-catastrophic security vulnerabilities. There are innumerous solutions for these kinds of risks and it's an absolute reality these solutions contribute very much to security, but there haven't been any efficient solutions found against Zero-Day malicious softwares yet. Machine learning methods which does a really good job on a lot of domains, is also a potential antidote against Zero-Day venom. Throughout this thesis, a stable solution other than traditional methods has been investigated in order to overcome all kinds of malwares. Instead of solutions consisting of complex, time-costly and heterogeneous features, as in various papers in the literature, a stable solution has been obtained with simple, low time cost and homogeneous properties. The 98.04% accuracy percentage obtained with this method shows that the method is quite successful.

February 2021, 65 pages

Key Words: Cyber Security, Malware Detection, Machine Learning, Deep Learning, API Calls.

TEŞEKKÜR

Çalışmalarım süresince destek veren danışmanım Dr. Öğr. Üyesi Yılmaz AR'a, görevimde gerekli izinleri kullanmam konusunda gerekli kolaylıkları gösteren amirim Fatih BARIŞ'a, evdeki çalışma zamanlarımda desteklerini bir an olsun eksik etmeyen değerli annem Mine KÖSEDAĞ, sevgili kardeşlerim Ece Nagihan YÜKSEL ve İsa Emir YÜKSEL ile çalışmamda her desteği eksiksiz veren biricik eşim Sümeyra YÜKSEL başta olmak üzere çalışmamda emeği geçen herkese teşekkür ederim.

Adnan Kutay YÜKSEL

Ankara, Şubat 2021

İÇİNDEKİLER

KISALTMALAR DİZİNİ

API	:	Application Programming Interface (Uygulama Programlama Arayüzü)
USD	:	U.S. Dollars (Amerikan Doları)
USB	:	Universal Serial Bus (Evrensel Seri Veriyolu)
KB	:	Kilobytes (Kilobayt)
GB	:	Gigabytes (Gigabayt)
IBAN	:	International Bank Account Number (Uluslararası Banka Hesap Numarası)
RAT	:	Remote Administration Tool (Uzaktan Yönetim Aracı)
PE	:	PorTablo ExecuTablo (Taşınabilir Yürütülebilir)
HTTP	:	Hyper Text Transfer Protocol (Hipermetin Aktarma Protokolü)
IP	:	Internet Protocol (İnternet Protokolü)
DNS	:	Domain Name Server (Alan Adı Sunucusu)
URL	:	Uniform Resource Locator (Tekbiçimli Kaynak Konumlayıcı)
KNN	:	K-Nearest Neighbour (K-En Yakın Komşu)
SVM	:	Support Vector Machine (Destek Vektör Makinesi)
DNN	:	Deep Neural Networks (Derin Sinir Ağları)
CNN	:	Convolutional Neural Networks (Evrişimli Sinir Ağları)
DBN	:	Deep Belief Networks (Derin İnanç Ağları)
SAE	:	Stacked Auto-Encoder (Yığın Otokodlayıcı)
IG	:	Information Gain (Bilgi Kazanımı)
RF	:	Random Forest (Rassal Orman)
2-D	:	Two-Dimensional (2 Boyutlu)
DOS	:	Disk Operating System (Disk İşletim Sistemi)
DLL	:	Dynamic-Link Library (Dinamik-Bağlı Kitaplık)
PSI	:	PrinTablo Strings Information (Yazdırılabilir Diziler Bilgisi)
BLSTM	:	Bidirectional Long Short-Term Memory (İki Yönlü Uzun Kısa Süreli Bellek)
GRU	:	Gated Recurrent Unit (Geçitli Tekrarlayan Birim)

BGRU Tekrarlayan	:	Bidirectional Gated Recurrent Unit (İki Yönlü Geçitli Birim)
LSTM	:	Long Short-Term Memory (Uzun Kısa Süreli Bellek)
ANN	:	Artificial Neural Networks (Yapay Sinir Ağları)
HMM	:	Hidden Markov Model (Saklı Markov Modeli)
MIST	:	Malware Instruction Set (Zararlı Yazılım Talimat Kümesi)
IB1	:	Instance-Based 1 (Örnek Temelli)
DKOM	:	Direct Kernel Object Manipulation (Doğrudan Çekirdek Nesne Manipülasyonu)
HCI	:	Human Computer Interaction (İnsan Bilgisayar Etkileşimi)
KVM	:	Kernel-Based Virtual Machine (Çekirdek Tabanlı Sanal Makine)
EXE	:	ExecuTablo File Extension (Yürütülebilir Dosya Uzantısı)
APIMDS	:	API-Based Malware Detection System (API Tabanlı Zararlı Yazılım Tespit Sistemi)
LTS	:	Long Term Support (Uzun Vadeli Destek)
ReLU	:	Rectified Linear Unit (Doğrultulmuş Doğrusal Birim)
ROC	:	Receiver Operating Characteristic (Alıcı İşletim Karakteristiği)
MD-5	:	Message Digest 5 (Mesaj Özeti)
SHA	:	Secure Hash Algorithm (Güvenli Özet Algoritması)

ŞEKİLLER DİZİNİ

ÇİZELGELER DİZİNİ

1. GİRİŞ

İnternet artık her yerdedir ve insan hayatının ayrılmaz bir parçası olmuştur. İnternet birçok yeteneği sayesinde sadece insanların değil, her türlü kamu ve özel sektör kurum ve kuruluşlarının da bağımlı olduğu bir kavram haline de gelmiştir. Her türden kurum ve kuruluşun (hükümetler, ordular ve sivil şirket ve kurumlar da dâhil) internet altyapısı üzerine inşa edilmesiyle verinin üretim hızı ve kapasitesi de oldukça artmıştır. Bu bağlamda insanlık, günlük olarak $2{,}5 * 10^{18}$ (2,5 Kentilyon) bayt veri oluşturmaktadır. Bu verileri göz önünde canlandırmak gerekirse, yüksekliği üst üste yığılmış 10 milyon blu-ray diski dolduracak ve bu da 4 Eyfel Kulesinin üst üste yüksekliğini ölçecek seviyededir. Bu büyük verinin bir kısmını şunlar oluşturmaktadır:

- 6 milyar Google araması,
- 27 milyar Whatsapp mesajı,
- 4 milyar Facebook gönderisi,
- 78 milyon Instagram gönderisi,
- 432 bin Instagram hikâyesi,
- 16 milyon Linkedin araması,
- 130 milyon Tumblr gönderisi,
- 14 milyon saat Netflix izlemesi,
- 183 milyon Skype Çağrısı,
- 215 bin e-posta ve
- 59 milyon saat Spotify dinlemesi (Anonymous 2020a).

Teknolojilerin baş döndürücü hızı ve onlara bağımlılık çok fazla artmıştır ve önümüzdeki yıllarda katlanarak artacak gibi görünmektedir. Bu göz kamaştırıcı ve üssel hız, bugün dünya verilerinin %90'ının yalnızca son 2 yılda yaratılmış olması gerçeğiyle açıkça gözler önüne serilebilir (Anonymous 2020b).

Siber uzay etkisini çok artırırken, buna bağlı olarak siber saldırganlar da çabalarını artırmıştır. Fidye, içeriden gelen tehdit, politika, rekabet, siber savaş, öfke veya diğer çeşitli nedenler gibi farklı sebeplerle siber saldırganlar, sistemlere ve cihazlara zarar vermenin, durdurmanın, değiştirmenin veya izlemenin farklı yollarını bulduğu gibi bu

yöntemler çoğunlukla, gittikçe artan bir bedel ödeten zararlı yazılımlar aracılığıyla uygulanmaktadır. Zararlı yazılımların küresel maliyeti 2015'te 500 milyar ABD doları iken, günümüzde 2 trilyon ABD dolarına ulaştı ve 2021 sonunda 6 trilyon ABD dolarına yaklaşması beklenmektedir. Zaman geçtikçe Siber Güvenlik ve zararlı yazılımlarla başa çıkmanın bilgi sistemlerinde her zaman büyük önemi olacaktır. (Anonymous 2020c). Bu maliyet için en somut örnek, İran'ın dışarıya kapalı ağ (intranet) olarak kurup işlettiği Natanz'daki nükleer tesisleri yönettiği ağa verilen hasardır.

Bazı zararların milyarlarca dolar seviyelerinde ifade edilmeleri mümkünken, bazı zararları ifade etmek için bu seviyeler bile yetersiz kalmaktadır. Eğitim sistemlerini engellemenin eğitimin durmasına, adli bir veri tabanının yok edilmesinin adaletsizliğe ve suçluların özgür kalmasına, sıhhi bilgilerde değişiklik yapılmasının yanlış muayene ve muhtemelen ölümlere yol açması gibi; bir süper güç ulusun askeri sistemlerine sızılarak, topyekûn bir savaş ile elde edebilecek sonuçların benzerleri kolaylıkla ve doğrudan elde edilebilecektir.

Tüm bu felaket senaryolarından sonra elbette bu tür felaketlerin üstesinden gelmek için yapılabilecek şeyler vardır. Sistemlerin korunması ve iyi bir siber güvenlik sağlanması için ilk adım, zararlı yazılımlara karşı iyi bir koruma gerektirir. Zararlı yazılımlar boyutlarıyla yalnızca birkaç KB'den GB boyutlarına kadar değişebileceği gibi, karakteristik, tür, işlev ve hedef olarak farklılaşabilmektedir. Bir zararlı yazılım bilgisayarınızı devre dışı bırakabilir, klavye ve fare hareketlerinizi ve tıklamalarınızı izleyebilir, IBAN numarası, banka kartı bilgileri, kişisel sırlar gibi özel ve hayati bilgilerinizi çalabilir, eylemlerinizi herhangi bir büyük verinin bir parçası olarak kullanabilir ve hatta cihazınızın zombi veya kripto para madenciliği botu olarak yasa dışı kullanılmasına yol açabilir. Görüldüğü gibi, herhangi bir zararlı yazılım, herhangi bir kişiye, sisteme, organizasyona veya kuruluşa, bunlara ordular hükümetler ve istihbarat teşkilatları da dâhildir, hayati zararlar verebilir.

Siber uzaydaki potansiyel tehlike ve zararların sayıca artışı ve çeşitlenmesiyle birlikte siber zararlara karşı yapılan çalışmalarda da pek çok gelişme yaşanmaktadır. Kötü amaçlı bilgisayar korsanları (black hat hacker) sistemlere girmenin yeni yollarını

bulacağından ve iyi amaçlı bilgisayar korsanları (white hat hacker) bunları ortadan kaldırmak için yeni yollar deneyeceğinden, bu savaş hiç bitmeyen bir savaş olarak sonsuz dek sürecektir. Ayrıca günümüzdeki zararlı yazılımlar incelendiğinde çoğunun otomatikleşmiş süreçlerle veya "script-kiddy/lamer" (hacker olmamalarına rağmen siber ortamdaki tehlikeli kişiler) olarak adlandırılan ve daha önceden yazılmış olan zararlı yazılımları inceleyerek bunlar üzerinde değişiklikler yaparak tekrar üreten kişiler tarafından gerçekleştirildiğini göstermektedir (Aliyev V., 2010).

Günümüzde siber güvenlik konularında bilgi sahibi olup olmamasına bakılmaksızın, herhangi bir programlama dilini siber güvenliğe yönelik kütüphaneleri de dâhil olmak üzere bilen bir geliştirici, script-kiddy veya lamer hâlen GitHub, GitBucket, BitBucket, Launchpad gibi sayısız kod depolama sitelerinde yer alan kodlar üzerinde çalışmalar yapabilir. Bu yaptığı çalışmalarda üzerinde çalıştığı kod daha önceden antivirüs yazılımlarının veritabanına alınarak zararlı yazılım olarak tanınmış olsa dahi, sadece birkaç değişken veya fonksiyon adını değiştirerek yeni bir zararlı yazılım üretebilir. Bu yazılım aslında bir önceki yazılımın yaptığı işin tamamen aynısını yapmakta, ancak içeriği antivirüs sistemlerini aldatmak amaçlı küçük değişikliklere uğratıldığı ve dolayısıyla ileriki bölümlerde değinilecek olan "imza değeri" (hash) değiştiği için antivirüs yazılımlarınca tanınmayacak ve başka bir yazılım olarak değerlendirilecektir. Böylece fazla bir uğraş göstermeden yeni bir zararlı yazılım üretilmiş olabilecektir.

İşte bu nedenle, tüm bu tehlikelerle başa çıkmanın verimli ve otomatikleştirilmiş bir yolunu sağlamak için yeni bir yaklaşıma ihtiyaç vardır. Bu sistemin en büyük gerekliliği de sadece kendisine bildirilen yazılımlara göre değerlendirme yapmaktan öte, cihazlarda gerçekleştirilen işlemlerin örüntülerini saptayarak bu örüntüler neticesinde daha önceden kendisine bildirilmemiş durumlarda da yorum yapabilme yetisidir.

Tez çalışmasının literatüre sağladığı katkı, hâlen yetersiz kalan konvansiyonel zararlı yazılımlarla mücadele yöntemlerin zaaflarını gidererek evrensel olarak zararlı yazılımları tespit edilmesini sağlayan bir çözüm ortaya koymaktır. Araştırmanın arkasındaki ana fikrin ayrıntılı olarak Bölüm 5.2'de açıklanacak, söz konusu bölüme kadar geleneksel yöntemlerin neler oldukları, günümüzde neden yetersiz kaldıklarını

açıklayarak zararlı yazılımla başa çıkmak için basitliği ve hızı korurken en verimli yolun ne olabileceği tartışılarak ortaya çözüm önerisi konacaktır.

2. TEORİK ARKAPLAN

Bu bölümde, zararlı yazılımlar hakkında bazı bilgiler verilecek, bunların farklı yöntemlerle nasıl saptandığına değinilecek ve bu saptamada geleneksel yöntemlerin neden yetersiz kaldığı açıklığa kavuşturulacaktır.

2.1 Zararlı Yazılım Nedir?

Önde gelen virüsten koruma şirketlerinden McAfee'ye (McAfee, 2020) göre "Malicious Software" veya kısaca "Malware" olarak bilinen "Zararlı Yazılım", herhangi bir programlanabilir cihaza, sunucuya veya ağa zarar vermek veya bu sistemleri kendi yararına kullanmak için tasarlanmış bir koddur. Kimi kullanıcı etkileşimi gerektiren, kimi gizlenmek için bir süre bekleyen, kimiyse kendisini zamanla diğer ağ bileşenlerine dağıtan pek çok farklı özellikte zararlı yazılım türleri bulunmaktadır.

2.2 Zararlı Yazılım Türleri

Zararlı yazılımın tanımı geniş olduğundan, kötü amaçlı yazılım türlerini ve sınırlarını ayırt etmek hiç kolay değildir. Birçoğunun, sistem başlangıcından başlayarak, kendisini kullanıcıdan gizlemeyi tercih etme, algılanmamak için kullanıcının izni olmadan işletim sistemine entegre etme gibi pek çok ortak özelliği bulunmaktadır. Bu benzerliklerin yanı sıra, literatürde birçok farklı türde zararlı yazılıma, tanımlara ve sınıflandırmaya neden olan birçok farklılık da bulunmaktadır. Bu farklılıklara göre üzerinde uzlaşı sağlanan kötü amaçlı yazılım türlerinden bazıları şunlardır:

2.2.1 Virüs:

Adından da anlaşılacağı gibi bir bilgisayar virüsü, grip virüsü gibi, sistemin temellerini etkiler. Bir virüsün temel amacı, mağdura olası maddi ve manevi zararlar ile sonuçlanacak şekilde bir cihazı veya sistemi bozmaktır. Virüsler; çoğunlukla bir uygulamanın başlatılması, bir tıklama veya giriş cihazlarında bir hareket gibi kullanıcı etkileşimi gerektirirler. Virüsün varlığı, sistemi tamamen yavaşlatabileceği için net olarak hissedilebilir. Diğer zararlı yazılımlarda olduğu gibi, bunlar kasıtlı olarak

silinmesi zor bir şekilde tasarlanmıştır. Bazı diğer zararlı yazılım türlerinin aksine kendisini ağa yayma ihtiyacı hissetmez. Örneğin solucanların (worm) aksine, ağa ve diğer cihazlara yayılmak yerine özel bir hedefleri vardır ve üzerinde çalışırlar.

2.2.2 Solucan (Worm):

Virüsün aksine bir solucan, diğer sistem cihazlarına bulaşmak için insan yardımına veya etkileşime ihtiyaç duymayan kötü amaçlı yazılım türüdür. Bulaştırmak için potansiyel istismarları ve sosyal mühendisliği kullanırlar. Ağlar aracılığıyla kendini kolayca yeniden üretebilir ve diğer sistemlere bulaşabilir. Aslında temel amaçları ağ üzerinden yayılmak ve başka cihazlara da kendini enjekte ederek ağı ele geçirmektir. Virüslerle karşılaştırıldığında, solucanların farkının tek tek cihazlar yerine ağ veya sistemleri hedef aldıkları söylenebilir.

2.2.3 Truva atı (Trojan):

Truva atı, bir bilgi işlem sistemine sızan ve fark edilmeden kalmaya çalışan ve bu esnada değerli kaynaklarınıza erişen bir tür kötü amaçlı koddur. Ancak bu tür zararlı yazılımlar, diğer ağ cihazlarına dağıtılması gerekmeyen, yalnızca tek bir cihaza odaklanan türdendir. Adını bugün Çanakkale olarak anılan antik kent Troya'da gerçekleşen Truva Savaşı'nda kullanılan ünlü Truva atlarından almıştır. Bu atlar, savaşın bir tarafının hedef şehre girdiği bir hile olarak kullanılmış ve bu şekilde savaşın seyrini değiştirmiştir. Bu benzetmeden yola çıkılabileceği üzere truva atlarının en büyük özelliği bu zararlı yazılımların hedef sisteme gizlice sızması ve içeride değerleri kaynaklara gizlice erişen ve yeri geldiğinde bu kaynakları yönlendiren yazılımlar olmasıdır.

2.2.4 Reklam yazılımı (Adware):

Reklam yazılımı, kullanıcının izni veya rızası olmadan kullanıcının aleyhine olacak şekilde reklam yayınlayan bir tür zararlı yazılımdır. Bu tür kötü amaçlı yazılımların “en zararsız zararlı yazılım çeşidi” olduğu söylenebilir, zira reklamlar yüzünden alışıldık ve konforlu kullanıcı deneyimini bozar, ancak bunun dışında sisteminize ciddi bir zarar vermez veya kendisini ağ yoluyla diğer cihazlara yaymaz. Bazı güvenilir ücretsiz

yazılımların kurulum sırasında başka bir yazılım yüklemeye çalıştığı da göz önüne getirildiğinde, bu tür zararlı yazılımlar yalnızca agresif reklam olarak değerlendirilebilir.

2.2.5 Casus yazılım (Spyware):

Casus yazılım, kendisini herhangi bir şekilde sisteme yükleyen ve kurban tarafından gerçekleştirilen eylemleri takip eden zararlı yazılım türüdür. Diğer zararlı yazılım türlerinin aksine casus yazılımlar eylemleri hassas olup olmadıklarına göre filtrelemez. Böylece girilen siteler, izlenen videolar, yüklenen resimler, paylaşılan yorumlar ve hatta oynanan oyunlar casus yazılımların konusu olabilir. Adından da anlaşılacağı üzere bu tür bir zararlı yazılım, arka planda sessizce çalışan bir casus olarak tanımlanabilir.

2.2.6 Kök kullanıcı takımı (Rootkit):

Bilindiği üzere tüm sistemler farklı hesap türlerine sahiptir. Ve genellikle bunlar, sistem üzerinde daha yüksek ayrıcalıklara sahip yönetici (süper - admin) kullanıcılar ve yönetici kullanıcılara kıyasla sistemde bazı belirli işleri yapabilen normal (yerel - local) kullanıcılardan oluşur. Kök kullanıcı takımı adı verilen bu zararlı yazılımlar da bunu istismar ederek resmi bir şekilde ayrıcalıklarını yükseltmenin teknik bir yolunu ararlar. Böylece sistem üzerinde elde ettikleri yönetici yetkisiyle daha derin ve daha zararlı eylemler gerçekleştirirler.

2.2.7 Arka kapı (Backdoor):

Bir arka kapının kendisi, bir zararlı yazılımın parçası veya başlı başına ayrı bir program olabilir. Bu zararlı yazılımlar da kök kullanıcı takımları (rootkit'ler) gibi yönetici (süper - admin) kullanıcı yetkilerini istismar ederler. Bir kök kullanıcı takımı (rootkit) ile bir arka kapı arasındaki temel fark, bir arka kapının, ayrıcalıklı bir kullanıcı olarak kimlik doğrulaması yapmadan geleneksel kimlik doğrulama aşamasını atlatmasıdır. Rootkit'ler ise, sahte iddialar ve illegal yöntemlerle ayrıcalıklı kullanıcılar olarak kimlik doğrulaması sağlarlar. Yani hem kök kullanıcı takımları hem de arka kapılar yönetici yetkileriyle işlemler yapıyor olsa da; arka kapılar bu aşamayı by-pass ederek yönetici yetkilerini ele geçirirken, kök kullanıcı takımları bu aşamayı atlamadan aslında kendilerine verilmemiş olan yönetici haklarını illegal bir şekilde kimlik doğrulaması aşaması esnasında sağlarlar.

2.2.8 Tuş kaydedici (Keylogger):

Bir tuş kaydedici, klavyede yaptığınız her eylemi takip edebilen bir kötü amaçlı yazılımdır ve bu nedenle başta şifreler olmak üzere hassas ve kritik tüm verileriniz ortaya çıkar. Bu hassas veriler sadece şifreleri değil, aynı zamanda tuş vuruşları, tıklamalar, ekran görüntüleri, kamera görüntüleri, ses kayıtları vb. olası hareketleri de denetler. Ayrıca tek tıkla bulaşma olasılığı düşünüldüğünde, bu tür zararlı yazılımlar oldukça tehlikelidir. Bazı yazılımlar tarama esnasında diğer zararlı yazılımları tespit edebilse de, tuş kaydedicilerin tespiti kolay değildir. Zira tuş kaydediciler, işletim sistemlerinin kullanıcının eylemlerini gerçekleştirmesi için sağladığı legal servisleri kullanır ve işletim sistemleriyle doğrudan entegre olarak çalışır. Bu tarz yazılımların tanınamadığı göz önüne alındığında format atma, yeni işletim sistemi yükleme gibi işlemler gerçekleştirilmediği sürece hassas verilerin uzun süreler boyunca takip edilebileceği, bunun da tazmini zor zararlar vereceği aşikârdır.

2.2.9 Fidye yazılım (Ransomware):

Fidye yazılımları, saldırı nedeniyle kullanılamaz hale getirdiği şifreli dosyaların şifresini çözmek veya saldırı yoluyla elde edilen hassas veya kişisel verileri yayınlamamak karşılığında kurbandan para talep eden kötü amaçlı yazılımlardır. Bu zararlı yazılımlar, iyi bilinen ve çözülmesi kolay algoritmalar yerine kendi benzersiz şifreleme algoritmalarını kullanır. Bu sayede kullanıcı veya başka herhangi bir yazılım tarafından bu dosyaların şifresi çözülmesi kolay olmadığından saldırgana boyun eğmek durumunda kalınabilir. Gizlilik için kullanıcı, geleneksel yöntemler yerine kripto para birimleri ile ödeme yapmaya yönlendirilir ve böylece adli makamlarca yakalanma ihtimali olmaz.

2.2.10 Damlalık (Dropper):

Türkçe tam bir karşılığı yaygınlaşmamış olan Dropper'lar aslında kötü amaçlı yazılım değil, zararlı yazılımı taşıyan yazılım yükleyicileridir. Genellikle bir dosya uzantısı yoktur ve dolayısıyla tespit edilmesi daha zordur. Bu türdeki zararlı yazılımlar adlarını aslında içindeki sıvıyı taşımaktan başka bir fonksiyonu bulunmayan, sadece içindeki sıvıyla anlam kazanan tıbbi damlalıktan alırlar. "Stuxnet" olarak bilinen ve giriş bölümünde değerlendirmesi yapılan ünlü siber saldırı İran'ın nükleer program

tesislerine bu türde zararlı yazılımın kullanılmasıyla gerçekleştirilmiştir. Kurban tarafından yapılan tek hata, bu tarzda bir dosya barındıran USB cihazını nükleer tesis iç ağındaki bir cihaza takmaktı. Burada bu damlalık yazılımın, kendi içerisinde barındırdığı ve asıl kötü niyetli işlemi gerçekleştirecek yazılımı sistemdeki cihaza "damlattığı" çıkarımı yapılabilir.

2.2.11 İndirici (Downloader):

İndiriciler, damlalıklar gibi farklı yapıya sahip başka bir kötü amaçlı yazılım türüdür. Kendileri kötü amaçlı yazılım değillerdir, ancak İnternet üzerinden belirli kaynaklardan zararlı yazılım indirmek için kodlara sahiptirler. Bu zararlı yazılımın, geliştiricileri için ana dezavantajı, İnternet bağlantısına olan bağımlılığıdır. İnternet bağlantısı olmadığı takdirde sisteme zarar verecek ekstra bir yazılım indirilemez ve dolayısıyla da zararlı yazılımın geliştiricilerinin hedeflediği zararlı işlemler gerçekleştirilememiş olur. Kendileri zararlı yazılım olmadıklarından, tipik zararlı yazılımlardan farklı özelliklere sahiptirler ve bu nedenle geleneksel yöntemlerle tespit edilmeleri diğer zararlı yazılım türlerine göre daha zordur.

2.2.12 Uzak yönetim araçları (Remote administration tools):

Uzaktan Yönetim Aracı olarak adlandırılan yazılımlar aslında, ağları veya cihazları uzaktan yönetmek için kullanılan resmi araçlardır. Beklendiği üzere, bu resmi araçlar yeterli güvenlik önkoşulları gerektirir. Ancak kötü niyetli eylemler için bu araçlar, güvenlik koşulları atlatılarak, istismar aracı olarak kullanılabilmektedir. "RAT" olarak kısaltılan bu yazılımlar kötü niyetli bir şekilde kullanılmak istendiğinde, saldırganların kurban bilgisayarda uzaktan kodlar, kabuklar veya her türlü kötü niyetli eylemi yönetici ayrıcalıklarıyla yürütmesini sağlar.

2.2.13 Botlar (Bots):

İsimleri "Robot"tan türetilmiş olan "bot" zararlı yazılım araçları, muhtemel kötü niyetli veya illegal otomatik görevleri yerine getiren kötü amaçlı yazılımlardır. Siber saldırı

ağları, kripto para madenciliği veya başka herhangi bir şüpheli etkinlik için botlar, büyük bir bot topluluğunun parçası olarak daha da etkin bir şekilde kullanılabilir. Çok sayıda bottan oluşan bu tür topluluklara "botnet", bu ağların gerçekleştirdiği saldırılara ise "botnet saldırıları" adı verilir. Zombi olarak adlandırılan sistemler de aslında yukarıda açıklanan bir "bot"un yerine getirdiği görevlerle benzer görevleri yerine getirmektedir.

2.3 Zararlı Yazılım Analiz Yöntemleri

Zararlı yazılım analizi, kötü amaçlı yazılımın davranışlarının, ilerlemelerinin ve özetle amacının belirlendiği çalışmadır. Bu davranışlar arasında dosyaların oluşturulması, silinmesi ve değiştirilmesi, kayıt defteri anahtarları, işletim sistemi prosesleri (process) ve bir kötü amaçlı yazılımın yapabileceği diğer tipik işlemler bulunur. Bu analiz bazı araçlar, eylemler ve süreçler gerektirir. Ve tüm bu işlemler aşağıda listelendiği ve açıklandığı gibi üç analiz yöntemi halinde gruplanabilir.

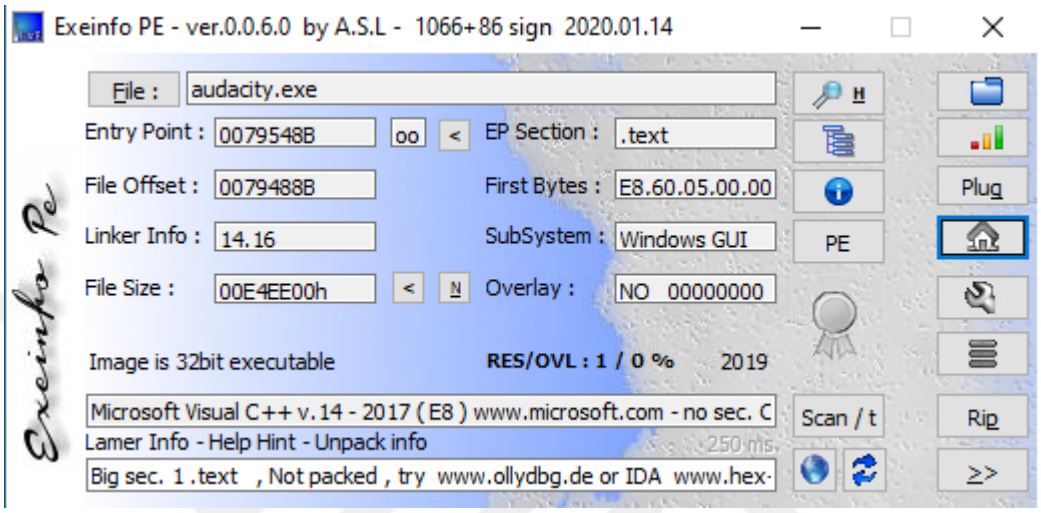
2.3.1 Statik zararlı yazılım analiz yöntemi

Yazılımın statik analizi, adından da anlaşılacağı üzere statik-durağan olarak, program fiilen çalıştırılmadan gerçekleştirilir. Bu yöntemde yazılımın kodu, kontrol akış grafikleri vb. bilgiler elde edilir (Damodaran, Anusha, vd. 2017). Bu yöntemde elde edilmeye çalışılan statik özelliklerden bazıları şunlardır: Özet (hash) değeri, Kimlik (identity) değeri, Dize (string) değerleri, Sağlama toplamı (checksum), Entropi, Başlık ve bölüm (section) adları, Derleme (compiling) bilgileri, Paket (package) bilgileri, Sertifikalar, Dizinler, Kaynaklar. Söz konusu statik özellikleri elde etmek için yaygın olarak kullanılan statik analiz araçlarından bazıları şunlardır: Exeinfo PE, PE Studio, IDA Pro vb. Bu yazılımlarla gerçekleştirilecek analizler aşağıda örneklendirilmiştir.

ExeinfoPE yazılımının vereceği bilgiler "Audacity" adında bir yazılım için Şekil 2.1'de örnek olarak verilmiştir. Buna göre bu program ile:

- Yazılımın adı,
- Yazılımın yürütülebilir olup olmadığını gösteren ve diğer bilgileri içeren başlık bilgileri,
- Çalıştırılabildiği işletim sistemi ve işlemci mimarisi (32 veya 64-bit)

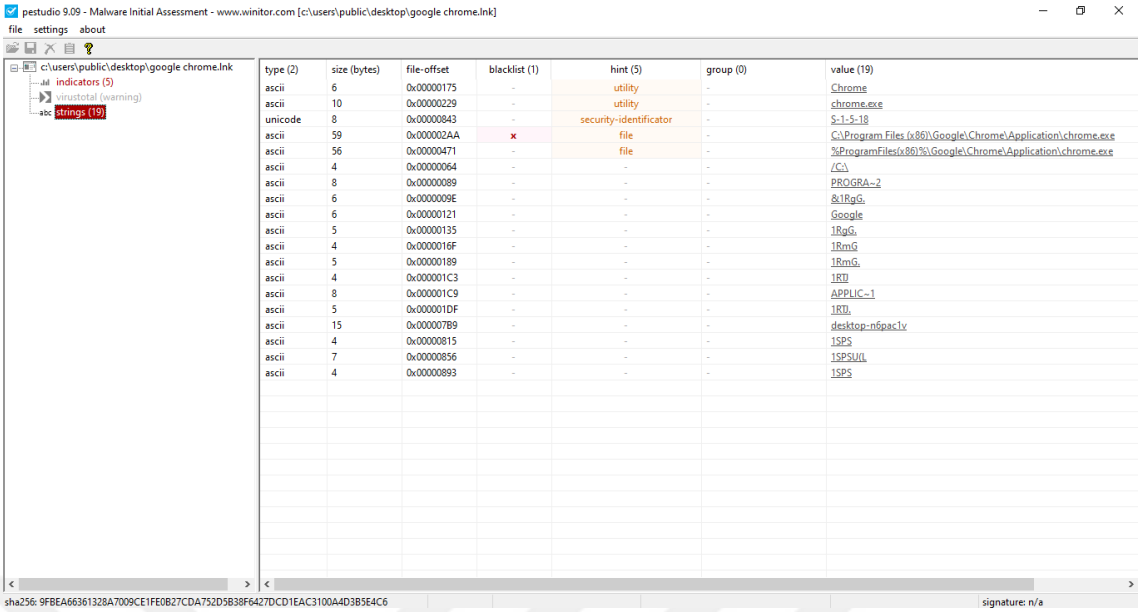
- Dosyanın başlık bilgilerinin büyüklüğü ve ofset (offset) değeri
- Dosyanın büyüklüğü
- Paketleme yazılımları ile dosyanın koruma altına alınıp alınmadığı ve paketlendiyse hangi algoritma ile paketlendiği
- Geliştirilme tarihi gibi birçok bilginin elde edilmesi mümkündür.



Şekil 2.1 Exeinfo PE Programıyla "Audacity" Adlı Yazılıma İlişkin Elde Edilen Bilgiler

PE Studio yazılımıyla elde edilebilecek bilgiler ise Şekil 2.2'de verilmiştir. Bu şekilde "Google Chrome" yazılımının içinde gerçekleştirilen faaliyetlere yönelik ön bilgiler verecek veriler görülmektedir. Bunlardan bazıları:

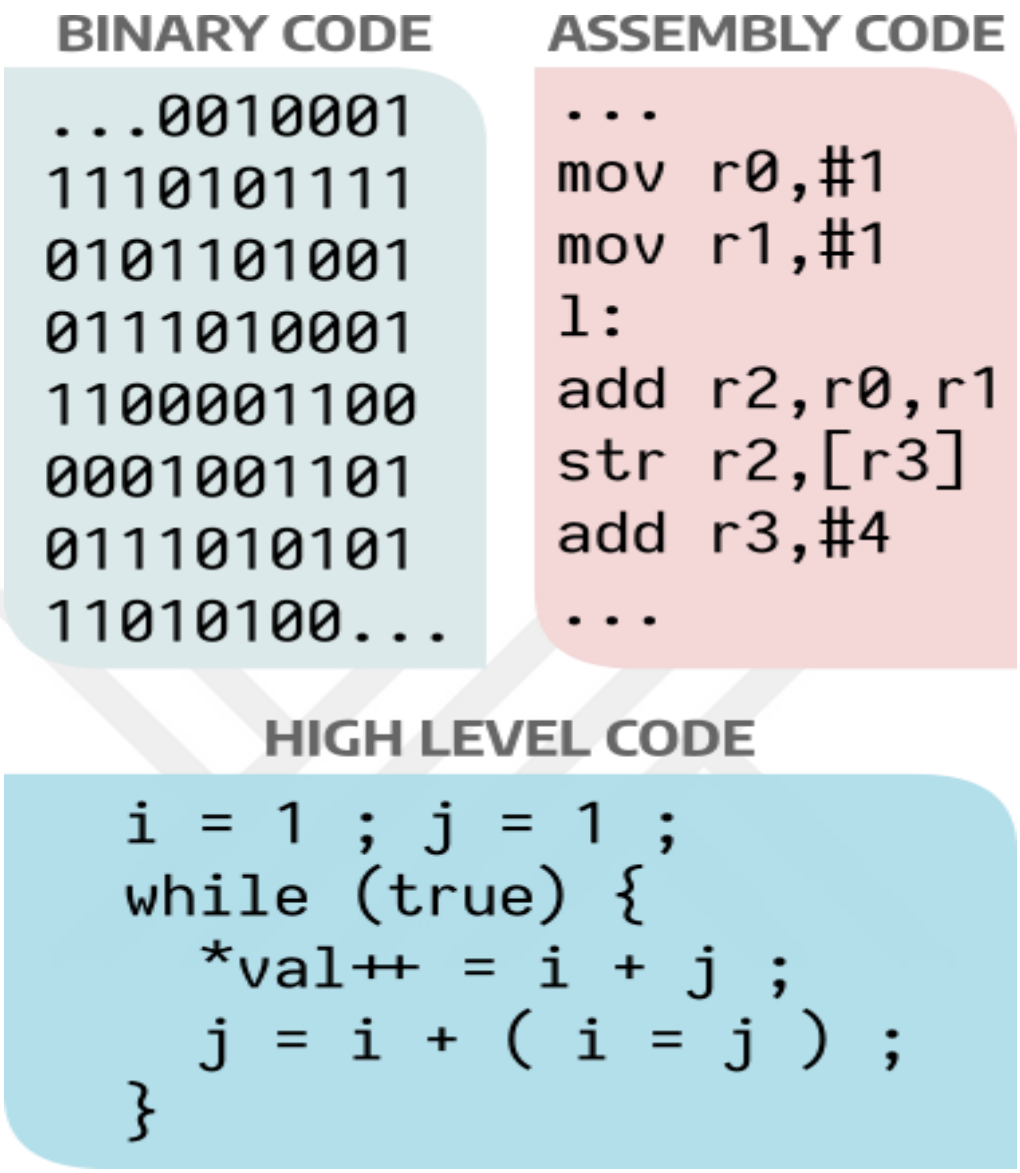
- Dosyayı çalıştıran işletim sistemi kullanıcısı
- Dosyayı çalıştıran bilgisayarın adı
- Dosyayı çalıştıran programın kendisi ve kısayolunun bulunduğu dizin ile
- Dosyanın kendisine verdiği isim gibi birçok bilgiler görülmektedir.



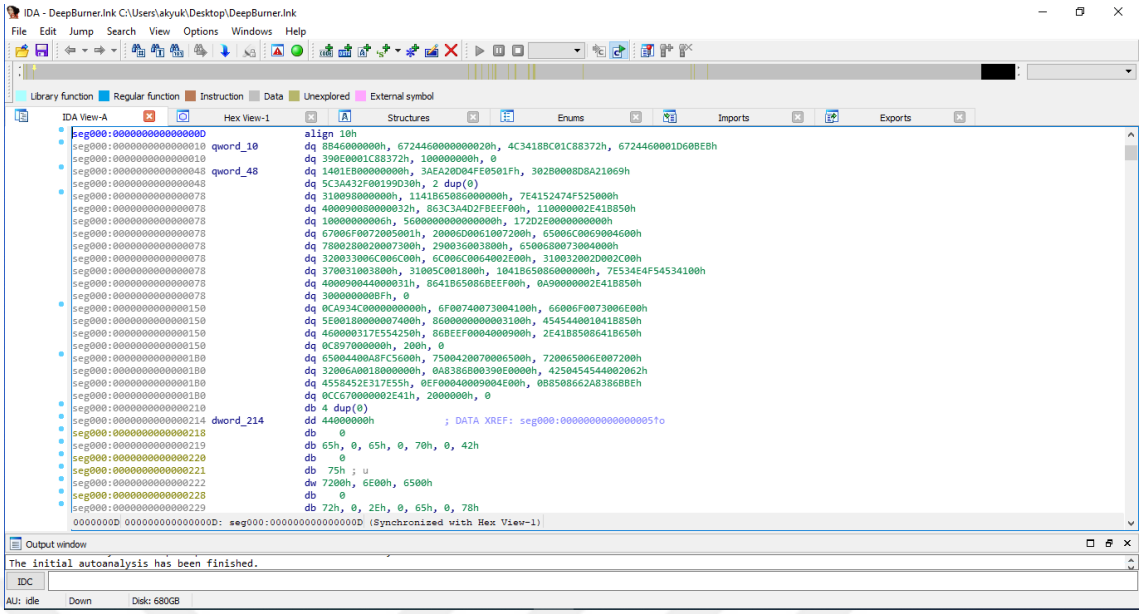
Şekil 2.2 PE Studio Programıyla "Google Chrome" Adlı Yazılıma İlişkin Elde Edilen Bilgiler

Dosyaların metaverilerinin yanısıra içeriklerine yönelik daha ayrıntılı bilgiler verebilen ve yaygın olarak kullanılan yazılımların en ayrıntılısı AidaPro'dur. İnsanlar tarafından Yüksek seviyeli programlama dillerinde (Java, C++, C# vb.) geliştirilen kodlar, bilgisayar tarafından önce Assembly sonra da Binary (ikili) koda dönüştürülerek yorumlanmaktadır. AidaPro adlı bu paket ayırıcının (disassembler) bize sağladığı fayda ise, paketlenme kod bulanıklaştırma gibi engelleri aşamasa da, Assembly dilinde kod içeriğini görmemizi sağlamasıdır. Bu sayede "if-else", "while-do" "goto" gibi program akışlarını istediğimiz gibi yönlendirerek yazılımın hangi durumlarda hangi işleri gerçekleştirdiğinin görülebilmesidir.

AidaPro adlı disassembler yazılımının çalışma mantığı Şekil 2.3'te, "Deep Burner" adlı yazılımın AidaPro ile elde edilen Assembly kodlarının bir kısmı Şekil 2.4'te verilmiştir.



Şekil 2.3 Yüksek Seviyeli Kod, Assembly Kod ve Binary (İkili) Kod (Hex-Rays 2020)

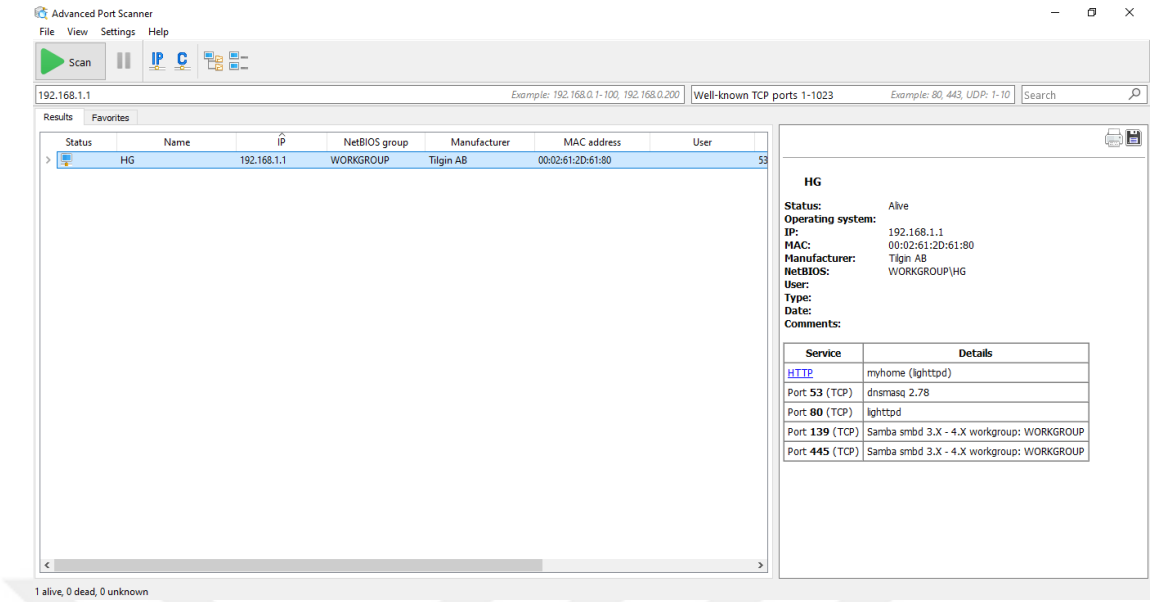


Şekil 2.4 “DeepBurner” Adlı Yazılımın AidaPro Programıyla İncelenmesi

2.3.2 Dinamik zararlı yazılım analiz yöntemi

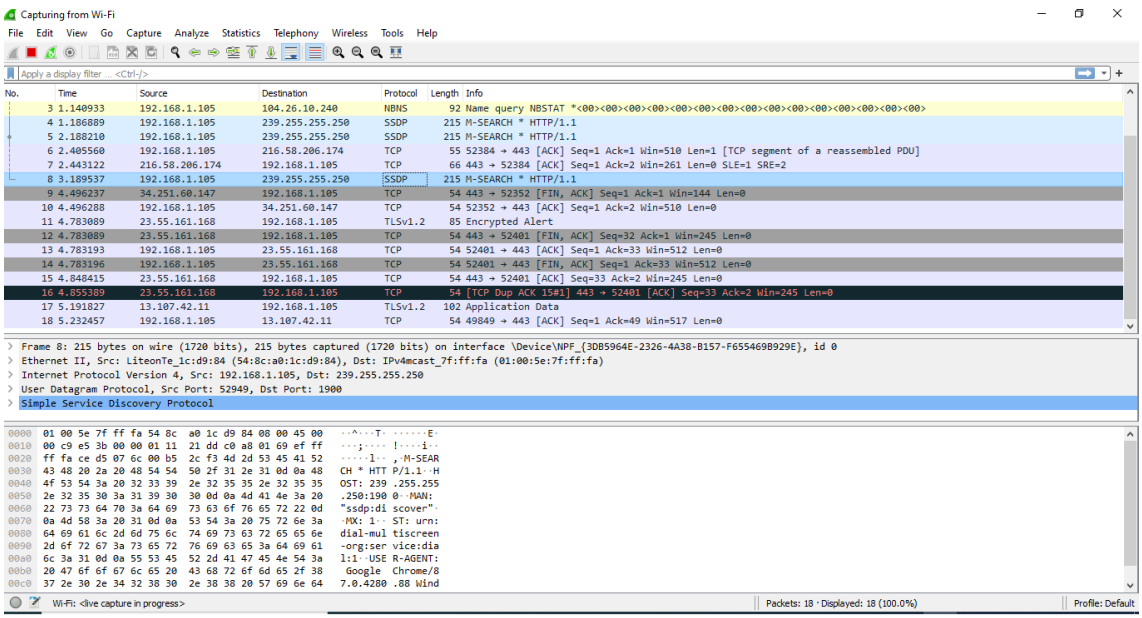
Yine adından anlaşılacağı üzere dinamik olarak gerçekleştirilen dinamik analiz zararlı yazılımın yürütülmesi sırasında davranışını gözlemler ve fonksiyon çağrıları, parametreler, bilgi akışı ve talimatlar (instructions) dâhil olmak üzere sistemin davranışını analiz eder. Bu yöntemde çıkarılan dinamik özelliklerden bazıları şunlardır: API Çağrıları (API Calls - bu tez çalışması bu özelliğe odaklanmaktadır), Kayıt defteri (registry) değişiklikleri, Donanıma erişim, HTTP Alma (GET)/ Gönderme(POST)/ Koyma (PUT)/ Silme (DELETE) istekleri, IP, DNS, URL bilgileri, Prosesler (process), Dosyalar. Bu özellikleri edinmek için yaygın olarak kullanılan dinamik analiz araçlarından bazıları şunlardır: PortScanner, Wireshark, Process Hacker, Procmon, Regshot, Procdot, OllyDebug vb. (Aslan ve Samet 2017). Bu yazılımlar kullanılarak elde edilecek bilgilere örnekler aşağıda verilmektedir.

Söz konusu araçlardan PortScanner IP adresi verilen bir cihazı tarayarak hangi portlarının iletişime açık olduğunu bildirir. Bu portları yazılımı çalıştırmadan önce ve sonra inceleyerek yazılımın herhangi bir saldırı gerçekleştirmek için kullanıcının izni olmaksızın port açıp açmadığı ortaya çıkarılabilir. Programın arayüzü Şekil 2.5’te gösterilmiştir.



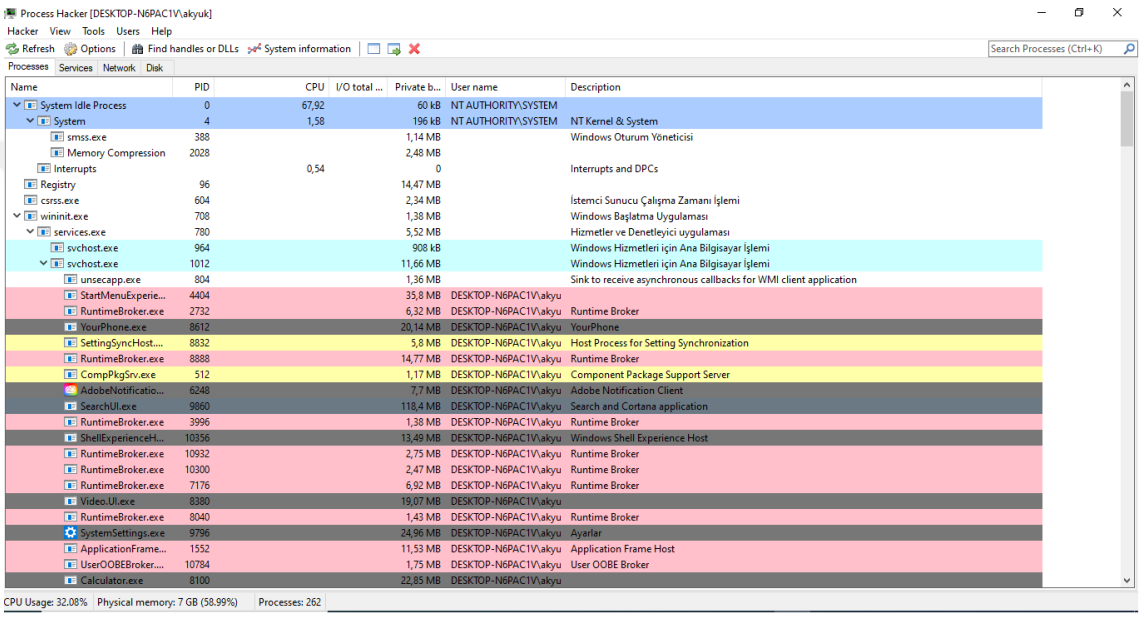
Şekil 2.5 PortScanner Programı ile Analizi Yapılacak Yazılımın Port Açtığının Tespiti

Wireshark programı ile çalıştırılan yazılımın hangi port/protokoller üzerinden hangi cihazlarla iletişim kurulduğu hangi paketlerin gönderildiği ve alındığı gibi birçok bilgi Şekil 2.6'da gösterildiği gibi tespit edilebilir. Burada herhangi bir güvenlik protokolü kullanılmadıysa, gönderilen ve alınan paketlerin içeriklerinin de okunabildiğini unutmamak gerekir.

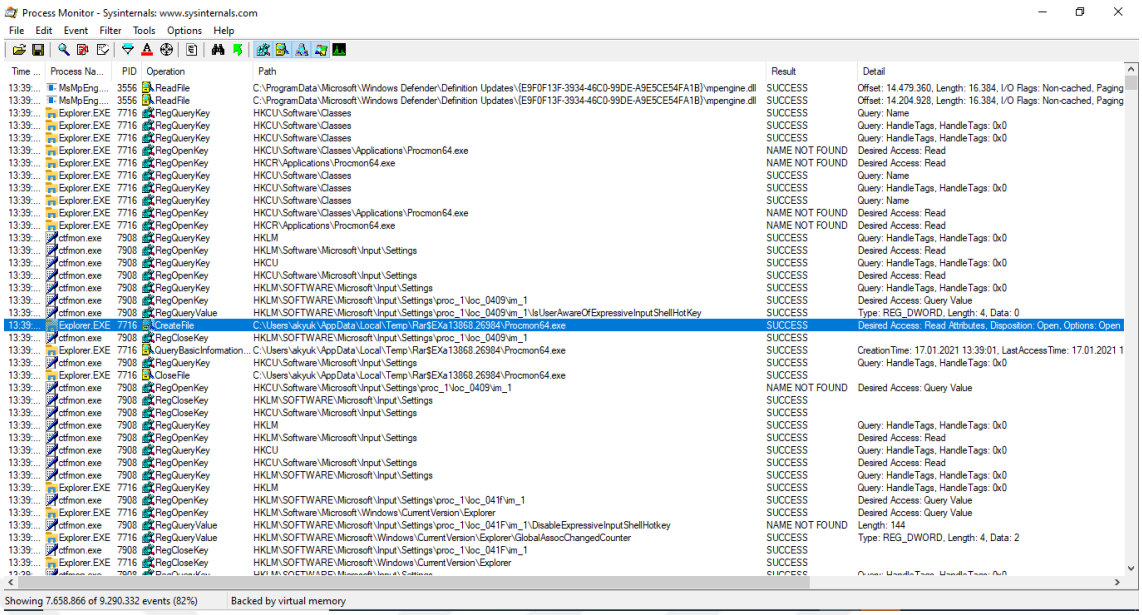


Şekil 2.6 Wireshark Programı ile Ağ Trafiğinin Tespit Edilmesi

Arayüzleri Şekil 2.7 ve Şekil 2.8'de verilen Procmon, Process Hacker ve Procdot gibi programlar ise, yazılımların hangi proses'leri çalıştırdıklarını, oluşturduklarını ve sildiklerini göstermektedir. Bu sayede, birçok zararlı yazılımın dikkat çekmemek için işletim sistemlerinin legal proses'lerine benzer proses'ler oluşturduklarından hareketle, işletim sistemlerinin oluşturup çalıştırdığı legal proses'lerin yanısıra, programdan beklenmeyen işlemlerin gerçekleştirilip gerçekleştirilmediği tespit edilmiş olur.

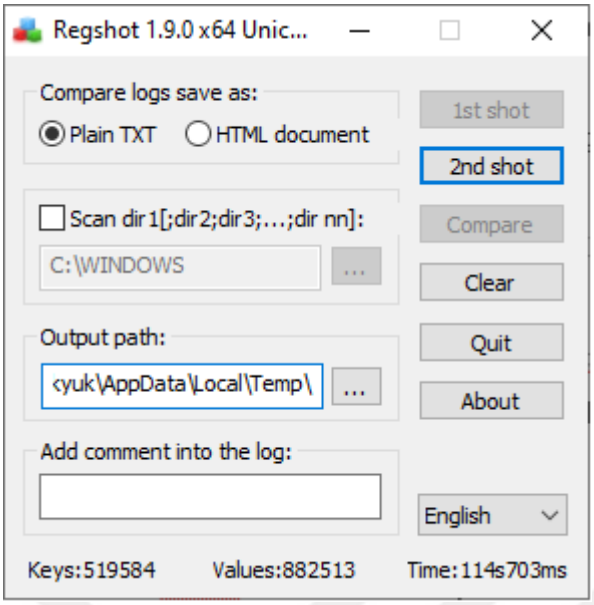


Şekil 2.7 Process Hacker Programı ile Hiyerarşik Yapı Altında İşlenen Proses'lerin Görülmesi



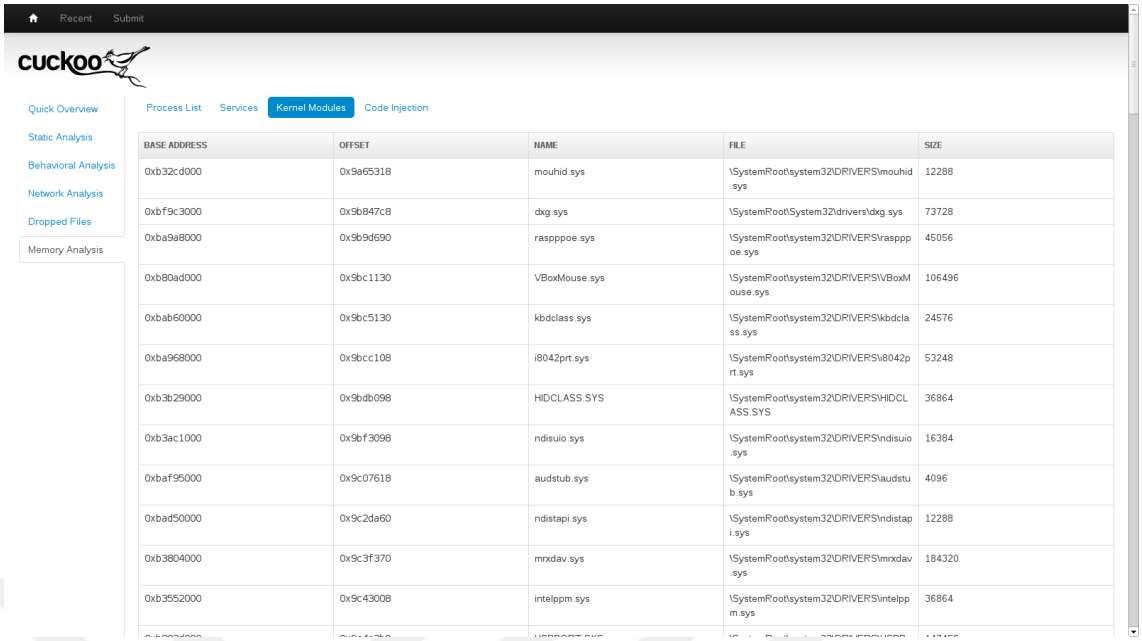
Şekil 2.8 Process Monitor Programı ile Hiyerarşik Yapı Altında İşlenen Proses'lerin Görülmesi

Regshot adı verilen program ise analizciye program çalıştırılmadan önce ve çalıştırıldıktan sonra hangi Registry (Kayıt Defteri) anahtarlarının silindiği, değiştirildiği ve eklendiğine dair bilgi sahibi vermek için kullanılmaktadır. Şekil 2.9'da örneği verildiği üzere bir programın kendisinden beklenenler dışında gerçekleştirdiği Registry değişiklikleri bu program vasıtasıyla izlenebilir.



Şekil 2.9 Yazılımın Değişiklik Yaptığı Registry (Kayıt Defteri) Değişikliklerinin Tespitine Yarayan Regshot Yazılımı Ekran Görüntüsü

Burada birçok araç/programa değinilmişken Cuckoo Sandbox'a ayrı bir başlık açmak gerekir. Cuckoo Sandbox yukarıda değinilen veya kapsam gereği burada değinilmeyen ve yazılımların gerçekleştirdiği her türlü dosya/kayıt defteri/proses/başlangıçta çalışan programlara ilişkin silme/değiştirme/yaratma/okuma/kopyalama gibi birçok hayati önemdeki işlemi bir sanal makine üzerinde gerçekleştirerek kullanıcıya istediği kriterler bazında raporlar sunan bir kum havuzudur.



Şekil 2.10 Cuckoo Sandbox ile Yazılımların Gerçekleştirdiği İşlemlerin Görülebildiği Bir Kullanıcı Arayüzü

Şekil 2.10'da görülen analiz parçacığının sadece Cuckoo Sandbox'ın yaptığı dinamik analizin hafıza analizini ilgilendiren kısımdaki kernel başlık bilgilerinin incelenmesi olduğu dikkate alındığında, programın ne kadar detaylı incelemeler sunduğu açıkça anlaşılabilecektir.

Cuckoo Sandbox'ın literatürde ve zararlı yazılım analizinde en yaygın kullanılan araç olmasının en başlıca sebepleri şunlardır:

- Ücretsiz ve %100 açık kaynak kodlu bir yazılımdır. Dolayısıyla da sektörde benzer hatta daha kısıtlı işlem gerçekleştirebilen ücretli yazılımlara göre çok daha avantajlıdır.
- Sadece yürütülebilir (executable) dosyaları değil, aynı zamanda PDF, Word, Excel gibi içlerine zararlı script (betik) veya makrolar yerleştirilmiş çok geniş ölçekte birçok programı da analiz edebilir.
- Yine sektördeki muadilleri de aynı işi yerine getirseler de, bu tezin de üzerinde çalışacağı API Çağrılarını en belirgin ve sorunsuz şekilde çıkarabilen kum havuzu yazılımıdır.
- Statik ve dinamik analizde belirtilen programların ortaya çıkardığı verilerin tamamını sağlayabilir. Yani bu kum havuzu ile dosyanın çalıştırılma saati, başlık bilgisi,

paketlenme bilgisi, dosya yürütüldüğünde hangi web siteleriyle hangi protokoller üzerinden bağlantı kurduğu, bu bağlantının ne kadar süre açık kaldığı, kapatılıp kapatılmadığı, bu bağlantılar üzerinden internet üzerinden dosya indirip indirmediği, indirdiyse hangi dizine indirdiği, yazılımın Registry ayarlarında silme/okuma/yazma/yaratma işlemlerinden herhangi birisini gerçekleştirip gerçekleştirmediği, hangi proses'leri ürettiği ve bunları ne kadar süre ile çalıştırarak hangi işlemleri gerçekleştirdiği, hangi API Çağrılarının hangi sıra ile çalıştırıldığı, bu API Çağrılarına işletim sisteminden döndürülen cevaplar, oluşturulan/silinen/değiştirilen/şifrelenen dosya ve dizinler gibi bir zararlı yazılım analizinde ihtiyaç duyulan bilgilerin neredeyse tamamını analizciye verebilmektedir.

- Kendi üzerinde tanımlı olan "yara" kuralları ile zararlı yazılımların yaygın olarak gerçekleştirdiği örüntülerden (pattern) tespit ettiklerini kullanıcıya bildirir.
- Her şeyden önemlisi tüm bunları bilgisayar üzerinde kurulan bir sanal makine ile yaptığı için; verilerin kaybolması, değiştirilmesi, çalınması gibi zararlı yazılımın gerçekleştirdiği işlemlerin tamamının izole edilmiş bir cihazda gerçekleştirilmesini sağlayarak zararlı yazılım analizi uzmanlarına büyük kolaylıklar sağlamaktadır.

Cuckoo Sandbox dışında web arayüzleri ile çalışan ancak hem daha kısıtlı özellikler sunan hem de ücretsiz sürümlerinde sadece belirli sayıda yazılımı incelemeye alan JoeSandbox, DRAKVUF gibi başka kum havuzları da bulunmaktadır. Ancak hiçbiri Cuckoo Sandbox'ın sağladığı avantajları sağlayamamaktadır.

2.3.3 Hibrit zararlı yazılım analizi

Hibrit Analiz Yöntemi, Bölüm 2.3.1 ve 2.3.2'de değinilen iki analiz yöntemi olan statik ve dinamik analiz yöntemlerinin dezavantajlarından kaçınmaya çalışırken avantajlarından da azami seviyede faydalanmayı amaçlamaktadır. Önceki iki yöntemin dezavantajları ile karşılaşıldıktan sonra, her iki yöntemin eksikliklerinden kaçmak için yeni bir entegre yöntem olması gerektiği bu yöntemin ortaya çıkmasındaki temel motivasyon olmuş ve bu şekilde zararlı yazılımla mücadeledeki başarı oranı daha da artmıştır.

2.4 Zararlı Yazılım Tespit Teknikleri

Zararlı yazılımların tespiti için temelde İmza-Tabanlı, Davranış-Tabanlı ve Bulgusal-Tabanlı (heuristic) olmak üzere 3 temel teknik kullanıldığı söylenebilir:

2.4.1 İmza-tabanlı tespit:

İmza-Tabanlı Tespit, en çok sıklıkla kullanılan zararlı yazılım tespit tekniğidir. İmza, belirli zararlı yazılımları tanımlamak için kullanılabilen bir bayt dizisidir ve imza tabanlı anti-virüs yazılımı, bilinen zararlı yazılımların imzalarının bir veritabanını tutar. Bu veritabanı yeni tehditler keşfedildikçe sık sık güncellenmelidir. Güncel bir imza veritabanı gerektirme dezavantajı olsa bile basit, hızlı ve en yaygın zararlı yazılım türlerine karşı oldukça etkilidir (Damodaran, Anusha, vd. 2017).

2.4.2 Davranış-tabanlı tespit:

Davranış-Tabanlı zararlı yazılım tespit teknikleri, programın zararlı mı yoksa zararsız mı olduğunu belirlemek için programın davranışını gözlemler. Kodun kendisini değil program davranışını arar. Fonksiyon veya değişken isimleri gibi program kodları değişse de, programın davranışı aynı veya benzer olacaktır; bu nedenle zararlı yazılım bu yöntemle hala kolayca tespit edilebilmekte ve kod bulanıklaştırma tekniklerine rağmen başarılı olunmaktadır (Aslan ve Samet 2017).

2.4.3 Sezgisel-tabanlı tespit:

Sezgisel-Tabanlı Tespitte, ilgili programın belirli karakteristik özelliklerini tespit etmek için makine öğrenimi tekniklerine kadar uzanan bir dizi işlemler gerçekleştirilir. Bu yöntem, tespite başlanmadan önce birkaç örnek kullanılarak eğitilir, ardından eğitilmiş sistem, zararlı yazılımları tespit etmek için yeni program örnekleri için çalıştırılır. Bu şekilde bilinmeyen zararlı yazılımlar belirli bir dereceye kadar yakalanabilir, ancak yine de karmaşık veya sıfır gün kötü amaçlı yazılımları tespit edilemez (Aslan ve Samet 2017).

2.4.4 Geleneksel metotların yetersizliği

Siber güvenlik alanında K-En Yakın Komşu (K-Nearest Neighbour - KNN), Naive Bayes Sınıflandırıcı, Karar Ağacı ve Destek Vektör Makinesi (Support Vector Machine

- SVM) gibi Makine Öğrenmesi yöntemlerinin heterojen yapıların zararlı niyetlerini ortaya çıkarmada vazgeçilmez araçlar olduğu kanıtlanmıştır (Kilgallon, S., De La Rosa, vd. 2017).

Elle yapılan analiz, antivirüs yazılımı kullanımı gibi geleneksel yöntemler, bugün için zararlı yazılımlarla başa çıkmak için yetersizdir. Çünkü bilgi teknolojilerinin ortaya çıkmasıyla, zararlı yazılım geliştirme uzun bir yol kat etmiş ve zararlı yazılımların sayısı inanılmaz bir şekilde artmıştır. Ayrıca yazılım ve kötü amaçlı yazılım geliştirme için her zamankinden çok daha fazla kaynak bulunmaktadır. Böylece, herkes internette bir kötü amaçlı yazılım örneğini kolayca bulabilmekte ve onu az çok değiştirerek antivirüs yazılımı aracılığıyla tespit edilemeyen yeni bir kötü amaçlı yazılımı üretebilmektedir.

Yukarıdaki nedenlerden dolayı, bir yazılımın zararlı olup olmadığını tespit etmek için bazı temel kurallara bağlı olmayan, durumdan duruma etkinliği azalmayan, stabil ve sağlam bir çözüm bulunması gerektiği açıktır. Makine öğrenimi yöntemlerinin yaptığı da budur. Bu makine öğrenimi yöntemleri sadece kurallara bağlı kalmamakta, aynı zamanda varsa zararlı yazılımların temel davranış kalıplarını da yakalayabilmektedir.

3. MAKİNA ÖĞRENMESİ YÖNTEMLERİ

Bu bölüm, makine öğrenimi yöntemlerini anlamak için gerekli teorik arka planın temellerini atacaktır. Daha sonra bu araştırma boyunca kullanılacak yöntemler hakkında ayrıntılı bilgi verilecektir.

3.1 Temeller

Makine öğrenimi, açık bir şekilde programlanmadan öğrenmek ve iyileştirmek için verileri kullanma becerisiyle sistemleri güçlendirmeyi amaçlayan bir yapay zekâ alt alanıdır (Rege ve Mbah 2018). Yapay Zekâyı birçok problemde statik olarak kullandıktan sonra araştırmacılarda, daha değişken problemlere daha dinamik çözümler oluşturma yönünde katkı sağlayabileceği fikri yaygınlaştı. Bu çalışmalar kapsamında şimdiye kadar çeşitli matematiksel modellere dayanan birçok farklı makine öğrenimi yöntemi geliştirilmiştir.

3.2 Yaygın olarak Kullanılan Makine Öğrenmesi Teknikleri

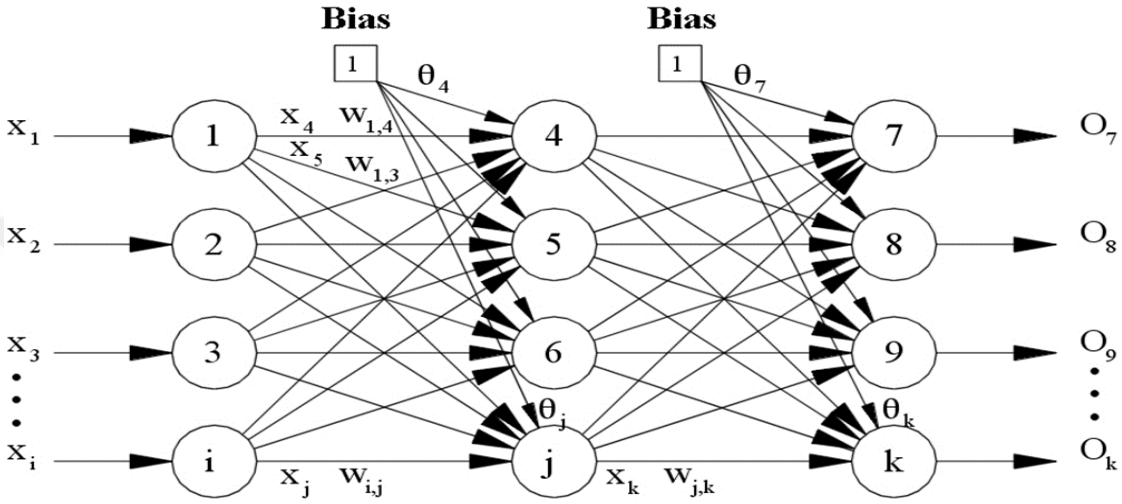
Farklı alanlar ve farklı amaçlar için başarıyla kullanılan ve farklı doğruluk oranlarıyla sonuçlanan birçok farklı makine öğrenimi yöntemi vardır. Literatür dikkate alınarak siber güvenlik alanında en yaygın kullanılan makine öğrenimi yöntemleri bu araştırmada kullanılacaktır. Bu yöntemler şu şekilde sıralanabilir: Derin Öğrenme, J48 Karar Ağaçları, K-En Yakın Komşular, Naive Bayes Sınıflandırıcı, Rassal Orman Ağacı ve Destek Vektör Makinesi. Farklı teknik ve mimarilerle çok farklı alanlarda başarılı sonuçlar ortaya koyan Derin Öğrenme bu çalışmanın ana odak noktası olmakla beraber diğer makine öğrenmesi yöntemlerine de değinilecektir.

3.2.1 Derin öğrenme

Derin öğrenme, bir dizi algoritmaya dayalı bir makine öğrenimi dalıdır. En başarılı yapay sinir ağları ve derin öğrenme yöntemlerinden bazıları Derin Sinir Ağlar (Deep Neural Networks-DNN), Evrişimli Sinir Ağları (Convolutional Neural Networks-CNN), Derin İnanç Ağları (Deep Belief Networks-DBN) ve Yığın Otokodlayıcıdır (Stacked Auto-Encoder-SAE). Burada değinilen birçok farklı mimarisiyle yapay sinir ağları son yıllarda giderek daha popüler hale gelmeye ve çok farklı alanlarda başarılı sonuçlar vermeye başlamıştır. Bu popülerlikle beraber bilgisayarla görme, konuşma tanıma ve

doğal dil işleme alanlarında da sıklıkla uygulanır hale gelmiştir. Sonuçta, araştırmalar derin öğrenmenin çoğu alanda geleneksel yöntemleri tamamen geride bıraktığını göstermektedir (Wang 2015).

Ortalama bir derin öğrenme mimarisi temel olarak Şekil 3.1'de gösterilen birim ve yapılardan oluşur:



Şekil 3.1 Ortalama Bir Derin Ağ Mimarisi (Anonymous 2020d)

Girdi Nöronları (Input Neurons - *1,2,3,i*): Bu düğümler, gerçek dünyadan girdi olarak gerçek verileri alır ve bunları bazı prosedürler altında işler. Problemin çözümünü etkileyen değişkenlere bağlı olarak giriş nöronlarının sayısı belirlenir.

Ağırlıklar (Weights - *X1,X2,X3,...):* Bunlar, ağ için girdi olarak gerçek girdiyle beraber kullanılan verilerdir. Bu verilerin değerleri bir alandan diğerine değişebilir. Kesin bir kural olmamakla beraber bu değerler başlangıçta rastgele atanarak öğrenmeye başlanır.

Çıktı Nöronları (Output Neurons - *7,8,9,k*): Bu düğümler, sorunun çözümüyle ilgili çıktı verilerini üretir. Çıkış nöronlarının sayısı, çözülecek problemin muhtemel kaç olasılıkla şekillenebileceğine göre belirlenir.

Çıktı (Output - *07,08,09*): Bunlar, çıktı nöronları tarafından üretilen çıktı verileridir.

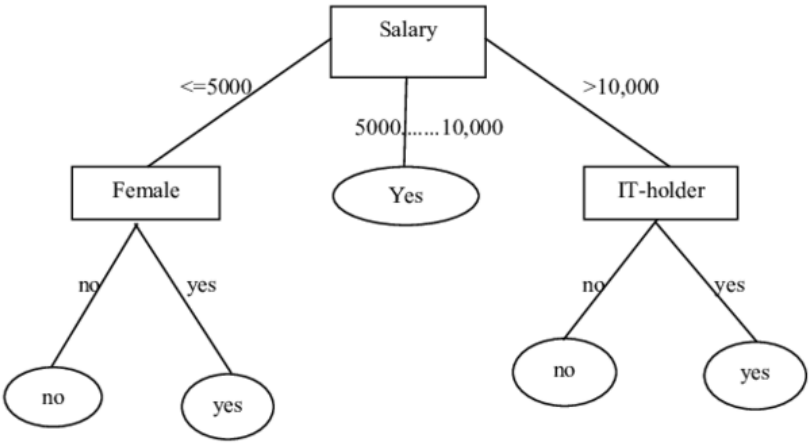
Gizli Nöronlar (Hidden Neurons - *4,5,6,j*): Bu düğümler, önceki katmandan alınan girdileri işleyerek doğrusallığı önler. Gizli katmanların sayısı ve bu katmanlarda yer alan nöron sayıları belli olmadığı gibi belirlenmesi konusunda bilimsel bir yöntem de yoktur, bir alandaki bir sorudan diğerine değişir.

$$f(x) = b + w1 \cdot x1 + w2 \cdot x2 + \ldots + wn \cdot xn \qquad (3.1)$$

Basit bir formül olarak, 3.1'deki formül herhangi bir nöronun herhangi bir çıktısının, gelen her girdinin çarpımlarının ve ağırlığının toplamı olduğunu göstermektedir. İnsan beyninin işleyişinden türetilen "$f(x)$", gelen bir sinyalin eşik değerini geçip geçmediğine göre bir sinyalin ateşlenip ateşlenmeyeceğine karar veren ve yaygın olarak kullanılan Sigmoid, Tanh, ReLU gibi çeşitleri olsa da hangi sorunda hangilerinin kullanacağının belirli olmadığı fonksiyonlardır.

3.2.2 J48 karar ağacı

J48, bilgi entropu ile karar ağaçlarının oluşturduğu C4.5 algoritmasının bir uygulamasıdır. Ağacın her düğümünde, kendisini birden çok alt kümeye en etkin şekilde ayıran özellik seçilir. Bölme, Bilgi Kazanımı (IG) veya Gini değerine göre yapılır. Karar verme için, en yüksek normalleştirilmiş IG'ye sahip özellik kullanılır (Wang 2015). J48 Ağacının gerçekleştirmesine dair bir örnek Şekil 3.2'de verilmiştir.

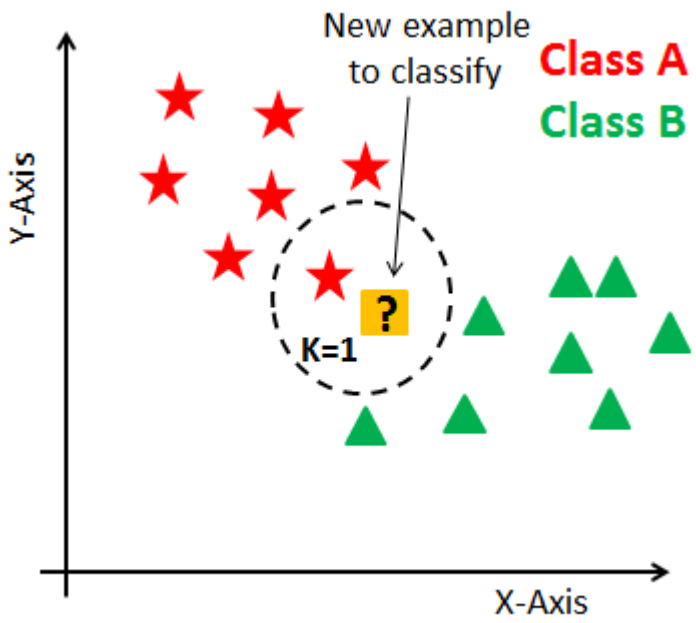


Şekil 3.2 J48 Karar Ağacı Gerçekleştirmesine Yönelik Bir Örnek (Researchgate 2020)

Bu algoritmada, model, veri kümesiyle ilgili sorulara göre eğitim yapar. Kök düğümden başlayarak, amacımız verileri homojen parçalara ayırmaktır. Bunu, grupları en iyi ayıran verileri bölümlere ayırarak yapılır ki böylece algoritmayı minimum aşamada dallandırarak istediğimiz sonucu elde edilebilsin.

3.2.3 K-En yakın komşu

K-En Yakın Komşu (K-Nearest Neighbors - KNN) en basit, ancak doğru sonuç veren makine öğrenimi algoritmalarından biridir. KNN parametrik olmayan bir algoritmadır, yani veri yapısı hakkında herhangi bir varsayımda bulunmaz. Karşılaştığımız gerçek dünya problemlerinde, veriler genel teorik varsayımlara nadiren uyarak parametrik olmayan algoritmaları bu tür problemler için iyi bir çözüm haline getirir. KNN model temsili, veri kümesi kadar basittir - öğrenmeye gerek yoktur, tüm eğitim verisi depolanır. Ve hem sınıflandırma hem de regresyon için kullanılabilir. K-En Yakın Komşu algoritmasının örnek bir gerçekleştirmesi Şekil 3.3'te verilmiştir.



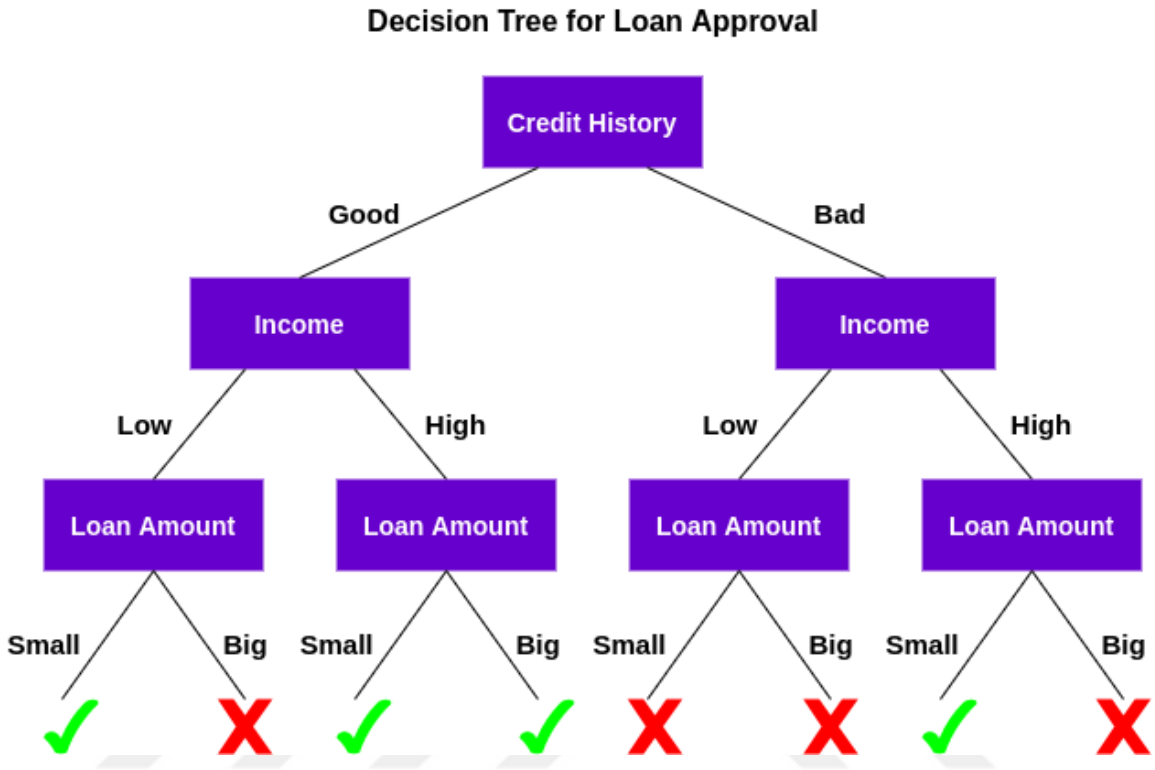
Şekil 3.3 K-En Yakın Komşu Algoritmasının Görsel Bir Gerçekleştirmesi (Anonymous 2020e)

Bu algoritma içinde, tek sayı olması gereken bir K değişkeni ayarlamamız beklenmektedir. Bu tek sayı, bilinmeyen bir nesnenin hangi sınıfa ait olduğunu belirler. Her bilinmeyen nesne için sınıf, bu nesnenin ait olduğu sınıf numarasının çoğunluğu tarafından belirlenir. Bir çoğunluğun oluşabilmesi ve bir eşitliğe meydan verilmemesi amacıyla K parametresi tek sayılardan belirlenmelidir.

3.2.4 Naive bayes sınıflandırıcı

Bu algoritma, girdi veri kümesinin özelliklerinin birbirinden bağımsız olduğunu önceden varsayan olasılıksal sınıflandırıcıdır. Ölçeklenebilirdir ve kayda değer sonuçlar üretmek için herhangi bir büyük eğitim veri kümesi gerektirmez (Apruzzese, Colajanni, vd. 2017).

3.2.5 Rassal karar ağacı



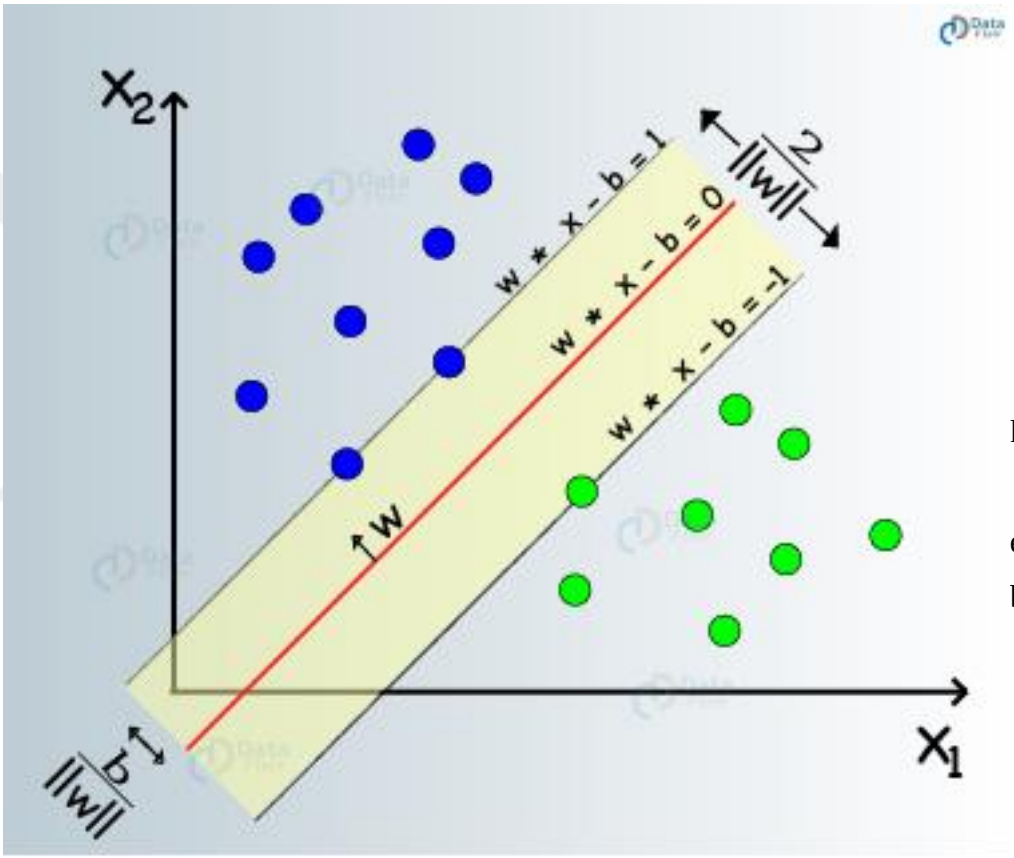
Şekil 3.4 Bir Rassal Karar Ağacını Gösteren Bir Grafik (Anonymous 2020f)

Rassal Karar Ağacı (Random Forest-RF), saptırma kavramının rastgele seçilmesine dayanan bir makine öğrenme yöntemidir. Bu tür bir kombinasyonun özelliği, her bir karar ağacının rastgele bir parametre vektöründen oluşturulmasıdır. Bir Rassal Karar Ağacı, her karar ağacı için bir özellik alt kümesini rastgele örnekler (Rastgele Alt Uzaylarda olduğu gibi) ve/veya her karar ağacı için bir eğitim verisi alt kümesini rastgele örnekler (Bagging yönteminde olduğu gibi) (Bernard, Heutte, vd. 2009).

Bu algoritma içinde aynı anda birden fazla ağaç işlenir. Ve bu işlem, ağaçların birbirleriyle ilişki kurmaması için kullanılır. Rassal Karar Ağacı gösteren bir örnek Şekil 3.4'te verilmiştir.

3.2.6 Destek vektör makinesi:

Bunlar, her bir örnek kategorisi arasındaki mesafeyi en üst düzeye çıkarmak amacıyla bir özellik uzayındaki veri örneklerini eşleştiren olasılıklı olmayan sınıflandırıcılardır. Giriş özellikleri üzerinde herhangi bir varsayımda bulunmazlar, ancak çok sınıflı sınıflandırmalarda kötü performans gösterirler. Bu nedenle ikili (binary) sınıflandırıcılar olarak kullanılmaları gerekir. Sınırlı ölçeklenebilirlikleri, uzun işlem sürelerine neden olabilir (Apruzzese, Colajanni, vd. 2018).



Şekil 3.5 Destek Vektör Makinesinin Gerçekleştirilmesi (Anonmyous 2020g)

Bu algoritmayı kullanırken, 2 boyutlu bir kategoril endirme için, iki sınıfın kümeleri arasında bir çizgi belirlenir. Ve tüm nesneler, bu sınıflar arasındaki çizgi ile ayrılmış bir sınıfa ait olarak tanımlanır. Destek Vektör Makinesi algoritmasının iki boyutlu örnek bir gerçekleştirmesi Şekil 3.5'te verilmiştir.

4. LİTERATÜR TARAMASI

Jiaz vd. (2019) kötü amaçlı yazılımları analiz etmek için farklı bir yaklaşım denemiştir. Yalnızca dinamik analizden değil, aynı zamanda statik analizden de yararlanmışlardır. Dinamik analiz için "Cuckoo Sandbox" adı verilen bir Zararlı yazılım analizi kum havuzu kullanılmış ve bu kapsamda 2300'den fazla özellik (feature) çıkarılmıştır. Statik analiz için PEFILE programı kullanılmış ve 92 öznitelik çıkarılmıştır. Özellikle, dosya başlıklarından 9 özellik, isteğe bağlı başlıktan 40 özellik, dosyanın bölümlerinden 10 özellik, DOS başlığından 17 özellik, dizin giriş kaynaklarından 4 özellik ve dizin girişi yükleme yapılandırmasından 20 özellik seçilmiştir. Dinamik analiz için Kayıt defteri, DLL'ler, API'ler ve özet bilgileri olmak üzere dört tür özellik kullanılmış ve bu dört özelliğin kombinasyonu kullanılarak, 9 farklı dinamik özellik kombinasyonundan istifade edilmiştir. Ayrıca muhtemelen araştırmaya en çok katkı sağladığını düşündükleri bazı özellikleri "önemli özellikler" olarak vurgulamışlardır. Araştırma, 39000'i kötü amaçlı yazılım olarak etiketlenen 49000 dosyanın binary kodlu dosyasından oluşan bir veri kümesi üzerinde gerçekleştirilmiştir. Makale, bir dosyayı analiz etmek için ne statik analiz ne de dinamik analizin yeterli olmadığının altını çizmektedir. Ve bu hibrit yöntemle, her ikisinden de yararlanırken, yaklaşık %97 doğruluğa erişmişlerdir.

Shijo vd. (2018), her iki yöntemin de kendi artıları ve eksileri olduğu için hem dinamik analiz hem de statik analizden oluşan hibrit bir analiz yöntemi kullanmanın avantajlarına odaklanmıştır. Zararlı yazılım yürütülebilir dosyalarının VirusShare topluluk web sitesinden toplandığı kendi veri kümeleriyle çalışmışlardır. Statik analiz için PSI (Yazdırılabilir dizeler bilgisi – PrinTablo Strins Information) değerleri, özellikler (feature) olarak çıkarılarak, dinamik analiz için özelliklerin çıkarılmasında "sistem çağrı sıklıkları" kullanılmıştır. Özellik belirlemede kıstas olarak veri kümesinde yer alan tüm API çağrıları yerine, veri kümesinde sezgisel olarak belirledikleri 2 eşiğinden daha fazla kez yer almış API Çağrıları dikkate alınmıştır. Ayrıca n-gramdan yararlanmayı değerlendirmiş ve biraz araştırma yaptıktan sonra sadece 3-API-çağrı-gramları ve 4-API-çağrı-gramlarını kullanmışlardır. Cuckoo Sandbox'tan istifade ederek statik yöntem, dinamik yöntem ve hibrit yöntem (hem dinamik hem de statik analizden oluşur) olmak üzere üç yöntem uygulanmıştır. Ve sırasıyla %95.8, %97.1,

%98.7 doğruluk oranlarına erişmişler, bunun sonucunda da "entegre yöntem" olan hibrit yöntemi en iyi olarak ilan etmişlerdir. İlginç bir şekilde bu, Rassal Ormanların yerini Destek Vektör Makinesi'nin aldığı ve Rassal Orman dışındaki yöntemlerin statik yöntem, dinamik yöntem ve hibrit yöntem (hem dinamik hem de statik analizden oluşan) açısından daha iyi doğruluk oranlarına sahip olduğu neredeyse tek makaledir. Bahsedilen doğruluk oranları sırasıyla %94.84, %96.65 ve %97.68'dir.

Kumar vd. (2019), çalışmalarında farklı bir yaklaşım sergilemiştir. Çoğunlukla kötü amaçlı yazılımın yürütülmesi tamamlanmadan önce farkına varmaya odaklanmışlardır. Diğer araştırmalarda, genel olarak tüm yazılım davranışları gözlemlenmekte ve ardından yazılımın zararlı mı yoksa zararsız mı olduğuna karar verilmektedir. Ancak bu makalede araştırmacılar, yazılım çalıştırıldıktan sonraki sürenin ilk 4 saniyesinde yazılımı zararsız veya zararlı yazılım olarak etiketlemeyi hedeflemişlerdir. Bu sayede zararlı yazılımın yürütülmesi bilgisayarda tamamlanmadan önce önlem almaya çalışmayı hedeflemişlerdir. Veri kümelerini Cuckoo Sandbox'ta çalıştıran yaklaşık 120000 örnekten oluşan başka bir makaleden toplayarak bu örnekleri VirusTotal'e göndererek gerçekten zararlı yazılım olup olmadığını çapraz kontrol etmişlerdir. Ulaşmak istedikleri hedef oldukça zor gözükse de iddia ettikleri yüzdelerden sonra başarılı olarak adlandırılabilmeleri mümkündür. En iyi puanları Rassal Orman Forest sınıflandırıcısını kullanarak %99.74'tür. Sırasıyla %96.73 ve %96.31 doğruluk oranları ile paketlenmiş ve gizlenmiş kötü amaçlı yazılım örnekleri üzerinde de yaklaşımlarını denediklerini iddia etmektedirler. Ayrıca, kötü amaçlı yazılımları ilk 4 saniyede sınıflandırırken (tespit yapılmamış sadece sınıflandırma üzerinde çalışılmıştır) %99.74'lük doğruluk, diğer makalelere kıyasla hatrı sayılır derecede yüksek ve tatmin edici bir orandır.

Liu vd. (2019) çalışmalarında 13518 zararlı ve 7860 zararsız örnek dâhil 21378 örnekten yararlanmışlardır, ancak örnekleri nereden çıkardıklarını belirtmemişlerdir. Dinamik analiz için Cuckoo Sandbox'ta örnekler çalıştırarak yazılımların davranışlarını sanal makinelerde görmektedirler. Ayrıca yaygın olarak 3 veya daha fazla API çağrı dizileri için bir eşik değeri belirlemiş ve dolayısıyla bir API çağrısı 3 veya daha fazla kez çağrılırsa, API çağrısı bir özellik olarak kabul edilmiştir. İlginç bir şekilde, bu çalışmamızda uygulanacağı üzere Kayıt defteri anahtarları (registry), klasörler vb. gibi

başka davranışlar kullanmamışlardır. Veri kümelerini eğitim kümesi, doğrulama kümesi ve test kümelerine bölmüşler ve tespit modeli olarak BLSTM'yi inşa etmişlerdir. "API çağrılarının" özellik olarak kullanıldığı bu çalışmada içlerinde GRU, BGRU, LSTM, BLSTM ve simpleRNN gibi birçok karmaşık işlemler sonunda en iyi derece olarak BLSTM yöntemi ile en iyi %97.85 puanını elde etmişlerdir. İkinci yöntem LSTM ise %97.55 doğruluk oranı vermiştir.

Pascariu vd. (2017) çalışmalarında özellikle proseslere (process) odaklanmaktadır. Ve bu prosesleri Microsoft Windows tarafından üretilen Sysmon adlı bir yardımcı program aracılığıyla kaydetmektedirler. Prosesleri kendileri kullanmazlar, ancak onları "bilinmeyen" (özellikle yazılımların adını benzersiz bir şekilde işleyen ve muhtemelen en yüksek riskli olan), "işletim sistemi" (explorer gibi işletim sistemi düzeyinde kullanılır. exe, svchost.exe vb.), "komut dosyası oluşturma" (cmd, powershell vb. gibi), "kullanıcı" (Word, excel vb. Günlük etkinlik programlarından oluşur) şeklinde sınıflandırmışlardır. Yalnızca Yapay Sinir Ağları (YSA) kullanırlar ve YSA, 5 giriş nöronu ile 3 çıkış nöronundan oluşur. Çalışma kayıtları (log) ana bileşen olarak kullanılırlar. Bunları kodlayıcılara girerler ve kodlamadan sonra YSA'ların girişlerini alırlar. Kayıtlarda 5 özyinelemeli üst (parent) yöntemi (function) dikkate alırlar (5'ten fazla varsa ihmal edilir). Sonuç olarak, tespit için iyi bir yöntem öne sürmüş olsalar da yöntemlerinin doğruluk yüzdelerini paylaşmamışlardır, bu da çalışmanın başarılı olup olmadığı konusunda net bir yargıya varmayı mümkün kılmamaktadır.

Hansen vd. (2016), bu makalelerinde belirttikleri ve daha önceden gerçekleştirdikleri ilk araştırmaya göre doğruluk oranlarını geliştirdiklerini belirtmişlerdir. Bu makalede bahsedilen son çalışmada, 270000'den fazla kötü amaçlı yazılımdan ve 837 iyi huylu yazılımdan oluşan bir veri kümesi kullanıyorlar ki böyle bir görev için bu derece yanlı bir veri kümesinin yanlılık sorunu oluşturabileceği aşikârdır. Ancak yine de uygulanan yöntemler üzerinde durulmaya değerdir. Bu çalışmada sanal makineler ve Cuckoo kullanılmıştır. Ancak, 200 API, 157 API sıklık sayısı, 14 API bölme sıklığı, 25 Ortak DLL olmak üzere toplam 396 özellik (feature) kullanılmıştır. Tespit için bunu yeterli bulmuşlar ancak kötü amaçlı yazılım aile sınıflandırması için yazılımların imzalarından da özellik olarak yararlanmışlardır. Çoğunlukla WEKA Aracı ile uyguladıkları Rassal

Orman'a odaklanarak veri kümesinin %67'sini eğitim ve %33'ünü test için kullanmış ve çalışmalarının sonunda "tespit" için %98.1 doğruluk elde ettiklerini belirtmişlerdir.

Damodaran vd. (2017)'nin çalışmasına gelince, daha geniş kapsamlı bir makale olması nedeniyle diğerlerinden biraz farklıdır; çünkü öncelikle dinamik, statik ve hibrit modeller hakkında bilgi verirken ayrıca akademik çalışmalarına ilişkin ayrıntılara değinmişlerdir. Yaygın olmayan Buster Sandbox Analyzer ile veri kümelerini hazırlamışlardır. Kötü amaçlı yazılım örnekleri, Harebot'tan 45 örnek, Güvenlik Kalkanı'ndan 50 örnek, Smart HDD'den 50 örnek, Winwebsec'ten 200 örnek, Zbot, ZeroAccess'ten 200 örnek ve 40 zararsız örnekten oluşan yazılım ailelerinden oluşturulmuştur. Toplama özellikleri için iki ortak odak noktası API çağrı dizileri (API Calls) ve işlem kodu (opcode) dizileridir. İşlem kodu için, tüm işlenenleri, etiketleri, yönergeleri vb. ve yalnızca call, push, add, test, jz gibi anımsatıcı kodlar korunmuştur. Diğer çalışmalardan farklı olarak, Cuckoo Sandbox yerine Buster Sandbox Analyzer'dan istifade edilmiştir. Farklı zararlı yazılım aileleri için %53.6-%99.8 arasında doğruluk oranlarına erişmişlerdir. Dinamik, statik ve hibrit yöntemleri denedikten sonra, raporlarını dinamik analizin daha maliyetli ancak daha iyi bir doğruluk oranına sahip olduğu gerçeğiyle sonuçlandırmışlardır. Gelecek çalışma olarak da, bu çalışmada Gizli Markov Modeli (Hidden Markov Modeli-HMM) denedikten sonra, bir Destek Vektör Makinesi (Support Vector Machine-SVM) yöntemi yaklaşımını tercih edeceklerine işaret etmektedirler.

Darshan vd. (2016), yalnızca Windows işletim sistemli bilgisayarlar üzerinde çalışmışlardır. Beklendiği üzere sanal makinelerden istifade ederek yazılımları bu alanda yaygın olarak kullanılan Cuckoo Sandbox üzerinde çalıştırmışlardır. Makine Öğreniminde kullanılan zararlı yazılımların davranışlarının bir formatı olan MIST (Malware Instruction Set)'ten ve n-gramlardan yararlanırlar. Sistem çağrılarını (API Çağrıları olarak da adlandırılır) çıkarırlar ve bunları n-gram olarak biçimlendirirler. Kapsamlı bir deneyden sonra, WEKA aracı içinde Bayesian Logistic Regression, SPegasos, IB1, Bagging, Part ve J48 gibi yöntemler üzerinde çalışmalarını tamamlayarak, çalışmalarını SPegasos ve 200/400/600 n-gram en iyi yöntem olarak tanımlamışlardır.

Aslan vd. (2017), herhangi bir makine öğrenimi yöntemiyle ilgili olmayan daha temel bir çalışma yapmışlardır. Bunun yerine insanlar tarafından gerçekleştirilen manuel analiz ile tespiti tercih etmişlerdir. 100'ü zararlı, 100'ü zararsız olmak üzere toplam 200 yazılım örneğinden oluşan ve farklı kaynaklardan çıkarılan kendi veri kümelerini kullanmışlardır. Ayrıca "Statik Analiz Araçları", "Dinamik Analiz Araçları", "Çevrimiçi Araçlar", "Araçların (tool) Kombinasyonu" olarak belirtilen yöntemler arasında en yüksek doğruluğun, "Araçların (tool) Kombinasyonu" kullanılarak elde edildiği sonucuna varmışlardır. Bu bahsedilen araçlar şunlardan oluşur: PEiD, PEView, IDAPro, Process Monitor, Wireshark, Cuckoo Sandbox, VirusTotal, Malwr. Ve statik analiz için PEiD, PEview, PE Explorer, PEBrowser Professional, BinText, UPX, MD5deep, Dependency Walker, Resource Hacker, IDA Pro, Hex Editors ve Hex-Ray Decompiler. Sonuçta, %87.5 oranıyla makale, makine öğreniminin bu tür manuel yöntemlerden çok daha etkili olduğunu gösteriyor.

Sun vd. (2018), korumalı alan günlüklerini tek tip ve iyi tanımlanmış, şekilli bir forma dönüştürmek için kendi veri kümelerini ve kendi yöntemlerini üretmişlerdir. Bu şekilde kayıt defteri anahtarı (registry) değişiklikleri, API çağrıları, mutex işlemleri vb. gibi çok çeşitli özellikleri kullanmaktadırlar. Tarzlarına; tüm metni küçük harfe dönüştürmek, yol sınırlayıcı olarak "/" kullanmak, web sitelerinin http ve https karakterlerini kaldırmak gibi bazı önlemlerin kullanıldığı "AMAR-Üretici Sistemi" (AMAR-Generator System) olarak adlandırırlar. Ayrıca API çağrılarını, esas olarak close_handle, reg_open, reg_create, reg_enumerate, reg_set, reg_query_key, reg_query, reg_del, open_file, create_file, copy_file, create_dir ve mutexler olarak sınıflandırmışlardır. Yeni bir yöntem işaret ettiklerinden, farklı yılların FFRI veri kümelerinde %74.87 - %100 arasında doğruluk oranlarına ulaştıklarını belirtmektedirler. Ancak gelecekteki işler için hala daha yapabilecekleri çok şey olduğu değerlendirilmektedir.

Lengyel vd. (2014), DRAKVUF adlı başka bir dinamik zararlı yazılım analiz sisteminin esas olarak bu makalede açıklandığını belirtmektedir. Çalışmalarını Xen Sanal Makinesi üzerinde yaparak Yürütme izleme, DKOM saldırılarının üstesinden gelme, bellek olaylarıyla dosya sistemi erişimini izleme ve bellekten silinen dosyaları işleme gibi yeni yöntemleri denemektedirler. Bunların etkinlikleri göz önüne alındığında çok karmaşık

bulunmaktadır. TDL4, Zeus, Shadowserver e.t.c. dâhil bazı zararlı yazılım örnekleri üzerinde çalışmışlardır. Sonunda, DRAKVUF'un zararlı yazılım örneklerini analiz etmenin iyi bir yolu olduğunu iddia etseler de ölçeklenebilir bir doğruluk oranı vermemektedirler. Dolayısıyla, çalışma ve yayımlandığı makalede geliştirilebilecek birçok husus olduğu değerlendirilmektedir.

Fujino vd. (2015)'in, çok yararlı bulduğumuz zararlı yazılımları tespit etmek için API Çağrısı konularını kullanan makalesi bizim de bu çalışmada yapacağımız gibi API çağrılarını temel almıştır. Çalışmalarına başa çıkılması gereken çok sayıda API çağrısı olduğunu, bu nedenle hangisinin özellik olarak seçileceğini bulmak için bir yönteme ihtiyaç duyulduğunu belirterek başlamışlardır. Kendi mantıkları dâhilinde bir hesaplama yaparak deneysel olarak belirlenen bir eşik değeri bulmuşlardır. API çağrıları üzerinde çalıştıktan sonra, API çağrısının değeri belirlenen eşiğin altındaysa atılır, ancak eşiğin üzerindeyse özellik olarak seçilir. Makalede bu eşik değerinin 0.1 ile 0.5 arasında olmasının optimal yaklaşım olacağını da belirtmişlerdir. Gözetimsiz (unsupervised) olarak devam ettikleri çalışmalarında makalelerinde atıfta bulundukları çalışmaların da gözetimsiz (supervised) öğrenmeyi kullandığını belirterek devam etmişlerdir. Ancak bu güzel çalışmanın devamında herhangi bir doğruluk oranı vermedikleri için çok iyi başladıkları çalışmanın yarım kalmış sayılabileceğini değerlendirilmektedir. Zaten ileride Bölüm 5'te anlatacağımız üzere çok daha basit bir metotla çok iyi bir doğruluk yüzdesini API Çağrılarını kullanarak almış durumda olduğumuz söylenebilir.

Kilgallon vd. (2017), hem dinamik analizden hem de statik analizden yararlandıkları için çalışmalarında hibrit zararlı yazılım analizini kullanmıştır. Rassal Ormanlar, Karar Ağaçları, Sinir Ağları, K-En Yakın Komşu, Destek Vektör Makinelerini denemişlerdir. Makalede belirtildiği üzere en iyi sonuçları sırasıyla Sinir Ağı ve Rassal Orman kullanılarak elde etmektedirler. 332 yazılımdan oluşan bir veri kümesi ile zararlı yazılım tespiti için %90 ve sınıflandırma için %92 olduğunu iddia etmektedirler. Ancak doğruluk oranlarından öte çalışmanın yaptığı ana katkının yazılımın tespitini yazılım çalışırken yapıyor olmaları olduğunu söylemek yerinde olacaktır.

Pirscoveanu vd. (2015), VirusShare'den indirdikleri yaklaşık 80000 örnekten oluşan kendi veri kümesini kullanmaktadır. Bu örnekleri Cuckoo Sandbox'ın korumalı

alanında (sanal makinada) çalıştırmakta ayrıca zararsız yazılımların yaygın olarak kullandığı bir beyaz liste oluşturmuşlardır. Bu çalışma ile gereksiz özelliklerden (feature) kurtulmayı amaç edinmişlerdir. Zararlı yazılımlarla başa çıkarken 4 tür bilgi kullanmaktadırlar: DNS bilgileri, erişilen dosyalar, muteksler ve kayıt anahtarları (registry). VirusTotal'ın etiketlerini başlıca 4 grup olarak kullanırlar: truva atı, potansiyel olarak istenmeyen program, reklam yazılımı, kök kullanıcı takımı (rootkit). Çalışmalarının sonunda, 160 ağaçla Rassal Orman algoritmasını kullanarak, %98 gibi iyi bir doğruluk oranını elde etmişlerdir.

Mehra vd. (2015), makalelerde çoğunlukla ihmal edilen çok önemli bir konu olan HCI'ye (İnsan Bilgisayar Etkileşimi) odaklanmaktadır. Bu, bazı kötü amaçlı yazılımların ihtiyaç duyulan herhangi bir insan etkileşimine bakılmaksızın çalıştığı, ancak bazılarının bir şaşırtma yöntemi olarak İnsan Etkileşimini gerektirdiği anlamına gelir. Makale, temel olarak, kum havuzlarının bazı kötü amaçlı yazılımları analiz etmek için yararlı araçlar olduğunun altını çizip, bunun insan tarafından tetiklenince yürütülmeye başlanan zararlı yazılımlar söz konusu olduğunda ise tamamen kabul edilemez olduğuna temas etmektedirler. Makale ayrıca, zararlı yazılım analizi için bazı yararlı araçları, olayla tetiklenen kötü amaçlı yazılımlarla başa çıkarken etkinlikleri açısından çeşitli korumalı alanları (sandbox) karşılaştırmaktadır.

Choudhury vd. (2018), mutekslerin açıklaması ve kötü amaçlı yazılım örneklerini analiz ederkenki önemlerini açıklamak dışında pek katkı sağlamamaktadır. İlk olarak Cuckoo korumalı alanını ve uygulanışını açıklayıp sonrasında günümüzde zararlı yazılım yazarlarının kodlarını paketledikleri ve bunun da zararlı yazılım analizcilerinin, kod çıplak gözle görülemeyeceği ve içeriği gizlendiği için statik analizde zorluklara neden olduğu gerçeğini vurgulamışlardır. Daha sonrasında zararlı yazılımların önemli işaretleri olan mutekslerle devam eden çalışmada; mutekslerin, sistem kaynaklarının eşzamanlı erişimini kontrol eden bayraklar/programlar olduğu belirtilmiştir. Bu nedenle sistemde birden fazla kod örneği çalışıyorsa yalnızca bir örneğin çalışmaya devam edeceği ve diğer yazılımların aynı anda daha fazla örnek çalışıyormuş gibi değerlendirilerek durdurulması neticesinde zararlı kodun amacına ulaşamayabileceği sonucuna ulaşılmıştır.

Udayakumar vd. (2017), çalışmalarında yeni bir şey ortaya koymak yerine literatürü gözden geçirmeye odaklanmıştır. Günümüzde kötü amaçlı yazılım tespitinin önemi ile başlayan çalışma statik analiz yerine dinamik analize odaklanılması gerektiğiyle devam etmektedir. Zararlı yazılım tespiti/sınıflandırması alanında yazılmış 38'den fazla makaleden bahsedilmiştir. Açıklanan literatür araştırmasından sonra, zararlı yazılım analizi hakkında temel seviyede bilgi vermektedirler. Bilgiler temelde ".exe" dosyaları ve ".dll" dosyaları hakkındadır. Kötü niyetli olanları bulmak için güvenli ".dll" dosyalarını kullanmanın faydalı olabileceği değerlendirmesine müteakip davranışlar karşılaştırılarak, "muhtemelen zararlı" ve "muhtemelen zararsız" .dll dosyaları belirlenir. Net olarak bir çalışma ortaya koymayan makale, son kısımlara doğru bu alandaki herhangi bir uygulama için bazı öneriler vermektedir ve bunu temelde özetle, Cuckoo Sandbox'ın kullanılmasıyla özetlenebilir. Sonuç olarak, daha önce de belirtildiği gibi makale ortaya bir çalışma veya doğruluk oranı koymayıp bir literatür taramasından ibarettir.

Walker vd. (2019), Cuckoo korumalı alanının literatür genelinde takdir edilen faydasının aksine ek bir özelliğine odaklanmakta, ve makaleyi bunun üzerine yoğunlaştırmaktadırlar. Çalışmada Cuckoo korumalı alanının zararlı yazılım örneklerini analiz ederken yararlı olduğunu, ancak "tehdit puanlama" özelliğinin verimsiz olduğunu belirmişlerdir. Malpedia'dan aldıkları kendi zararlı yazılım örnekleriyle çalışarak bazı araştırmalar yaptıktan sonra Cuckoo Sandbox'ının "tehdit puanlama" özelliğinin geliştirilmesi gerektiğine işaret etmektedirler. Ancak kendi yaptığımız değerlendirmede Cuckoo'nun resmi sayfasında ve uygulama arayüzünde verilen bilgilere bakılacak olursa, uygulama geliştiricileri zaten bu özelliğin yeni bir özellik olduğunu ve yine de iyileştirme için çalışmalara devam ettiklerini kendileri belirtmektedirler. Bu nedenle, bir deneme özelliğinin üzerinde yoğunlaşarak bunu bir "tehdit" olarak etiketlemenin her türlü yazılımın tüm davranış ve karakteristiklerini veren en gelişmiş araç olan Cuckoo'ya haksızlık olduğu söylenebilir.

Jamalpor vd. (2018) ile ilgili olarak, makalelerinin makine öğrenimi ile ilgisi yoktur, ancak teknikleri ve ortamları analiz etmek için açıklamalar bulunmaktadır. İlk olarak Malwr (şimdi Cuckoo olarak adlandırılan), JoeSandbox, ThreatExpert vb. gibi yaygın sanal alanlardan bahsedip, sonrasında, kernel132.dll dosyalarını kerne1.dll'ye

kopyalayan "m1.exe" adlı kötü amaçlı bir kod örneğini incelemişlerdir. Burada bu ifadenin "kernel" (k harfi – e harfi – r harfi – n harfi – e harfi – l harfi) değil, "kerne1" (k harfi – e harfi – r harfi – n harfi – e harfi – 1 sayısı) şeklinde bir aldatmaca olduğu gözden kaçırılmamalıdır. Sonuç olarak siber saldırılar her geçen gün arttığı için zararlı yazılım örneklerini analiz etmenin de daha fazla zaman aldığını düşündürmekte ve literatürde yaygın olarak yer aldığı üzere, KVM, VMWare, VirtualBox e.t.c. gibi sanal makinelerde Cuckoo gibi Sandbox'ları kullanmanın zararlı yazılım örnekleriyle başa çıkarken en verimli ve en güvenli çözüm olduğu sonucuna varılmaktadır.

Areeba Irshad vd. (2019) makalelerinde, Genetik Algoritma ile çıkarılan özellikleri (feature) kullanmıştır. Bazıları 1. API Çağrıları, 2. Kayıt anahtarları, 3. Windows Dizinleri, 4. Windows DLL dosyası, 5. Windows sisteminin EXE dosyası olan bazı olası özelliklerle başlayıp Genetik Algoritmadan istifade ederek en değerli 41 özelliği belirlemişlerdir. Çalışmadaki veri kümesi, 121'i zararlı yazılım etiketli ve 115'i zararsız yazılım etiketli olmak üzere 236 örnekten oluşmaktadır. Üç farklı sınıflandırıcı kullandıklarını belirtmektedirler. Son olarak, Destek Vektör Makinesi ile %81.3, Naive Bayes sınıflandırıcı için %64.7 ve Rassal Orman Sınıflandırıcısı için %86.8 doğruluk oranlarına sahip oldukları belirtmektedirler. İleride Bölüm 5'te de değinileceği üzere daha basit ve evrensel çözümler gerekirken, çok heterojen özellikler üzerinde, hem de genetik algoritma ile (bu durum zaman/donanım/bilgi maliyetlerini artırmaktadır) çalışarak elde ettikleri yüzdelerin düşük olduğunu itiraf etmek hiç yanlış olmayacaktır.

Literatürde yer alan yayınlar yukarıda ayrıntılarıyla beraber incelenmiştir. Bu yayınlardan birçoğu uyguladıkları yöntem ile başarı yüzdelerini vermiş, ancak birçoğu ya çok kısıtlı veri kümeleri kullanmış ya da kullandıkları veri kümelerini paylaşıma açmamışlardır. Bu tezde yürütülen çalışmayı kıyaslama ihtiyacı bulunduğunda, çalışma boyunca kullanılan ve Bölüm 5.3'te değinilen iki veri kümesi dışında bir veri kümesine rastlanmamıştır. Dolayısıyla Bölüm 6'da Çizelge 6.2'de yapılan nihai kıyaslama için yeteri kadar veri bulunamamış ve iki veri kümesinden hiçbir çalışmada başarı yüzdesi verilmeyen (Catak ve Yazi 2019) veri kümesinden istifade edilememiştir. Sadece (Ki, Kim, vd. 2015) veri kümesinin kendi hazırlayıcıları tarafından ortaya konan başarı yüzdeleri kıyaslamaya alınabilmiştir. Yalnız bu kıyaslamayı yaparken, şu hususların dikkate alınması yaptığımız çalışmanın önemini daha açıkça ortaya koyacaktır:

- Çalışma Makine Öğrenmesi veya Derin Öğrenme yöntemlerini kullanmamaktadır.
- Çalışmada ileriki bölümlerde değinileceği üzere denge problemi oluşmaması için bu tez çalışmasında gerçekleştirildiği üzere zararlı-zararsız yazılımların eşitlenmesi gibi bir yöntem izlenmemiştir.
- Çalışmanın elde ettiği doğruluk yüzdelerine kıyasla, bu tezde uygulanan ve ileride değinilecek olan 12 katlı k-Fold doğrulamasının daha geniş ve gerçekçi sonuçlar vereceği açıktır.
- Hepsinden önemlisi çalışma, hâlen zararlı yazılım tarama ve antivirüs sitelerinde "zararlı" olduğu bilinen yazılımları ilk önce bu veritabanları vasıtasıyla ayıklamıştır. Dolayısıyla bu tezde hiçbir zararlı yazılım veritabanından istifade etmeden sadece Makine Öğrenmesine dayanan bu tezdeki çalışmanın elde ettiği başarı yüzdesi takdire değerdir.

Bu tezdeki çalışmanın başarısının objektif olarak ortaya konması maksadıyla yapılan çalışmalar ilişkin dikkate değer bir başka husus ise literatür taramasında değinilen onlarca yayından API Çağrılarının kullanarak bir başarı yüzdesi elde eden ve bu başarının hangi veri kümesi ile elde edildiğine değinen bir çalışma bulunamamıştır. Örneğin;

- (Shijo ve Salim 2018) ve (Fujino, Murakami, vd. 2015) API Çağrı frekanslarını-yer alma sıklıklarını,
- (Sun, Fujino vd. 2018) benzer API Çağrıları gruplandığındaki API Çağrı frekanslarını,
- (Pirscoveanu, Hansen, vd. 2015) yazılımın ilk dizisinde yürütülen API Çağrılarını,
- (Damodaran, Anusha vd. 2017) bir yazılım yürütülürken API Çağrılarının gerçekleşme sırasını,
- (Ijaz, Durad, vd. 2017) başarısız, başarılı ve toplam API Çağrı sıklığı sayıları dikkate alarak yapılan çalışmalar olsa da ya kendi hazırladıkları veri kümelerini kullanmışlar ya da yayınlarında kullandığı veri kümelerini belirtmemişlerdir.

Dolayısıyla Bölüm 6'da ayrıntılı olarak anlatılacak olan karşılaştırmada API Çağrılarını içeren iki veri kümesinden sadece birisi kullanılabilmiştir. Kıyaslama

yapılırken de yukarıdaki hususların dikkate alınması sağlıklı bir karşılaştırma için oldukça önemlidir.

5. PRATİK ÇALIŞMA

5.1 Genel Bakış

Yukarıda Makine Öğrenmesi yöntemlerinden bahsedilirken belirtildiği gibi “öğrenme” süreci, makine öğrenmesinde çok önemli bir aşamadır. Bu yüzden araştırmamızda öncelikle makineye “girdi” olarak adlandırılan veriler verilecek, sonrasında, "makine öğrenimi yöntemi" olarak adlandırılan bir yapı oluşturulacak ve en son da öğrenebilmesi için ağı eğitilecektir ki bu da "öğrenme" olarak adlandırılır. Bu kavramlar, aşağıda uygulandığında somut hale gelecektir.

Aşağıdaki bölümlerde ayrıntılara daha derinlemesine girebilmek için, önce veri kümesi seçimi ve özellik seçimi için temel anlayışı açıklamanın uygun olacağı değerlendirilmektedir.

5.2 Araştırmanın Arkasında Yer Alan Ana Fikir

Bölüm 1’de yüzeysel olarak bahsedildiği gibi, zararlı yazılımların tespiti veya sınıflandırılması konusunda çok sayıda araştırma vardır. Ancak çoğu, araştırmaları ve bunların uygulamalarını daha karmaşık hale getiren geniş bir özellik (feature) yelpazesinden faydalanmaktadır. Bizim görüşümüze göre ise temel bir yapıya ve daha hızlı gerçekleştirmeye ihtiyaç bulunmaktadır. Çoğu makale, bu tezin 2.3.1 ve 2.3.2 bölümlerinde ele alınan birçok farklı ve karmaşık özelliğe (feature) odaklanmaktadır. Ancak biz, "API Çağrılarının" bir yazılımın tam olarak ne yürüttüğü ve nasıl davrandığını en net olarak ortaya koyan özellik olacağı değerlendirilmektedir. Bu bakış açısından, dinamik zararlı yazılım analizinden çıkarılan API Çağrılarına bağlı olarak zararlı yazılımların tespiti konusunda bir temele sahip olunacaktır. Doğruluk oranını olabildiğince yüksek tutarken sadelik ve hız en üstte tutacak olmamız çalışmamızın en büyük ve göze çarpan özelliği olacaktır.

Bunu başarmak için bizce ilk olarak "API Çağrısı özellikleri" yaklaşımı benimsenmiştir. Bu yaklaşımda, olası ve yaygın zararlı yazılım davranışlarının az ya da çok ortak API Çağrıları kullandığı gerçeğinden yararlanılmıştır. Bu davranışlar için yaygın zararlı yazılım davranışları ve olası API Çağrıları gösterildiği gibidir:

- Tuş vuruşu kaydı için: FindWindowsA, ShowWindow, GetAsyncKeyState, SetWindowsHookEx, RegisterHotKey, GetMessage, UnhookWindowsHookEx vb.
- Ekran yakalama için: GetDC, GetWindowDC, CreateCompatibleDC, CreateCompatibleBitmap, SelectObject, BitBlt, WriteFile vb.
- Hata ayıklama (Anti-Debugging)'dan kaçınmak için: IsDebuggerPresent, CheckRemoteDebuggerPresent, OutputDebugStringA, OutputDebugStringW vb.
- İndiriciler için: URLDownloadToFile, WinExec, ShellExecute vb.
- DLL Enjeksiyonu için: OpenProcess, VirtualAllocEx, WriteProcessMemory, CreateRemoteThread vb.
- Damlalıklar için: FindResource, LoadResource, SizeOfResource, LockResource vb.
- Kayıt Anahtarı Defterini (Registry) değiştirmek için: RegCloseKey, RegOpenKeyExA, RegDeleteValueA vb.

"API Çağrısı Özellikleri" yaklaşımıyla beklenenden düşük oranlara ulaşılması üzerine "En Yaygın API Çağrıları" yaklaşımı benimsendi. Bu yaklaşımla, en sık kullanılan API Çağrılarının, herhangi bir zararlı yazılım olasılığı hakkında önemli ipuçları verebileceği kabul edilmiştir. Bu iki yaklaşım, aşağıdaki bölümlerde daha ayrıntılı olarak açıklanacaktır.

5.3 Veri Kümesi

Bu tez boyunca, Bölüm 5.2'de açıklanan nedenlerden dolayı, API Çağrılarını iki veri kümesinin bize izin verdiği potansiyel özellikler (feature) olarak tercih edilecektir. İlk olarak, "API Çağrı Sırası Analizine Dayalı Kötü Amaçlı Yazılımları Algılamak için Yeni Bir Yaklaşım" (Ki, Kim, vd. 2015) ile birlikte yayınlanan ve tamamen API Çağrılarına dayanan APIMDS (API Tabanlı Zararlı Yazılım Algılama Sistemi) Veri Kümesi kullanılacaktır. Veri kümesinin boyutu 112.7 MB'dir ve 23146 yazılım örneğinden oluşan bir ".csv" dosyasıdır. Örneklerin 14131'i "zararlı", 3137'si "zararsız" olarak, 5878'i ise etiketsiz (yani zararlı mı zararsız mı olduğu bilinmeyen) durumdadır. Farklı yazılımlar farklı sayıda API Çağrısı kullandığından, örneklerin sütun sayısı belirli değildir.

Veri kümesinin yapısı incelenecek olursa; ilk sütun yazılım türü ve bazı bilgiler hakkında fikir veren bir dizedir. İlk sütundaki dizeyle ilgili üç olasılık vardır:

- Dize boşsa, yazılım "etiketsizdir", yani yazılımın zararlı mı yoksa zararsız mı olduğu bilinmemektedir.
- Dizide "virüs değil" (“not-a-virus”) ifadesi geçiyorsa, yazılım "zararsız” olarak etiketlenir.
- Dize boş değilse ve "virüs değil" (“not-a-virus”) ifadesi içermiyorsa, yazılım "zararlı” yazılım olarak etiketlenir.

İkinci olarak, başka bir veri kümesinde ilk veri kümesinden elde ettiğimiz iyi başarıyı karşılaştırmak için, "Windows PE zararlı yazılım sınıflandırması için bir karşılaştırma API Çağrısı veri kümesi" veri kümesi (Catak ve Yazi 2019) kullanılmıştır. Kaggle'da da yer alan veri kümesi toplamda yaklaşık 2.2 GB'lık iki farklı dosyadan oluşmaktadır ve her ikisi de “.txt” türünde dosyalardır. İlk dosya API Çağrılarını, ikinci dosya ise zararlı yazılım türünü içerir. Yani ilk dosyadaki zararlı yazılımın türü ikinci dosyada karşılık gelen satır numarasına yazılı bulunmaktadır.

Bu ikinci veri kümesi, 7107 zararlı yazılım örneğinden oluşur. Tamamı zararlı yazılımlar olan bu yazılımlardan 832'si "Casus Yazılım", 379'u "Reklam Yazılımı", 891'i "Damlalık" tır. "İndirici", "Truva Atı", "Solucan", "Virüs", "Arka Kapı" türlerinin her biri 1001’er örneğe sahiptir. Farklı yazılımlar, ilk veri kümesinde olduğu gibi farklı sayıda API Çağrısı kullandığından, örneklerin sütun sayısı belirli değildir. Tüm örnekler zararlı yazılım olduğundan, sorunumuz ilk veri kümesinde olduğu gibi tespit (detection) değil, bir sınıflandırma problemi olacaktır.

Çalışma boyunca, kolaylık sağlamak için, (Ki, Kim, vd. 2015) tarafından yayınlanan veri kümesine atıfta bulunan "birinci veri kümesi" ve (Catak ve Yazi, 2019) tarafından yayınlanan veri kümesine atıfta bulunan "ikinci veri kümesi" terimleri kullanılacaktır.

5.4 Veri Kümesi Hazırlama ve Özellik Çıkarma

Amaçlarımız doğrultusunda, ilk aşamada ilk veri kümesini uygulamaya hazır hale getirmek için yapılması gereken bazı çalışmalar gereklidir. Verileri işlemek için Linux

Bash betikleri kullanılarak Ubuntu 18.04 LTS ve 12 GB RAM özellikli makinede yapılan hazırlık ve işlemler aşağıdaki gibidir:

- Veriler çift tırnaklı dizeler içinde sunulduğundan, verilerin üzerinde çalışacak temiz dizeler olması için tüm başlangıç ve bitiş çift tırnak işaretleri kaldırılmıştır.
- Öğrenme sürecine olan ilgisizlikleri nedeniyle, 14131 adet kötü niyetli ve 3137 adet zararsız olmak üzere toplam 17268 yazılım elde tutularak, 5878 adet etiketsiz yazılım atılmıştır.
- Özellik (feature) çıkarma aşaması için iki grup API Çağrısı belirlenmiştir. Birinci grup "en az bir zararlı yazılımda gerçekleşen ancak herhangi bir zararsız yazılımda yer almayan (makalenin geri kalanında Sözde-Zararlı API Çağrıları olarak adlandırılan) API Çağrıları"dır. İkinci grup, "zararsız yazılımlardan en az birinde gerçekleşen ancak herhangi bir zararlı yazılımda yer almayan (makalenin geri kalanında Sözde-Zararsız API Çağrıları olarak adlandırılan) API Çağrıları" olmuştur. Ancak şaşırtıcı bir şekilde, kötü niyetli grup olarak adlandırılan ilk grup 599 API Çağrısına sahipken, iyi huylu olarak adlandırılan ikinci grupta yalnızca 5 API Çağrısı olduğu görülmüştür. (Tüm veri kümesinde bulunan farklı API Çağrısı sayısı 1165'tir.) Özellik seçim aşamasındaki detaylı çalışma ve araştırmada esas alınan basitlik anlayışı göz önüne alındığında, 604 özelliğin (604 = 599 adet zararlı yazılımlardan + 4 adet zararsız yazılımlardan çıkarılan) çok fazla olduğu değerlendirilmiştir. Ayrıca özelliklerin sadece zararlı veya sadece zararsız yazılımlardan seçilmesinin, makine öğrenmesine dayanan çalışmamızı yalnızca doğrusal çalışan bir regresyon haline getireceği ve bunun da bir derin öğrenmenin değil bir nevi sabit bir algoritmanın sonucu olacağı değerlendirilerek bu gerçekler neticesinde başka çözümler aranmıştır.
- Daha uygun bir çözüm arayışımızda daha basit bir yaklaşım ele alınmıştır. Bunun için de diğer API Çağrılarına bağlı API Çağrılarının tespit edilmesinin yerinde olacağı düşünülmüştür. Burada "bağlı" teriminden kastedilen, bir API Çağrısının veri kümesinde sadece diğer API Çağrısı ile birlikte yer alması, diğer API Çağrısının olmadığı örneklemlerde ise yer almamasıdır. Bu şekilde çıkaracağımız özellikleri (feature) sadeleştirebileceğimizi düşündük. Yaklaşık 40 saatlik bir betik çalıştırdıktan sonra, başka bir API Çağrısına bağımlı olan tüm API Çağrıları tespit edilmiştir. Bu yaklaşımla "Sözde Zararlı API Çağrısı" sayısı 599'dan 387'ye ve "Sözde Zararsız API

Çağrısı" sayısı 5'ten 4'e düşürüldü (böylece toplam bağımsız API Çağrısı sayısı 391 oldu). Yalnız 391 sayısının aradığımız basit ve etkin bir çözüm için hala fazla olduğu değerlendirilmiştir.

- Bazı ilerlemelerin kaydedildiği, ancak iyileştirme için hala yer olduğu düşünülmüştür. "Sözde Zararlı API Çağrıları" ve "Sözde Belirsiz API Çağrıları" için (daha önce bahsedilen iki grup halinde gruplandırılamayan- yani hem en az bir zararlı yazılımda hem de en az bir zararsız yazılımda yer alan API Çağrıları), en fazla sıklıkla yer alma sayısı dikkate alınacaktır. "Sözde Zararsız API Çağrıları" için, sayı 4 olduğundan ve zaten yeterince düşük olduğundan hiçbir şey yapılmayacaktır.
- "Sözde Zararlı API Çağrıları", yazılımlarda geçme sayılarına göre azalan sırada listelenmiştir. Tüm veri kümesinde 85'ten (ki bu sonuçta çıkarmak istediğimiz özellik sayısına göre belirlediğimiz sezgisel bir kriterdir) daha az sayıda gerçekleşen API Çağrıları, nihayet "Sözde Zararlı API Çağrısı" 44 adet olmak üzere belirlenmiştir.
- "Sözde Belirsiz API Çağrıları", yazılım örneklerinde geçme sayılarına göre azalan sırada listelenmiştir. 79000'den (ki bu da sonuçta çıkarmak istediğimiz özellik sayısına göre belirlediğimiz sezgisel bir kriterdir) daha az sayıda gerçekleşen API Çağrıları, toplam "sözde belirsiz API Çağrısı" 16 adet olmak üzere belirlenmiştir.
- Son olarak özellik seçimi, toplamda 64 özellik olmak üzere 44'ü Sözde Zararlı, 4'ü Sözde Zararsız ve 16'sı Sözde Belirsiz API Çağrısı seçilmiştir. Bu şekilde daha önceki maddelerde belirtilen doğrusallık gibi bir sorunun da tamamen üstesinden gelinmiştir. Bu sayede API Çağrıları (tarafımızca tasarlanan) karakteristik özelliklerine göre ayrılarak en sık kullanılanları seçilerek en başta hedeflediğimiz etkin ve basit bir uygulamaya yönelik adımlar atılmıştır.
- Veri kümesinde zararlı ve zararsız örneklerin eşitsizliğini dikkate alarak, bir dengesizlik-yanlılık sorunu yaşamamak için zararlı yazılımların sayısının (10994), zararsız yazılımların sayısına (3137) eşitlenmesi gerektiği değerlendirilmiştir. Bu dengesizlik-yanlılık sorununa çare olacak olsa da sayılardan da anlaşılacağı üzere veri kümesinde verilerin yarısından fazlasının kaybına sebep olmuştur.

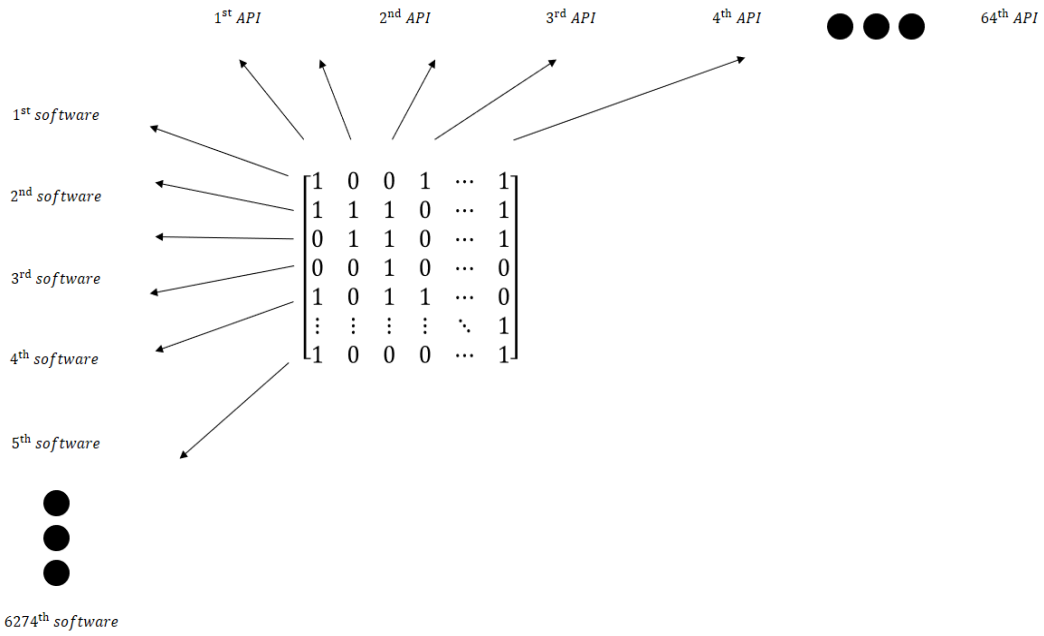
Yukarıda anlatıldığı şekilde seçilen 64 özellik ile elde edilen doğruluk oranı %81.06 olmuştur. Bu başarı oranı ayrıntılı olarak incelendiğinde, "Zararlı/Zararsız Karakterli API Çağrıları Yaklaşımı" olarak adlandırabileceğimiz bu detaylı tasarlanmış özellik hazırlama sürecine kıyasla oldukça düşük görünmektedir. Dolayısıyla API Çağrılarının

veri kümesindeki yer alma sayılarıyla ilgilenen "En Çok Kullanılan API Çağrıları Yaklaşımı" olarak adlandırabileceğimiz bir başka özellik seçme yaklaşımının daha faydalı olabileceği değerlendirilmiştir. Buna göre veri kümesindeki tüm API Çağrılarının veri kümesinde yer alma sayıları hesaplanmış ve tüm API Çağrıları, öğrenme sürecine olası katkılarını temsil ettiğini değerlendirdiğimiz toplam yer alma sayılarının azalan sırasına göre sıralanmıştır. Ve "çoğunlukla meydana gelen" API Çağrılarının ilk kaç tanesinin doğruluk oranına ne kadar katkıda bulunduğunu görmek için (sezgisel olarak) 10, 20, 40, 60, 80, 100, 200, 300 ve 400 sayıları seçilmiştir. "Zararlı/Zararsız Karakterli API Çağrıları Yaklaşımı"nda yetersiz bulduğumuz doğruluk oranları "En Çok Kullanılan API Çağrıları Yaklaşımı" ile birlikte önemli ölçüde artmış ve basit özellik çıkarma süreci değerlendirildiğinde fazlasıyla tatminkâr seviyeye gelmiştir. "En Çok Kullanılan API Çağrıları Yaklaşımı"na yönelik daha fazla değerlendirme, karşılaştırma ve bilgi Bölüm 6'da ayrıntılı olarak verilecektir.

5.5 Özelliklerin (Feature) Temsil Edilmesi

Literatürde, API Çağrılarını çeşitli şekillerde kullanan birçok araştırma vardır. API Çağrı frekanslarını-yer alma sıklıklarını kullanan (Shijo ve Salim 2018) ve (Fujino, Murakami, vd. 2015), benzer API Çağrıları gruplandığındaki API Çağrı frekanslarını kullanan (Sun, Fujino vd. 2018), yazılımın ilk dizisinde yürütülen API Çağrılarını kullanan (Pirscoveanu, Hansen, vd. 2015), bir yazılım yürütülürken API Çağrılarının gerçekleşme sırası (Damodaran, Anusha vd. 2017) ve hatta başarısız, başarı ve toplam API Çağrı sıklığı sayısı (Ijaz, Durad, vd. 2017)'nı dikkate alarak özellik çıkaran çalışmalara rastlanmıştır. Ancak bu tez çalışmasında basitlik önde tutularak sadece bir API Çağrısının çağrılıp çağrılmadığıyla (gerçekleştirilip gerçekleştirilmediğiyle) ilgilenilecektir.

Seçilen özelliklerin temsili için duruma en uygun görünen "Kelime Torbası" (Bag of Words) adlı bir notasyon kullanılacaktır. Bu gösterim için Şekil 5.1'deki yapıyı oluşturmak için aşağıdaki hazırlıkların yapılması gerekecektir.



Şekil 5.1 Araştırmamızda Kullanacağımız "Kelime Torbası" (Bag of Words) Gerçekleştirmesi

Girdi matrisine "A" diyelim. Eğer $x^{inci} yazılım$, y^{inci} özellik (API)'yi içeriyorsa, o halde $A[x][y] = 1$ olacaktır. Eğer $x^{inci} yazılım$, y^{inci} özellik (API)'yi içermiyorsa, o halde $A[x][y] = 0$ olacaktır. Şekil 5.1'de verilen tüm 0 ve 1 değerleri bu anlatıma göre rastgele verilmiş örneklerdir. Burada anlattıklarımıza göre şekli yorumlarsak, $1^{inci} yazılım$, $1^{inci} API$'yi içermekte (çünkü $A[0][0] = 1$), ancak $2^{nci} API$'yi içermemektedir (çünkü $A[0][1] = 0$).

$A[6274][64]$, 6274 satıra (3137 zararlı ve 3137 zararsız örnekten oluşan toplam yazılım sayısından türetilmiştir) ve 64 sütuna (44 Sözde Zararlı, 4 Sözde Zararsız ve 16 Sözde Belirsiz API Çağrısı olarak adlandırılan API Çağrısından çıkarılmıştır) sahip bir matristir.

Çıktı için başka bir matrisimiz olacak ve buna "B" diyelim. Eğer $x^{inci} yazılım$ zararlıysa $B[x] = 1$ olmasını, eğer $x^{inci} yazılım$ zararsızsa $B[x] = 0$ olması beklenecektir. $B[6274][1]$ matrisi, 6274 satır (3137 zararlı ve 3137 zararsız örnekten oluşan toplam yazılım sayısından türetilmiştir) ve sadece 1 sütundan (yazılımın zararlı

olup olmaması üzerinden elde edilecek olan ikili-binary durumdaki tek bir sonuçtan türetilmiştir) oluşmaktadır.

5.6 Gerçekleştirme

5.6.1 Derin öğrenme

Derin Öğrenme yöntemi Python3 programlama dili ve Keras kütüphanesi kullanılarak gerçekleştirilmiştir. Özellik boyutundan türetilen ağın giriş boyutu önce 64 ve daha sonra 10, 20, 40, 60, 80, 100, 200, 300 ve 400 olarak seçilmiştir. Sebepleri önceki bölümlerde temel olarak açıklandığı gibi Bölüm 6'da yine daha detaylı olarak açıklanacaktır. İkili-binary sonuçtan türetilerek (0'a yakınsadıkça zararsız ve 1'e yakınsadıkça zararlı olma olasılığı artacaktır), ağın çıktı boyutu yalnızca 1'dir. Sezgisel olarak, özellik sayılarına göre katman sayıları Çizelge 5.1'de verildiği şekilde belirlenmiş ve aradaki gizli katman büyüklük ve nöron sayıları "üçte iki çoğunluk" yazısız kuralı dikkate alınarak sezgisel olarak belirlenmiştir. Eğitimde Python3 ortamı ve kullanılan Keras kütüphanesinden istifade edilmiştir.

Değişen girdi, çıktı, gizli katman büyüklük ve sayısı parametrelerine ilaveten, belirtilen tüm mimarilerde optimal sonucu verdiğini değerlendirdiğimiz bazı parametreler de bulunmaktadır. Bunların detayına girildiğinde, $Learningrate$ (Öğrenme Hızı) 0,001'dir. Doygunluk (Saturation) probleminin üstesinden gelmek ve çıktı değerinin 0 ile 1 arasında kararlı ve mantıklı bir şekilde gidip gelebilmesi için, son (çıktı) katmanında kullanılan son aktivasyon fonksiyonu olarak Sigmoid seçilmiş, son aktivasyon dışında tüm katmanların tüm aktivasyon fonksiyonları ReLU (Rectified Lineer Unit) olarak belirlenmiştir. Bu iki fonksiyonun formülleri Eşitlik 5.1 ve 5.2'de verilmiştir.

$$Sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{5.1}$$

$$ReLU(x) = max(0, x) \tag{5.2}$$

Sonrasında, ağın öğrenmesi için 12, 20 ve 50 epoğa ($epochs$) ayarlanmış sezgisel bir toplu iş boyutu ($batchsize$) yeterli olmakla beraber çalışmalarda en optimal değeri veren sayı 12 olmuştur. Değerler, "İkili Çapraz Entropi" (Binary Cross Entropy) kullanılarak optimize edilmiş ve son olarak sonuçlar, 12 Katlı (12-Fold Cross) kullanılarak gerçekçi sonuçlar elde edebilmek için karıştırma (shuffle) yapılarak doğrulanmıştır.

Çizelge 5.1 Özellik Sayılarına Göre Gizli Katman Sayısı ve Bu Katmanlarda Yer alan Gizli Nöron Sayıları

ÖZELLİK SAYISI	TESPİT İÇİN MİMARİ	SINIFLANDIRMA İÇİN MİMARİ
10	10-8-1	10-8
20	20-16-12-8-1	20-16-12-8
40	40-25-18-12-8-1	40-25-18-12-8
60	60-40-25-18-12-8-1	60-40-25-18-12-8
80	80-50-32-20-12-8-1	80-50-32-20-12-8
100	100-75-50-30-20-12-8-1	100-75-50-30-20-12-8
200	200-130-80-50-30-20-12-8-1	200-130-80-50-30-20-12-8
300	300-200-120-80-50-30-20-12-8-1	300-200-120-80-50-30-20-12-8
400	400-250-160-120-80-50-30-20-12-8-1	400-250-160-120-80-50-30-20-12-8

Örneğin, 100 özelliğe sahip algılama sinir ağı mimarimiz (Şekil 3.1'e bakınız), girdi katmanımızın 100 nöronu vardır ve 1 boyutlu bir çıktı katmanı vardır. Bu, ağın 100 nörondan 0-1 değerleri arasında bilgi alacağı ve sorunun çözümüne yönelik 1 çıktı nöronundan 0-1 değerleri arasında 1 adet değer vereceği anlamına gelir (Girdilerde ve çıktılar “1” değerinin “Zararlı”, “0” değerinin “Zararsız” anlamına geleceğini Bölüm 5.5’te belirtilmiştir). Giriş ve çıkış arasındaki katmanlara “Gizli Katmanlar” olarak adlandırılır. Çizelgede verilen “100-75-50-30-20-12-8-1” mimarisi değerlendirildiğinde 6 gizli katmanımız olduğu ve bu katmanlarda sırasıyla o kadar nöron olduğunu görülecektir. Ayrıca yürütülen çalışmalarda özellik (feature) sayısı gibi, katman

sayısının da belirli bir dereceye kadar yüzdeyi artırdığı ancak bir dereceden sonra yüzdeye bulunduğu katkıya kıyasla fazla donanım/zaman maliyeti ortaya çıkardığı görülerek yukarıda zaman-fayda ikileminde optimal sonuçlar alınan değerlerin sunulduğu unutulmamalıdır. Elde edilen en iyi başarı (accuracy) değeri %90.34 olmuştur.

5.6.2 J48 karar ağacı

Python3 ortamı ve Keras kütüphanesi kullanıldığında, veri kümesinin %80'i eğitim için, kalan %20'si test için kullanılmıştır. Gini kriteri $maximumDepth = 3$, $minimumNumberOfInstancesPerLeaf = 5$ve diğer parametreler varsayılan olarak seçilmiştir. Elde edilen başarı (accuracy) yüzdesi %89.29 olmuştur.

5.6.3 K-En yakın komşu

Python3 ortamı ve Keras kütüphanesi kullanıldığında, veri kümesinin %80'i eğitim için, kalan %20'si test için kullanılmıştır. Öklid mesafesi $K = 5$ olarak seçilmiş ve diğer parametreler varsayılan olarak bırakılmıştır. Elde edilen başarı (accuracy) yüzdesi %90.01 olmuştur.

5.6.4 Naive bayes sınıflandırıcı

Python3 ortamı ve Keras kütüphanesi kullanıldığında, veri kümesinin %80'i eğitim için kullanıldı ve kalan %20'si varsayılan hiperparametreler ile test etmek için kullanılmıştır. Elde edilen başarı (accuracy) yüzdesi %59.00 olmuştur. Burada başarı yüzdesinin düşüklüğünün Naive Bayes Sınıflandırıcısının olasılıksal bir algoritma olmasından kaynaklandığının altını çizmek gerekir.

5.6.5 Rassal orman

Python3 ortamı ve Keras kütüphanesi kullanıldığında, veri kümesinin %80'i eğitim için kullanıldı ve kalan %20'si $bagSize = 100$, $iterations = 100$, $inimumNumberOfInstancesPerLeaf = 5$, $batchSize = 12$ parametreleri ve varsayılan olarak bırakılan diğer parametrelerle test etmek için kullanılmıştır. Elde edilen başarı (accuracy) yüzdesi %89.77 olmuştur.

5.6.6 Destek vektör makinesi

Python3 ortamı ve Keras kütüphanesi kullanılarak çalışma gerçekleştirilmiştir. Veri kümesinin %80'i eğitim için kullanılmış ve kalan %20'si doğrusal çekirdek ile test etmek için kullanılmıştır. Diğer parametreler varsayılan olarak bırakılmıştır. Elde edilen başarı (accuracy) yüzdesi %90.09 olmuştur.

6. SONUÇLAR, KARŞILAŞTIRMA VE TARTIŞMA

6.1 Sonuçlar ve Karşılaştırma

Doğruluk oranı, Karışıklık Matrisi ROC (Alıcı İşlem Karakteristiği) Eğrisine göre Çizelge 6.1'de gösterildiği şekilde hesaplanacaktır.

Çizelge 6.1 Doğruluk Hesaplaması için Karışıklık Matrisi

		Tahmin Edilen Sınıf	
		Hayır	Evet
Gerçekteki Sınıf	Hayır	Gerçek Negatif (True Negative-TN)	Yalancı Pozitif (False Positive-FP)
	Evet	Yalancı Negatif (False Negative-FN)	Gerçek Pozitif (True Positive-TP)

Çizelge 6.1'e göre;

$$(Accuracy)\ Doğruluk = \frac{TN + TP}{TN + FP + FN + TP} \tag{6.1}$$

$$(Precision)\ Kesinlik = \frac{TP}{FP + TP} \tag{6.2}$$

$$(Sensitivity)\ Hassasiyet = \frac{TP}{FN + TP} \tag{6.3}$$

$$(Specificity)\ Özgünlük = \frac{TN}{TN + FP} \tag{6.4}$$

$$F1\ Puanı = 2 * \frac{Kesinlik\ * Hassasiyet}{Kesinlik + Hassasiyet} \tag{6.5}$$

Daha ileri bir analiz için aşağıdaki bilgiler not edilmelidir. Bölüm 5.5'te bahsedildiği gibi, ilk olarak "Sözde Zararlı" ve "Sözde Zararsız" API Çağrılarının bulunduğu "karakterizasyon" işlemi yürütülmüştür. Bu yöntemin sakıncalarının farkına varıldıktan sonra, özelliklerin tüm veri kümesindeki oluşumlarına göre seçilmesi için başka bir yaklaşım benimsenmiştir.

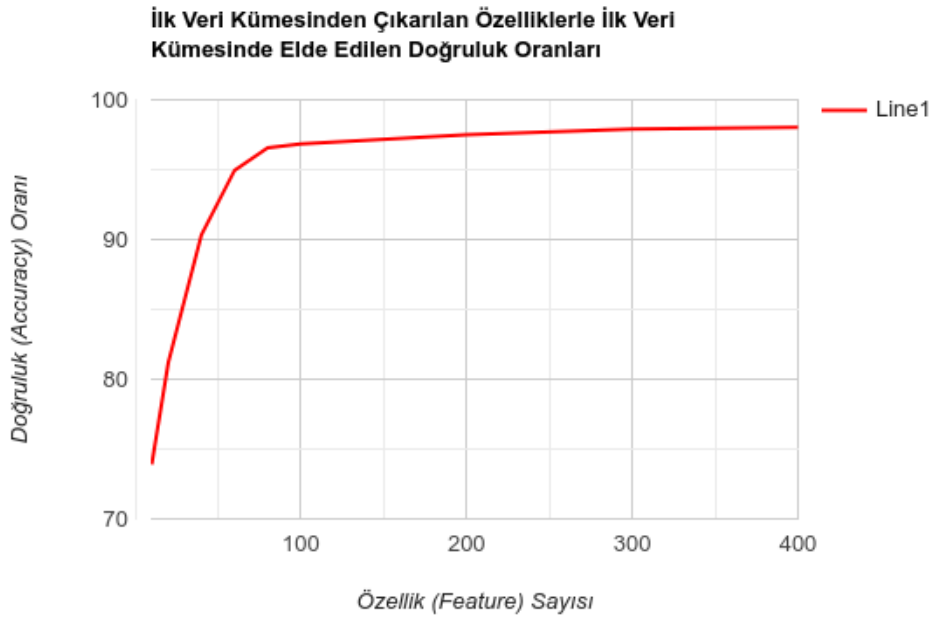
Daha ayrıntılı değinmek gerekirse, bu süreçler şu şekilde özetlenebilir:

- **İlk Veri Kümesinde "Zararlı/Zararsız Karakteristikli API Çağrıları Yaklaşımı":** 64 sözde karakteristik API Çağrısı bulunmuş ve özellik olarak kullanılmıştır. Elde edilen sonuçlar özelliklerin belirlenmesi aşamasında yapılan titiz ve detaylı çalışma göz önüne alındığında tatmin edici görülmemiştir.
- **İlk Veri Kümesinde "En Çok Kullanılan API Çağrıları Yaklaşımı":** "Zararlı/Zararsız Karakteristikli API Çağrıları Yaklaşımı" ile beklenenden düşük sonuçlar alınması üzerine en çok kullanılan 10, 20, 40, 60, 80, 100, 200 ve 400 API Çağrıları özellik olarak seçilmiştir. Elde edilen sonuçlarda doğruluk seviyesi oldukça yüksektir.
- **Özellikleri Başka Bir Veri Kümesiyle Test Etme:** "En Çok Kullanılan API Çağrıları Yaklaşımı" ile beklenen sonuçları gördükten sonra, aynı özellik kümeleri ikinci veri kümesinin özellikleri olarak test edilmiştir. İlk veri kümesinden çıkarılan özellikler başka bir (İkinci) veri kümesindeki modelleri tanımlamada başarılı olduğu için sonuçlar fazlasıyla tatmin edici bulunmuştur. Ayrıca bu test için bir başka zorluk da, ikinci veri kümesinin yalnızca sınıflandırmaya izin veren yapısı (veri kümesinin tamamının zararlı yazılımlardan oluşması, hiç zararsız yazılım içermemesi ve etiketlerinde zararlı yazılım sınıflarını barındırması), birinci veri kümesinin ise sınıflandırma değil tespit gerektiren hem zararlı hem de zararsız yazılımlardan oluşmasıdır. Bu bize, farklı görevler ve farklı veri kümeleri için bu yaklaşımın çok uygulanabilir ve evrensel bir çözüm olduğunu göstermiştir.
- **Yaklaşımları Başka Bir Veri Kümesi ile Test Etme:** Yukarıdaki aşamada tatmin edici sonuçlarla karşılaşılınca, başka bir (ikinci) veri kümesinde özelliklerin test edilmesi yerine yaklaşımın test edilmesi fikri hâsıl olmuştur. Bu kapsamda "En Çok

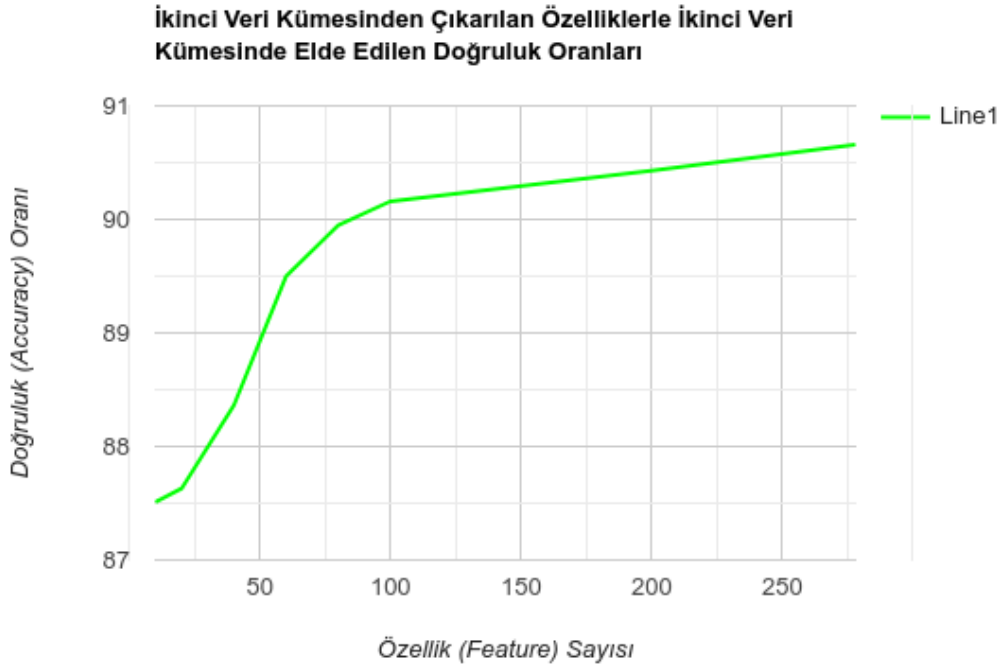
Kullanılan API Çağrıları Yaklaşımı" başka bir (ikinci) veri kümesinde test edilmiştir. Bu kapsamda, ikinci veri kümesinde en çok kullanılan API Çağrıları çıkarılmış ve 10, 20, 40, 60, 80, 100, 200 ve 278 adetten oluşan en çok kullanılan API Çağrıları özellik (feature) olarak belirlenmiştir. Burada ilk veri kümesinde 300 ve 400 özellikler test edilmişken, ikinci veri kümesinde yer alan benzersiz API Çağrı sayısı 278 olduğundan, 300 ve 400 özellik sayılarının yerine mecburiyetten 278 özelliğin tercih edilebildiğine dikkat edilmelidir. İlk veri kümesinden daha düşük olsa da sonuçlar yine tatmin edicidir. Bunun ikinci veri kümesindeki benzersiz API Çağrısı sayısının (278) çok az olmasından kaynaklandığı ve yeteri kadar benzersiz API Çağrısı bulunan herhangi bir veri kümesinde sonuçların ilk veri kümesinde olduğu kadar tatmin edici olacağı aşikârdır.

Şekil 6.1, 6.2 ve 6.3'te çeşitli çalışmalara göre özellik (feature) sayılarının doğruluk (accuracy) oranlarına nasıl katkıda bulunduğunu gösterilmektedir. Burada özellik sayıları arttıkça başarı oranlarının da (gittikçe azalan bir ivmeyle) arttığı görülmektedir. Dolayısıyla, ihtiyaç varsa, elde mevcut zaman donanım gibi maliyet faktörlerine göre özellik sayılarının belirlenerek doğruluk oranlarının istenilen derecede göz ardı edilebilir olması da sistemin esnekliğini göstermektedir.

Ayrıca bu şekillerde verilen doğruluk oranlarından hareketle, çalışmanın sadece çalışılan veri kümesiyle değil diğer veri kümeleriyle de geçerlenebilir olması çalışmanın ve aynı zamanda tespit veya sınıflandırma problemlerinin her ikisinde de başarılı olması, üzerinde çalışılan soruna evrensel bir çözüm getirebileceğine dair bir gösterge olarak kabul edilmelidir. Bunun yanı sıra, veri kümesinin zararlı/zararsız ve zararlı yazılım türleri olmak üzere çeşit ve sayısının artmasıyla elde edilen doğruluk oranlarının da üzerinde oldukça stabil bir çözüm sağlanabileceği de açıktır.

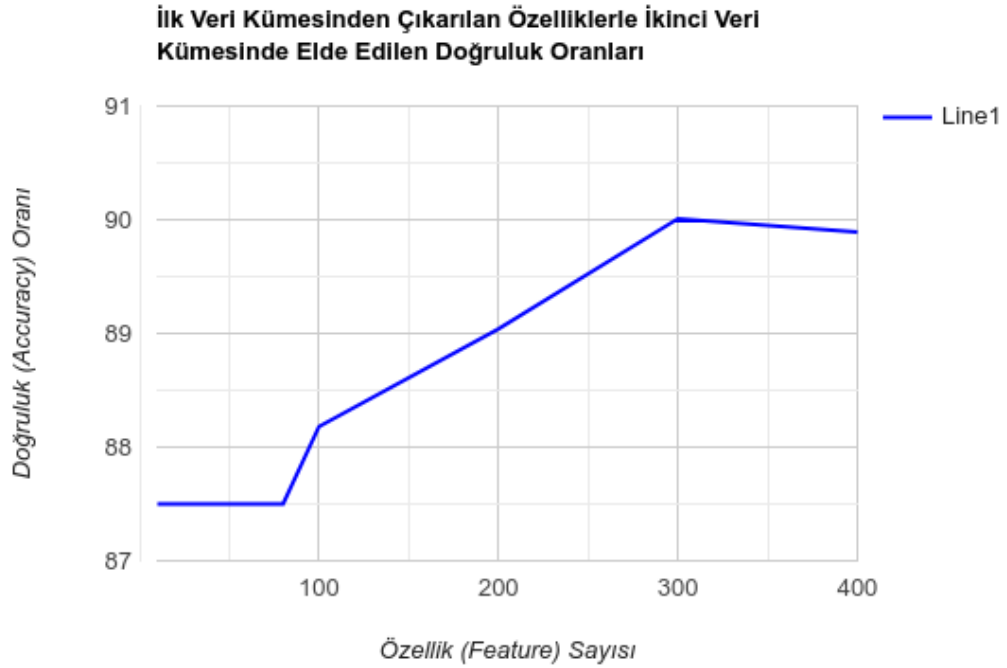


Şekil 6.1 İlk Veri Kümesinden Çıkarılan Özelliklerle İlk Veri Kümesinde Elde Edilen Doğruluk Oranları



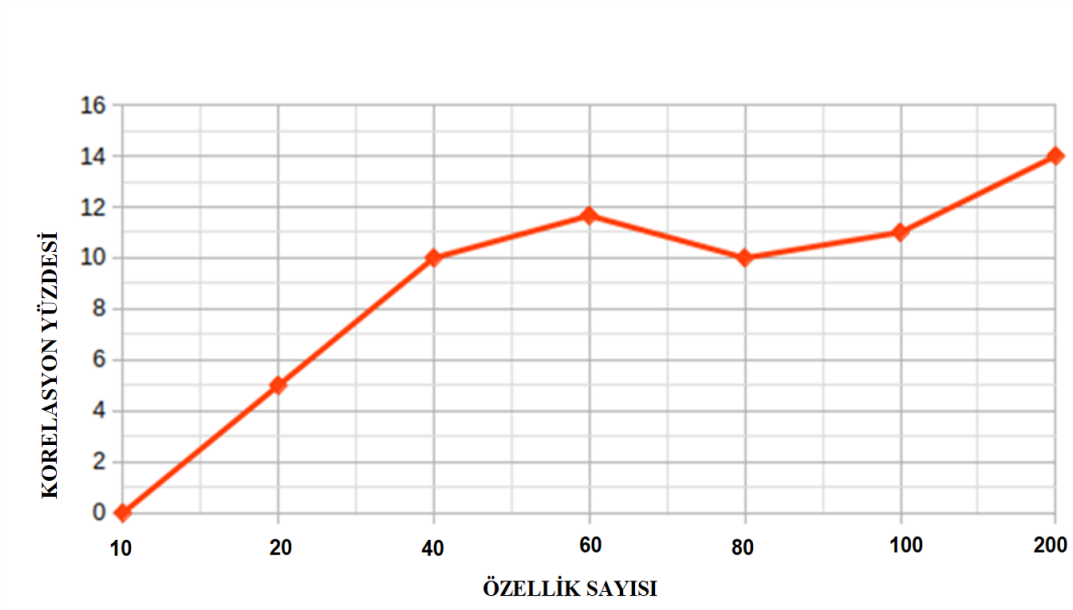
Şekil 6.2 İkinci Veri Kümesinden Çıkarılan Özelliklerle İkinci Veri Kümesinde Elde Edilen Doğruluk Oranları

Şekil 6.3 İlk Veri Kümesinden Çıkarılan Özelliklerle İkinci Veri Kümesinde Elde Edilen Doğruluk Oranları



Bir başka karşılaştırma da, iki veri kümesinin "En Çok Kullanılan API Çağrıları", yani özellikleri arasındaki muhtemel bir korelasyon kurulabilip kurulamayacağı konusunda gerçekleştirilmiştir. Yapılan analizde veri kümelerinin en sık kullanılan API Çağrıları arasında anlamlı bir korelasyon bulunamamıştır. Bunun da ilk veri kümesinin tespit (detection) çözümleri için uygun olup hem zararlı hem de zararsız örneklerden oluşurken, ikinci veri kümesinin yalnızca zararlı örneklerden oluşan ve sınıflandırma sorununa yönelik bir veri kümesi olmasından ve her iki veri kümesinin belirli tipteki yazılımlardan oluşarak farklı karakteristik göstermelerinden kaynaklandığı değerlendirilmektedir. Korelasyon grafiği Şekil 6.4'te gösterilmektedir.

Şekil 6.4 Birinci ve İkinci Veri Kümelerinden Çıkarılan Özelliklerin Arasındaki Muhtemel Korelasyonun Yakalanmasına Dair Çalışma



Makine öğrenmesi metotlarıyla yukarıdaki esaslarla elde edilen başarı yüzdeleri çeşitli

Çizelge 6.2 Çeşitli Makine Öğrenmesi Metotlarıyla Elde Edilen Çeşitli Metrikler Ölçüsünde Başarı Yüzdeleri

metriklere göre Çizelge 6.2'de verilmiştir.

METRİK	J48 KARAR AĞACI	K-EN YAKIN KOMŞU	NAIVE BAYES	RASSAL ORMAN	DESTEK VEKTÖR MAKİNESİ	DERİN ÖĞRENME
Doğruluk	**% 89.29**	**% 90.01**	**% 59.00**	**% 89.77**	**% 90.09**	**% 98.04**
Kesinlik	**% 52.51**	**% 73.67**	**% 57.09**	**% 90.01**	**% 83.38**	**% 96.19**
Hassasiyet	**% 89.09**	**% 65.07**	**% 87.30**	**% 97.51**	**% 55.83**	**% 98.74**
Özgünlük	**% 90.38**	**% 94.79**	**% 27.93**	**% 52.81**	**% 97.57**	**% 96.87**
F1	**% 66.08**	**% 69.10**	**% 69.05**	**% 94.03**	**% 66.98**	**% 97.45**

Literatür taraması ve kıyaslamada yaşanan sorunlar Bölüm 4'te ayrıntılı olarak belirtilmiştir. Ayrıca Bölüm 4'te değinilen hususlar dikkate alındığında Çizelge 6.2 VE 6.3'te görüldüğü üzere bu tezin elde ettiği sonuçlar oldukça tatmin edicidir. Burada ilk satırda belirtilen (Ki, Kim, vd. 2015) çalışması hakkında çalışmanın makine öğrenmesi/derin öğrenme algoritmalarını değil, imza tabanlı algılama, DNA dizisi algoritması vb. bir dizi çalışmadan oluşan karmaşık bir yöntem kullandığına dikkat çekmek gerekir. Yani çalışma yazılımları ilk önce antivirüs tabanlarında aramakta, eğer zaten zararlı yazılım olarak bilinen bir yazılımsa direkt olarak "zararlı" şeklinde etiketleyerek sorunun bir kısmını zaten imza tabanlı algoritmalarla çözmektedir. Dolayısıyla bahsedilen çalışmalar, araştırmamıza kıyasla çok karmaşık olup, daha fazla zaman gerekmektedir. Sonuç olarak (birinci çalışmada olduğu gibi) yazılımların imzalarını (MD5, SHA-1, SHA-256, vb. özet değerleri) zararlı yazılımların veri tabanlarıyla karşılaştırarak daha önceden tespit edilen zararlı yazılımlardan istifade edilmeyen, elle yazılan hiçbir algoritma gerektirmeyen ve makine öğrenmesi/derin öğrenme yöntemleri kullanan araştırmamızın doğruluk oranlarının çok başarılı olduğu rahatlıkla söylenebilir.

DOĞRULUĞU ELDE EDEN ÇALIŞMA	ÖZELLİKLERİN ÇIKARILDIĞI VERİ KÜMESİ	KULLANILAN YAKLAŞIM	DOĞRULUĞUN ELDE EDİLDİĞİ VERİ KÜMESİ	KULLANILAN ÖZELLİK SAYISI	ÇÖZÜLEN PROBLEM TÜRÜ	ELDE EDİLEN DOĞRULUK
(Ki, Kim, vd. 2015)	1'inci Veri Kümesi	İmza-Tabanlı Tespit, Genetik Algoritma vb. birçok karmaşık çalışmalar bütünü	1'inci Veri Kümesi	-	Tespit	%99.88
Çalışmamız	1'inci Veri Kümesi	Zararlı/Zararsız API Çağrıları Yaklaşımı	1'inci Veri Kümesi	64	Tespit	%81.06
Çalışmamız	1'inci Veri Kümesi	En Çok Kullanılan API Çağrıları Yaklaşımı	1'inci Veri Kümesi	10	Tespit	%73.89
Çalışmamız	1'inci Veri Kümesi	En Çok Kullanılan API Çağrıları Yaklaşımı	1'inci Veri Kümesi	20	Tespit	%81.19
Çalışmamız	1'inci Veri Kümesi	En Çok Kullanılan API Çağrıları Yaklaşımı	1'inci Veri Kümesi	40	Tespit	%90.34
Çalışmamız	1'inci Veri Kümesi	En Çok Kullanılan API Çağrıları Yaklaşımı	1'inci Veri Kümesi	60	Tespit	%94.96
Çalışmamız	1'inci Veri Kümesi	En Çok Kullanılan API Çağrıları Yaklaşımı	1'inci Veri Kümesi	80	Tespit	%96.57
Çalışmamız	1'inci Veri Kümesi	En Çok Kullanılan API Çağrıları Yaklaşımı	1'inci Veri Kümesi	100	Tespit	%96.84
Çalışmamız	1'inci Veri Kümesi	En Çok Kullanılan API Çağrıları Yaklaşımı	1'inci Veri Kümesi	200	Tespit	%97.50
Çalışmamız	1'inci Veri Kümesi	En Çok Kullanılan API Çağrıları Yaklaşımı	1'inci Veri Kümesi	300	Tespit	%97.91
Çalışmamız	1'inci Veri Kümesi	En Çok Kullanılan API Çağrıları Yaklaşımı	1'inci Veri Kümesi	400	Tespit	%98.04
Çalışmamız	1'inci Veri Kümesi	En Çok Kullanılan API Çağrıları Yaklaşımı	2'nci Veri Kümesi	10	Sınıflandırma	%87.5
Çalışmamız	1'inci Veri Kümesi	En Çok Kullanılan API Çağrıları Yaklaşımı	2'nci Veri Kümesi	20	Sınıflandırma	%87.50
Çalışmamız	1'inci Veri Kümesi	En Çok Kullanılan API Çağrıları Yaklaşımı	2'nci Veri Kümesi	40	Sınıflandırma	%87.50
Çalışmamız	1'inci Veri Kümesi	En Çok Kullanılan API Çağrıları Yaklaşımı	2'nci Veri Kümesi	60	Sınıflandırma	%87.50
Çalışmamız	1'inci Veri Kümesi	En Çok Kullanılan API Çağrıları Yaklaşımı	2'nci Veri Kümesi	80	Sınıflandırma	%87.50
Çalışmamız	1'inci Veri Kümesi	En Çok Kullanılan API Çağrıları Yaklaşımı	2'nci Veri Kümesi	100	Sınıflandırma	%88.18
Çalışmamız	1'inci Veri Kümesi	En Çok Kullanılan API Çağrıları Yaklaşımı	2'nci Veri Kümesi	200	Sınıflandırma	%89.04
Çalışmamız	1'inci Veri Kümesi	En Çok Kullanılan API Çağrıları Yaklaşımı	2'nci Veri Kümesi	300	Sınıflandırma	%90.01

Çizelge 6.3 Farklı Yaklaşımlar, Veri Kümeleri, Özellik Sayı ve Seçimleriyle Derin Öğrenme Yöntemiyle Elde Edilen Farklı Doğruluk Değerlerinin Kıyaslanması (Devam)

Çalışmamız	1'inci Veri Kümesi	En Çok Kullanılan API Çağrıları Yaklaşımı	2'nci Veri Kümesi	400	Sınıflandırma	%89.89
Çalışmamız	2'nci Veri Kümesi	En Çok Kullanılan API Çağrıları Yaklaşımı	2'nci Veri Kümesi	10	Sınıflandırma	%87.51
Çalışmamız	2'nci Veri Kümesi	En Çok Kullanılan API Çağrıları Yaklaşımı	2'nci Veri Kümesi	20	Sınıflandırma	%87.63
Çalışmamız	2'nci Veri Kümesi	En Çok Kullanılan API Çağrıları Yaklaşımı	2'nci Veri Kümesi	40	Sınıflandırma	%88.36
Çalışmamız	2'nci Veri Kümesi	En Çok Kullanılan API Çağrıları Yaklaşımı	2'nci Veri Kümesi	60	Sınıflandırma	%89.50
Çalışmamız	2'nci Veri Kümesi	En Çok Kullanılan API Çağrıları Yaklaşımı	2'nci Veri Kümesi	80	Sınıflandırma	%89.95
Çalışmamız	2'nci Veri Kümesi	En Çok Kullanılan API Çağrıları Yaklaşımı	2'nci Veri Kümesi	100	Sınıflandırma	%90.16
Çalışmamız	2'nci Veri Kümesi	En Çok Kullanılan API Çağrıları Yaklaşımı	2'nci Veri Kümesi	200	Sınıflandırma	%90.43
Çalışmamız	2'nci Veri Kümesi	En Çok Kullanılan API Çağrıları Yaklaşımı	2'nci Veri Kümesi	278	Sınıflandırma	%90.66

6.2 Tartışma

Daha ayrıntılı ve kapsamlı tartışma için, çıkarılacak belirli çıkarımlar ve alınacak dersler bulunmaktadır:

- Makine öğrenimi yöntemlerinden yararlanarak, karmaşık algoritmalar veya hazırlık süreçleri olmadan, zararlı yazılımları tespit etmek ve sınıflandırmak mümkündür.
- Özelliklerin sayısı, doğruluk yüzdesine neredeyse sonsuza kadar katkıda bulunmaktadır. Ancak bir aşamadan sonra, özellik sayısının artmasıyla, bulunulan katkı tüketilen zamana kıyasla ciddi azalış göstermektedir. Bu özelliklerin sayısını algoritmanın hızını ve basitliğini koruyacak şekilde mümkün olduğunca ve optimal bir dereceye kadar (ki bu çalışma için 100-200 olduğu söylenebilir) artırmanın, sonrasında ise zaman/donanım maliyeti açısından artırmamanın mantıklı olacağı değerlendirilmektedir.
- İlk veri kümesindeki doğruluk oranlarının, ikinci veri kümesinden daha iyi olduğu görülmektedir. Bu da ilk veri kümesinde yer alan geniş benzersiz API Çağrısı sayısından kaynaklanmaktadır. Yani, doğru şekilde etiketlenmiş yeterli sayıda zararlı ve zararsız yazılıma (ve dolayısıyla benzersiz API Çağrısına) sahip olabilirsek, zararlı

yazılım algılama sorununa daha evrensel bir çözüm üreterek "Derin Öğrenen Antivirüs"ü oluşturabilmek mümkün olacaktır.

7. SONUÇ

Araştırmamızın birçok açıdan başarılı olduğunu söylemek mümkündür. Tanımlanmış sayıda özellik; tamamen homojen olan (sadece bir tür özellikten oluşan-API Çağrıları) ve başka herhangi bir özellik türü eklemeden bir mantık dâhilinde belirli sayıda elde edilen özellikler kullanılmıştır. Literatürdeki birçok doğruluk oranından daha iyi bir başarı elde edilmiştir. Bu doğruluk oranları, birden fazla makine öğrenimi yöntemi ve farklı veri kümeleri için de iyi seviyededir.

Makine öğrenimini kullanarak kötü amaçlı yazılım tespiti için daha basit ve daha hızlı yöntemler konusu üzerinde durularak hızlı ve sağlam bir çözüm bulunmuştur. Siber Güvenlik bir zekâ savaşıdır ve araştırmalar daha etkin, daha hızlı ve daha basit yöntemlere odaklanmalıdır. Ayrıca, "insan etkileşimi gerektiren/olay tarafından tetiklenen" zararlı yazılımlarla (Mehra ve Pandey 2015) ve "sınırlı zaman aralıklarında kötü amaçlı yazılımları tespit etme" (Kumar, Mukhopadhyay vd. 2019) ve (Irshad, Maurya vd. 2019) çalışmalarında olduğu gibi daha kompleks çalışmalar da Gelecekteki İş olarak not edilmektedir.

Bu tez sürecinde gerçekleştirdiğimiz çalışmalardan sonra, yeterli sayıda örneğe sahip olunduğunda bir "Derin Öğrenen Antivirüs" oluşturmanın çok daha kolay olacağı, bunun kötü amaçlı bilgisayar korsanlarına (black hat hackers) ağır bir yük getireceği ve evrensel bir siber güvenlik anlayışına yeni bir bakış açısı getireceği kuşkusuzdur.

KAYNAKLAR

Aliyev, V. (2010). Using honeypots to study skill level of attackers based on the exploited vulnerabilities in the network.

Anonymous. 2020a. Web Site: https://influencermarketinghub.com/social-media-statistics/, Son Erişim: 04.01.2021.

Anonymous. 2020b. Web Site: https://www.dihuni.com/2020/04/10/every-day-big-data-statistics-2-5-quintillion-bytes-of-data-created-daily/, Son Erişim: 04.01.2021.

Anonymous. 2020c. Web Site: https://www.safetydetectives.com/blog/malware-statistics/, Son Erişim: 04.01.2021.

Anonymous 2020d. Web Site: https://www.cse.unsw.edu.au/~cs9417ml/MLP2/ BackPropagation.html, Son Erişim: 04.01.2021.

Anonymous 2020e. Web Site: https://morioh.com/p/620a3a2faf6c, Son Erişim: 04.01.2021.

Anonymous 2020f. Web Site: https://www.analyticsvidhya.com/blog/2020/05/decision-tree-vs-random-forest-algorithm/, Son Erişim: 04.01.2021.

Anonymous 2020g. Web Site: https://data-flair.training/blogs/svm-support-vector-machine-tutorial/, Son Erişim: 04.01.2021.

Apruzzese, G., Colajanni, M., Ferretti, L., Guido, A. and Marchetti, M., 2018, May. On the effectiveness of machine and deep learning for cyber security. In 2018 10th International Conference on Cyber Conflict (CyCon) (pp. 371-390). IEEE.

Aslan Ö., Samet R. 2017. Investigation Of Possibilities To Detect Malware Using Existing Tools. In 14th International Conference On Computer Systems And Applications (IEEE/ACS), pp.1277-1284.

Bernard, S., Heutte, L. and Adam, S., 2009, June. On the selection of decision trees in random forests. In 2009 International Joint Conference on Neural Networks (pp. 302-307). IEEE.

Catak, F.O. and Yazı, A.F., 2019. A benchmark API call dataset for windows PE malware classification. arXiv preprint arXiv:1905.01999.

Choudhury, T., Jain, S. K., Aradhya, N. and Kumar, P., 2018. An Entire Dynamic Malware Examination with Near Investigation of Conduct Examination Sandboxes. In Second International Conference on Green Computing and Internet of Things (ICGCIoT), pp. 583-590. IEEE.

Chumachenko, K., 2017. Machine learning methods for malware detection and classification.

Damodaran, A., Di Troia, F., Visaggio, C. A., Austin, T. H., & Stamp, M. (2017). A comparison of static, dynamic, and hybrid analysis for malware detection. Journal of Computer Virology and Hacking Techniques, 13(1), 1-12.

Darshan, S.S., Kumara, M.A. and Jaidhar, C.D., 2016, December. Windows malware detection based on cuckoo sandbox generated report using machine learning algorithm. In 2016 11th International Conference on Industrial and Information Systems (ICIIS) (pp. 534-539). IEEE.

Fujino, A., Murakami, J. and Mori, T., 2015, January. Discovering similar malware samples using api call topics. In 2015 12th annual IEEE consumer communications and networking conference (CCNC).

Hansen, S.S., Larsen, T.M.T., Stevanovic, M. and Pedersen, J.M., 2016, February. An approach for detection and family classification of malware based on behavioral analysis. In 2016 International conference on computing, networking and communications (ICNC) (pp. 1-5). IEEE.

Hex-Rays 2020. Web Site: https://www.hex-rays.com/products/ida/. Son Erişim: 04.01.2021.

Ijaz, M., Durad, M.H. and Ismail, M., 2019, January. Static and dynamic malware analysis using machine learning. In 2019 16th International bhurban conference on applied sciences and technology (IBCAST) (pp. 687-691). IEEE.

Irshad, A., Maurya, R., Dutta, M.K., Burget, R. and Uher, V., 2019, July. Feature Optimization for Run Time Analysis of Malware in Windows Operating System using Machine Learning Approach. In 2019 42nd International Conference on Telecommunications and Signal Processing (TSP) (pp. 255-260). IEEE.

Jamalpur, S., Navya, Y.S., Raja, P., Tagore, G. and Rao, G.R.K., 2018, April. Dynamic malware analysis using cuckoo sandbox. In 2018 Second international conference on inventive communication and computational technologies (ICICCT) (pp. 1056-1060). IEEE.

Kilgallon, S., De La Rosa, L., & Cavazos, J. 2017. Improving the effectiveness and efficiency of dynamic malware analysis with machine learning. In 2017 Resilience Week (RWS) (pp. 30-36). IEEE.

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), p.659101.

Kumar, N., Mukhopadhyay, S., Gupta, M., Handa, A. and Shukla, S.K., 2019, August. Malware Classification using Early Stage Behavioral Analysis. In 2019 14th Asia Joint Conference on Information Security (AsiaJCIS) (pp. 16-23). IEEE.

Lengyel, T.K., Maresca, S., Payne, B.D., Webster, G.D., Vogl, S. and Kiayias, A., 2014, December. Scalability, fidelity and stealth in the DRAKVUF dynamic malware analysis system. In Proceedings of the 30th Annual Computer Security Applications Conference (pp. 386-395).

Liu, Y. and Wang, Y., 2019, March. A robust malware detection system using deep learning on api calls. In 2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC) (pp. 1456-1460). IEEE.

McAfee. 2020. Web Site: https://www.mcafee.com/enterprise/en-us/security-awareness/ransomware/what-is-malware.html, Son Erişim: 27.12.2020.

Mehra, M. and Pandey, D., 2015, December. Event triggered malware: A new challenge to sandboxing. In 2015 Annual IEEE India Conference (INDICON) (pp. 1-6). IEEE.

Pascariu, C. and Barbu, I.D., 2017, June. Dynamic analysis of malware using artificial neural networks: Applying machine learning to identify malicious behavior based on parent process hirarchy. In 2017 9th International Conference on Electronics, Computers and Artificial Intelligence (ECAI) (pp. 1-5). IEEE.

Pirscoveanu, R.S., Hansen, S.S., Larsen, T.M., Stevanovic, M., Pedersen, J.M. and Czech, A., 2015, June. Analysis of malware behavior: Type classification using machine learning. In 2015 International conference on cyber situational awareness, data analytics and assessment (CyberSA) (pp. 1-7). IEEE.

Rege, M. and Mbah, R.B.K., 2018. Machine learning for cyber defense and attack. DATA ANALYTICS 2018, p.83.

Researchgate. 2020. Web Site: https://www.researchgate.net/Şekil/The-typical-decision-tree-model-J48-are-the-improved-versions-of-C45-algorithms-or-can_fig1_288807267, Son Erişim: 04.01.2021.

Shijo, P.V. and Salim, A.J.P.C.S., 2015. Integrated static and dynamic analysis for malware detection. Procedia Computer Science, 46, pp.804-811.

Sun, B., Fujino, A., Mori, T., Ban, T., Takahashi, T. and Inoue, D., 2018. Automatically generating malware analysis reports using sandbox logs. IEICE TRANSACTIONS on Information and Systems, 101(11), pp.2622-2632.

Udayakumar, N., Anandaselvi, S. and Subbulakshmi, T., 2017, December. Dynamic malware analysis using machine learning algorithm. In 2017 International Conference on Intelligent Sustainable Systems (ICISS) (pp. 795-800). IEEE.

Walker, A., Amjad, M.F. and Sengupta, S., 2019, January. Cuckoo's malware threat scoring and classification: Friend or foe?. In 2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC) (pp. 0678-0684). IEEE.

Wang, Z., 2015. The applications of deep learning on traffic identification. BlackHat USA, 24(11), pp.1-10.