T.C.
GEBZE TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

METAMORFİK KÖTÜCÜL KODLARIN GERÇEK ZAMANLI TESPİTİ

NECMETTİN ÇARKACI

YÜKSEK LİSANS TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

GEBZE
2016

T.C.
GEBZE TEKNİK ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

METAMORFİK KÖTÜCÜL KODLARIN GERÇEK ZAMANLI TESPİTİ

NECMETTİN ÇARKACI
YÜKSEK LİSANS TEZİ
BİLGİSYAR MÜHENDİSLİĞİ ANABİLİM DALI

DANIŞMANI
PROF. DR. İBRAHİM SOĞUKPINAR

GEBZE
2016

T.R.

GEBZE TECHNICAL UNIVERSITY

GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

REAL TIME METAMORPHIC MALWARE DETECTION

NECMETTIN ÇARKACI

A THESIS SUBMITTED FOR THE DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTER ENGINEERING

THESIS SUPERVISOR

PROF. DR. İBRAHİM SOĞUKPINAR

GEBZE

2016

GEBZE TEKNİK ÜNİVERSİTESİ	**YÜKSEK LİSANS JÜRİ ONAY FORMU**

GTÜ Fen Bilimleri Enstitüsü Yönetim Kurulu'nun 27/06/2016 tarih ve 2016/43 sayılı kararıyla oluşturulan jüri tarafından 26../08../2016 tarihinde tez savunma sınavı yapılan Necmettin ÇARKACI'nın tez çalışması Bilgisayar Mühendisliği Anabilim Dalında YÜKSEK LİSANS tezi olarak kabul edilmiştir.

JÜRİ

ÜYE (TEZ DANIŞMANI)	: Prof. Dr. İbrahim SOĞUKPINAR
ÜYE	: Doç. Dr. Ali Gökhan YAVUZ
ÜYE	: Yrd. Doç. Dr. Burcu YILMAZ

ONAY

Gebze Teknik Üniversitesi Fen Bilimleri Enstitüsü Yönetim Kurulu'nun

...../...../..... tarih ve ...../..... sayılı kararı.

İMZA/MÜHÜR

ÖZET

Kötücül kodlar metamorfik kod gizleme yöntemleri sayesinde her bir nesilde büyüklüğü, yapısı ve çalışma şekli farklı fakat işlevselliği aynı olan yeni kötücül kodlar üretmektedirler. Böylece imza tabanlı geleneksel kötücül kod tespit sistemlerini atlatabilmektedirler. Çalışmamız kapsamında MAIL (Malware Analysis Intermediate Language) dilinin ürettiği özet yapılar kullanılarak metamorfik yapıdaki kötücül kodların tespitini yapan bir yöntem geliştirilmiştir. MAIL diline ait kodların frekans değerleri için öznitelik çıkarma, öznitelik seçimi ve sınıflandırma algoritmalarının doğruluk ve performans açısından incelemesi gerçekleştirilmiştir. Sistem metamorfik kötücül kod yöntemlerini kullanan NGVCK, G2, VCL32, PSMPC kötücül kod sınıfları üzerinde test edilmiş olup MAIL yapısına ait 21 adet özet öznitelik kod kümesi içerisinden 2 adet öznitelik seçilerek %90 oranında öznitelik azaltımı ile %100 oranında kötücül kod tespit başarımı sağlamıştır.

Anahtar Kelimeler: Kötücül Kod Tespiti, Metamorfik Kötücül Kod, Durağan Analiz, Öznitelik Seçimi, Örüntü Tanıma, Sınıflandırma.

SUMMARY

Malwares can create new malware samples which have different size, structure and operation mode but same functionality in each metamorphic code generation via malicious code obfuscation methods. So they bypass traditional signature-based malware detection systems. In this study, a pattern recognition based system that detects metamorphic malware by using summary structure of Malware Analysis Intermediate Language (MAIL) has been proposed. For the term frequency of MAIL language codes, feature extraction, feature selection and classification algorithm is researched in terms of accuracy and performance. The system is tested with metamorphic malware construction kits NGVCK, G2, VCL32, PSMPC and achieve %100 accuracy with 2 of 21 MAIL feature and implement %90 feature reduction.

Key Words: Malware Detection, Metamorphic Malware, Static Analysis, Feature Selection, Pattern Recognition, Classification.

TEŞEKKÜR

Başta, yüksek lisans eğitimimde ve akademik hayatımda desteğini ve yardımlarını hiçbir zaman esirgemeyip bilgisi ile bu çalışmanın oluşmasının yolunu açan danışmanım Prof. Dr. İbrahim SOĞUKPINAR'a,

ve tez çalışma sürecinde göstermiş olduğu sabır ve verdiği desteklerden dolayı sevgili eşim Nimet Hande ÇARKACI ile oğlum Muhammed Yusuf ÇARKACI'ya en içten teşekkürlerimi sunarım.

İÇİNDEKİLER

SİMGELER ve KISALTMALAR DİZİNİ

Simgeler ve Kısaltmalar		Açıklamalar
API Call	:	Application program interface call
DFA	:	Deterministic finite automaton
G2	:	Second generation virus kit
IDF	:	Inverse document frequency
MAIL	:	Malware analysis intermated language
MPCGEN	:	Mass code generation
NGVCK	:	Next generation construction kit
NOP	:	No operation code
TF	:	Term frequency
TF-IDF	:	Term frequency - Inverse document frequency
VCL32	:	Virus creation lab

ŞEKİLLER DİZİNİ

TABLOLAR DİZİNİ

1. GİRİŞ

Akıllı telefonlarla birlikte taşınabilir ortamların yaygınlık kazanması, e-devlet, e-yönetişim, e-ticaret alanındaki uygulama ve kullanıcı sayısındaki artış, internet teknolojilerini kullanan Google, Facebook, Amazon vb. gibi küresel ölçekli büyük şirketlerin ortaya çıkışı ile birlikte önceleri vandalizm veya hobi olarak gelişen siber korsanlık faaliyetleri yavaş yavaş siber hırsızlık, daha da ötesinde hedef odaklı saldırı tiplerine dönüşmüş durumdadır. Bu gelişmelere paralel olarak devletler siber ordular kurmaya ve birbirilerine yönelik siber saldırılar organize etmeye başlamıştır. Bu saldırılarda karmaşıklığı gelişmiş, profesyonel gruplar tarafından geliştirilen gelişmiş kötücül kodlar kullanılmaktadır.

İnternet kullanımının yaygınlaşması ve bilginin daha kolay ulaşılabilir hale gelmesi ile birlikte zararlı kodlar ve bu kodları otomatik üreten araçlar her seviyeden kullanıcının rahatlıkla ulaşabileceği hale gelmiştir. Günümüzde az bir bilgiyle etkili ve güçlü kötücül kodlar kolaylıkla kullanılabilir ve üretilebilir durumdadır. Bu durum kötücül kod sayısını artıran önemli faktörler arasındadır. AV-TEST enstitüsünün 2016 raporuna göre her gün 390.000 adet yeni kötücül kod ortaya çıkmaktadır [1].

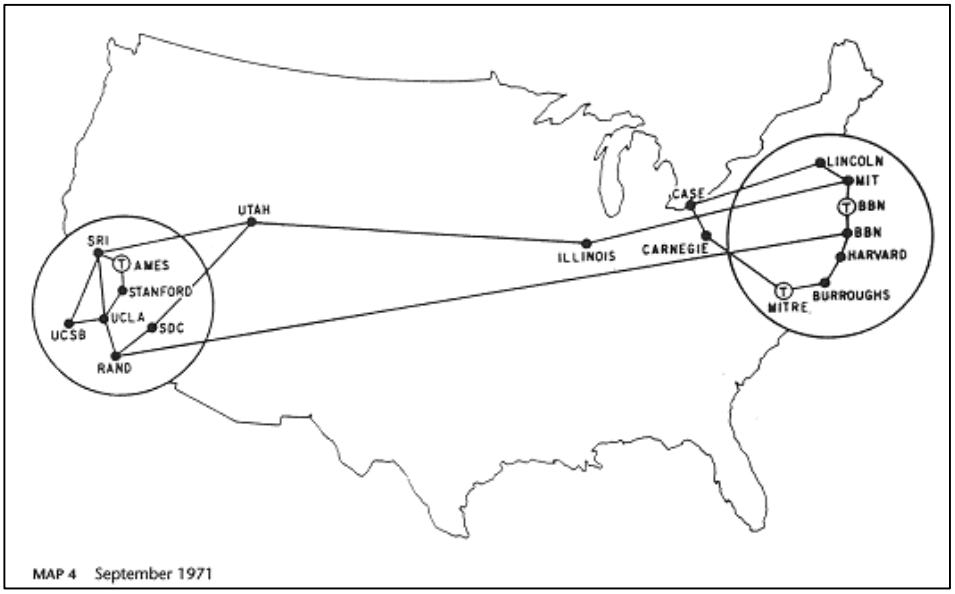
Zararlı yazılımları kötü amaçlar için kullanan kitlenin artması ve zararlı yazılımların siber saldırı gibi özel amaçlar için profesyonel gruplar tarafından kullanılması kötücül kodların çeşitliliğini, karmaşıklığını ve sayısını arttırdığından dolayı bu kodların tespitinde ideal bir yöntem henüz mevcut değildir.

Kötücül kodların çeşitliliği ve karmaşıklığını artırmada kötücül kodlara değişme özelliği katan polimorfik ve metamorfik yöntemler yaygın olarak kullanılmaktadır [2]. Bu yöntemler sayesinde bu tarz kötücül kodlar geleneksel imza tabanlı kötücül kod tanıma sistemlerini atlatabilmektedirler.

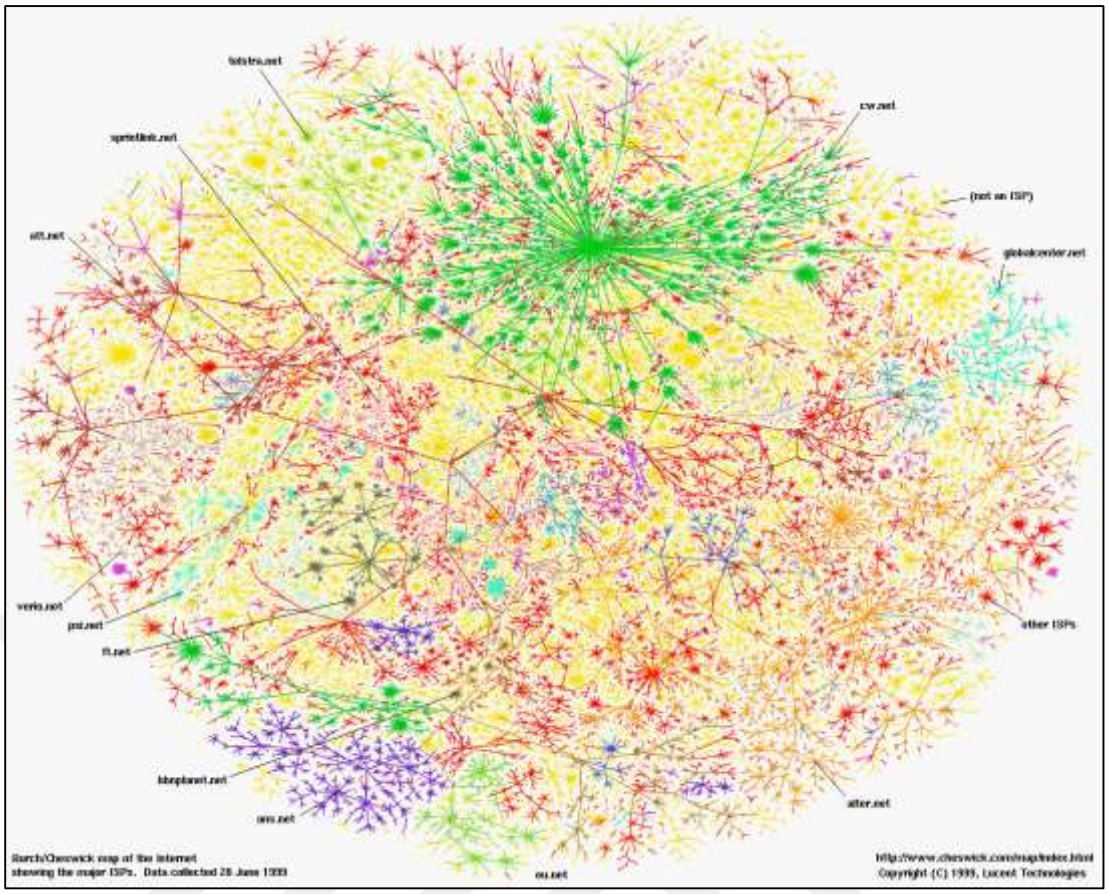
Son yıllarda metamorfik yapıdaki kötücül kodları tespit etmeye yönelik makine öğrenmesi tabanlı yöntemler geliştirilemeye başlanmıştır. Bununla birlikte öznitelik sayısındaki büyüklük gerçek zamanlı olarak kötücül kodların tespiti önünde büyük bir engel teşkil etmektedir. Çalışmamızda kötücül kod tespiti için değerli öznitelikleri seçerek daha az öznitelikle, kötücül kodları yüksek başarım oranıyla tespit edecek bir yöntem geliştirilmiştir.

1.1. Motivasyon

İnternet başlangıçta birbiriyle güven sorunu olmayan küçük bir grup için tasarlanmıştı. İnterneti tasarlayanlar güvenlikten daha çok sorumluluğun dağıtılmasına odaklanmışlardı. İlerleyen yıllarda büyük finans işlemleri, veri paylaşımları için kullanılacak büyük kullanıcı kitlesine sahip bir ağ olacağı düşünülmemişti. Bununla birlikte son otuz yıl içerisinde internetin genel kullanıma da açılmasıyla birlikte ilk ağ tasarımına göre inanılmaz bir büyüme göstermiştir. Şekil 1.1: Arpanet 1971 ağ haritasında görüleceği gibi 1971 yılında ARPANET küçük bir ağ olarak tasarlanmışken Şekil 1.2: İnternet 1999 ağ haritasında görüleceği üzere bundan otuz yıl sonra 2000'li yıllarda devasa büyüklüğe ulaşmıştır. Günümüzde ise neredeyse elektriğin var olduğu her yer internet ile ulaşılabilir durumdadır. Dünya dışında uydular dahi iletişim için bazı internet kanallarını kullanmaktadır. İnternetin yaygınlaşmasındaki bu artış güvenli açıklıklarında ki artışı da beraberinde getirmiştir. Bugün yaygın internet ağının ulaşım gücü ve saldırılara karşı zayıf yapısı nedeniyle, dünyanın farklı bölgelerinde yaşayan insanlar birbiri için tehlike arz edecek durumdadırlar. Sistemlerin internet vasıtasıyla herkes tarafından ulaşılabilir olması, saldırganlar için çok uzaklardaki sistemlere, çok farklı amaçlar için, basit yöntemler kullanarak saldırabilme imkânı sağlamıştır. Bu nedenle günümüzde bilgi iletişim altyapı sistemlerinin internetin ilk dönemlerine göre saldırıya maruz kalma potansiyeli daha yüksektir.

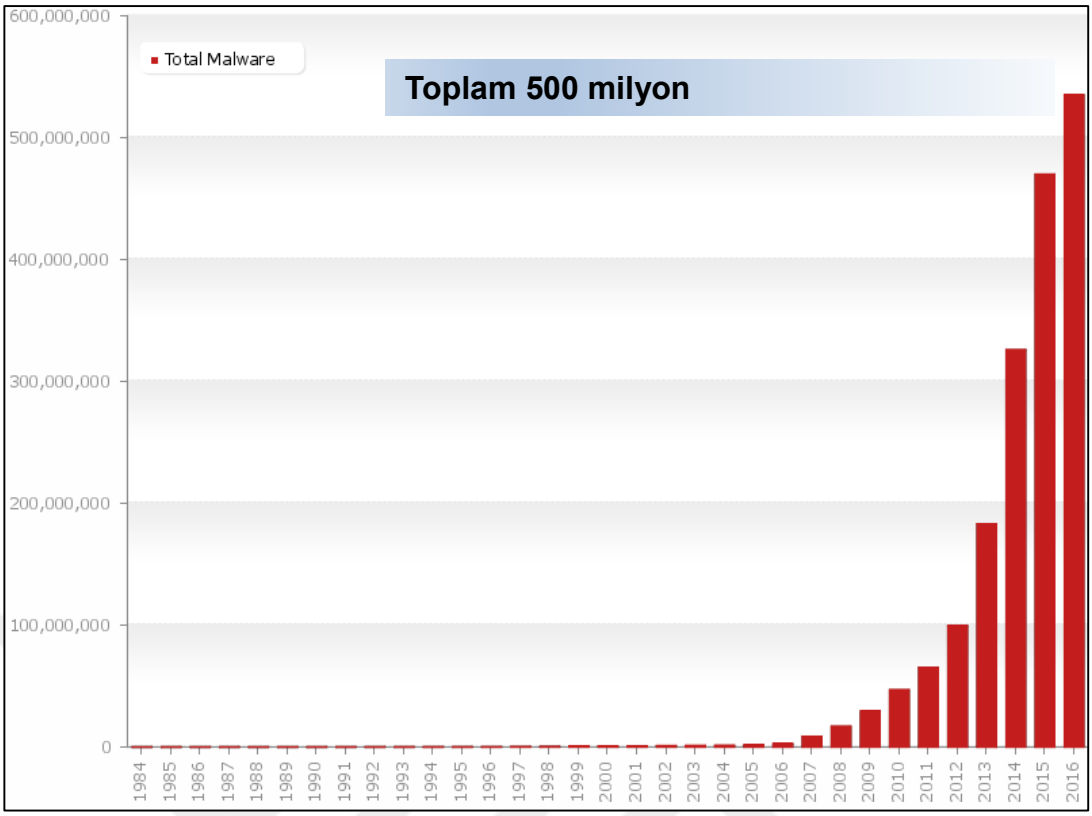


Şekil 1.1: Arpanet 1971 ağ haritası.

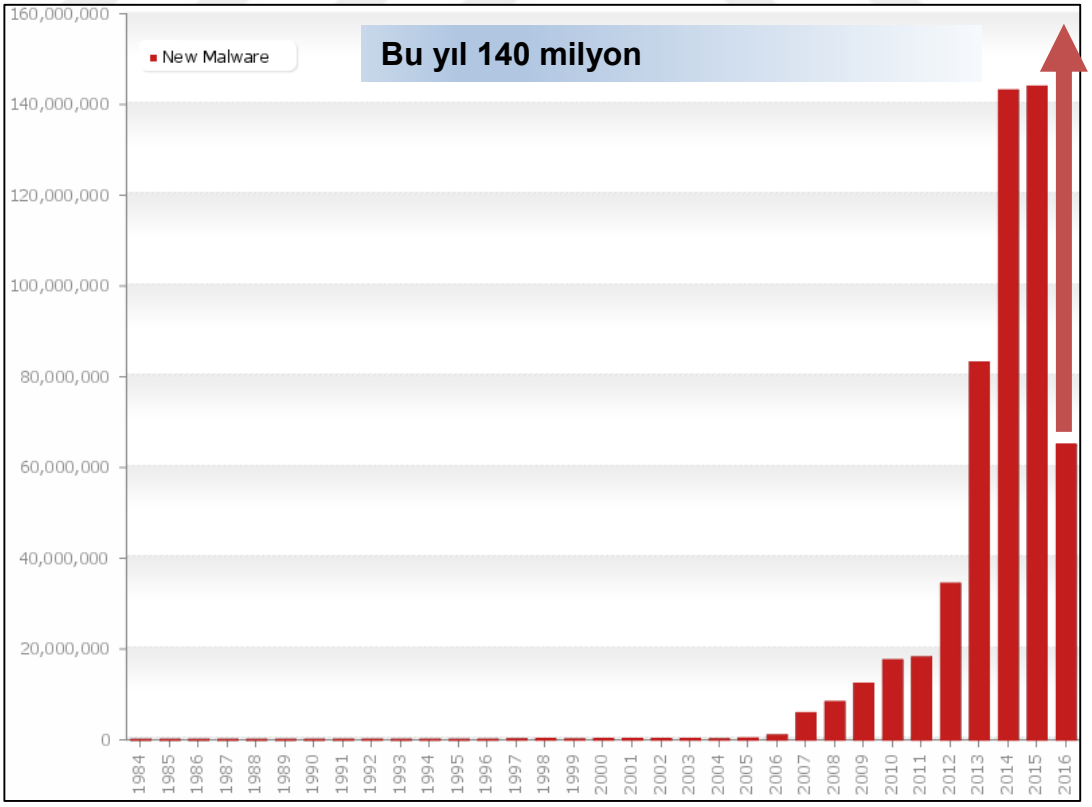


Şekil 1.2: İnternet 1999 ağ haritası.

Şekil 1.3: AV Test enstitüsü yıllara göre toplam kötücül kod artış oranı grafiğinde görüldüğü üzere 2016 yılı itibariyle toplam kötücül kod sayısı 550 milyona yaklaşmıştır. 2012 yılından itibaren yıllık kötücül kod artış oranı %90'lar seviyesinde seyretmektedir. Buna ek olarak Şekil 1.4: AV Test enstitüsü yıllara göre yeni kötücül kod artış oranı grafiğinde görüleceği üzere bu kötücül kodların yaklaşık yarısına yakını son iki yıl içerisinde ortaya çıkmış kötücül kodlardan oluşmaktadır. Bu durum kötücül kodların çok hızlı şekilde büyüdüğü ve her geçen gün çeşitliliğinin daha da arttığının açık göstergesidir. Özellikle 2007 yılında kötücül kod sayısındaki artışın belirgin olarak ortaya çıkması akıllı telefonların ortaya çıkması ile açıklanabilir. Kötücül kod sayısındaki 2013 ve 2014 sonrası büyük sıçrama ise dünya ölçeğinde bilgi işlem altyapılarında ortaya çıkan sıçrama ile doğrudan bağlantılıdır. Kötücül kod sayısındaki bu hızlı artış büyük veri işleme ve makine öğrenmesi yöntemleri ile büyük verilere ait özniteliklerin tespiti araştırma konularını önemini vurgular niteliktedir.

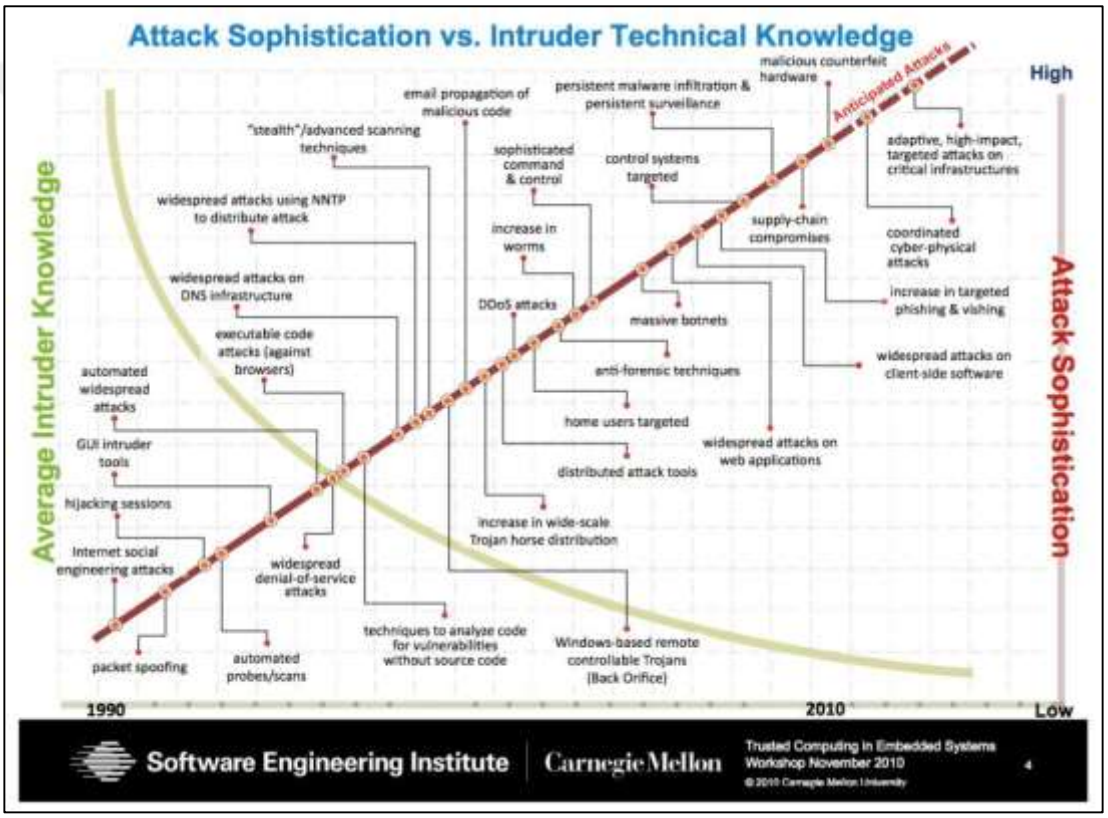


Şekil 1.3: AV Test enstitüsü yıllara göre toplam kötücül kod artış oranı grafiği.



Şekil 1.4: AV Test enstitüsü yıllara göre yeni kötücül kod artış oranı grafiği.

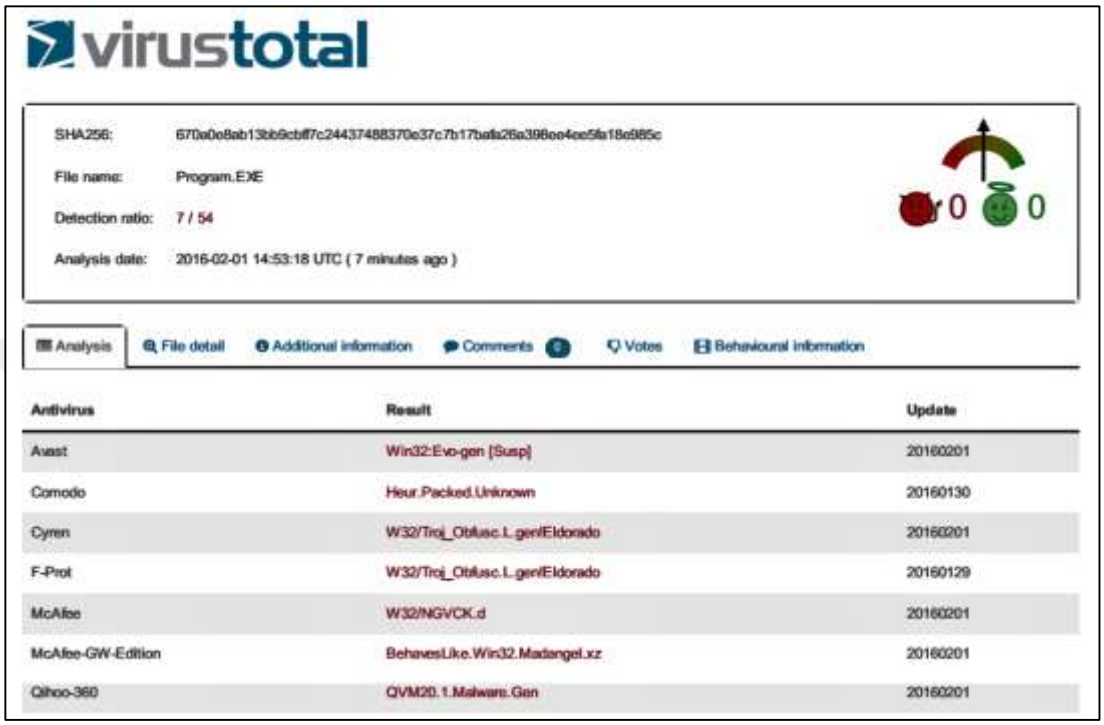
Bunun yanı sıra “Z0mbi” kod isimli kötücül kod yazarı tarafından kaynak kodları açık halde 2000 yılında yazılan, karmaşık metamorfik kod gizleme yöntemleri kullanan, Zmist virüsünün güncellenmiş yeni sürümü ZmistiK 2012 yılında anti virüs şirketleri tarafından tespit edildi [3]. Bu durum değişme özelliğine sahip 2000’li yıllarda oluşturulan metamorfik kodların, kötücül yazılım tespit sistemlerine yakalanmadan hala etkinliklerini sürdürebildiklerinin açık bir göstergesidir. Yeni ortaya çıkan kötücül kodların yanı sıra geçmiş yıllara ait, kötücül olarak tespit edilmiş, birçok kötücül kodun metamorfik yöntemler sayesinde tanınmadan etkinliğini günümüzde de devam ettirebildiğinin açık kanıtıdır.



Şekil 1.5: Saldırı karmaşıklığı ile saldırgan teknik bilgisinin tarihsel gelişim grafiği.

Günümüzde kötücül yazılım oluşturma araçları internet ortamında kolaylıkla ulaşılabilecek durumdadır. Bu durum sınırlı teknik bilgiye sahip kişilerin potansiyel olarak yıkıcı kötücül yazılımlar oluşturabilmelerine olanak sağlamaktadır [4]. Carnegie Mellon Üniversitesi Yazılım Mühendisliği tarafından yapılan saldırıların karmaşıklığı ve saldırganların teknik bilgisinin tarihsel gelişimini gösteren Şekil 1.5’deki Saldırı karmaşıklığı ile saldırgan teknik bilgisinin tarihsel gelişim grafiğinden de görüleceği üzere zamanla saldırılar daha da karmaşık hale gelmekte bununla birlikte

ortalama saldırganın saldırı için ihtiyaç duyduğu teknik bilgi gittikçe azalmaktadır. İlk dönemlerde küçük saldırılar için büyük oranda teknik bilgi gerekirken, günümüzde çok az teknik bilgiyle otomatikleştirilmiş araçları kullanarak büyük ölçekli karmaşık saldırılar yapılabilmekte, kritik altyapılar tahrip edilebilmektedir.



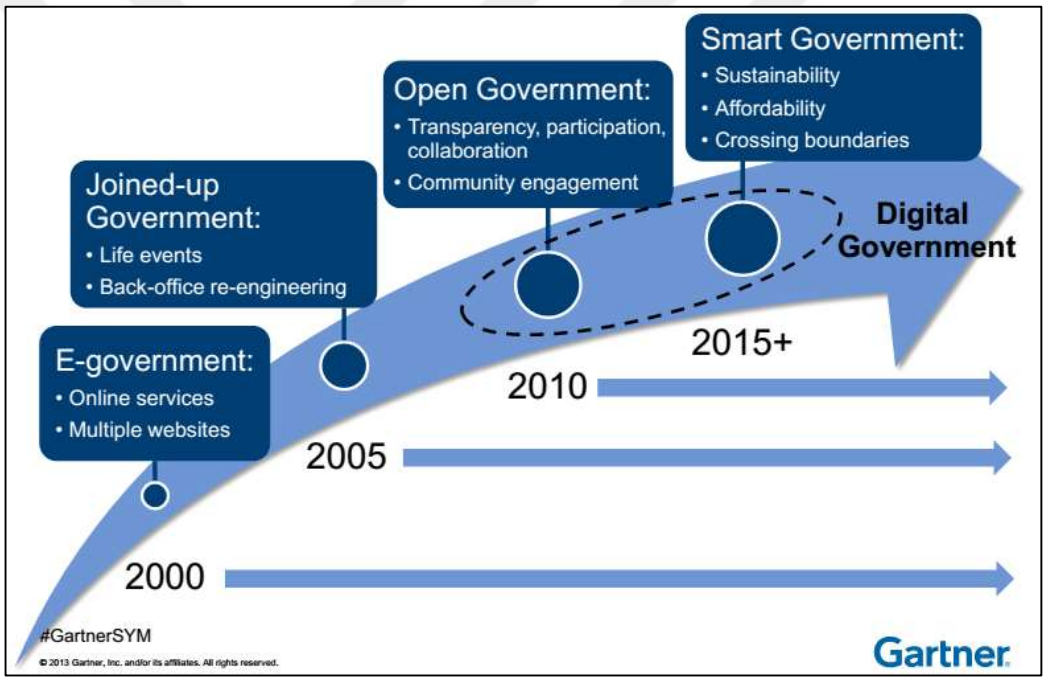
Şekil 1.6: Metamorfik yapıdaki kötücül kod örneğinin VirusTotal değerlendirme raporu.

Çalışmamız kapsamında analiz ettiğimiz NGVCK metamorfik kötücül kod sınıfına ait kötücül kod örneklerinden birini VirusTotal[1] sanal yöresinde bulunan mevcut anti virüs sistemleri üzerinde test ettiğimizde, Şekil 1.6'daki sonuç raporunda görüldüğü üzere kötücül kod örneğimiz yaygın olarak kullanılan 54 adet antivirüs sisteminden sadece 7 tanesi (%13) tarafından kötücül kod olarak tespit edilebilmiştir. Bunu yanı sıra sadece 1 tanesi (McAfee[2]) tarafından kötücül kod sınıfı doğru olarak tespit edilebilmiştir. VirusTotal sanal yöresi uç kullanıcılar tarafından yaygın olarak kullanılan neredeyse tüm anti-virüs yazılımlarını kötücül kod tespiti için kullandığı düşünüldüğünde problemin büyüklüğü daha da ortaya çıkmaktadır.

Akademisyenler, ticaret dünyasından uzmanlar, gelecek bilimciler ve sosyal bilimcilerden oluşan uzman gruplara hazırlattığı bilişim sistemlerinin beşer yıllık

[1]Kötücül kod analiz ortamı : https://www.virustotal.com/
[2]Kötücül kod tespit sistemi : http://www.mcafee.com/

gelişim öngörü grafikleri ile ün yapmış Gartner araştırma şirketi tarafından hazırlanan Şekil 1.7: E-Yönetişim, e-devlet, akıllı yönetim sistemleri gelişim grafiğine göre e-devlet ve e-yönetişim uygulamaları 2010 yılı itibariyle büyük oranda sanal ortama taşındığı görülmektedir. Hatta 2015 yılı itibariyle dijital devlet yaklaşımının oturacağı, 2015 yılı sonrasında akıllı yönetim sistemlerine doğru ilerleneceği öngörülmektedir. Hali hazırda dünyanın bazı bölgelerinde seçimlerin elektronik ortamda yapıldığı, seçmenlerin elektronik ortam üzerinden alınan kararlarla yönetime katkıda bulunduğu, elektronik imza sayesinde, birçok resmi işlemin sanal ortamda gerçekleştiği, hatta eski kâğıt ortamındaki evrakların taranarak sanal ortama taşınarak buradan işlendiği düşünüldüğünde dijital dünyanın artık yönetim alanın ayrılmaz bir parçası olduğu söylenebilir.



Şekil 1.7: E-Yönetişim, e-devlet, akıllı yönetim sistemleri gelişim grafiği.

Bürokratik işlemleri azaltıp, hayatımızı kolaylaştıran, bize zaman ve para tasarrufu sağlayan e-devlet ve e-yönetişim alanındaki bu gelişmelere paralel olarak sanal dünyaya bilgi teknolojileri ile bağlı kişiler ve devletlerin kritik bilgileri bu geniş sahada saldırılara açık hale geldi. Bu saldırılara örnek olarak İran nükleer santrallerinde sistemin ısıtma sisteminde arızaya neden olarak sistemi tahrip eden Stuxnet solucanı gösterilebilir.

Amerika ve Avrupa Birliği ülkeleri için kritik altyapıların önem sıralaması Tablo 1.1: AB ve ABD için önem sırasına göre kritik altyapılar [5]'te gösterilmiştir. Bununla birlikte bu tabloda görülen bilgi ve iletişim dışındaki kritik altyapıların birçoğu sanal ortama taşınmış; sanal ortamda iş gören bilgisayarlar tarafından kontrol edilip yönetilmektedir. Bu kritik altyapı kaynaklarına yönelik sanal dünyada yapılan hacktivist eylemlerin ve yeni sanal hırsızlık modellerinin popülerlik ve yaygınlık kazanması ve kötücül kodların bu saldırılarda etkin şekilde kullanılması kötücül kod sayısı, çeşitliliğini artıran diğer bir önemli etkendir [6].

Tablo 1.1: AB ve ABD için önem sırasına göre kritik altyapıları.

AB	ABD
Su	Su
Gıda	Enerji
Sağlık	Ulaşım
Enerji	Tarım ve Gıda
Finans	Kolluk Hizmetleri
Ulaşım	**Bilgi ve iletişim**
Sivil Yönetim	Bankacılık ve Finans
Bilgi ve İletişim	Bayındırlık Hizmetleri
Uzay ve Araştırmaları	Federal ve Yerel Hizmetler
Kimyasal ve Nükleer Endüstri	Acil Hizmetler
Kamu Düzeni ve Güvenlik Alanı	

Devletler siber ordular kurmaya ve birbirilerine yönelik siber saldırılar organize etmeye başlamıştır. Bu saldırılarda karmaşıklığı gelişmiş, profesyonel gruplar tarafından geliştirilmiş gelişmiş kötücül kodlar kullanmaktadırlar. Bu konuda son örnek Çin Halk Cumhuriyeti ordusunun Amerika Birleşik Devletlerinin kritik bilgi kaynaklarına ve ekonomik anlamda büyük şirketlerine yönelik siber saldırıları gösterilebilir [7]. Bu saldırılar sonrası ABD başkanı Hüseyin Barack Obama'nın Amerika için yeni dönemde en büyük tehdit siber terörizmdir açıklaması sorunun önemi ortaya koyar niteliktedir. Beyaz saray siber güvenlik uzmanı Richard Clark'ın "Casusluk artık çok kolaylaştı. Eskiden Washington'daki Rus Elçiliği'nde çalışan bir KGB ajanının bir FBI ajanı ayartması çok zordu. Ama şimdi Moskova'da oturuyorsun.

Ve hiçbir risk olmadan binlerce sayfa çalabiliyorsun. Eskinin casuslarına artık gerek yok.", "Eskiden Sinop'ta büyük bir kulemiz vardı. Rusya'daki konuşmaları dinliyorduk. Ama şimdi buna ihtiyaç yok. Kimse radyo frekansı kullanmıyor. Ulusal Güvenlik Ajansı'nın (NSA) Maryland'deki kampüsünden bütün dünyadaki internet trafiğini izliyoruz." [8], [9] sözleri sanal dünyanın siber savaşlar için etkin şekilde kullanıldığının açık göstergesidir. ABD, Çin, Rusya, Kuzey Kore, İsrail, İran, Suriye ve gelişmiş Avrupa ülkeleri dünyada siber savaş için ordu kuran ve eylem planları olan başlıca ülkeler arasındadır. Bu savaşlara örnek olarak Estonya'ya yapılan siber saldırı gösterilebilir. Estonya'ya yapılan bu siber saldırı neticesinde otuz gün boyunca bankalar, devlet daireleri, marketler işlem yapamaz hale gelmiş, adeta hayat durmuştur. Güvenlik uzmanları tarafından Estonya'ya yapılan bu saldırı gelecekte olabilecek bir savaşta siber saldırıların etkisini ölçmek için yapılmış bir tatbikat olarak yorumlanmıştır. Bu mücadelede büyük ekipler tarafından hazırlanmış sıfır gün ataklarını bünyesinde barındıran karmaşıklığı gelişmiş, hedef odaklı kötücül yazılımlar kullanılmaktadır. Bu yazılımların en önemli örnekleri İran ve Rusya'nın nükleer santrallerine sızıp zarar veren Stuxnet ve türevi Duqu solucanlarıdır.

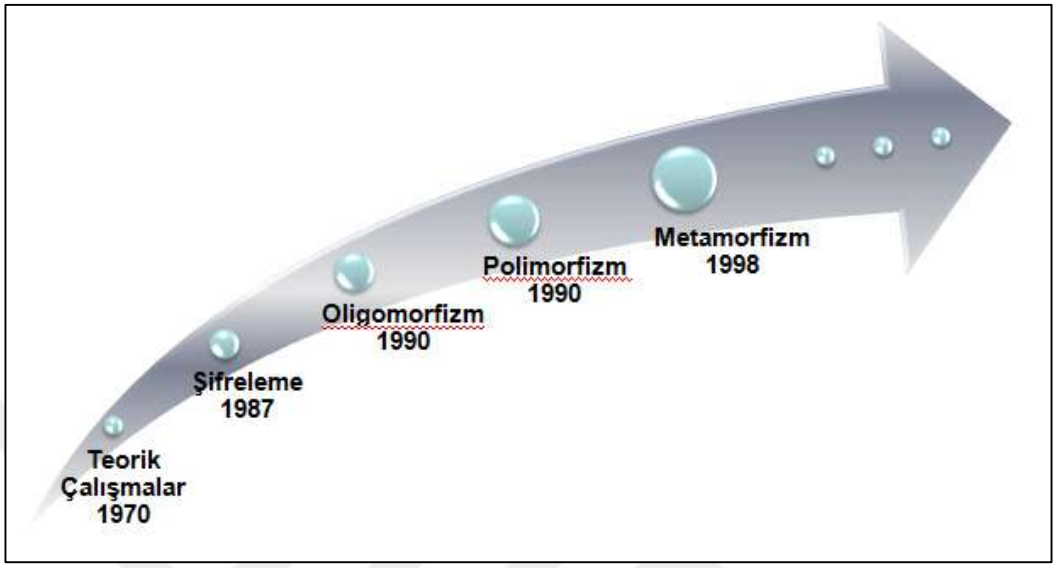
Yukarıda açıklanan nedenlere dayalı olarak bilinmeyen kötücül kodların sayısında, karmaşıklığında ve çeşitliliğindeki hızlı artış, mevcut anti-virüs sistemleri tarafından sıklıkla kullanılan bilinen kötücül kodların tespitine yönelik geliştirilmiş imza tabanlı yöntemlerin artık gözden geçirilmesi gerektiğinin ve büyük veri analizine dayalı makine öğrenmesi destekli yeni yöntemlerin geliştirilmesi gerektiğinin göstergesidir.

Bu amaçla son dönemlerde bilinmeyen kötücül kodların tespitine yönelik makine öğrenmesi temelli yöntemler geliştirilmeye başlanmış ve başarılı sonuçlar elde eden çalışmalar yapılmıştır. Bununla birlikte öznitelik uzayının büyüklüğü, kötücül kodların kendilerini gizlemek amacıyla metamorfik ve polimorfik kod gizleme yöntemlerini kullanmaları gibi nedenlerle kötücül kodların gerçek zamanlı tespiti konusu araştırmacılar ve anti virüs şirketleri tarafından önemini sürdürmeye devam etmektedir.

Bu tez kapsamında, Bölüm 2'de metamorfik kötücül kod gizleme yöntemleri üzerinde durulacak olup kuramsal temeller ve konuyla ilgili yapılmış çalışmaların açıklanması ile devam edilecektir. Bölüm 3'de araştırma soruları ve çalışma kapsamında takip edilen metodoloji açıklanmıştır. Bölüm 4'de yaptığımız deneysel çalışmalar ve sonuçları paylaşılmış olup makale Bölüm 5 ile sonuçlandırılmıştır.

2. KURAMSAL TEMELLER

2.1. Kod Gizleme Yöntemleri

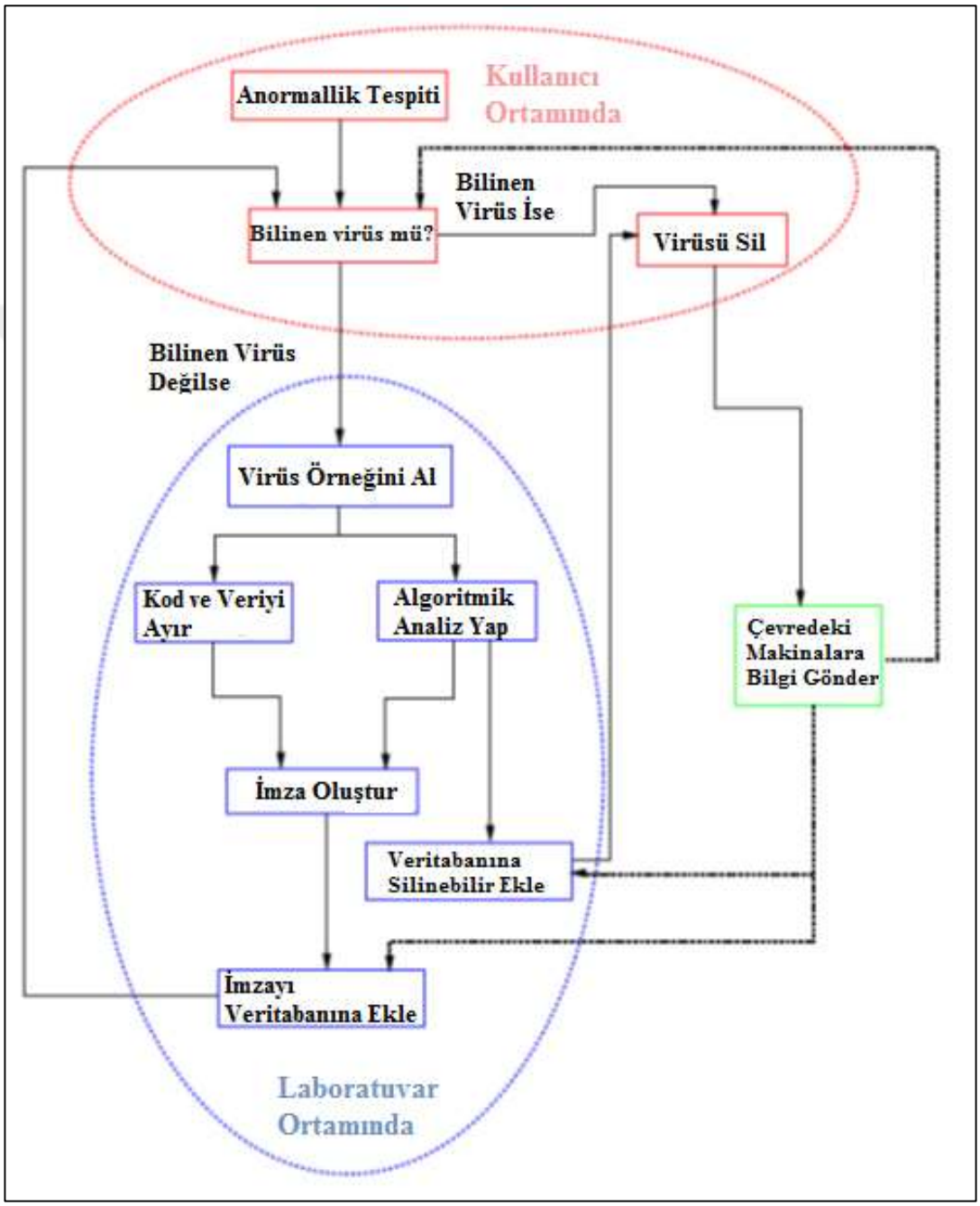


Şekil 2.1: Kötücül kod gizleme yöntemlerinin zamana göre gelişim grafiği.

Kötücül yazılımlar kendilerini gizlemek amacıyla Şekil 2.1: Kötücül kod gizleme yöntemlerinin zamana göre gelişim grafiğinde görülen şifreleme, oligomorfizm, polimorfizm ve metamorfizm yöntemlerin kullanmaktadırlar; bu yöntemler sayesinde geleneksel imza tabanlı kötücül yazılım tespit sistemleri tarafından tanımalarını zorlaştırmakta, hatta imkânsız hale getirmektedirler [2], [10], [11].

Geleneksel kötücül kod tespit sistemlerinde Şekil 2.2'de IBM tarafından geliştirilen ilk imza tabanlı anti virüs tespit sisteminin akış şemasında da görüleceği üzere kötücül kod yayıldığında kötücül kod araştırma uzmanları kodu analizi ederek bu kodun ikili yapısını kullanan bir algoritmayla bu koda ait kötücül kod imza oluşturur ve bunu kötücül kod imza veri tabanına ekleyerek veri tabanını günceller. Daha sonra bu kötücül kod sisteme yayılmak istediğinde ya da kendini farklı bilgisayarlara kopyaladığında, sistem oluşturulmuş kötücül kod imza veri tabanını kullanılarak kötücül kodu tespit eder ve etkisiz hale getirir. Bu uygulama hızlı ve etkili bir çözüm olmasına karşı kötücül kodların polimorfik ve metamorfik yöntemleri kullanarak ikili kod yapılarını değiştirmeleri ve böylece kendilerini tanınmaz hale getirmeleri nedeniyle polimorfik ve metamorfik yapıda olan kötücül kodları tespit

etmekte verimli olmaktadırlar [11]. Bununla birlikte günümüzde ticari kötücül kod tespit programlarının çoğu sezgisel yöntemlere göre daha hızlı ve daha yüksek doğruluk oranına sahip olması nedeniyle geleneksel imza tabanlı kötücül kod tespit tekniklerini kullanmaktadır [12].



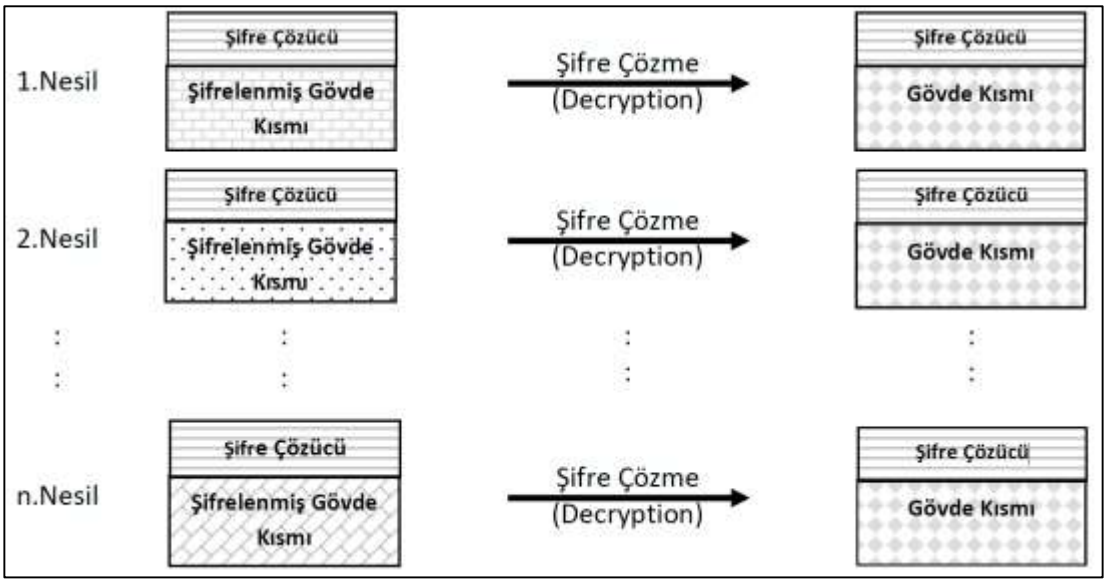
Şekil 2.2: İmza tabanlı kötücül kod tespit sistemlerinin çalışma yapısı IBM kötücül kod tespit sistemi örneği.

```
remnux@remnux:~/Desktop$ pehash torrentlocker.exe
File:   torrentlocker.exe
md5:    4cbeb9cd18604fd16d1fa4c4350d8ab0
sha1:   eaa4b62ef92e9413477728325be004a5a43d4c83
ssdeep:
6144:NZ3RRJKEj8Emqh9ijN9MMd30zW73KQgpc837NBH:LhRzwEm7jEjz+/g2o7fH
```

Şekil 2.3: Kötücül kod için imza örneği.

Günümüzde kötücül kodların tespiti ve tanımlanması için kullanılan imza algoritmaları genelde md5, sha2, sha256, sha512 gibi özet kod algoritmalarından oluşmaktadır. Şekil 2.3: Kötücül kod için imza örneğinde görüldüğü gibi Linux işletim sisteminde tanımlı pehash özet kod programı kullanılarak md5, sha1, ssdeep özet kod algoritmaları tarafından üretilmiş torrentlocker.exe kötücül koduna ait imzalar görülmektedir. Bununla birlikte kötücül kodların bünyesinde tanımlı ayırt edici byte dizileri de imza tabanlı kötücül kod sistemleri tarafından kullanılmaktadır. Ayrıca kötücül kodların ayrı ayrı bölümlerinin özet kodları da imza olarak kullanılmaktadır.

2.2. Şifreleme Yöntemi İle Kod Gizleme



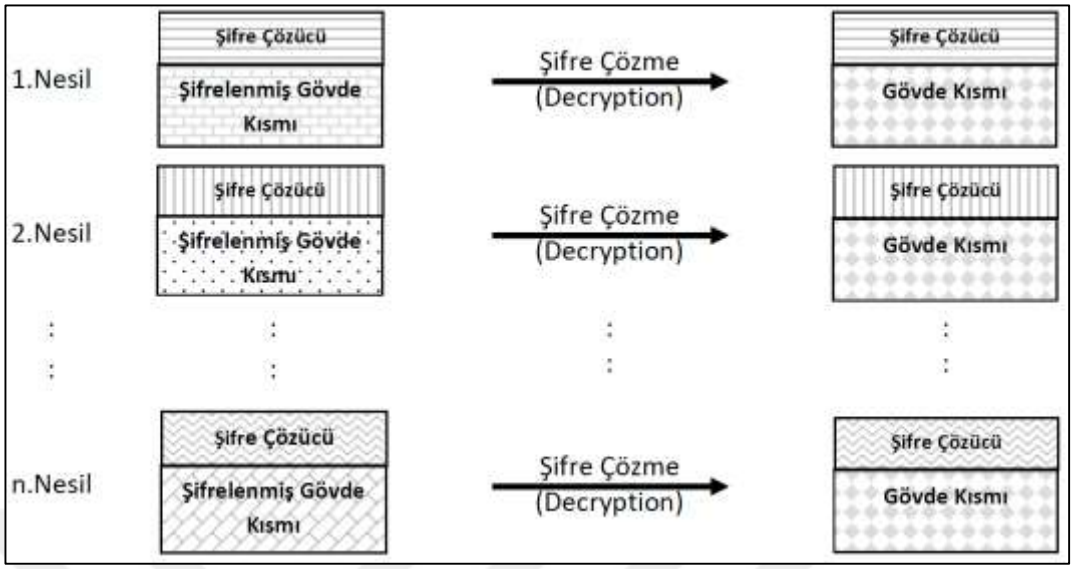
Şekil 2.4: Şifrelenmiş kötücül kodların yapısı.

Kötücül kodların kendilerini gizleme yöntemlerinden en kolayı kodun işlevsel kısmının şifrelenmesi yöntemidir. Bu yöntem DOS işletim sistemi zamanında beri kullanılmaktadır. Bu yöntemi kullanan ilk kötücül yazılımlardan biri Cascade'dir. [2] Bu yöntemin uygulamasında Şekil 2.4: Şifrelenmiş kötücül kodların yapısı görüleceği kötücül kod iki parçaya ayrılır. İlk kısım sabit bir şifre çözücünün bulunduğu kodun ilk bölümü. İkinci kısım ise kötücül kodun işlevsel kısmını oluşturan virüsün şifrelenmiş gövde kısmı. Şekilde de görüldüğü üzere kötücül kodun her bir neslinde şifre çözücü aynıdır. Bu şifre kod örneklerinden biri Tablo 2.1: Cascade kötücül yazılımının sabit şifre çözücü kodunda görülebilir. Bununla birlikte bu kötücül yazılımlar sabit ve değişmez şifre çözücü kullandıklarından dolayı tespit edilmeleri kolaydır. Kötücül yazılım tespit sistemleri durağan bir yöntemle kodu hiç çalıştırmadan kodun şifrelenmiş gövde kısmını çözmeden (decrypt), sadece şifre çözücüyü içeren örüntüye bakarak bu kötücül yazılımları tespit edebilmektedir. Günümüzde bu yöntem kullanılarak şifrelenmiş kötücül yazılımlar geleneksel anti-virüs sistemleri tarafından kolayca tespit edilebilmektedirler [2].

Tablo 2.1: Cascade kötücül yazılımının sabit şifre çözücü kodu.

```
lea     si, Start     ; position to decrypt (dynamically set)
mov     sp, 0682      ; lenght of encrypted body (1666 bytes)

Decrypt:
xor     [si],si       ; decryption key/counter 1
xor     [si],sp       ; decryption key counter 2
inc     si            ; increment one counter
dec     sp            ; decrement the other
jnz     Decrypt       ; loop until all bytes are decrypted

Start:
                      ; Encrypted/Decrypted Virus Body
```

2.3. Oligomorfik Yöntemle Kod Gizleme



Şekil 2.5: Oligomorfik kötücül kodların yapısı.

Oligomorfik kötücül kodlar, kötücül kod tespit sistemlerinden korunmak için şifreleme yöntemi ile kötücül kod gizleme bölümünde açıklanan sistemle birlikte aynı yöntemi kullanır; kötücül kodun yapısını şifreleyerek gizleme yöntemi. Bununla birlikte oligomorfik yöntemle kod gizlemenin şifreleme yönteminden ayrıldığı nokta, Şekil 2.5: Oligomorfik kötücül kodların yapısında da görüleceği üzere birden fazla şifre çözücü kullanması ve her yeni nesilde farklı şifre çözücü anahtar kullanmasıdır [2]. Yani yapısında sabit şifre çözücü bulunmaz, bunun yerine şifre çözücü anahtar kümesi üreten fonksiyon bulundurur. Her yeni nesilde yapısında bulunan yüzlerce farklı şifre çözücü anahtarlardan birini rastgele kullanır.

Örneğin W95/Memorial kötücül yazılımı bünyesinde 96 farklı şifre çözücü anahtar yapısı bulundurur. Tablo 2.2: W95/Memorial oligomorfik kötücül kodunun iki farklı nesilde şifre çözme anahtar yapısı örneğinde görüleceği üzere oligomorfik kötücül kod nesilden nesile anahtarını değiştirir. Böylelikle sabit şifre çözücü anahtar örüntüsüne göre kötücül kodların tespit edilmesi yöntemi pratik bir yöntem olmaktan çıkar.

Bununla birlikte oligomorfik sistemler sonlu sayıda şifre çözücü anahtar bulundururlar. Bu durum sınırlayıcı bir etkendir ve kodun tüm nesillerinin tahmin edilmesi ya da izlenmesine olanak sağlar. Her bir nesilde yeni anahtar kullanmasına karşı toplam kullanılan anahtar sayısı sınırlıdır. Bu nedenle oligomorfik kötücül kodlar

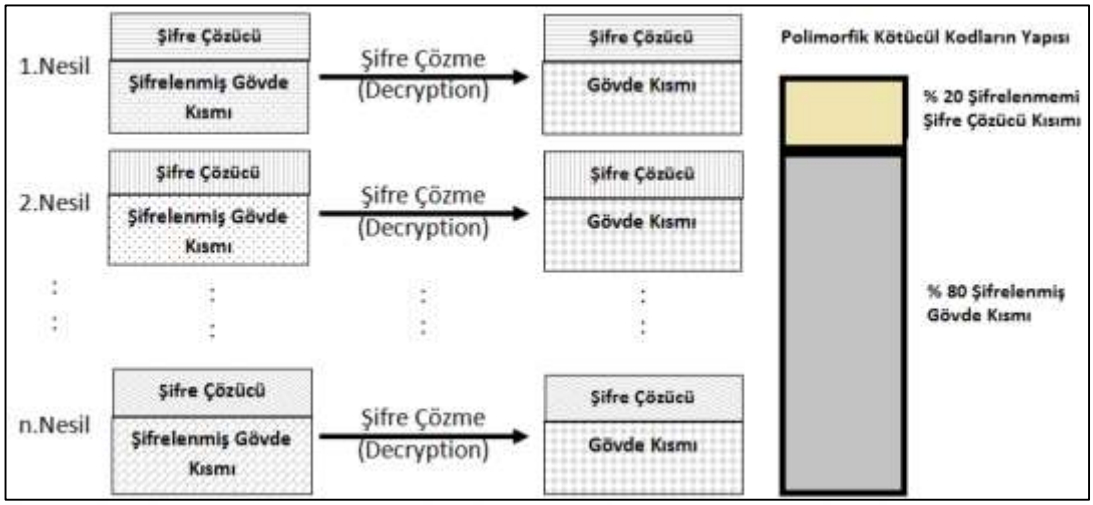
polinom zamanda tespit edilebilirler. Örneğin W95/Memorial kötücül koduna ait 96 adet kod, polinom zamanda denemeyle elde edilebilir. Daha sonra bu anahtarlar kullanılarak W95/Memorial kötücül kod sınıfına ait kod örneği en fazla 96 adet denemeyle tespit edilebilir.

Tablo 2.2: W95/Memorial oligomorfik kötücül kodunun 2 farklı nesilde şifre çözme anahtar yapısı.

1.Nesil	mov ebp,00405000h ; select base
	mov ecx,0550h ; this many bytes
	lea esi,[ebp+0000002E] ; offset of "Start"
	add ecx,[abp+00000029] ; plus this many bytes
	mov al,[ebp+0000002D] ; pick the first key
	Decrypt:
	nop ; junk
	nop ; junk
	xor [esi],al ; decrypt a byte
	inc esi ; next byte
	nop ; junk
	inc al ; slide the key
	dec ecx ; are there any more bytes to decrypt?
	jnz Decrypt ; until all bytes are decrypted
	jmp Start ; decryption done, execute body
	; Data area
	Start ; encrypted/decrypted virus body

2.Nesil	mov ecx,0550h ; this many bytes
	mov ebp,013BC000h ; select base
	lea esi,[ebp+0000002E] ; offset of "Start"
	add ecx,[abp+00000029] ; plus this many bytes
	mov al,[ebp+0000002D] ; pick the first key
	Decrypt:
	nop ; junk
	nop ; junk
	xor [esi],al ; decrypt a byte
	inc esi ; next byte
	nop ; junk
	inc al ; slide the key
	loop Decrypt ; until all bytes are decrypted
	jmp Start ; decryption done, execute body
	; Data area
	Start ; encrypted/decrypted virus body

2.4. Polimorfik Yöntemle Kod Gizleme



Şekil 2.6: Polimorfik kötücül kod yapısı ve kodun evrimleşmesi.

Kötücül kodun şifrelenerek gizlenmesi yöntemini kullanan bir diğer teknik olan polimorfik yöntemle kod gizlemede ise oligomorfik yöntemden farklı olarak sonsuz sayıda şifre çözücü anahtar bulundurulur. Bunu sağlamak için Win32/Coko gibi bazı kötücül kodlar çok katmanlı şifreleme tekniği kullanırken, Win32/Crypto gibi bazıları da Random Decryption Algorithm (RDA) tabanlı şifre çözücü anahtar tekniği kullanır. Böylece kötücül kodlar çok katmanlı ve çok şifreli yapısı sayesinde polimorfik kod gizleme yöntemi ile kaba saldırı (brute-force) ataklarına karşı dirençli bir mekanizma oluşturmuş olur. Polimorfik yöntemi ilk kullanan 1260 kötücül yazılımının şifre çözücü anahtar yapısı Tablo 2.3'de gösterilmiştir.

Polimorfik yöntemle kod gizleme tekniğinde Şekil 2.6: Polimorfik kötücül kod yapısı ve kodun evrimleşmesinde gösterildiği gibi kötücül yazılımın yaklaşık %20'lik kısmı şifrelenmemiş olarak tutulur ve diğer %80'lik kötücül yazılımın asıl işi yapan kısmı şifrelenmiş olarak bulundurulur. Yazılımın şifrelenmiş gövde kısmını çözecek şifre çözücü anahtarın bulundurulduğu %20'lik şifrelenmemiş kısım polimorfik yöntemin katmanlı ve sonsuz şifreleme yöntemine sahip yapısı sayesinde her nesilde farklı olur. Program çalıştırıldığında şifreli gövde kısmı şifre çözücü anahtar sayesinde belleğe çözülerek kod aktif hale getirilip çalıştırılır.

Polimorfik yöntem kullanan kötücül yazılımlarda yapılan sezgisel analizlerde sık sık verimsiz rastgelelik fonksiyonları kullanıldığı gözlenmiştir [2]. Bu durum durağan kötücül yazılım tespit sistemlerinin polimorfik kötücül yazılımları tespit

etmelerinde kolaylık sağlamaktadır. Hatta kötücül kodun yapısında bulunan bazı özel karakterler (wildcard string'ler) kullanılarak bile başarım oranı yüksek polimorfik kötücül kod tespitleri yapılabilmektedir [2]. Bunun yanı sıra polimorfik kötücül kodlar çalıştırıldıklarında bellek üzerine çözdükleri kötücül yazılımın gövde kısmı Şekil 2.6: Polimorfik kötücül kod yapısı ve kodun evrimleşmesinde görüldüğü gibi her nesilde aynı olduğundan dolayı dinamik analiz yöntemi kullanılarak, kum havuzları ve ya öykünücüler yardımıyla kötücül kodun sabit, her nesilde aynı kalan gövde kısmının eşleştirmesiyle polimorfik kötücül kodlar tespit edilebilmektedir. Buna karşı dinamik yöntemle analizin çeşitli dezavantajları Bölüm 3.2.5 dinamik analizle kötücül kod tespitinin dezavantajları bölümünde detaylı olarak açıklanmıştır.

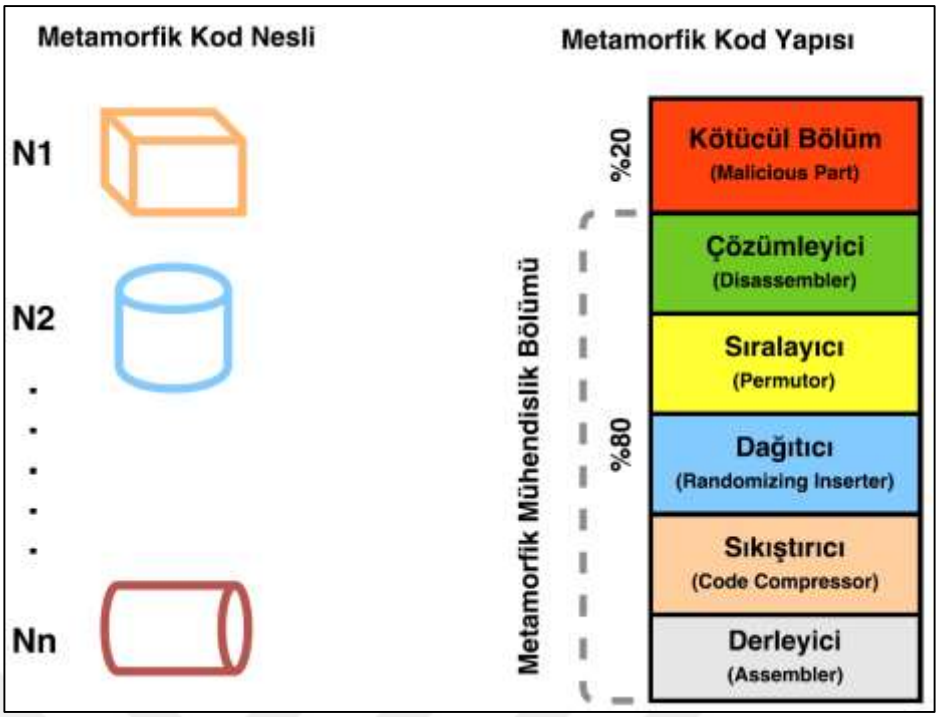
Tablo 2.3: 1260 polimorfik kötücül kodunun şifre çözme anahtar yapısı.

```
; Group 1 - Prolog Instructions

inc si                          ; optional, variable junk
mov ax,0E9B                     ; set key 1
clc                             ; optional variable junk
mov di,012A                     ; offset of Start
nop                             ; optional, variable junk
mov cx,0571                     ; this many bytes - key 2

; Group 2 - Decryption Instructions

Decrypt:
xor [di],cx                     ; decrypt first word with key 2
sub bx,dx                       ; optional, variable junk
xor bx,cx                       ; optional, variable junk
sub bx,ax                       ; optional, variable junk
sub bx,cx                       ; optional, variable junk
nop                             ; non-optional junk
xor dx,cx                       ; optional, variable junk
xor [di],ax                     ; decrypt first word with key 1

; Group 3 - Decryption Instructions

inc di                          ; next byte
nop                             ; non-optional junk
clc                             ; optional, variable junk
inc ax                          ; slide key 1
loop Decrypt                    ; until all bytes are decrypted - slide key 2

;random padding

Start:                          ; encrypted/decrypted virus body
```

Şifreleme, oligomorfizm ve polimorfizm yöntemlerinde kötücül kodun asıl işlevi gerçekleştiren kötücül bölümü olan yaklaşık %80'lik kısmı şifrelenmiş olarak tutulur, şifreli kısmı çözecek bilgilerin tutulduğu %20'lik kısım ise şifrelenmemiş olarak bulunur. Program çalıştırıldığında şifrelenmiş bu kısım bellek üzerine çözülerek program çalışır. Şifreleme ve oligomorfizm yöntemlerinde kodu çözecek anahtar kötücül kodun içerisinde yer aldığı için polinom zamanda denemeyle anahtarlar elde edilebilmiş ve kötücül kodlar bu anahtar takip edilerek tespit edilebilmiştir. Şifreleme ve oligomorfizm yöntemleri kötücül kodların kendilerini gizlemek için kullandıkları en naif yöntemlerdir. Polimorfik kötücül kod gizleme yönteminde ise teorik olarak sonsuz sayıda rastgele anahtar üretecek birden fazla şifreleme algoritması yer alması nedeniyle kendisinden sınırsız sayıda şifrelenmiş tür oluşturabilecek bir yapı tasarlanmıştır. Bununla birlikte kötücül kodlar çalıştırıldıklarında bellek üzerinde çözülmüş halleri aynı olacağından dinamik analiz yöntemi ve kum havuzları yardımıyla polimorfik kötücül kodlar kolaylıkla tespit edilebilmişlerdir. Polimorfik yapıdaki kötücül kodların uygulamasında ortaya çıkan problemler, dinamik analiz yöntemi ile tespit edilebilmelerinin getirdiği dezavantajlar nedeniyle kötücül kod geliştiricileri tarafından metamorfik yapıdaki kodlar popüler ve yaygın kullanılır hale gelmiştir.

2.5. Metamorfik Yöntemle Kod Gizleme

Metamorfik kötücül kodlar polimorfik kötücül kodlardan farklı olarak Şekil 2.7 metamorfik kötücül kod anatomisi ve kod neslinin sembolik gösteriminde görüleceği üzere kodun %20'lik çalışan kısmı şifrelenmekte, bununla birlikte %80'lik kısım ise metamorfik kod üretim ve dönüştürme araçlarından oluşmaktadır. Bu araçlar sayesinde metamorfik yapıdaki kötücül kodlar her bir nesilde koda gövdenin şifrelenmesinin yanı sıra kodun yapısını değiştirerek kodun tanınmasını engelleyici özellikler ekleyebilmektedirler. Bu nedenle Şekil 2.7'de sembolik olarak gösterildiği gibi şifrelenmiş gövde kısmı çözüldüğünde her bir nesilde yapısı, büyüklüğü, tipi farklı kodlar ortaya çıkmaktadır. Kod yapısının farklı olmasının yanı sıra bu durum kodun işlevselliğini ve davranışını değiştirmemektedir. Koda eklenen işlevsellikler genelde şaşırtma amaçlı, iş yapmayan ya da mevcut kod bloklarının sırasının değiştirilmesi şeklindedir. Bu yöntemlerden bazılar aşağıda açıklanmıştır;



Şekil 2.7: Metamorfik kötücül kod anatomisi ve kod neslinin sembolik gösterimi.

2.5.1. İşlevsiz Döngü ve Komut Eklenmesi

Kod analizini zorlaştırmak ve kodun yapısını değiştirmek için kaynak kodun içerisine boş döngü ya da gerçek anlamda işlevselliği olmayan boş komutlar eklenir. Bu amaçla kullanılan Tablo 2.4: İşlevsiz ölü kod örneklerinde görülen komutlar işlemcinin kayıt sisteminde ve bellekte herhangi bir değişiklik yapmamaktadır [11]. Bu nedenle bunlara işlevsiz kod (No Operation NOP) denir. Bununla birlikte işlemcinin bayrak (flag) durumunda değişikliğe neden olurlar. Örneğin bir değişkene 0 eklediğimizde, işlemci tarafından işlem yapılır fakat işlem sonucu değişmez.

Tablo 2.4: İşlevsiz ölü kod örnekleri.

Komut		**İşlem**
ADD	reg, 0	reg ←reg+0
MOV	reg, reg	reg ←reg
OR	reg,0	reg ←reg | 0
AND	reg, -1	reg ←reg& -1

Tablo 2.5: Geri döndürülebilir ölü kod örnekleri.

Komut		Yorum
PUSH	**CX**	AX değeri önce yığına eklenir. Daha sonra AX üzerinde veya yığın belleğinde herhangi bir değişiklik yapılmadan önce AX tekrar aynı konumuna geri döndürülür.
…	**…**	
POP	**CX**	
INC	**AX**	DX değeri önce 14 artırılır daha sonra DX herhangi bir işlemde kullanılmadan eski değerine geri döndürülür.
…	**…**	
SUB	**AX,1**	

Bu yöntemin bir diğer çeşidi ölü kodlar ekleyip çıkarmaktır. Örneğin Tablo 2.5: Geri döndürülebilir ölü kod örneklerinde de görüldüğü gibi kuyruğa herhangi bir değişken eklenir ve bu değişkenle ilgili herhangi bir işlem yapılmadan tekrar kuyruktan çıkarılırsa bu durum program işleyişinde zaman kaybından başka bir değişikliğe neden olmaz [11]. Bununla birlikte yapılan bu değişiklik kodun imzası üzerinde belirgin bir değişikliğe neden olur. Bu yöntemin farklı tekniklerle aynı mantık çerçevesinde uygulanması ile eklenen kodun programın işleyişinde herhangi bir değişiklik yapmaması nedeniyle ölü kod ekleme yöntemi olarak adlandırılmaktadır. W32.Evol kötücül kodu tarafından ölü kod ekleme tekniklerinin kötücül kodun imzası üzerindeki yaptığı değişim Tablo 2.6: W32.Evol kötücül yazılımının ölü kod yöntemi ile metamorfik değişiminde gösterilmiştir [2].

Tablo 2.6: W32.Evol kötücül yazılımının ölü kod yöntemi ile metamorfik değişimi.

W32.Evol orijinal kötücül kodu		**W32.Evol metamorfik kötücül kodu**	
İkili kod dizisi	Assembly Kodu	İkili kod dizisi	Assembly Kodu

C7060F000055
C746048bec5151
	mov [esi], 5500000Fh
	mov [esi+0004],
		BF0F000055
		893E
		5F
		52
		B640
		BA8BEC515
		53
		8BDA
		895E04
			mov edi, 5500000Fh
			mov [esi], edi
			pop edi
			push edx
			mov dh, 40
			mov edx, 5151EC8Bh
			push ebx
			mov ebx,ebx
			mov [esi+0004], ebx

imza : C7060F000055C746048BEC5151		imza : BF0F000055893E5F52B640BA8BEC5151538BDA895E04	

2.5.2. Komutları Eşdeğeri İle Değiştirmek

Bu teknikte işlemci üzerinde aynı anlama gelen, aynı işlevselliğe sahip kodların yerleri değiştirildiğinde programın işlevi açısından sonuç aynı olacağından işlemciye ait aynı işlevi gören komutlar eş değerleriyle değiştirilerek programın işlevi değiştirilmeden imzasında değişiklik yapılır. Örneğin "mov eax,0", "xor ax,eax", "and eax,0", "sub eax,eax" komutlarının her biri eax işaretçisini 0 olarak işaretler [13]. Böylece hepsi aynı sonucu farklı komutlarla üretir. Bu nedenle bu komut dizileri birbirilerinin yerine kullanılması programın çalışması açısından bir değişikliğe neden olmaz. Tablo 2.7: W95.Bistro kötücül yazılımının komutları eşdeğeri ile değiştirme yöntemi ile metamorfik değişim örneğinde W95.Bistro kötücül kodunun "test esi, esi" kodu yerine "or esi, esi", "or edi, edi" yerine "test edi, edi" ve yine benzer şekilde "move bp, esp" komutu yerine "push esp", "pop ebp" eş değer komutları kullanılarak kodun mutasyona uğratılmış halinde görüldüğü üzere programın işleyişinde değişiklik olmamasına rağmen kodun imzası değişikliğe uğramıştır [2], [13]. Özetle komutlar değişmiş olmasına rağmen kodun işlevselliği aynı kalmıştır. Bununla birlikte yapılan her değişiklik kodun imzasında değişikliğe neden olmuştur.

Tablo 2.7: W95.Bistro kötücül yazılımının komutları eşdeğeri ile değiştirme yöntemi ile metamorfik değişim örneği.

W95.Bistro orijinal kötücül kodu		**W95.Bistro metamorfik kötücül kodu**	
İkili kod dizisi	Assembly Kodu	İkili kod dizisi	Assembly Kodu
55 8BEC 8B7608 85F6 743B 8B7E0C 09FF 7434 31D2	push ebp mov ebp, esp mov esi, dwordptr [ebp + 08] test esi, esi je 401045 mov edi, dwordptrr[ebp + 0c] or edi,edi je 401045 xor edx, edx	55 54 5D 8B7608 09F6 743B 8B7E0C 85FF 7434 28D2	push ebp push esp pop ebp mov esi,dwordptr [ebp + 08] or esi, esi je 401045 mov edi, dwordptr [bp +0c] test edi, edi je 401045 sub edx, edx
imza: 558BEC8B760885F6743B8B7E0C09FF743431D2		imza: 55545D8B760809F6743B8B7E0C85FF743428D2	

2.5.3. Komutların Sırasının Değiştirilmesi

Tablo 2.8: Metamorfik yöntemlerde komut sırasının değiştirilmesi örneği.

Kod Düzeni 1	Kod Düzeni 2

mov eax,0F
push ecx
add esi,ebx
	add esi,ebx
	mov eax,0F
	push ecx

Birçok programda programın işlevselliği bozulmadan bağımsız değişkenlerin sırası değiştirilebilir. Bununla birlikte bu basit değişiklik kodun ikili yapısını değiştirerek imzasında farklılaşmaya neden olur. Böylece aynı işlevselliğe sahip kodun imza tabanlı kötücül kod tanıma sistemleri tarafından farklı iki program gibi algılanmasına neden olur. Tablo 2.8: Metamorfik yöntemlerde komut sırasının değiştirilmesi örneğinde görüldüğü gibi birbirinden bağımsız kodların sıralamaları kodun anlamı değişikliğe uğramadan değiştirilebilir [11]. Bu durum programın işlevselliğinde herhangi bir değişime neden olmaz.

2.5.4. Değişkenlerin Değiştirilmesi

Tablo 2.9: W95.Regswap kötücül yazılımının değişkenleri değiştirme yöntemi ile metamorfik değişim örneği.

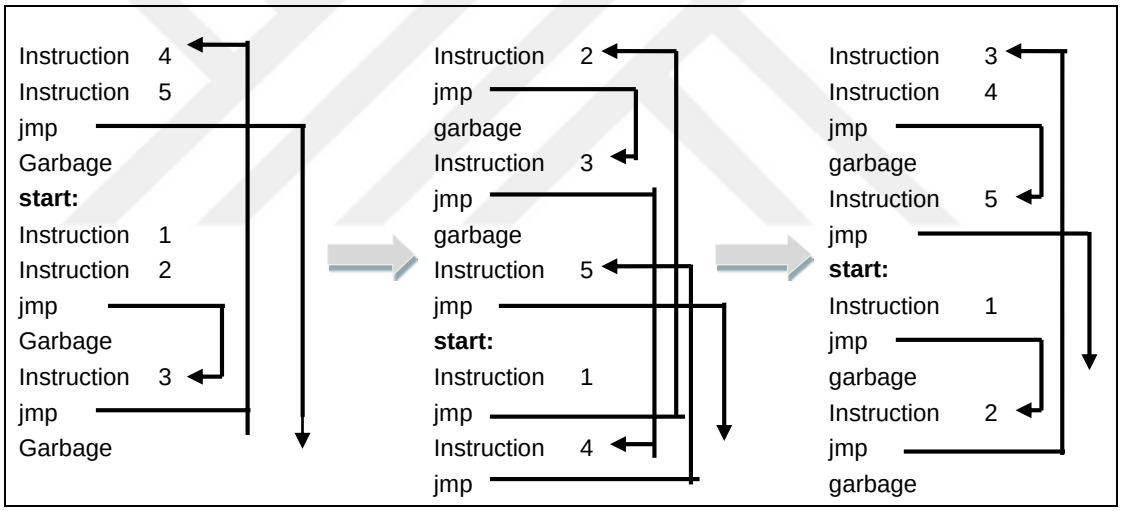
W95.Regswap orijinal kötücül kodu		W95.Regswap metamorfik kötücül kodu	
İkili kod dizisi	Assembly Kodu	İkili kod dizisi	Assembly Kodu

5A
BF04000000
8BF5
B80C000000
81C28800000
8B1A
 	
899C8618110
000
	pop edx
	mov edi
	mov esi, ebp
	mov eax, 000Ch
	add edx, 0088h
	add ebx, [edx]
	
	mov eax*4+0000111
	8],ebx
		58
		BB04000000
		8BD5
		BF0C000000
		81C08800000
		8B30
		
		89B4BA18110
		000
			pop eax
			mov ebx, 0004h
			mov edx, ebp
			mov edi, 000Ch
			add eax, 0088h
			mov esi, [eax]
			
			mov [edx+edi*4+00001
			118], esi

İmza: 5ABF040000008BF5B80C00000081C2 880000008B1A899C8618110000		İmza 58BB040000008BD5BF0C00000081C08800 00008B3089	

Bu yöntemde kötücül yazılımın farklı nesillerinde farklı bellek değişkenlerini kullanılarak kodun imzası değiştirilir. Diğer yöntemlerde olduğu gibi bu yöntemde de kodun işlevselliği değişmediği halde ikili kod yapısı değişime uğramaktadır. Bununla birlikte bu yöntem diğerlerine göre kötücül kodun imzası üzerindeki yaptığı değişiklik daha azdır. Tablo 2.9: W95.Regswap kötücül yazılımının değişkenleri değiştirme yöntemi ile metamorfik değişim örneğinde W95.Regswap kötücül yazılımının "edx", "edi" ve "esi" bellek değişkenleri yerine "eax", "ebx" ve "edx" bellek değişkenleri kullanarak değişime uğramış farklı versiyonlarına ait imzalarında renklendirilmiş aynı kalan bölümlerinden de anlaşılacağı üzere imzanın büyük bölümü aynı kalmıştır [13]. Bu yöntem tek başına karmaşıklığı yüksek bir çözüm sunmaz bu nedenle diğer yöntemlerle bir arada kullanılarak etkisi daha artırılabilir.

2.5.5. Kontrol Kod Bloklarının Yerinin Değiştirilmesi



Şekil 2.8: ZPerm kötücül kodunun kontrol kod bloklarının yerinin değiştirilmesi.

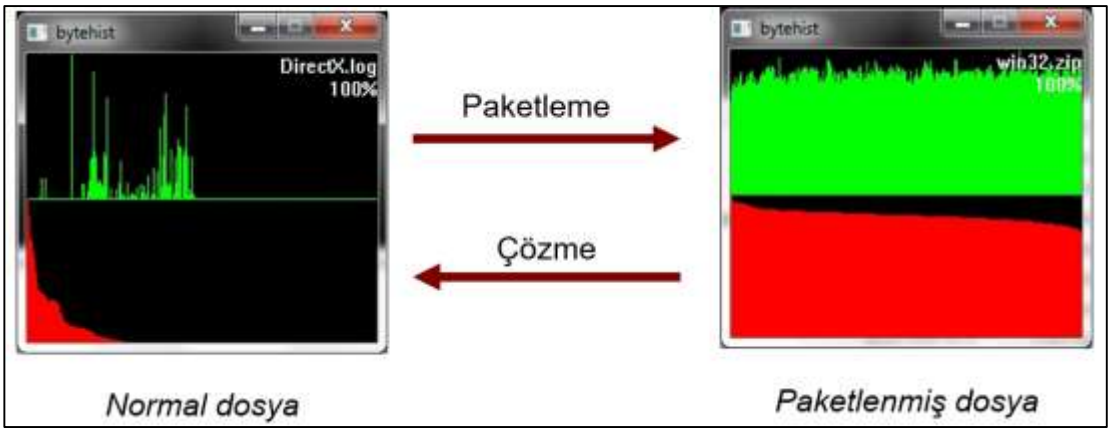
Programlar her ne kadar tanımlanmış komut listesi üzerinde sıralı bir şekilde çalışma prensibine sahip olsa da, işlemci üzerinde tanımlı "jump", "move" vb. döngü ve kontrol komutları sayesinde çalışma sırası olarak bellek üzerinde tanımlı komut listesinin farklı adımlarına atlayarak programın çalışmasına bu noktadan devam edebilir. Programa ait çalışma sırasındaki bu değişiklik komut sırasında geriye veya ileriye doğru olabilir. Bu durumun doğası gereği kontrol komutları arasında kalan komut satırları çalışma sırası değişmeyecek şekilde birbirinden bağımsız bloklar halinde düşünebilir. Bu blokların farklı dizilime sahip olmaları programın akışında

herhangi bir değişikliğe neden olmaz. Kontrol bloklarının program akış listesi içerisinde birbirine referans verdikleri sürece sıralı bir şekilde tutulmak zorunda değildir. Kötücül kod yazarları bu fikirden hareketle programa ait kontrol bloklarını kod listesi içerisinde farklı dizilimlerini kullanarak kötücül kodun her sürümünde farklı bir imza oluşturabilmesini sağlamıştır. Örneğin Zperm kötücül kodunun Şekil 2.8: ZPerm kötücül kodunun kontrol kod bloklarının yerinin değiştirilmesi örneğinde görüldüğü gibi farklı üç versiyonunda kod yapısı ve sırası değiştirilmiştir. Bununla birlikte kod bloklarının başlangıç ve bitiş noktalarını birbirine referans verdiği için bloklarının çalışma sırası aynı kalmıştır [2]. Böylede programın her bir sürümde işlevselliği aynı kalmış, imzası değişikliğe uğramıştır. En etkili metamorfik değişim yöntemlerindendir. Değişim belli bir kod bloğunda ya da belli komutlarda yapılabilir.

2.5.6. Kodun Paketlenmesi

Kodun paketlenmesi fikri, kod içerisindeki benzer yapıların birbirine referans vererek tutulmasıdır. Böylece kod parçası disk üzerinde daha küçük yer kaplayacak ve daha hızlı transfer edilebilecektir. Bu anlamda en bilindik algoritma Huffman Encoding algoritmasıdır ve en yaygın kullanılan paketleme algoritmaları olan zip, tar, rar programlarının kullandığı yöntemler temelde bu algoritmaya dayanır.

Günlük kullanım içerisinde yaygın olarak kullanılan bu teknik kötücül kodlar tarafından kendilerini kötücül kod tespit sistemlerine karşı kamufle etmek amacıyla da sıkça kullanılmaktadır. Bu yöntemle, kötücül kod paketleme yazılımlarını kullanarak kodu paketleyip ikili kod yapısını değiştirerek imza tabanlı kötücül kod sistemlerinin atlatabilmektedir. Bu maksatla kötücül kodlar tarafından kullanılan en yaygın algoritma UPX'dir. Paketlenmiş ve paketlenmemiş kodların yapısı birbirinden farklıdır. Bununla birlikte kötücül kodlar paketlendiğinde kötücül kodların işlem kodları homojene yakın bir dağılım gösterir ki bu da normalde karşılaşılacak bir durum değildir. Şekil 2.9: Kötücül kodun paketlenmiş ve paketlenmemiş kodlarının byte histogramlarında görüleceği üzere paketlenmiş ve paketlenmemiş kodların byte histogramı birbirinden oldukça farklıdır. Bu nedenle kötücül kodun paketlendiği işlem koduna ait dağılım üzerinde tespit edilebilir. Bununla birlikte bu durum paketlenmiş kodun kötücül olup olmadığı ve çözülebileceği anlamına gelmemektedir. Örneğin zararsız bir kod zip formatın paketlenerek dağıtılabilir. Bununla birlikte dağıtılan kod zararlı bir kötücül kod örneği de olabilir.



Şekil 2.9: Kötücül kodun paketlenmiş ve paketlenmemiş kodlarının byte histogramları.

Metamorfik kötücül kodlar açıklanan işlevsiz döngü ve komut eklenmesi, komutların eş değeri ile değiştirilmesi, birbirinden bağımsız komutların sırasının değiştirilmesi, değişkenlerinin değiştirilmesi, kontrol bloklarının yerlerinin değiştirilmesi yöntemlerini kullanarak işlevselliklerini koruyacak şekilde yapılarını değiştirebilmekte, geleneksel kötücül kod tespit sistemlerini atlatabilmektedirler. Son zamanlarda yapısını değiştiren polimorfik ve metamorfik kötücül kodların tespitine yönelik Bölüm 3 İlgili Çalışmalar kısmında açıklandığı üzere bilimsel çalışmalar yapılmaktadır. Bununla birlikte kötücül kodların gerçek zamanda tespiti için, hız ve başarım anlamında henüz ideal bir yöntem geliştirmemiş olup hala yeni çalışmalar yapılmakta ve özgün çözümler üretilmeye açık bir alan olmayı sürdürmektedir. Özellikle yöntemlerin tanıma performansları ve hızları başlıca çalışma alanlarındandır. Bu anlamda başkalaşan kötücül yazılımların gerçek zamanda tanınması en kritik çalışma alanıdır. Bu sayede kötücül yazılımın etkisi görülmeden tespiti ve etkisizleştirilmesi sağlanmış olabilecektir.

2.5.7. Metamorfik Kod Dönüştürme Yönteminin Faydaları

Metamorfik yöntemler hatalı programlama veya program tasarımı sonucunda oluşan açıklıkların kullanılarak belleğin taşırılması ve programın komutları arasına dışarıdan komut eklenmesi ile yapılan “buffer over flow” saldırılarına karşı koruma amacıyla kullanılabilir. Bellek taşması hataları bir kez tespit edildiğinde, farklı ortamlarda aynı girdi verileriyle tekrar tekrar oluşacaktır. Bu durumda bu zafiyetin istismar edilerek saldırı amaçlı kullanılmasına imkân sağlayacaktır. Günümüzde

bellek taşması zafiyeti en sık kullanılan saldırı yöntemleri arasındadır. Bununla birlikte metamorfik yöntemler “buffer over flow” gibi yazılımın her kopyasında aynı hatayı veren açıklar için koruyucu yararlı bir yöntem olarak kullanılabilir. Metamorfik yöntemle kod yapısı değiştirildiğinde programın farklı sürümlerinde farklı yapı ve büyüklükte programlar oluşacağı için, her bir programda hatanın referans verdiği bellek açıklığı farklılaşacaktır [14]. Bu nedenle metamorfik yöntem kullanıldığında açıklık programda potansiyel olarak var olmaya devam edecektir; bununla birlikte yazılımın bir kopyası için kullanılan açıklık diğeri için “otomatik” olarak kullanılamayacaktır. Kötücül kod tasarımcıları için programın her bir kopyası için ayrı ayrı bellek taşması analizi yapılmasına neden olacaktır. Bu durumda bir kere hatayı bul ve her defasında kullan taktiğini bertaraf edilecektir. Üstelik bu özellik programa en az miktarda metamorfik yöntem uygulanarak kazandırılabilmektedir.

3. İLGİLİ ÇALIŞMALAR

3.1. Durağan Analiz Yöntemi İle Kötücül Kod Tespiti

Durağan kötücül kod analiz tekniğini kullanan yöntemlerde assembly ve program kodları analiz objesi olarak dikkate alınır. Analiz işlemi program çalıştırılmadan sadece bu kodlar üzerinde yapılır. Bu yöntem aynı zamanda kaynak kod analizi olarak da bilinir. Diğer bir teknik olan dinamik yöntemden en önemli farkı kötücül kodun çalıştırılmadan analizinin yapılmasıdır.

Durağan analiz işleminde genel olarak önce ikili kod disassemble edilip ikili kod yapısı assembler komutlarına dönüştürülür. Daha sonra bu assemly kodları üzerinden kontrol ve veri akış analiz teknikleri ile kötücül kod tespiti yapılır. Durağan analiz tekniği ile kötücül kod tespit işleminde elimizde kötücül kodun işlem kodu (OpCode), derleyici modeli, işlem çağrıları, üst veri bilgileri, iki kod dizileri vb. gibi zengin bir kaynak yer alır.

Çizge tabanlı analiz, n-gram analiz, özvektör analizi, gizli markov zinciri (Hidden Markov Model Analiz) vb. yöntemler durağan tespit sistemleri için kullanılan analiz yöntemleri arasındadır. En sık çizge tabanlı analiz ve n-gram analiz yöntemi tercih edilmesi ile birlikte son yıllarda makine öğrenmesi yöntemleri yüksek başarım oranları elde edilmiştir. Konuyla ilgili çalışmalar aşağıda açıklanmıştır.

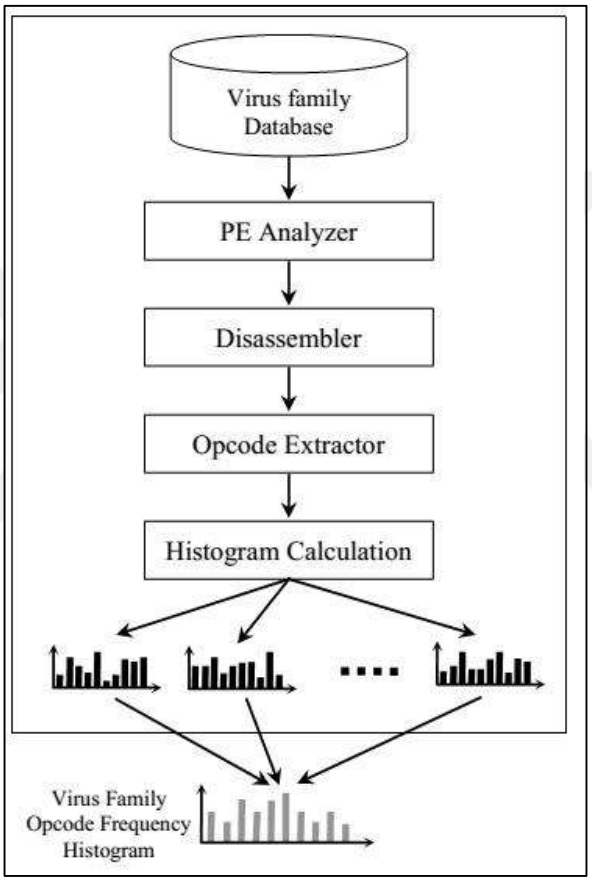
3.1.1. Durağan Analiz Yöntemini Kullanan İlgili Çalışmalar

Klasik durağan özellik kullanarak kötücül yazılım tanıma yöntemi M. G. Schultz ve arkadaşları tarafından [15] ‘te önerilmiştir. T. Abou-Assaleh ve arkadaşları tarafından kötücül koda ait byte dizilerinin n-gram ağırlıkları hesaplanarak tanıma gerçekleştiren yöntem geliştirilmiştir [16]. Zhang ve arkadaşları da çalışmalarında kötücül kod özelliklerini tespit etmek için n-gram analizi fikri kullanarak başarılı sonuçlar elde etmiştir [17]. Yine n-gram hesaplamayı kullanan ve başarılı sonuçlar elde eden durağan analiz tekniğine dayalı yöntemler içinde Ye ve arkadaşlarınınki gösterilebilir [18].

Wing ve arkadaşları tarafından metamorfik kötücül kodların işlem kodları (OpCode), örüntü analiz yöntemlerinden olan gizli markov zinciri (Hidden Markov

Model (HMM)) kullanılarak makine öğrenmesi yaklaşımıyla analiz edilmesiyle benzerlik tabanlı kötücül kod tespit yöntemi geliştirildi [4].

Durağan analiz yönteminde işlem kodları (OpCode) kullanarak kötücül kod tespiti yapan bir diğer yöntemde Rad ve arkadaşları tarafından geliştirilen kötücül kodun kullandığı işlem kodlarının histogramlarının çıkartılarak kötücül kodların sınıflandırılması yöntemidir [19]. Çalışmada sadece Windows işletim sistemi üzerinde tanımlı PE (Portable Executable) programlar analiz edilmektedir.



Şekil 3.1: Histogram analizi ile kötücül kod tespit yöntemi.

Şekil 3.1: Histogram analizi ile kötücül kod tespit yönteminde [19] akış şeması gösterilen çalışmada ilk olarak PE formatında dosyalarının ikili kod yapısını çözümlenip (disassembly) programın işlem kodlarını elde ediltikten sonra, bu kodların her birinin sadece ilk sekizlik kodu (byte) alınmaktadır. Çalışmada bu şekilde 256 işlem kod tipi belirlenmiştir. Daha sonra her bir kötücül kod için x-ekseni bu 256 adet işlem kodunu (OpCode) temsil edecek şekilde 1,….,256 arası sayılardan ve y-ekseninde kullanılan işlem kod (OpCode) sayısını verecek şekilde histogramlar

oluşturulmaktadır. Son olarak oluşturulan aynı aileye ait kötücül kod türlerinin histogramları birleştirilip ortalaması alınarak o kötücül kod türü için genel ortalama bir kötücül kod histogramı oluşturulmuştur. Oluşturulan bu histogram virüs sınıfına ait işlem kod sıklık özniteliği temsil histogramı olarak adlandırılmaktadır. Geliştirilen yöntemin testi için bir önceki yöntemde vurgulanan tüm gelişmiş metamorfik kod gizleme yöntemlerini bünyesinde barındıran NGVCK metamorfik kötücül kodunun 200 farklı türü üzerinde işlem yapıldı. Geliştirilen yöntemde sınıflandırma histogramların farklılığı üzerinden yapılmaktadır. Eğer programın farklılık değeri eşik değerinden az ise program o kötücül kod ailesine aittir denilmektedir. Yapılan çalışmada fark metriği olarak Minkows-form fark metriği kullanılmıştır [19]. Çalışmada 40 iyi huylu, 40 NGVCK kötücül kod ailesine ait ve 20 tane de diğer metamorfik kötücül kod ailelerine ait toplam 100 kötücül kod üzerinde testler yapılmış, testlerde sonucunda %100 kötücül kod tespit başarımı ile %0 yanlış pozitif değer elde edilmiştir. Geliştirilen yöntemin avantajı uygulama basitliği ve yüksek doğruluk oranı olarak gözlenirken test veri kümesinin küçüklüğü ve yöntemin gerçek zamanlı olarak uygulanabilirliğindeki zorluklar çalışmanın dezavantajı olarak ön plana çıkmaktadır. Benzer şekilde fark metriği olarak kosinüs benzerlik teoremini kullanan çalışmalarda da yaklaşık sonuçlar elde edilmiştir [20], [21].

Armoun S. E. ve Hashemi'de kötücül kodların işlem kodlarının, otomata yapısında formlara dönüştürülerek standardize (normalization) edilmesi ve bu sayede farklı bir uzaya transfer edilerek tespit edilebilmesine dayalı genel bir yaklaşım önermişlerdir. İlgili çalışmada öncelikle gizleme yöntemlerinin her biri Augmented DFA yapısında otomata formlarına dönüştürülüp, bu form kullanarak kaynak kod içerisinde yer alan gizleme kodlarının tespiti sağlanmıştır. Kaynak kod içerisinde yer alan bu gizleme kodlarının elenmesiyle kötücül yazılımın geleneksel kötücül kod tespit sistemleri tarafından tespit edebilir standart hale dönüşmesi sağlanmıştır [22].

Durağan analiz tekniklerinden olan ve çalıştırılabilir kodlarda değerlerin değişim ve yayılımının izini süren Balakrishnan tarafından ortaya atılan değer kümesi analizi yaklaşımı Leder ve arkadaşları tarafından temel alınarak çalıştırılabilir kodda yer alan değerlerin karakteristik kümesini ortaya çıkararak kötücül yazılımların sınıflandırılması ve gerçek zamanlı taramasını yapabilen yöntem geliştirilmiştir [23].

Runwal metamorfik kötü amaçlı yazılım tespiti için kötücül kodların işlem kodlarını çizge üzerine transfer ederek burada çizge tabanlı benzerlik üzerinden kötücül kodların tespitini yapan etkili bir yöntem önermiştir [24].

Benzer şekilde Victoria Üniversitesi'nde Alam S. ve arkadaşları da metamorfik yapıdaki kötücül kodların işlem kodları için MAIL (Malware Analysis Intermated Language) isimli bir ara dil yapısı önermiştir [25]. MAIL dili aracılığı ile kötücül koda ait işlem kodlar MAIL üzerinde tanımlı daha özet yapılara dönüştürülmektedir. Elde edilen bu özet kodlara ait kontrol akışları çizge tabanlı akış yapısı üzerinde temsil edilmekte ve bu yapı üzerinde çizge izomorfizmi üzerinden benzerlik analizine tabi tutarak kötücül kod tespiti yapılmaktadır. Çalışmada % 98,9 kötücül kod başarımı elde edilmiş olup, % 4,5 yanlış pozitif değeri gözlenmiştir. Alam ve arkadaşları tarafından önerilen diğer bir çerçeve çalışmada gerçek zamanlı olarak metamorfik kötücül kodları tespit edebilen, akış penceresi (Sliding Window) tabanlı bir yöntem önerilmiştir. Bu yöntem ile % 99,08 kötücül kod tespit başarımı elde edilirken, %0,93 yanlış pozitif oranıyla bir önceki çalışmaya göre yanlış pozitif değeri çok daha düşük bir yöntem geliştirilmiştir.

3.1.2. Durağan Analiz Yönteminin Avantajları

- Kod çalıştırılmadığından sistem kötücül koddan etkilenme riski taşımaz.
- Dinamik analize göre daha hızlı analiz yapılır.
- Birçok kod aynı anda kolayca analiz edilebilir.
- Elde edilen kod verileriyle tekrar tekrar farklı analizler yapılabilir.
- Elde edilen veriler, zamana ve çalışılan ortama göre değişmez.
- Analiz için kötücül kodun diassembler edilmiş yapısı zengin bir kaynak oluşturur.
- Kötücül kod tasarlayıcıları tarafından uygulanan dinamik analiz ortamı tespit yöntemleri bertaraf edilir.

3.1.3. Durağan Analiz Yönteminin Dezavantajları

- Kod gizleme yöntemleri kodun işlevsel yapısını değiştirmeden semantik yapısında değişikler yapar. Bu durum durağan analiz için yanıltıcıdır.
- Kod gizleme yöntemleri iyi huylu yazılımlarında kötücül kod olarak etiketlemesine neden olabilir. Yanlış pozitif oranı daha yüksek sonuçlar ortaya çıkabilir.

- Paketleme ve şifreleme yöntemleri kodun disassemble edilmesi önleyebilir. Kod hiç analiz edilemeyebilir.
- Kötücül kod tespitinde önemli kaynaklardan olan kötücül koda ait işletim sistemi ve ağ üzerindeki davranış bilgisi elde edilemez.

3.2. Dinamik Yöntemler Metamorfik Kötücül Kod Tespiti

Dinamik kötücül kod tespit yöntemlerinde, durağan kötücül kod tespit yöntemlerinden farklı olarak, analiz işlemi program çalıştırılarak, programın çalışma anındaki davranışlarının incelenmesi tekniği kullanılarak yapılır. Dinamik yöntemle kötücül kod tespit teknikleri arasında çizge tabanlı analiz, n-gram analiz ve çeşitli makine öğrenmesi yöntemleri gösterilebilir. Bu analiz yöntemleri içinde çizge tabanlı analiz yöntemlerine yönelik çalışmalar diğer yöntemlere göre daha yaygın olarak kullanılmaktadır. Dinamik analiz yöntemi genelde kötücül kod olup olmadığı henüz tespit edilmemiş bilinmeyen kötücül kodların ya da polimorfik veya metamorfik yöntemler kullanarak işlevselliği aynı fakat ikili yapıları farklı nesiller oluşturarak kendini gizleyip çeşitlilik sağlamış kötücül kodların tespitinde kullanılır [12].

Şekil 3.2: Dinamik analiz tekniğinde davranış tabanlı analiz adımları.

Davranışa göre dinamik analiz tekniğinde standart yöntem Şekil 3.2: Dinamik analiz tekniğinde davranış tabanlı analiz adımlarında belirtildiği gibi üç bölümden oluşur. İlk olarak programın davranışları izlenerek kaydedilir, sonra kaydedilen davranışlardan davranış özellikleri için özet yapılar oluşturulur, son olarak davranış tabanlı tespit ve sınıflandırma algoritmaları tasarlanır [12].

Dinamik analiz çalışmalarında kötücül kodun davranışları iki farklı teknik yaklaşımla izlenir. İlk yöntemde açık kaynak sistem öykünücülerin ya da sanal makinelerin kaynak kodları değiştirilip, komut sanallaştırma sürecine davranış izleme yapısı eklenerek kötücül kodların davranış verilerinin toplaması yöntemidir. Bu yöntemde davranışların incelenmesi, takibi, kaydedilmesi sanal makine izleme katmanında yapılır. Bu uygulaması kolay olmayan bir yöntemdir. Bu yönteme kullanan sistemlere Anubis[3], Qemu[4] ve Cuckoo[5] kum havuzu örnek olarak gösterilebilir. Bu yöntemde bilgisayar sistemi tam olarak kontrol altına alınır ve kötücül kodun davranışları için daha doğru ve kesin veriler sağlaması garanti edilir [12].

Güvenlik açısında analiz altındaki kötücül yazılımın internet ortamı ile bağlantı kurması istenmeyen bir durumdur. Bu nedenle analiz sırasında sanal makinanın internet bağlantısı kesilerek yerel ağda çalışılır. Bununla birlikte kötücül kodlar ağ üzerinde işlem yapmak isteyebilir ve bu veriler kötücül kodun tespitinde önemli davranış verileri olma potansiyeli sahip olabilir, hatta kötücül yazılım ağ bağlantısı olmadan hiç çalışmayabilir, bu durum yazılımın dinamik yöntemle analizini imkânsız hale getirebilir. Bu nedenle sanal makinanın ya da öykünücünün kaynak kodları değiştirilerek MikroTik Router OS[6] vb. gibi sistemlerin ağ yönlendirme özelliği yardımıyla sanal ağ ortamı oluşturulup internetteki çevrimiçi DNS, FTP, HTTP, E-posta vb. servislerin sanal olarak canlandırılması yöntemiyle kötücül kodların sıkça kullandığı DNS adres değiştirme, FTP indirme, Web tarayıcı yönlendirme E-posta gönderme, Port yönlendirme, TCP ve UDP paket gönderme gibi ağ üzerindeki davranışlarına yönelik bilgiler elde edilebilir [12].

İkinci yöntemde kötücül kodların davranış takibi misafir işletim sistemi katmanında yapılır. Diğer bir deyişle sanal makine sadece bir sanal uygulama ortamı değil, aynı zamanda kötücül kodun davranışlarını yakalayacak ve analiz edecek diğer ayıklama (debugger) ve tersine mühendislik araçlarının da bulunduğu ortamdır. Bu yöntemi kullanan sistemlere CFISandbox[7] örnek olarak gösterilebilir. Kötücül kodun davranışlarının izlenmesi için işletim sistemi tarafından sağlanan servislerin ve ara yüzlerin direkt olarak aynı katmanda kullanılabilmesi bu yöntemin avantajları

[3]Anubis kötücül yazılım tespit sistemi : https://anubis.iseclab.org/
[4]Qemu emulator : http://wiki.qemu.org/Main_Page
[5]Cuckoo kum havuzu : https://www.cuckoosandbox.org/
[6]MikroTik OS yönlendirici : http://www.mikrotik.com/software.html
[7]CWSandbox kum havuzu : https://www.threattrack.com/

arasındadır. Daha hızlı ve daha güçlü kontrol ortamı sağlar. Bunların yanında kötücül kodun davranış takibini yapan tersine mühendislik araçları ile kötücül kodun aynı ortamda çalışması ve bunun doğal sonucu olarak tersine mühendislik araçlarının kötücül kod tarafından etkilenme riski altında bulunması ve bu durumun kötücül kodun tersine mühendislik araçlarına yönelik karşı tespit teknolojilerini kullanmasını kolaylaştırması bu yöntemin dezavantajları arasındadır.

3.2.1. Dinamik Analizde Kullanılan Kum Havuzları

Dinamik analiz tekniği ile kötücül kod tespiti, kötücül kodun sanal makinalar, sanal kum havuzları veya öykünücüler kullanılarak analiz edilmesi ve davranışlarının takip edilmesi yöntemine dayanır. Bu amaçla kullanılan sanal analiz ortamları ikiye ayrılır, bir kısmı çevrimiçi; kötücül kodun yüklendiği ve analiz sonrasında kötücül kodun davranış özetini rapor olarak veren kum havuzları, diğeri ise masaüstü ortamda çalışan kum havuzları. Masaüstü ortamda çalışan kum havuzları içerisinde açık kaynak kodlu olanlar aynı zamanda analiz işlemi esnasın kaynak kodları değiştirilerek özelleştirilebilir.

Şekil 3.3: Dinamik analiz tekniği ile kötücül kod tespit yöntemlerinde sık kullanılan sanal makinalar.

Şekil 3.3: Dinamik analiz tekniği ile kötücül kod tespit yöntemlerinde sık kullanılan sanal makinalar listesinde verilen en sık kullanılan çevrimiçi kum havuzları içerisinde Anubis, ThreatExpert, Comodo, Malbox ve VMRay sayılabilir. En çok kullanılan çevrim dışı sanal kum havuzları içinde VMware, Oracle Virtualbox, Qemu, Sandboxie, JoeSecurity, NormanBox, Remnux örnek verilebilir.

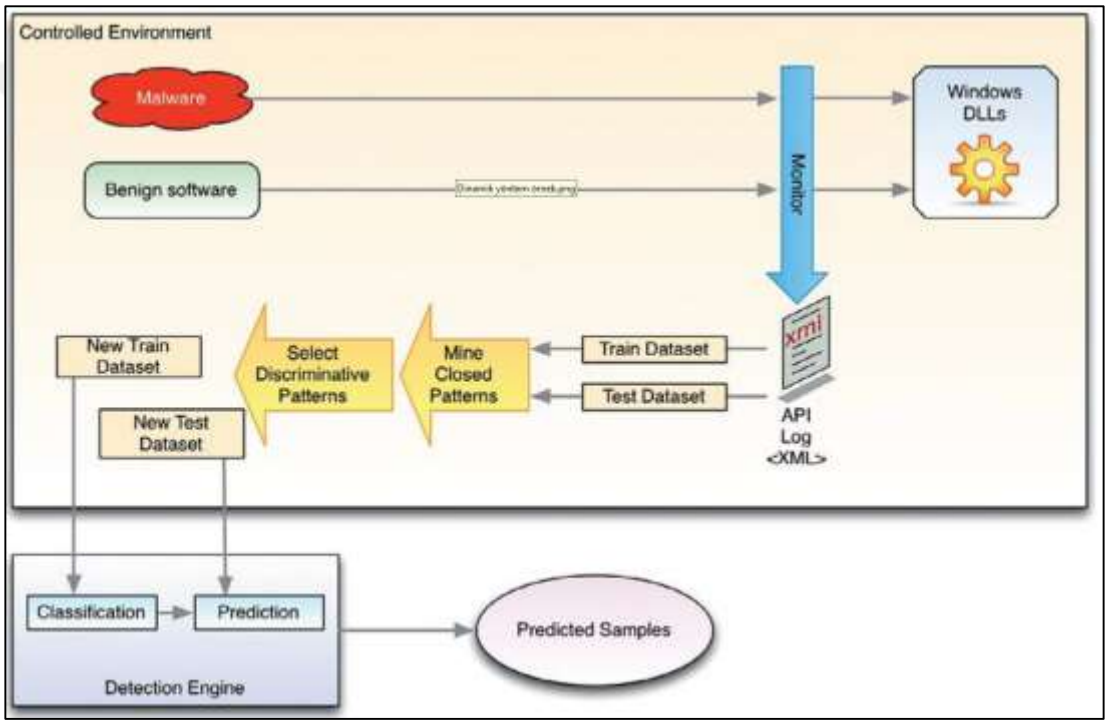
3.2.2. Dinamik Analizde İşlem Çağrıları (API Call)

İşlem çağrıları (API Call), Windows işletim sisteminde, çalışan program ile işletim sistemi arasında bağlantı görevi gören ara yüzlerdir. Kötücül kodlar bir sisteme girdiklerinde amaçlarından ilki sistemi ele geçirmek ve sistemde de yüksek kullanıcı yetkilerine sahip olmaktır. İkinci hedef ise sistemde trafiği izlemek, dış bir sistemle bilgi alış verişinde bulunmak, bilgileri çalmak ve sızıntı sağlamak vb. gibi işlemler olacaktır. Tüm bu işlemler sırasında kötücül kod tarafından işletim sistemine ait işlem çağrıları sıkça kullanılır. Kötücül kodlar tarafından işletim sistemine ait işlem çağrılarının sıkça kullanılması dinamik teknik kullanan kötücül kod tespit sistemlerinde analizcileri bu çağrıların izlenmesine yönlendirmiştir. Bu nedenle dinamik olarak kötücül kod tespiti yapan sistemler tarafından kötücül koda ait işlem çağrılarının takip edilmesi, sınıflandırılması ve bu çağrıların oluşturduğu dizilere göre davranış yapıları oluşturarak kötücül kodların tespit edilmesi yaygın olarak başvurulan yöntemler arasındadır. Bununla birlikte analiz ortamının etkilenmesi riski, çalışma zamanın uzunluğu gibi nedenlerden dolayı API çağrıları algoritmalar tarafından direkt olarak işleme tabi tutulmak için uygun değildir [12]. Bu nedenle programın davranışlarının izlenmesi sonucu elde edilen API çağrı günlük kayıtları (log) genel işlenecek bir yapıya dönüştürülerek analiz işlemi yapılır. API çağrılarının izlenmesi dinamik yöntemin özünü oluşturur. Dinamik analiz yönteminde temelde yapılan işlem kötücül kodun kullandığı API çağrılarının takibini yapmaktır.

3.2.3. Dinamik Analiz Yöntemini Kullanan İlgili Çalışmalar

Willems Carsten ve arkadaşları kötücül yazılımın kullandığı API çağrılarını analiz ederek kötücül kod için davranış analizi yapan bir yöntem geliştirdiler. Geliştirdikleri yöntemin diğer API çağrı yöntemlerinden farklı olarak kötücül kodun davranışlarını takip etmek için ayıklama (debugging) yerine Şekil 3.4: Dinamik analiz

tekniğinde api yakalama yöntemi ile metamorfik kötücül kod sınıflandırma sisteminde çalışma yöntemi gösterilen "API hooking" yöntemini kullanmasıdır. "API hooking" yöntemi sayesinde kötücül kodların karşı analiz yöntemlerinden etkilenmemektedir [26]. Bu yöntemde kötücül yazılım tarafından oluşturulan ve değiştirilen dosyalar, kötücül kodun Windows kayıt defterinde yaptığı değişiklikler, oluşturulan süreçler (process), giriş yapılan sanal bellek alanları, açılan ağ bağlantıları ve gönderim yapılan paketler, kötücül kod çalışmadan önce yüklenen kütüphaneler bilgisini takip edip rapor haline getirilerek analiz edilmektedir. Çalışma kapsamında kötücül kod tespit çalışmaları CWSandbox kum havuzu kullanılarak yapılmıştır.



Şekil 3.4: Dinamik analiz tekniğinde api yakalama yöntemi ile metamorfik kötücül kod sınıflandırma sistemi.

Ahmadi ve arkadaşları da WinAPIOverride32 aracını kullanarak kötücül kodun davranışlarını izlemek amacıyla "API hooking" tekniğini kullanmışlardır. Bu sayede kırılma noktaları (break point) koyarak ayıklama (debugging) yapmadıkları için kötücül kodun karşı analiz tekniklerine karşı savunucu bir kötücül kod tespit sistemi geliştirmişlerdir. Geliştirilen yöntemin denenmesi için 635 adet kodun bulunduğu bir veri seti kullanmışlardır. Bu veri setindeki programların 267 tanesi kötücül kodlardan oluşmuştur. Yöntem PE (Portable executable) dosyalar üzerinde test edilmiştir. Analiz işlemi Şekil 3.4'de de görüleceği üzere beş adımdan oluşmaktadır. İlk olarak kötücül

kodun davranışları “API hooking” yöntemi ile takip edilip ele geçirilmekte, daha sonra ele geçirilen bu günlük (log) kayıtlarından API çağrı dizileri alınmaktadır. Günlük kayıtlarından tekrarlayıcı şablon (iterative pattern) ile özellik çıkarma işlemi yapılarak ayırt edici şablonlar elde edilmektedir. Sonuç olarak ta Fisher hesaplama algoritması ile öznitelik seçimi yapılarak, kötücül kodun hangi sınıfa ait olduğu tahmin etmek için sınıflandırıcı kullanılmaktadır [27]. Yöntemin avantajları yüksek tespit hızı ve düşük oranda yanlış alarm olarak ortaya çıkmaktadır.

Lee ve arkadaşları da kötücül yazılımların yapısı değişmesine rağmen yazılımın ana yapısında bulunan ve ana işlemleri gerçekleştiren değişkenlerin hiçbir zaman değişmeyeceği, anlamsal yapısını koruyacağı fikrinden hareketle statik analiz yöntemini kullanarak kötücül yazılımların API çağrılarını çağrı çizgesinde tasvir ederek, kötücül yazılımların anlamsal imzasını (semantic signature) çıkaran ve böylece kötücül kodların karakteristiğini ortaya koyan ve buna göre kötücül kod tespiti yapan bir yöntem önermişleridir [28]. Zhang ve arkadaşları Destek Vektör Makineleri (Support Vector Machine) kullanarak Kaba Kümeleme (Rough Sets) yöntemi ile öznitelik sayısını azaltarak tespit yapan yöntem önermişlerdir [29].

Waganer ve arkadaşları kötücül dosyalardan API çağrı listelerini elde edip bunların benzerlik matrislerini kullanarak kötücül yazılımları tespit yöntemi önermişlerdir [30]. Benzer şekilde Tian ve arkadaşları da API çağrılarının özelliklerini elde ederek örüntü tanıma yöntemi ile kötücül kodların tespit edilmesini sağlayan yöntem üzerinde çalışmışlardır [31]. Burguera ve arkadaşları, gerçek zamanda sistem çağrılarının merkezi bir sunucuya gönderilerek tespit yapan bir yöntem önermişlerdir [32]. Tsyganok ve arkadaşları da kötücül yazılımların API çağrıları ve dosyalar üzerinde yaptığı değişiklikleri inceleyerek kötücül yazılımlar için dinamik sınıflandırma yöntemi kullanmıştır [31].

Yine Xue ve arkadaşları kötücül yazılımın çalışma sürecinde sistem çağrı dizilerini bir araya getirilmesi ile ortaya çıkan örüntüler üzerinden kötücül yazılımı tespiti yöntemi önermişlerdir [33].

Dinamik analiz teknikleri kullanılarak kötücül kod tespitinde kötücül kodu doğal ortamdan izole ederek çevreye zarar vermeyeceği, hareketlerinin kolayca gözlenebileceği ve hızlı şekilde analiz yapılabilecek kum havuzlarında çalıştırılması yöntemi yaygın olarak tercih edilmektedir. Bununla birlikte kötücül kodlar bu analiz ortamını tanıyıcı teknikler kullanarak analizi engelleyici veya zorlaştırıcı teknikleri sıklıkla kullanmaktadır.

3.2.4. Dinamik Analiz Yönteminin Avantajları

- Oligomorfik, polimorfik ve paketlenerek gizlenmiş kötücül kodlar içi ideal bir tespit yöntemidir.
- Metamorfik kötücül kodlarda metamorfik yöntemler kodun ikili yapısını, büyüklüğünü, çalışma yöntemini değiştirmesine karşı kodun kötücül işlemler yapan çekirdek kısmı aynı kalmaktadır. Davranış analizi ile metamorfik kötücül kod tespiti için daha verimli veriler elde edilir.
- Kötücül kodların ağda yaptıkları işlemler kötücül kod tespit yöntemlerine önemli veriler sağlar. Dinamik analiz sayesinde koda ait ağ davranış bilgisi elde edilebilir.
- Kötücül kod parçalarını üzerinde taşımayan sisteme bulaştıktan sonra ağ üzerinden indirerek kullanan kodların tespitinde daha güçlü bir yöntemdir.

3.2.5. Dinamik Analiz Yöntemin Dezavantajları

- Analiz işlemi durağan analize göre daha yavaştır.
- Bir anda sadece tek bir kötücül kod için analiz yapılabilmektedir.
- Tersine mühendislik araçları kötücül yazılımla aynı ortamı paylaştığı için kötücül yazılımın bu araçları etkileme ihtimali var.
- Kötücül kod çalışır durumda olduğu için karşı analiz işlemleri yapabilmesine imkân tanımaktadır.
- Kötücül kodların büyük çoğunluğu yaygın olarak kullanılan dinamik analiz ortamlarını, sanal makinanın türünü tespit edebilmekte ve analiz esnasında aktif olmamaktadır.
- Kötücül kodlar tarafından kullanılan mantıksal saldırılara karşı dayanıksızdır. Örneğin zamanlamalı çalışma, rastgele ortama göre çalışma.
- Analizi zorlaştıracak, boş anlamsız işlem süreçleri üretilerek davranış analiz sürecini karmaşıklaştırmakta ve analiz zamanını uzatmaktadır.

3.2.6. Dinamik Analiz Ortamı Tespit Teknikleri

Daha öncede belirttiğimiz gibi kötücül kodlar dinamik olarak analiz edildiklerinde aktif durumda olduklarından bellek ve işletim sistemi üzerindeki

kayıtları kontrol ederek çalışılan ortamın analiz ortamı ya da kum havuzu olup olmadığını kontrol edebilmektedirler. Analiz ortamında çalıştıklarını tespit ettiklerinde işlemi yarıda kesebilmekte ya da analiz işlemini karmaşıklaştıran işlemler uygulayabilmektedirler. Anoire Issa tarafından [34]'te yapılan çalışma kapsamında Zeus ve SpyEye kötücül kodları tarafından kullanılan karşı analiz tekniklerden bazıları aşağıda açıklanmıştır.

İşletim sistemi üzerinde çalışan programların çoğu kendi dosya sistemlerine sahiptirler. Örneğin Sandboxi kum havuzu "SbieDll.dll" dosyasını kullanır. Bununla birlikte kötücül kodlar bu dll dosyasının sistem üzerinde yüklü olup olmadığını kontrol ederek Sandboxie kum havuzu altında çalıştıkları kontrol etmektedirler. Benzer şekilde Oracle Virtualbox sanal makinesini çalıştırıldığında sistem içinde kendi süreçlerini yürütür. Kötücül kodlar bu süreçleri kontrol ederek VirtualBox'ı tespit edebilmektedirler. Örneğin süreç listesinde "VboxServices.exe" ya da "VBoxTray.exe" adında bir süreci aktif durumda olması kötücül kodlar için VirtualBox sanal ortamında çalışıldığını göstermektedir.

Kötücül kodlar masaüstü ortamlarda çalışan kum havuzları ve sanal makinalar yanı sıra çevrimiçi çalışan ortamlarda çalışan kum havuzu ve sanal makinaları da tespit edebilmektedirler. Buna örnek olarak ticari olarak hizmet veren çevrimiçi kötücül kod analiz ortamı Anubis verilebilir. Anubise ait bazı çevresel değişkenler sistemi kullanan kullanıcılar tarafından ulaşılabilir durumdadır. Bunlardan olan ürün numarası "76487-337-8429955-22614" kötücül kodlar tarafından kontrol edilerek çalışma ortamının Anubis olduğu tespit edilmektedir. Benzer şekilde CWSandboxi kum havuzuda sabit ürün numarası "76487-644-3177037-23510" üzerinden, JoeBox kum havuzu da "55274-640-2673064-23950" ürün numarası üzerinden tespit edilebilmektedir.

Windows işletim sistemi tarafından sağlanan "dbghelp.dll" ayıklayıcı kütüphanesi dinamik analiz için kullanılan analiz programlarının çoğu tarafından ana ayıklayıcı olarak kullanılmaktadır. Kötücül kodlar bu kütüphanenin belleğe yüklenip yüklenmediğini kontrol ederek kendileri için bir ayıklama işlemi yürütüldüğünü kontrol ederler. Kötücül kod analizcileri tarafından en sık kullanılan ayıklama programlarından Olly Debugger tarafından halen daha ana ayıklama kütüphanesi olarak "dbghelp.dll" kullanılmaktadır.

Kötücül kodlar tarafından kullanılan diğer bir sanal makine tespit tekniği de kayıt defterinin kontrol edilmesidir. En basit yöntemlerden biride olsa bugün hala kötücül kodlar tarafından kullanılan geçer yöntemlerden biridir. Örneğin en eski ve

profesyonel kum havuzlarından biri olan Norman kullanıcı adı olarak kayıt defterinde "CurrentUser" kullanmaktadır. Kötücül kodlar bu kayıt defteri içeriğini takip ederek Norman kum havuzunu tespit etmektedirler.

Kötücül kodlar tarafından kayıt defterleri, kullanılan dinamik kütüphanelerin takibi, yürütülen süreçler, ürün numaraları, işletim sistemi tarafından sabit bellek değeri üzerinde çalıştırılan kütüphanelerin takibi gibi yöntemler kullanılarak kötücül kodlar tarafından kum havuzları ve sanal makinalar tespit edilebilmektedir.

4. METODOLOJİ

4.1. Araştırma Soruları ve Hipotezler

Çalışma öncesi elde bulunan veriler ve sorular;

- Metamorfik kötücül kodların asıl işlevi yapan kısımları kodun içinde dağınık yapıda dahi olsa mevcuttur. Her bir nesilde dağınık bir yapıda dahi olsa tekrar etmektedir.
- Kötücül kodların gizleme kısımları değişebilir. Kodu anlamsız hale dönüştürmek için fazladan, eşdeğer kodlar bulunabilir.
- Her bir nesilde tekrar eden OpCode dizileri var mı?
- Kötücül kod kısmına ait örüntüler yakalanarak virüsler diğer uygulamalardan ayrılabilir mi?
- Kötücül kod dizileri için kullanılan özetleme algoritmalarıyla daha küçük öznitelik kümeleriyle daha yüksek başarımlar elde edilebilir mi?
- Kötücül kodların gerçek zamanlı tespiti için kullanılacak örüntüler elde edilebilir mi?

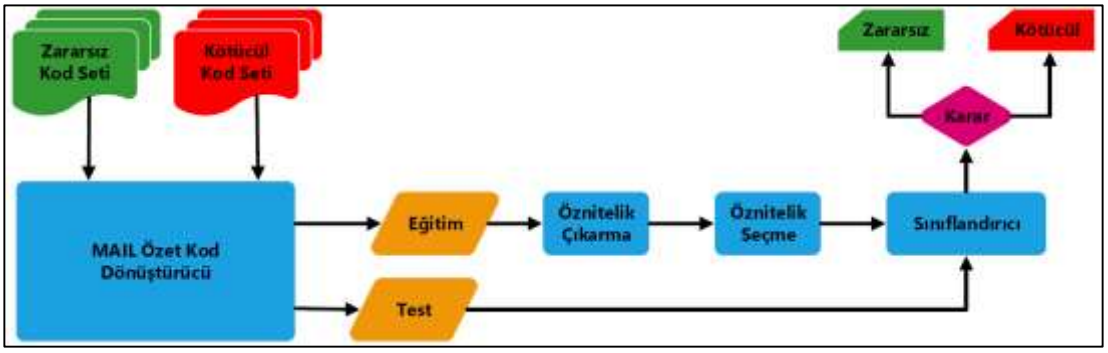
Çalışma öncesi elde bulunan verilen ve sorulardan yola çıkarak çalışma kapsamında cevap aranacak sorular ve geliştirilen hipotezler aşağıdaki gibi belirlenmiştir.

- MAIL diline ait cümlelerin terim sıklığı TF (Term Frequency), ters doküman sıklığı IDF (Inverse Term Frequency) ve bu iki değerin çarpımı niteliğindeki TF-IDF değerleri metamorfik kötücül kodların tespitinde kullanılabilir mi?
- Metamorfik kötücül kodların tespitinde MAIL dilinde tanımlanmış ayırt ediciliği en yüksek öznitelikler hangileridir?
- Ayırt ediciliği yüksek öznitelikler bilgisi kullanılarak daha az öznitelikle daha hızlı ve başarım oranı daha yüksek bir yöntem oluşturulabilir mi?
- Geliştirilen model için en iyi öznitelik seçim algoritması hangisidir?
- Model için en uygun sınıflandırma algoritması hangisidir?

4.2. Analiz Yöntemi Seçimi

Kötücül kodların gerçek zamanlı tespiti, kötücül kodlar sisteme yayılmadan, yüklenme sırasında ya da yüklendikten sonra henüz aktif hale gelmeden tespit edilmesi olarak tanımlanmaktadır. Bu nedenle dinamik analiz yöntemlerinde kötücül kod çalıştırılarak analiz edildiği için kod aktif hale gelmekte karşı analiz yöntemleri uygulayabilmekte, analiz zamanını uzatabilmektedir. Bununla birlikte durağan analiz yöntemi sayesinde kötücül kodlar çalıştırılmadan henüz aktif hale gelmeden analiz edilmesi, gerçek zamanlı kötücül kod tespiti için daha uygun analiz ortamı sunmaktadır. Bununla beraber Bölüm 3.1.3'de açıklanan durağan analiz yönteminin dezavantajları nedeniyle kötücül kod tespiti uzayabilmekte, kod gizleme yöntemleri nedeniyle bazı durumlarda imkânsız hale gelebilmektedir. Bu durumda öznitelik kümesinin seçimi, büyüklüğü çalışma zamanın açısında önemli rol oynamaktadır. Çalışmamız kapsamında avantajları ve dezavantajları karşılaştırıldığında gerçek zamanlı kötücül kod tespit sistemleri için en uygun analiz yöntemi olarak durağan analiz yöntemi ortaya çıkmıştır. Bununla birlikte durağan analiz yöntemine ait kötücül kodların gerçek zamanlı tespitini engelleyen dezavantajlar öznitelik seçim sürecinde giderilmiştir.

4.3. Metodoloji

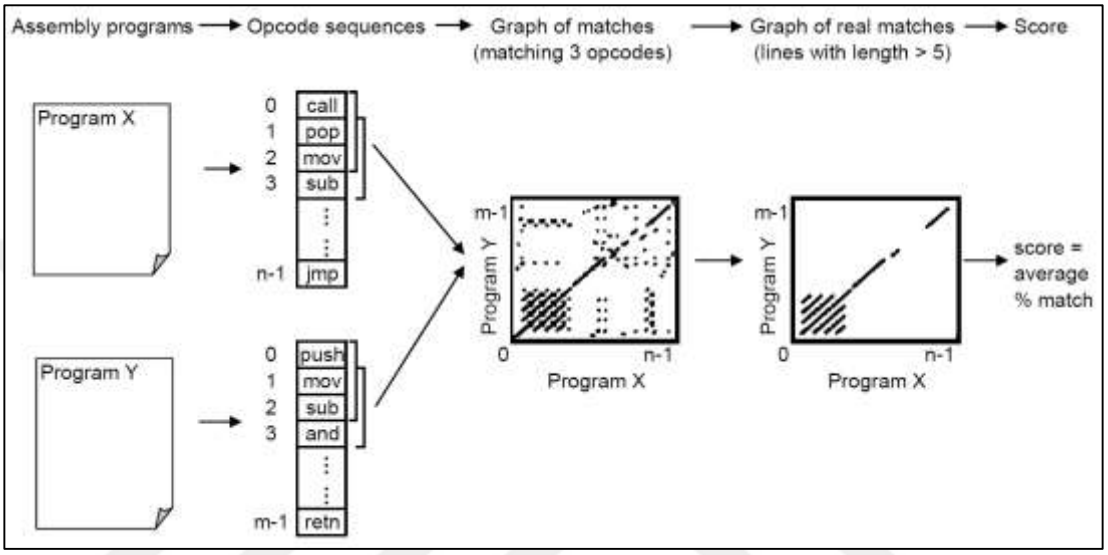


Şekil 4.1: Metamorfik kötücül kod tespit sistemi.

Çalışmamız kapsamında Şekil 4.1: Metamorfik kötücül kod tespit sistemi akış diyagramında gösterilen metodoloji takip edilmiştir. Veri madenciliği yöntemlerinde sıkça takip edilen bu metodoloji gereği öncelikle eğitim ve test için gerekli veri seti toplanmıştır. Daha sonra öznitelik çıkarma işlemi için yöntem seçilmiştir. İlgili sistem

için MAIL diline ait cümlelerin TF, IDF, TF-IDF değerleri öznitelik olarak belirlenmiştir. Öznitelik kümesini azaltmak ve sistemin başarımını artırmak için öznitelik seçimi yapılmıştır. Son olarak sistem için en uygun sınıflandırma algoritması belirlenerek sistemin başarımı ölçülmüştür.

4.3.1. Veri Setinin Belirlenmesi



Şekil 4.2: İşlem kod benzerlik analiz yöntemi.

Wing [4]'de 2.5 Metamorfik Yöntemle Kod Gizleme bölümünde açıklanan kamuflaj yöntemlerini kullanan NGVCK (Next Generation Virus Creation Kit), G2 (Second Generation Virus Kit), VCL32 (Virus Creation Lab for Win32), MPCGEN (Mass Code Generation) kötücül kodlarının her birinden yaklaşık 10 örnek nesil ve Cygwin öykünücüsünden alınan yaklaşık 20 koddan oluşan bir kümede Şekil 4.2: İşlem kod benzerlik analiz yöntemi çalışma yöntemi verilen kodların işlem kod benzerlik durumunu test eden çalışma yapmıştır. Çalışma sonucunda kötücül kodların kendi sınıfları içinde işlem kod tabanlı benzerlik oranı NGVCK kötücül kodu hariç diğer kötücül kodlar için %34 ile %96 arasında değerlerde bulunmuştur [4]. Bununla birlikte NGVCK metamorfik kötücül kodu için benzerlik oranı %0 olarak gözlenmiştir. Çalışma sonucunda NGVCK metamorfik gizleme yöntemlerini en etkili kullanan kötücül kod olarak belirlenmiştir. Bu nedenle metamorfik kötücül tanıma üzerine yapılmış daha sonraki çalışmaların çoğunluğunda [4], [35], [36], [19], [37], [38], [39], [40], [41] NGVCK kötücül kod sınıfı metamorfik kötücül kodların tanınmasında örnek kötücül kod kümesi olarak kullanıldı ve çalışmaların başarımları

NGVCK tanınması üzerinden ölçüldü. Bununla birlikte NGVCK kötücül kod kümesini diğer metamorfik kötücül kod sınıflarından ayıran kötücül kod gizleme yönteminin Bölüm 2.5.5'te açıklanan kod bloklarının yerlerinin değiştirilmesi özelliği olduğu tespit edildi. Bu nedenle çalışmamız kapsamında metamorfik kötücül kod gizleme yöntemlerini en iyi derecede temsil yeteneğine sahip NGVCK kötücül kod sınıfı metamorfik kötücül kod veri seti olarak kullanılmıştır. Bunun yanı sıra metamorfik kötücül kod çeşitliliğini artırmak, benzer yöntemleri kullanan metamorfik kötücül kodların birbirinden ayırt edebilecek bir sistem oluşturabilmek, yapılan diğer çalışmalarla sistemin başarım oranını karşılaştırılabilmesi için ortak veri seti üzerinde test edilebilmesi amacıyla G2, VCL32, PSMPC metamorfik kötücül kod sınıfları da veri setine dâhil edilen diğer kötücül kod sınıfları olmuştur.

Kötücül kod sistemlerinin başarımının ölçülmesinde kötücül kod tespit başarısı yanı sıra yanlış pozitif oranındaki düşüklükte önem taşımaktadır. İyi bir kötücül tespit sistemi zararsız kodları kötücül olarak etiketlememelidir. Bu durumun test edilebilmesi için Windows işletim sistemine ait program dosyaları ile Cygwin öykünücüsüne ait çalıştırılabilir kodlar veri setine dâhil edilmiştir.

Veri seti Visual C++, Visual C#, Visual Basic ve GCC gibi farklı derleyiciler tarafından derlenmiş, Windows ve Linux gibi farklı platformlarda çalışan, çalıştırılabilir formattaki "exe" dosyaları yanı sıra "dll" uzantılı kütüphaneleri de kapsayacak şekilde çeşitliliği yüksek geniş veri seti olarak oluşturulmuştur. Çalışma zamanın ölçülebilmesi, gerçek zamanda kötücül kod tespit özelliğinin test edilebilmesi için diğer çalışmalara oranla daha büyük bir veri seti üzerinde çalışılmıştır. Veri setindeki örnek dosyaların sayısının büyüklüğü, kötücül kod sınıfları ve türlerindeki çeşitlilik, zararsız kod kümesinin büyüklüğü geliştirilen kötücül kod sisteminin test edilmesi için güçlü bir test ortamı oluşturmuştur.

4.3.2. Öznitelik Çıkarma Yöntemi Seçimi

İlgili çalışmalar bölümünde de açıklandığı üzere işlem kodlar kötücül kodları gerçek zamanlı olarak tespit eden yöntemlerde sıkça kullanılmaktadır. Bununla birlikte tespit yöntemlerinde işlem kodların öznitelik olarak kullanılmasının kaçınılmaz bazı dezavantajları mevcuttur. Bunlardan bazıları;

- Farklı derleyici kullanılması işlem kodlara ait örüntüleri değiştirecektir. Bu nedenle elde edilen örüntüler kodun farklı bir derleyici kullanılması durumunda geçerliliğini yitirecektir.
- Aynı derleyicinin farklı optimizasyon seviyeleri farklı assembly kodları üretmektedir. Derleyicinin farklı optimizasyon seviyesinde kodun derlenmesi işlem kod örüntülerinin etkisini azaltacaktır.
- Kodun farklı işletim sistemleri için derlenmesi işlem kod örüntülerini değiştirecektir. Kodun yapısında farklılaşmaya neden olacaktır.
- Kuramsal temeller kısmında açıklanan metamorfik ve polimorfik değişiklikler kötücül kodun yeni sürümünde yapısal değişikliklere neden olacağından kodun işlem kod örüntüleri değişecektir.
- İşlem kod seviyesinde yaklaşık olarak 900 kadar işlem kod mevcuttur. Öznitelik sayısının fazla olması kötücül kod tespit başarım oranını artırırken tespit için gerekli zamanı da uzatacaktır. Bununla birlikte öznitelik sayısının az tutulması başarım oranını düşürürken çalışma zamanını azaltacaktır. Bu durum yüksek başarımlı gerçek zamanlı bir tespit sistemi oluşturulması açısından ikilem oluşturacaktır.

Geliştirilen sistem için yukarıda açıklanan nedenlerden dolayı platform, derleyici ve işlemciden bağımsız, polimorfik ve metamorfik kod gizleme yöntemlerine karşı dirençli, işlem kodları daha az sayıda işlem kod kümesi ile temsil kabiliyetine sahip ara bir dil kullanılması zorunluluğu ortaya çıkmıştır.

Bu amaçla daha önceki [25], [41], [38], [39] çalışmalarında kullanılan ve başarılı sonuçlar elde edilen MAIL özet dil yapısı çalışmamızda geliştirilen sistem için kodlara ait işlem kod sistemini temsil eden üzerinde çalışılacak ara dil olarak kabul edildi.

4.3.3. MAIL Özet Dil Yapısı

MAIL, ikili program kodlarını çözümleyip (disassembly), programın kontrol akış diyagramının daha iyi tanımlanabilmesi ve daha kolay analiz edilebilmesi amacıyla programın işlem kodları (OpCode) üzerinden [25]'de tanımlanan 21 adet MAIL cümlesinden oluşan özet bir yapıya dönüştüren ara bir dildir. MAIL platform bağımsızlığı, kötücül kod analizlerinde otomasyon ve optimizasyon yönelik güçlü yanları nedeniyle tercih edilmiştir. MAIL dilinin tercih edilmesindeki en önemli

etkenlerden birisi de işlemci ve platform bağımsız yöntem geliştirilmesine olanak sağlamasıdır. Böylece MAIL dil yapısı kullanarak geliştireceğimiz yöntem Windows, Linux işletim sistemi ortamları, 32 ve ya 64 bit işlemciler için tasarlanmış programlar, hatta mobil ortamlar için çalışabilecektir.

Günümüzde işlemciler yaklaşık olarak 900 kadar farklı işlem koda sahiptirler. Her bir işlemci modeli kendi işlem kod kümesine sahiptir. Bu nedenle bu işlem kodların birçoğu benzer işlemleri yapmaktalar. Durağan analiz esnasında daha hızlı analiz yapmak ve gereksiz işlem kodların elemek için MAIL dili içerisinde bu 900 kadar işlem kod 21 adet MAIL cümlesine dönüştürülmektedir. MAIL dilinde amaç kodun semantik yapısını koruyarak koda ait assembly program yapısını daha özet bir dille ifade etmektir. Aynı zamanda MAIL derleyici, optimizasyon seviyesi ve platform farklarından dolayı ortaya çıkan farklılıkları en aza indirerek kodun anlamsal bütünlüğünü koruyarak kodların assembly kod yapısı için temsil kabiliyeti yüksek bir ara dil olarak karşımıza çıkmaktadır.

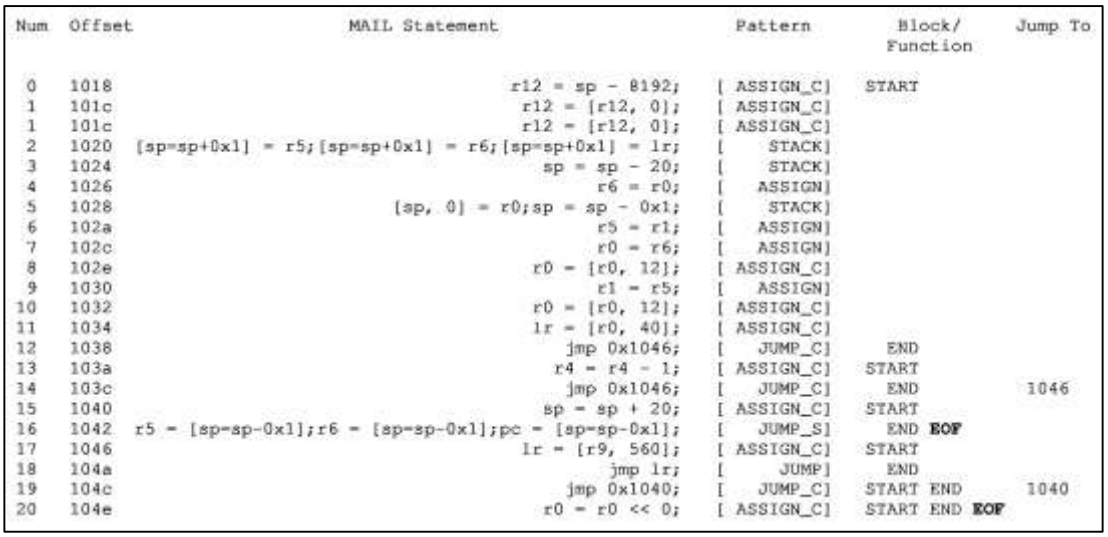
Assembly diline ait işlem kodlar işlevlerine göre gruplandırılabilirler. Örneğin assembly dilinde ADD, SUB, MUL, DIV, FSIN, FCOS, PADDW, PSUBW, ADDPS, ADDPD, PMULLD, PAVGW, DPPD, SHR ve SHL gibi işlem kodlar aritmetik işlemler yaparken, AND, OR, NOT gibi işlem kodlar mantıksal işlemler gerçekleştirmektedir. Bunu yanı sıra MOV, CMOV, XCHG, PUSH, POP, LODS, STOS, MOVS, MOVAPS, MOVAPD, IN, OUT, INS, OUTS, LAHF, SAHF, PREFETCH, FLDPI, FLDCW, FXSAVE, LEA ve LDS gibi işlem kodlar veri transferi yaparken, MP, CALL, RET, CMP, CMPS, CMPPS, PCMPEQW, REP ve LOOP gibi işlem kodlar kontrol işlem kodları olarak işlev görmektedir. Özellikle metamorfik kötücül kodların ayırt ediciliğinde en yüksek değere sahip kodlar kontrol işlem kodlarından oluştuğunu problem ifadesi kısmında ayrıntılı tartışılmıştı.

MAIL dili assembly diline ait işlem kodların bu gruplama özelliğinden yararlanarak bu yaklaşık 900 kadar olan işlem kod listesini Tablo 4.1: MAIL diline ait işlem kod açıklamaları tablosunda ayrıntılı olarak verilen 21 adet işlem koda dönüştürmektedir. Bu kodlar ASSIGN, ASSIGN CONSTANT, CONTROL, CONTROL CONSTANT, CALL, CALL CONSTANT, FLAG, FLAG STACK, HALT, JUMP, JUMP CONSTANT, JUMP STACK, LIBCALL, LIBCALL CONSTANT, LOCK, STACK, STACK CONSTANT, TEST, TEST CONSTANT, UNKNOWN, NOTDEFINED işlem kodlarından oluşmaktadır. Bu kodlar genel olarak, atama, çağırma, kontrol, test gruplarından oluşmaktadır.

MAIL dili analiz edilecek kod parçasını önce disassembly etmekte. Daha sonra assembly kodlarını MAIL diline tercüme ederek kendi içinde tanımlamaktadır.

Tablo 4.1: MAIL diline ait işlem kod açıklamaları.

İşlem Kod	**Açıklama / Örnek**
ASSIGN	Atama işlemi. Örn : EAX=EAX+ECX;
ASSIGN CONSTANT	Sabit değer içeren atama. Örn : EAX=EAX+0x01;
CONTROL	Bilinmeyen bir noktaya yapılan atlama (jump). Örn : if (ZF == 1) JMP [EAX+ECX+0x10];
CONTROL CONSTANT	Bilinen bir noktaya yapılan atlama. Örn : if (ZF == 1) JMP 0x400567;
CALL	Hedefi bilinmeyen çağrı. Örn :CALL EBX;
CALL CONSTANT	Hedefi bilinen çağrı.Örn : CALL 0x603248;
FLAG	İşaretçi. Örn : CF = 1;
FLAG STACK	Yığın için işaretçi. Örn : EFLAGS = [SP=SP-0x1];
HALT	Durdurucu. Örn : HALT;
JUMP	Hedefi bilinmeyen atlama. Örn : JMP [EAX+ECX+0x10];
JUMP CONSTANT	Hedefini bilinen atlama. Örn : JMP 0x680376
JUMP STACK	Yığın üzerinde geri atlama. Örn : JMP [SP=SP-0x8]
LIBCALL	Kütüphane çağırma. Örn : compare(EAX, ECX);
LIBCALL CONSTANT	Sabit parametre içeren kütüphane çağırma. Örn : compare(EAX, 0x10);
LOCK	Kilitleme. Örn : LOCK
STACK	Yığın. Örn : EAX = [SP=SP-0x1];
STACK CONSTANT	Sabit değer içeren yığını. Örn : [SP=SP+0x1] = 0x432516;
TEST	Test. Örn : EAX and ECX;
TEST CONSTANT	Sabit değer içeren test. Örn : EAX and 0x10;
UNKNOWN	Bilinmeyen assembly komutu. Çevirisi yapılmıyor.
NOTDEFINED	Varsayılan örüntü. Örn: Her yeni komut oluşturulduğunda atanan varsayılan değer.



Num	Offset	MAIL Statement	Pattern	Block/ Function	Jump To
0	1018	r12 = sp - 8192;	[ASSIGN_C]	START	
1	101c	r12 = [r12, 0];	[ASSIGN_C]		
1	101c	r12 = [r12, 0];	[ASSIGN_C]		
2	1020	[sp=sp+0x1] = r5;[sp=sp+0x1] = r6;[sp=sp+0x1] = lr;	[STACK]		
3	1024	sp = sp - 20;	[STACK]		
4	1026	r6 = r0;	[ASSIGN]		
5	1028	[sp, 0] = r0;sp = sp - 0x1;	[STACK]		
6	102a	r5 = r1;	[ASSIGN]		
7	102c	r0 = r6;	[ASSIGN]		
8	102e	r0 = [r0, 12];	[ASSIGN_C]		
9	1030	r1 = r5;	[ASSIGN]		
10	1032	r0 = [r0, 12];	[ASSIGN_C]		
11	1034	lr = [r0, 40];	[ASSIGN_C]		
12	1038	jmp 0x1046;	[JUMP_C]	END	
13	103a	r4 = r4 - 1;	[ASSIGN_C]	START	
14	103c	jmp 0x1046;	[JUMP_C]	END	1046
15	1040	sp = sp + 20;	[ASSIGN_C]	START	
16	1042	r5 = [sp=sp-0x1];r6 = [sp=sp-0x1];pc = [sp=sp-0x1];	[JUMP_S]	END **EOF**	
17	1046	lr = [r9, 560];	[ASSIGN_C]	START	
18	104a	jmp lr;	[JUMP]	END	
19	104c	jmp 0x1040;	[JUMP_C]	START END	1040
20	104e	r0 = r0 << 0;	[ASSIGN_C]	START END **EOF**	

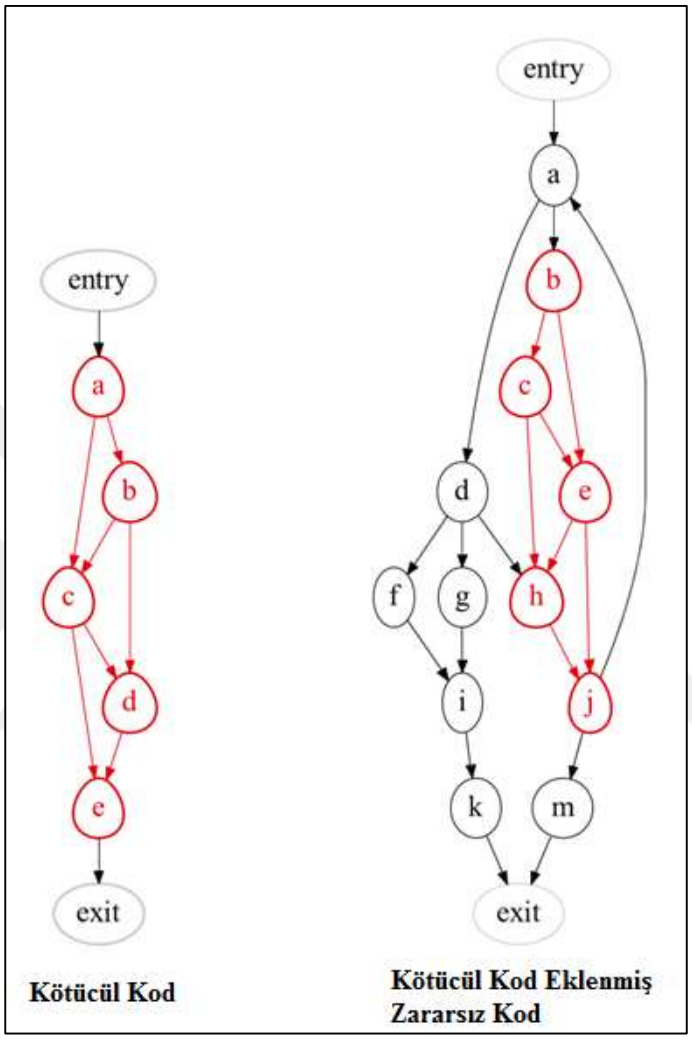
Şekil 4.3: Örnek kodun MAIL dilinde özetlenmiş işlem kod ve kontrol akış şeması.

MAIL dili üzerinde sadece işlem kodların MAIL diline ait karşılıkları tanımlanmamakta, aynı zamanda koda ait kontrol akış yapısı da blok halinde tanımlanmaktadır. Şekil 4.3: Örnek kodun MAIL dilinde özetlenmiş işlem kod ve kontrol akış şemasında "Block / Function" sütununda görüleceği üzere her bir kontrol komutu MAIL dili içerisinde blok fonksiyonun bitişini belirtmektedir. Bu yöntemle MAIL dilinde kontrol işlemleri birbirinden ayrı blok fonksiyonlar olarak tanımlanabilmektedir. Aynı zamanda "Jump To" sütunu altında bulunan değerlerden anlaşılacağı üzere her bir kontrol işlemi sonrasında programın atladığı sıra numarası "offset" değeri belirtilmektedir. Bu sıra numarası takip edilerek programa ait kontrol akış yapısı kolayca çıkartılabilmektedir.

Metamorfik kötücül kodlar tarafından kullanılan kodun imzası üzerinde en büyük değişimi yaratan en etkili metamorfik değişim yönteminin kod bloklarının değişimi olduğu daha önce kuramsal temeller bölümünde açıklanmıştı. MAIL dili yardımıyla elde edilen bu kontrol akış blokları Şekil 4.4: MAIL diline ait kod bloklarının kontrol akış çizgesi üzerinde temsili edilerek kötücül kod ve kötücül kodun başkalaşım geçirmiş hali ya da kötücül kodun kendini kopyaladığı zararsız koddaki sürümü çizge izomorfizm benzerliği üzerinden tespit edilebilmektedir. [38]'de bu yöntem kullanılarak kötücül kod tespiti yapılmış olup %99.2 başarım oranı, 3.07 yanlış pozitif değeri elde edilmiştir.

Bununla birlikte çalışmamız kapsamında MAIL diline ait bu kontrol blokları değil daha önceki çalışmalarda üzerinde durulmayan MAIL diline ait işlem kodların frekansları üzerinde durulacaktır. Bu amaçla MAIL dilinde tanımlı Şekil 4.3: Örnek

kodun MAIL dilinde özetlenmiş işlem kod ve kontrol akış şemasında “Pattern” sütunu altında bulunan Tablo 4.1: MAIL diline ait işlem kod açıklamaları verilen MAIL işlem kodlarının terim frekans, doküman frekans ve terim ve ters doküman frekans değeri çarpı öznitelik vektörü olarak kullanılacaktır.



Şekil 4.4: MAIL diline ait kod bloklarının kontrol akış çizgesi üzerinde temsili.

4.3.4. TF, IDF, TF-IDF Değerleri Üzerinden Öznitelik Çıkarma

Öznitelik çıkarma yönteminde doküman tanıma işlemlerinde sıkça kullanılan ve yüksek başarım oranına sahip olan TF, IDF, TF-IDF hesaplama yöntemleri MAIL diline ait cümleler üzerinde uygulanarak öznitelik vektörleri oluşturulmaktadır. Her bir öznitelik vektörü MAIL dilinde tanımlı toplam 21 adet işlem koda ait TF, IDF, TF-IDF değerlerinden oluşan üç adet öznitelik vektöründen oluşmaktadır. TF değerinin hesaplanması (4.1)’de belirtilen TF denklemi uygun olarak 21 adet MAIL işlem

kodunun her birinin program içerisinde ortaya çıkma sayısının, programdaki toplam işlem koda bölümüyle elde edilmektedir. Örneğin CONTROL işlem kodu program içerisinde 20 defa geçiyorsa ve program içerisinde toplam 100 adet işlem kod varsa. TF öznitelik vektörü içerisinde CONTROL işlem koduna ait TF değeri 20/100 = 0.5 olacaktır.

IDF özniteliği (4.2)'de verilen denkleme uygun olarak MAIL diline ait 21 adet özniteliğin işlem gören toplam doküman sayısının işlem kod içerisinde geçtiği toplam doküman sayısına bölünmesi sonucu elde edilen sayısının logaritması alınarak hesaplanmaktadır. Örneğin CONTROL işlem kodu 100 doküman içerisinde geçiyorsa ve toplamda işlem gören 1000 adet doküman varsa CONTROL işlemin kodunun IDF değeri 100/1000=10 ve buradan log(10)=1 olarak bulunacaktır.

Benzer şekilde TF-IDF öznitelik değeri (4.3)'de verilen denkleme göre üzere TF ve IDF değerlerinin çarpım sonucu olduğu için daha önce hesaplanmış TF ve IDF vektörlerindeki değerlerin çarpımı bize TF-IDF değerini verecektir. Örneğin CONTROL işlem kodu için daha önce TF değeri 0.5 ve IDF değeri 1 olarak bulunmuştu buradan CONTROL işlem koduna ait TF-IDF değeri 0.5*1=1 olduğu söylenecektir.

$$tf(term, doc) = \frac{\mathrm{f(term, doc)}}{\mid \mathrm{f(term, doc): term} \in \mathrm{doc} \mid} \tag{4.1}$$

$$idf(term, D) = \log \frac{|\mathrm{D}|}{\mid \mathrm{doc} \in \mathrm{D} : \mathrm{term} \in \mathrm{doc} \mid} \tag{4.2}$$

$$tf - idf\ (term, doc, D) = tf(term, doc)\ x\ idf(term, D) \tag{4.3}$$

4.3.5. Öznitelik Seçim Yöntemleri

Oluşturulan öznitelik vektörleri için öznitelik seçim işlemi ile değerli öznitelikler seçilerek gereksiz, sınıflandırmaya katkısı olmayan öznitelikler filtrelenerek öznitelik sayısı daha da azaltılması ve başarımın artırılması hedeflenmiştir. Bu amaçla karar ağaçlarına dayalı Bilgi Kazanımı (InformationGain), Kazanç Oranı (GainRatio) algoritmalarıyla, yapay sinir ağları temelli Destek Vektör Makinaları (Support Vector Machine (SVM)), Destek (ReliefF) algoritmaları yanı sıra Ki-Kare Dağılımı (ChiSquare) öznitelik seçim algoritmaları ile oluşturulan TF, IDF-

TF-IDF öznitelik kümeleri üzerinde değerli özniteliklerin seçimi ve sistemin başarımı üzerindeki etkisi araştırılmıştır.

4.3.6. Uygun Sınıflandırma Algoritmasının Seçimi

MAIL diline ait işlem kodların TF, IDF, TF-IDF öznitelik vektörleri için en uygun sınıflandırma algoritmalar araştırılmıştır. Bu amaçla, farklı yaklaşımlar ve yöntemler uygulayan sınıflandırma algoritmalarından weka ortamında tanımlı örnekler seçilerek sistemin eğitiminde kullanılarak sistem için uygunluğu test edilmiştir. Bu amaçla Karar Ağaçları (C4.5), NaiveBayes, En Yakın Komşu (KNN), Rastgele Orman (Random Forest), Destek Vektör Makinaları (Support Vektör Machine (SVM)), K-Yıldız (KSTAR), Biçimsel (LOGISTCS) sınıflandırma algoritmaları geliştirilen sistemin sınıflandırma başarımı üzerinde test edilecek algoritmalar olarak belirlendi.

4.3.7. Sistemin Başarım Ölçüm Metrikleri

Model için uygun öznitelik çıkarma, öznitelik seçme ve sınıflandırma algoritmalarının belirlenmesi için modelin başarım oranı 10 çapraz doğrulama yöntemi kullanılarak her defasında farklı test ve eğitim veri seti ile ölçülmektedir. 10 çapraz doğrulama yönteminden amaç sistemdeki her bir değerin test ve eğitim verisi olarak kullanılarak sistemin farklı farklı eğitim ve test kümeleriyle doğruluk değerini ölçmektir.

Sitemin başarımı için ölçüm metriği olarak (4.4)'te belirtilen doğru pozitif oranı TPR (True Positive Rate) ve (4.5)'te verilen yanlış pozitif oranı FPR (False Positive Rate) denklemlerindeki hesaplama yöntemleri kullanılmıştır. Doğru pozitif oranı TPR değeri, sınıfı kötücül olan kodlardan kötücül olarak tespit edilenlerin sayısının -doğru pozitif sayısının TP'nin- yine TP ile kötücül olan fakat zararsız olarak etiketlenen -yanlış negatif FN- sayısının toplamına bölümü ile elde edilir. Benzer şekilde yanlış pozitif oranı FPR değeri, zararsız kodların zararsız olarak etiketlenen -yanlış pozitif FP- sayısının, FP ile zararsız kodların zararlı olarak etiketlenen -doğru negatif TN- sayısının toplamına bölümü ile elde edilir.

$$TPR = TP/(TP + FN) \quad (4.4)$$

$$FPR = FP/(TN + FP) \quad (4.5)$$

5. DENEYSEL ÇALIŞMALAR VE BULGULAR

Çalışmalar Intel Core i5-4570 CPU 3.2 Ghz işlemci ve 4Gb bellek üzerinde çalışan 64 bit Ubuntu işletim sistemi üzerinde gerçekleştirilmiştir. Öznitelik seçimi, sistemin eğitilmesi ve sınıflandırma algoritmaları için veri madenciliği ve örüntü tanıma çalışmalarında sıkça kullanılan weka kütüphanesi kullanılmıştır. Sınıflandırıcıların çekirdek fonksiyonları ve parametreleri için weka kütüphanesinde varsayılan olarak tanımlı değerler kullanıldı.

5.1. Veri Toplama

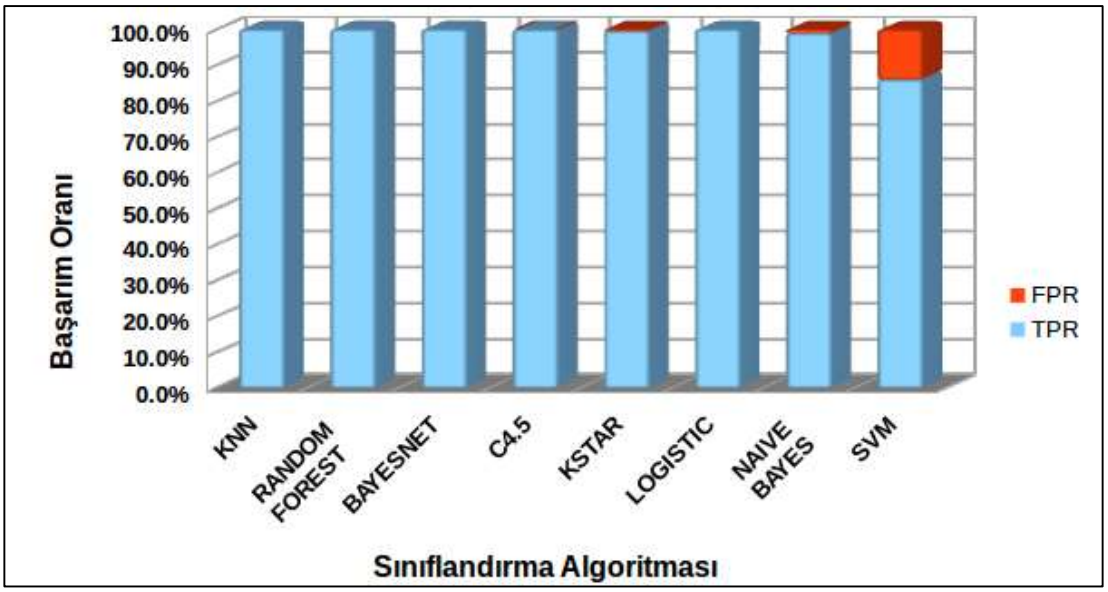
Veri seti olarak Tablo 5.1: Kötücül ve zararsız kod veri setinde dağılımı açıklanan 944 adet metamorfik kötücül kod ve 509 adet zararsız kod örneği kullanılmıştır. Kötücül kodlar VXHeaven (http://vxheaven.org/) sanal yöresinde yer alan daha önceki [4], [35], [36], [19], [37], [38], [39], [40], [41] çalışmalarda kullanılan NGVCK, G2, VLC32, PSMPC kötücül kod üretme araçları kullanılarak üretilmiştir. Zararsız kodlar windows 7 işletim sistemi program dosyalarından ve Cygwin öykünücüsünden derlenmiştir.

Tablo 5.1: Kötücül ve zararsız kod veri seti.

Kötücül Kod Kümesi				**Zararsız Kod Kümesi**	
NGVCK	G2	VLC32	PSMPC	CYGWIN	WIN7
294	250	200	200	313	196

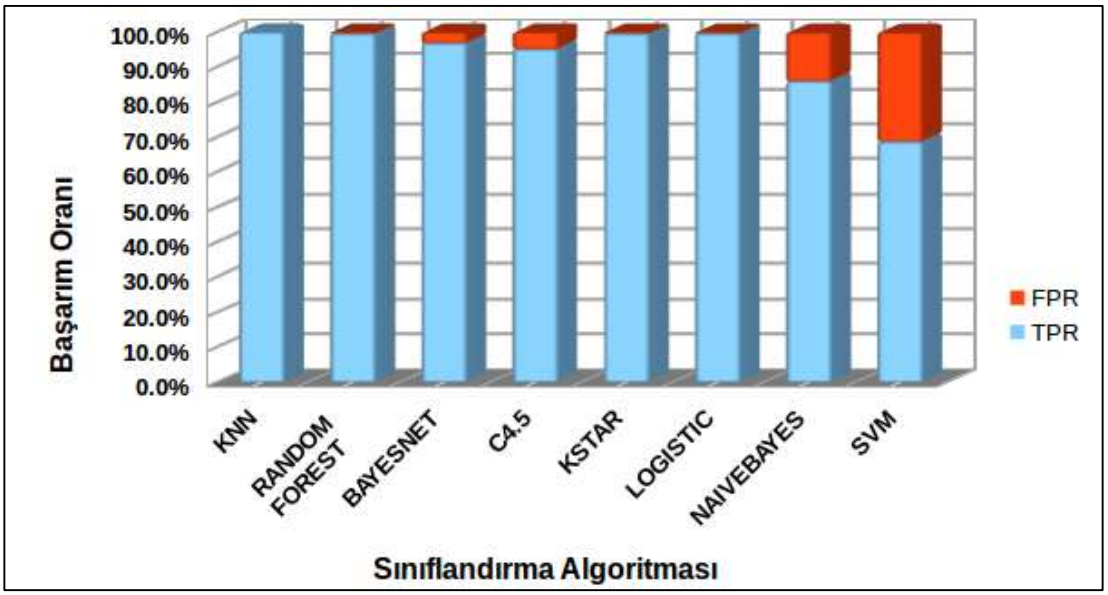
5.2. Öznitelik Çıkarımı ve Sınıflandırma

Öznitelik Çıkarımı ve Sınıflandırma Şekil 5.1: Terim frekans (TF) öznitelik çıkarma yönteminin başarım grafiğinde görüleceği üzere öznitelik çıkarma yöntemi olarak MAIL diline ait 21 adet işlem kodun (cümlenin) TF değeri kullanıldığında KNN, Random Forest, Bayesnet, Logistic sınıflandırma algoritmaları %100 tespit başarımında bulunmuştur. Bununla birlikte en kötü sınıflandırma algoritması %86 başarım oranı ile SVN olmuştur. Yine sırasıyla C4.5 %99.9, KStar %99.5 ve Naivebayes %98.5 kötücül kod tespit başarım oranına sahip olmuştur.

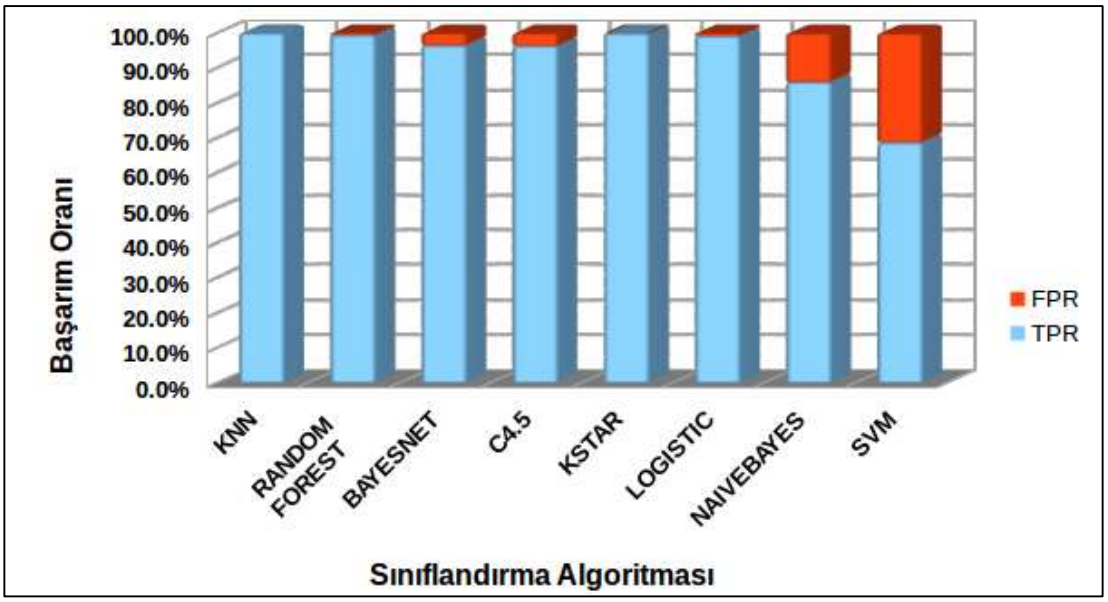


Şekil 5.1: Terim frekans (TF) öznitelik çıkarma yönteminin başarım grafiği.

Şekil 5.2: Ters doküman (IDF) öznitelik çıkarma yöntemi başarım grafiğinde görüleceği üzere öznitelik çıkarma yöntemi olarak MAIL diline ait 21 adet işlem kodun (cümlenin) IDF değeri kullanıldığında KNN algoritması %100 tespit başarımında bulunmuştur. Bununla birlikte en kötü sınıflandırma algoritması %68.7 başarım oranı ile SVN olmuştur. Yine sırasıyla Random Forest %99.6, Logistic %99.6, KStar %99.6, BayesNet %96.9, C4.5 %95.2 ve Naivebayes %86 kötücül kod tespit başarım oranına sahip olmuştur.



Şekil 5.2: Ters doküman (IDF) öznitelik çıkarma yöntemi başarım grafiği.



Şekil 5.3: TF-IDF öznitelik çıkarma yöntemi başarım grafiği.

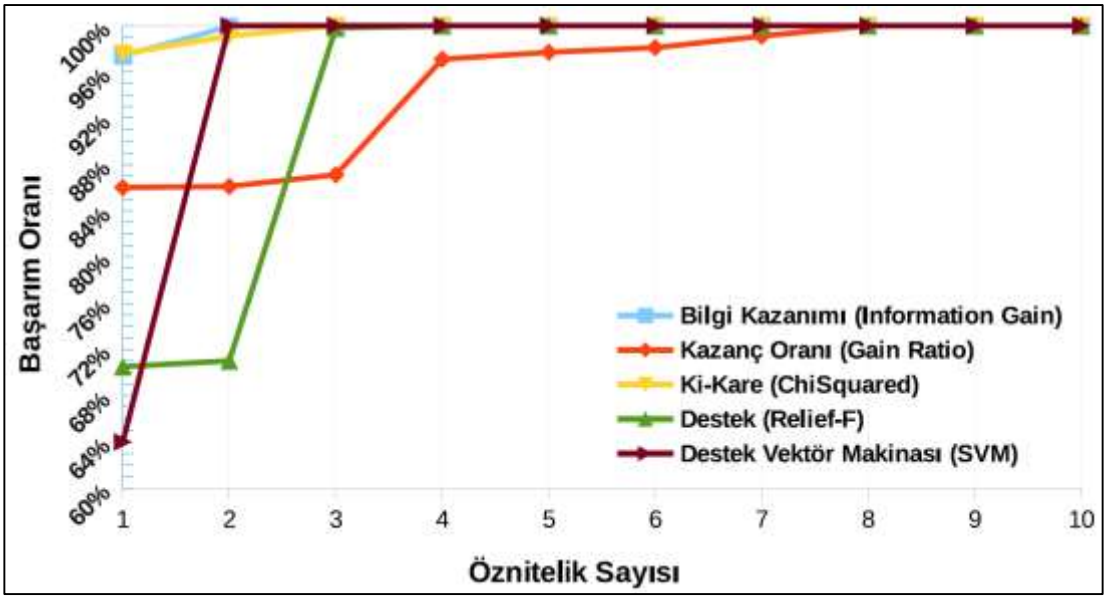
Şekil 5.3: TF-IDF öznitelik çıkarma yöntemi başarım grafiğinde görüleceği üzere öznitelik çıkarma yöntemi olarak MAIL diline ait 21 adet işlem kodun (cümlenin) TF-IDF değeri kullanıldığında KNN algoritması %100 tespit başarımında bulunmuştur. Bununla birlikte en kötü sınıflandırma algoritması %68.7 başarım oranı ile SVN olmuştur. Yine sırasıyla KStar %99.6, Random Forest %99.4, Logistic %99.1, BayesNet %96.5, C4.5 %96.4 ve Naivebayes %86 kötücül kod tespit başarım oranına sahip olmuştur. Bu sonuçlar TF-IDF, TF ve IDF çarpım sonucu olduğu için beklenen sonuçlardır. IDF değerinden gelen değerler TF'in başarım oranını düşürmüştür.

Tablo 5.2: Öznitelik vektörleri ˙için sınıflandırma algoritmalarının doğruluk değerleri.

Sınıflandırma Algoritması	**Öznitelik Çıkarma Yöntemi ve Başarım oranları**					
	TF		**IDF**		**TF-IDF**	
	TPR	**FPR**	**TPR**	**FPR**	**TPR**	**FPR**
KNN	100.0%	0.0%	100.0%	0.0%	100.0%	0.0%
RandomForest	100.0%	0.0%	99.6%	0.4%	99.4%	0.6%
BayesNet	100.0%	0.0%	96.9%	3.1%	96.5%	3.5%
Logistic	100.0%	0.0%	99.6%	0.4%	99.1%	0.9%
C4.5	99.9%	0.1%	95.2%	4.8%	96.4%	3.6%
Kstar	99.5%	0.5%	99.6%	0.4%	99.9%	0.1%
Naivebayes	98.9%	1.1%	86.0%	14.0%	86.0%	14.0%
SVM	86.0%	14.0%	68.7%	31.3%	68.7%	31.3%

MAIL diline ait cümlelerin TF, IDF, TF-IDF değerleri ile oluşturduğumuz öznitelik vektörleri kötücül kod tespitinde Tablo 5.2: Öznitelik vektörleri ˙için sınıflandırma algoritmalarının doğruluk değerleri tablosunda sonuçlar verildiği üzere Karar Ağaçları (C4.5), NaiveBayes, En Yakın Komşu (KNN), Rastgele Orman (Random Forest), Destek Vektör Makinaları (Support Vektör Machine (SVM)), K-Yıldız (KSTAR), Biçimsel (LOGISTCS) sınıflandırma algoritmaları ile %68.7 ile %100 arasında değişen oranlarda başarım göstermiştir. Deneyim sonuçlarına göre en yakın komşuluk algoritması KNN her üç öznitelik değeri için %100 başarım göstererek oluşturulan model için en uygun sınıflandırma algoritması olarak saptanmıştır. Bununla birlikte Randomforest, Bayesnet, Logistic sınıflandırma algoritmaları da model için kullanılmaya aday diğer algoritmalardır. TF özniteliği, IDF, TF-IDF özniteliklere göre daha başarılı sonuçlar vererek oluşturulan model için en uygun öznitelik çıkarma yöntemi olarak belirlenmiştir. Bununla birlikte sınıflandırma algoritması olarak KNN kullanılması durumunda IDF, TF-IDF öznitelikleri model için öznitelik çıkarma yöntemi olarak kullanılmaya aday diğer yöntemlerdir.

5.3. Öznitelik Seçimi ve Sınıflandırma



Şekil 5.4: Öznitelik seçim algoritmaları başarım oranları grafiği.

Öznitelik seçim yöntemlerinin başarım oranına etkisini araştırmak amacıyla Bilgi Kazanımı (InformationGain), Kazanç Oranı (GainRatio), Ki-Kare Dağılımı

(ChiSquare), Destek Vektör Makinaları (Support Vector Machine (SVM)), Destek (ReliefF) öznitelik seçim algoritmaları MAIL dili üzerinden elde edilen cümlelere ait TF öznitelik kümesi üzerinde kullanılmıştır. Öznitelik seçim algoritmalarının öznitelik sayısına göre başarım oranına etkisi Şekil 5.4: Öznitelik seçim algoritmaları başarım oranları grafiğinde ve Tablo 5.3: Öznitelik seçim algoritmaları başarım oranları tablosunda verilmiştir. Kullanılan algoritmaların tümü sınıflandırma algoritması KNN olduğu durumda %100 oranında başarılı sınıflandırma yapmıştır. Özellikle SVM algoritması 1 adet öznitelikle en düşük başarım oranına sahip algoritma iken seçtiği 2 adet öznitelikle %100 başarım sağlayarak sistemin geliştirilen sistem için gerçek zamanlı tespit sağlayacak küçüklükte öznitelik sayısına ulaşılmasını sağlamıştır. Information gain ve chisquared algoritmaları Relief-F algoritmasına göre daha başarılı olurken, en kötü öznitelik seçim algoritması olarak gain ratio gözlenmiştir. Gain ratio algoritması MAIL diline ait 7 adet işlem kodun TF öznitelik değeri ile %100 tespit başarımı sağlayabilmiştir. Öznitelik seçim algoritmalarının %100 başarım sağlayan öznitelik grupları ve sayısı ayrıntılı olarak Tablo 5.3'de verilmiştir.

Tablo 5.3: Öznitelik seçim algoritmaları başarım oranları.

Öznitelik Sayısı	**Information Gain**	**Gain Ratio**	**ChiSquared**	**Relief-F**	**SVM**
1	97,40%	86,00%	97,60%	70,50%	64,00%
2	99,93%	86,10%	99,10%	71,00%	100,00%
3	99,93%	87,10%	100,00%	99,80%	100,00%
4	100,00%	97,10%	100,00%	100,00%	100,00%
5	100,00%	97,70%	100,00%	100,00%	100,00%
6	100,00%	98,10%	100,00%	100,00%	100,00%
7	100,00%	99,10%	100,00%	100,00%	100,00%
8	100,00%	100,00%	100,00%	100,00%	100,00%
.	100,00%	100,00%	100,00%	100,00%	100,00%
.	100,00%	100,00%	100,00%	100,00%	100,00%
21	100,00%	100,00%	100,00%	100,00%	100,00%

Tablo 5.4: %100 başarım için seçilen değerli öznitelikler.

Öznitelik Seçim Algoritması	Öznitelik Sayısı	Öznitelikler
SVM	2	JUMP_C, TEST_C
Chi-Squared	3	JUMP_C, ASSIGN, CALL_C
Information Gain	4	JUMP_C, ASSIGN, CALL_C, STACK
Relief-F	4	ASSIGN, CALL_C, ASSIGN_C, TEST_C,
GainRatio	8	CALL_C, CALL, JUMP_S, CONTROL_C, JUMP, TEST, CONTROL, STACK_C

Tablo 5.4: %100 başarım için seçilen değerli öznitelikler incelendiğinde MAIL diline JUMP_C işlem kodunun neredeyse bütün öznitelik algoritmaları tarafından değerli öznitelik olarak seçildiği gözlenmiştir. Metamorfik kötücül kodlar için en etkili başkalaşım yönteminin kötücül koda ait kod bloklarının yerlerini değiştirmek olduğu düşünüldüğünde ve JUMP_C işlem kodunun MAIL dilinde kontrol işlemlerini tanımlamak için kullanıldığı göz önüne alındığında MAIL kodunun kötücül kodlara ait kontrol akış işlemlerini kendi dili içerisinde iyi şekilde temsil ettiği söylenebilir. Bununla birlikte SVM tarafında iki değerli öznitelikten biri olarak hesaplanan TEST işlem kodu beklenmedik bir sonuçtur. TEST assembly kodunun AND işlem koduyla aynı işlevi görmektedir. Komutları eş değeri ile değiştirme metamorfik yöntemi tarafından TEST işlem kodunun AND yerine AND işlem kodunu TEST yerine kullanılarak kod yapısının değiştirilmesi TEST kodunu ayırt ediciliği yüksek değerli özniteliklerden kıldığı düşünülmektedir.

Bunun yanı sıra seçilen diğer ayırt ediciliği yüksek değerli öznitelikler ASSING, ASSIGN_C, CALL_C, STACK, STACK_C, TEST_C, CONTROL, CONTROL_C MAIL işlem kodlarıdır. Listeden görüleceği üzere önemli öznitelikler genelde atama, çağırma, kontrol işlemlerini yürüten yani metamorfik başkalaşım yöntemleri tarafından sıkça kullanılan işlem kodların MAIL dilindeki karşılıklarıdır.

Yaklaşık 900 kadar assembly işlem kodu üzerinden yapılan çalışmalar MAIL dilinde en fazla 8 adet işlem kod ile temsil edilmektedir. MAIL diline işlem kodları kullanan çalışmalar öznitelik olarak OpCode değerlerini kullanan göre başarımı daha yüksek sonuçlar elde edilebilmektedir. Bununla birlikte MAIL dili TF hesaplama yönteminin öznitelik çıkarma, SVM algoritmasının öznitelik seçme, KNN

algoritmasının sınıflandırma, yöntemi olarak kullanıldığı durumda 2 adet öznitelikle gerçek zamanda tespit başarımına ulaşılmıştır.

Çalışma kapsamında geliştirdiğimiz yöntemin ilgili çalışmalar kısmında açıklanan benzer veri seti üzerinde çalışan diğer çalışmalarla veri seti büyüklüğü ve başarım oranlarına göre karşılaştırması Tablo 5.5: Metamorfik kötücül kod tanıma yöntemleri başarım oranları tablosunda verilmiştir. Tabloda görüleceği üzere çalışma kapsamında geliştirilen yöntem %100 başarım elde eden diğer yöntemlere göre daha büyük ve daha fazla çeşitliliğe sahip veri seti üzerinde test edilmiştir. Geliştirdiğimiz yöntem MAIL dili sayesinde platform, derleyici bağımsızdır. Çalışma zamanı açısından 900 kadar öznitelikle kötücül kod tespiti yapan diğer yöntemlere göre 2 adet öznitelikle kötücül kod tespitinde bulunduğu için daha yüksek performansa sahiptir. Aynı zamanda yanlış pozitif oranı diğer çalışmalara göre daha düşüktür. MAIL dil yapısını kullanan diğer çalışmalar MAIL-CFG, MARD, SWOD çalışmalarında kötücül kod tanıma yöntemi MAIL diline ait kontrol akış bloklarının çizge tabanlı karşılaştırması üzerine kurulmuştur. Bununla birlikte çalışmamız kapsamında bu dil yapısına ait cümlelerin (işlem kod) karşılaştırması üzerinden yöntem geliştirilmiş olup bu yöntemin diğer çalışmalara oranla daha yüksek başarım elde ettiği gözlenmiştir.

Tablo 5.5: Metamorfik kötücül kod tanıma yöntemleri başarım oranları tablosu.

Sistem	**Tarih**	**Yazar**	**Başarım Oranı**		**Veri Kümesi**	
			TPR	**FPR**	**Zararsız**	**Kötücül**
Opcode-HMM	2006	Wong	90%	2%	40	200
Chi-Squared	2012	Toderici	98%	2%	40	200
Opcode-Graph	2012	Runwal	100%	1%	41	200
Histogram	2013	Rad	100%	0%	40	60
Opcode-SD	2013	Shanmugam	98%	0.5%	40	800
Opcode-HMM	2013	Auistin	93.5%	0.5%	102	77
MAIL-CFG	2013	Alam	98.9%	4,50%	2330	1020
MARD	2014	Alam	99.2%	3.07%	1137	250
SWOD	2015	Alam	99.08%	0.93%	1101	250
MAIL-Frekans	**2016**	Çarkacı	**100%**	**0%**	**537**	**920**

6. SONUÇ

Son dönemlerde polimorfik ve metamorfik kötücül kod tespitinde örüntü tabanlı, makina öğrenmesi yaklaşımlarını kullanan çalışmalarla başarılı sonuçlar elde edilmeye başlanmıştır. Bununla birlikte öznitelik çıkarımı, öznitelik kümesinin büyüklüğü gibi nedenlerden dolayı gerçek zamanlı metamorfik kötücül kod tespiti yapacak bir yöntem henüz tam olarak geliştirilmemiştir. Çalışmamızda geliştirdiğimiz yöntem metamorfik kötücül kodları 21 adet MAIL özet kodu içerisinden 2 adet değerli öznitelik seçerek %90 oranında öznitelik azaltımı ile %100 oranında başarımla tespit etmiştir. Yöntem için öznitelik çıkarma yönteminde TF, öznitelik seçme yönteminde SVM, sınıflandırıcı olarak KNN en iyi sonuçları veren algoritmalar olmuştur. Bununla birlikte SVM ve KNN algoritmaları için farklı çekirdek fonksiyonların ve parametrelerin başarım üzerindeki etkisi gelecek araştırma konuları arasındadır.

KAYNAKLAR

[1] Web1,(2016), https://www.av-test.org/en/statistics/malware/, (Erişim Tarihi: 20/06/2016).

[2] Szor P., (2005), "The Art of Computer Virus Research and Defense", 1st Edition, Addison-Wesley Professional.

[3] Web2,(2016),https://home.mcafee.com/virusinfo/virusprofile.aspx?key=921255, (Erişim Tarihi: 20/06/2016).

[4] Wong W., Stamp M., (2006), "Hunting for metamorphic engines", Journal of Computer Virology and Hacking Technique, 2 (3), 211-229.

[5] Brunner E. M., Suter M., (2009), "Uluslararası Kritik Bilgi ve Altyapıların Korunması El Kitabı 2008-2009", 1.Baskı, Güvenlik Çalışmaları Merkezi, İsviçre.

[6] Web3,(2016),https://securelist.com/analysis/kaspersky-securitybulletin/73038/kaspersky-security-bulletin-2015-overall-statistics-for-2015/, (Erişim Tarihi: 20/06/2016).

[7] Web4,(2016), http://www.reuters.com/article/2013/09/17/us-cyberattacks-china-idUSBRE98G0M720130917, (Erişim Tarihi: 20/06/2016).

[8] Clarke R. A., Knake R. K., (2011), "Siber Savaş", 1. Baskı, İstanbul Kültür Üniversitesi / Bilim Dizisi, İstanbul, Türkiye.

[9] Alkan M., (2012), "Siber Güvenlik ve Siber Savaşlar", Siber Güvenlik Siber Savaşlar TBMM internet Komisyonu.

[10] You I., Yim K., (2010), "Malware Obfuscation Techniques: A Brief Survey", Fifth International Conference on Broadband, Wireless Computing, Communication and Applications (BWCCA 2010), 297-300, Fukuoka, Japan, 4-6 November.

[11] Rad B. B., Masrom M., Ibrahim S., (2012), "Camouflage in Malware: from Encryption to Metamorphism", IJCSNS International Journal of Computer Science and Network Security, 12 (8), 74-83.

[12] Ying C., Qiguang M., Jiachen L., Lin G., (2013), "Abstracting minimal security-relevant behaviors for malware analysis", Journal of Computer Virology and Hacking Techniques, 9 (4), 193-204.

[13] Rad B. B., Masrom M., (2010), "Metamorphic Virus Variants Classification Using Opcode Frequency Histogram", 14th WSEAS International Conference on COMPUTERS (ICCOMP 2010), 147-155, Sofia, Bulgaria, 23-25 July.

[14] Gao X., Stamp M., (2005), "Metamorphic Software for Buffer Overflow Mitigation", 3rd Conference on Computer Science and its Applications, San Diego, California, USA, 28-30 June.

[15] Schultz M. G., Eskin E., Zadok E., Stolfo S. J., (2001), "Data Mining Methods for Detection of New Malicious Executables", IEEE Symposium on Security and Privacy (SP '01), IEEE Computer Society, Washington, DC, USA, 14-16 May.

[16] Abou-Assaleh T., Cercone N., Keselj V., Sweidan R., (2004), "N-gram-based detection of new malicious code", 28th Annual International Computer Software and Applications Conference (COMPSAC '04), 41-42, Washington DC, USA.

[17] Zhang B., Yin J., Hao J., Zhang D., Wang S., (2007), "Malicious codes detection based on ensemble learning". In Xiao B., Yang L. T., Ma J., Muller-Schloer C., Hua Y, Editors, "4th international conference on Autonomic and Trusted Computing (ATC'07) ", 468-477, Springer-Verlag, Berlin, Heidelberg.

[18] Yanfang Y., Tao L., Yong C., Qingshan J., (2010), "Automatic malware categorization using cluster ensemble", 16th ACM SIGKDD international conference on Knowledge discovery and data mining (KDD '10), 95-104, ACM, New York, NY, USA,.25-28 July.

[19] Rad B. B., Masrom M., Ibrahim S., (2012), "Opcodes histogram for classifying metamorphic portable executables malware", International Conference on E-Learning and E-Technologies in Education (ICEEE 2012), 209-213, Lodz, Poland, 24-26 September.

[20] Stolfo S., Wang S., Li W., (2007), "Towards stealthy malware detection", In: Somesh C. M., Douglas J., Dawn M., Cliff S. W., Editors, "Malware detection", Springer, USA.

[21] Siddiqui M., Wang MC., Lee J., (2008), "Data mining methods for malware detection using instruction sequences", 26th IASTED international conference on artificial intelligence and applications, 358-363, Anaheim, CA, USA, 11-13 February.

[22] Armoun S. E., Hashemi S., (2012), "A General Paradigm for Normalizing Metamorphic Malwares", 10th International Conference on Frontiers of Information Technology (FIT '12), Islamabad, Pakistan, 1-19 December.

[23] Leder F., Steinbock B., Martini P., (2009), "Classification and detection of metamorphic malware using value set analysis", 4th International Conference on Malicious and Unwanted Software (MALWARE 2009), 39-46, Montreal, Quebec, Canada, 13-14 October.

[24] Runwal N., Low R. M., Stamp M., (2012), “Opcode graph similarity and metamorphic detection”, Journal of Computer Virology and Hacking Technique, 8 (2), 37-52.

[25] Alam S., Horspool R. N., Traore I., (2013), “MAIL: Malware Analysis Intermediate Language - A Step Towards Automating and Optimizing Malware Detection”, 6th International Conference on Security of Information and Networks (SIN 2013), Aksaray, Türkiye, 26-28 November.

[26] Carsten W., Thorsten H., Felix F., (2007), “Toward Automated Dynamic Malware Analysis Using CWSandbox”, Journal IEEE Security and Privacy, 5 (2), 32-39.

[27] Ahmadi M., Sami A., Rahimi H., Yadegari B., (2013), “Malware detection by behavioural sequential patterns”, Computer Fraud & Security, 2013 (8), 11-19.

[28] Lee J., Jeong K., Lee H., (2010), “Detecting metamorphic malwares using code graphs”. 2010 ACM Symposium on Applied Computing (SAC '10), Sierre, Switzerland, 22 - 26 March.

[29] Zhang B., Yin J., Hao J., (2006), “Using RS and SVM to detect new malicious executable codes”, First international conference on Rough Sets and Knowledge Technology, Chongqing, China, 24-26 July.

[30] Wagener G. , State R. , Dulaunoy A., (2008), “Malware behaviour analysis”, Journal of Computer Virology and Hacking Techniques, 4 (4), 279–87.

[31] Tsyganok K., Tumoyan E., Babenko L., Anikeev M., (2012), “Classification of polymorphic and metamorphic malware samples based on their behavior”. Fifth International Conference on Security of Information and Networks (SIN '12), 111-116, Jaipur, India, 22-27 October.

[32] Burguera I., Zurutuza U., Nadjm-Tehrani S., (2011), “Crowdroid: behaviorbased malware detection system for android”. 1st ACM workshop on Security and privacy in smartphones and mobile devices (SPSM ’11), 15–26, Chicago, IL, USA, 17-21 October.

[33] Xue J., Hu C., Wang K., Ma R., Zou J., (2009), “Metamorphic malware detection technology based on aggregating emerging patterns”. 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human (ICIS '09), Seoul, Korea, 24-26 November.

[34] Issa A., (2012), “Anti-virtual machines and emulations”, Journal of Computer Virology and Hacking Techniques,, 8 (4), 141-149.

[35] A. Toderici, M. Stamp., (2013), “Chi-squared Distance and Metamorphic Virus Detection”, Journal of Computer Virology and Hacking Techniques,, 9 (1), 1-14.

[36] Runwal N., Low R. M., Stamp M., (2012), "Opcode graph similarity and metamorphic detection", Journal of Computer Virology and Hacking Techniques,, 8 (2), 37-52.

[37] G. Shanmugam, R. M. Low, M. Stamp., (2013), "Simple substitution distance and metamorphic detection", Journal of Computer Virology and Hacking Techniques, 9 (3),159-170.

[38] Alam S., Horspool R. N., Traore I., (2014), "MARD: A Framework for Metamorphic Malware Analysis and Real-Time Detection", 28th International Conference on Advanced Information Networking and Applications, 480-489, Victoria, BC, Canada, 13-16 May.

[39] Alam S., Horspool R. N., Traore I., Sogukpinar İ., (2015), "A framework for metamorphic malware analysis and real-time detection", Computers & Security, 48 (2), 212-233.

[40] Austin T. H., Filiol E., Josse S., Stamp M., (2013), "Exploring Hidden Markov Models for Virus Analysis: A Semantic Approach.", 46th Hawaii International Conference on System Sciences (HICSS), 5039-5048, Washington, DC, USA, 7-10 January.

[41] Alam S., Traore I., Sogukpinar İ., (2015), "Annotated Control Flow Graph for Metamorphic Malware Detection", The Computer Journal, 58 (10), 2608-2621.

ÖZGEÇMİŞ

Lisans eğitimi İstanbul Bilgi Üniversitesi Bilgisayar Bilimleri Bölümünde tamamlayan Necmettin ÇARKACI, bir süre özel ve kamu sektöründe çalıştıktan sonra Gebze Teknik Üniversitesinde Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Bölümünde kötücül kod tespit yöntemleri üzerine araştırmacı olarak çalışmaya başladı. Halen aynı üniversitede metamorfik ve polimorfik kötücül kod tespit sistemleri üzerine çalışmalarına devam etmektedir.