GENETİK PROGRAMLAMA KULLANILARAK MOBİL ZARARLI YAZILIMLARIN OTOMATİK OLARAK ÜRETİLMESİ

AUTOMATIC GENERATION OF MOBILE MALWARES USING GENETIC PROGRAMMING

EMRE AYDOĞAN

YRD. DOÇ. DR. SEVİL ŞEN AKAGÜNDÜZ

Tez Danışmanı

Hacettepe Üniversitesi

Lisansüstü Eğitim-Öğretim ve Sınav Yönetmeliğinin

Bilgisayar Mühendisliği Anabilim Dalı için Öngördüğü

YÜKSEK LİSANS TEZİ olarak hazırlanmıştır

2014

EMRE AYDOĞAN'nın hazırladığı "**Genetik Programlama Kullanılarak Mobil Zararlı Yazılımların Otomatik Olarak Üretilmesi**" adlı bu çalışma aşağıdaki jüri tarafından **BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI**'nda **YÜKSEK LİSANS TEZİ** olarak kabul edilmiştir.

Başkan
Doç. Dr. Ebru SEZER . . . . . .

Danışman
Yrd. Doç. Dr. SEVİL ŞEN . . . . . .

Üye
Yrd. Doç. Dr. HARUN ARTUNER . . . . . .

Üye
Yrd. Doç. Dr. Erhan MENGÜŞOĞLU . . . . . .

Üye
Yrd. Doç. Dr. MURAT AYDOS . . . . . .

Bu tez Hacettepe Üniversitesi Fen Bilimleri Enstitüsü tarafından **YÜKSEK LİSANS TEZİ** olarak onaylanmıştır.

Prof. Dr. Fatma SEVİN DÜZ
Fen Bilimleri Enstitüsü Müdürü

ETİK

Hacettepe Üniversitesi Fen Bilimleri Enstitüsü, tez yazım kurallarına uygun olarak hazırladığım bu tez çalışmasında;

- tez içindeki bütün bilgi ve belgeleri akademik kurallar çerçevesinde elde ettiğimi,
- görsel, işitsel ve yazılı tüm bilgi ve sonuçları bilimsel ahlak kurallarına uygun olarak sunduğumu,
- başkalarının eserlerinden yararlanılması durumunda ilgili eserlere bilimsel normlara uygun olarak atıfta bulunduğumu,
- atıfta bulunduğum eserlerin tümünü kaynak olarak gösterdiğimi,
- kullanılan verilerde herhangi bir değişiklik yapmadığımı,
- ve bu tezin herhangi bir bölümünü bu üniversite veya başka bir üniversitede başka bir tez çalışması olarak sunmadığımı

beyan ederim.

26/09/2014

EMRE AYDOĞAN

ÖZET

GENETİK PROGRAMLAMA KULLANILARAK MOBİL ZARARLI YAZILIMLARIN OTOMATİK OLARAK ÜRETİLMESİ

Emre AYDOĞAN

Yüksek Lisans, Bilgisayar Mühendisliği

Danışman: Yrd. Doç. Dr. Sevil ŞEN AKAGÜNDÜZ

Ağustos 2014, 82 sayfa

Son yıllarda mobil cihazların kullanımı önemli ölçüde artış göstermiştir. Bu akıllı cihazlar, e-posta okuma ve yazmadan, internetten gezinmeye kadar birçok kullanışlı işlevi zaman ve yer kısıtlaması olmaksızın kolay bir şekilde yapabilmemize olanak sağlamaktadır. Bu sebeplerden dolayı, mobil cihazlar günlük yaşantımızın vazgeçilmez bir parçası olmuşlardır. Bu akıllı cihazlar, mobil cihazlara zarar vermek için virüs geliştiricilerin de ilgisini çekmiştir. Son yıllarda, birçok yeni mobil zararlı yazılım ortaya çıkmıştır. Bu nedenle, bu gibi tehditlerden mobil cihazlarımızı korumak için güvenlik şirketleri şimdiden yeni çözüm önerileri geliştirmişlerdir. Fakat bu çözümlerin, mobil cihazların kısıtlı kaynakları ile bilinmeyen zararlı yazılımları tespit edebilmesi oldukça zor ve araştırılması gereken bir konudur. Bu çalışmada, varolan zararlı yazılımlardan genetik programlama yöntemi kullanılarak yeni, evrimleşmiş zararlı yazılımlar geliştirilmesi ve hali hazırdaki antivirüs sistemlerinin bu bilinmeyen zararlı yazılımların tespitine yönelik etkinliği değerlendirilecektir. Deney sonuçları, uygulama marketlerinde bulunan statik analiz yöntemini kullanan antivirüs çözümlerinin zayıflığını ve mobil cihazlar için yeni tespit etme yöntemlerinin gerekliliğini göstermektedir.

Anahtar Kelimeler: mobil zararlı yazılımlar, statik analiz, karıştırma, evrimsel hesaplama, genetik programlama

ABSTRACT

AUTOMATIC GENERATION OF MOBILE MALWARES USING GENETIC PROGRAMMING

Emre AYDOĞAN

Master of Science, Computer Engineering Department

Supervisor: Asst. Prof. Dr. Sevil ŞEN AKAGÜNDÜZ

August 2014, 82 pages

The number of mobile devices has increased dramatically in the past few years. These smart devices provide many useful functionalities accessible from anywhere at anytime, such as reading and writing e-mails, surfing on the Internet, showing facilities nearby, and the like. Hence, they become an inevitable part of our daily lives. However the popularity and adoption of mobile devices also attract virus writers in order to harm our devices. So, many security companies have already proposed new solutions in order to protect our mobile devices from such malicious attempts. However developing methodologies that detect unknown malwares is a research challenge, especially on devices with limited resources. This study presents a method that evolves automatically variants of malwares from the ones in the wild by using genetic programming. We aim to evaluate existing security solutions based on static analysis techniques against these evolved unknown malwares. The experimental results show the weaknesses of the static analysis tools available in the market, and the need of new detection techniques suitable for mobile devices.

Keywords: mobile malware, static analysis, obfuscation, evolutionary computation, genetic programming

TEŞEKKÜR

Tez çalışmamın her aşamasında değerli katkı ve eleştiriyle yol gösteren, beni her zaman çalışmaya teşvik eden ve güven veren danışmanım Sayın Yrd. Doç. Dr. Sevil ŞEN AKAGÜNDÜZ'e içtenlikle sonsuz teşekkürlerimi sunarım.

Ayrıca önemli yorum ve değerlendirmeleri ile katkıda bulunan jüri üyelerim Sayın Doç. Dr. Ebru SEZER'e, Sayın Yrd. Doç. Dr. Harun ARTUNER'e, Sayın Yrd. Doç. Dr. Erhan MENGÜŞOĞLU'na ve Sayın Yrd. Doç. Dr. Murat AYDOS'a teşekkür ederim.

Tez çalışmamım her aşamasında manevi olarak yanımda olan ve her türlü desteği esirgemeyen arkadaşlarım Levent KARACAN'a ve Niyazi SIRT'a ve üniversiteden arkadaşlarım Çağdaş BAŞ'a, Aysun KOÇAK'a, Pelin AKTAŞ'a, Cemil ZALLUHOĞLU'na, Handan GÜRSOY'a ve Abdurrahman BAYRAK'a değerli arkadaşlıkları için teşekkürlerimi sunarım. Ayrıca Hacettepe Üniversitesi Bilgi Güvenliği Araştırma Laboratuvarı öğrencileri ve laboratuvar arkadaşlarım Ahmet İlhan AYŞAN'a, Hüseyin Burhan ÖZKAN'a ve Rahem ABRI'ye arkadaşlıkları, destekleri ve keyifli bir çalışma ortamı sundukları için teşekkür ederim.

Bu çalışma Türkiye Bilimsel ve Teknolojik Araştırma Kurumu (TÜBITAK-112E354) tarafından desteklenmiştir. Yüksek lisans eğitimim boyunca verdiği burs ile maddi olarak desteklerinden dolayı TÜBİTAK'a teşekkür ederim.

Son olarak, tüm eğitim hayatım boyunca zorluklara rağmen beni destekleyen, cesaretlendiren ve bana gönülden inanan sevgili aileme tüm kalbimle teşekkür ederim.

İÇİNDEKİLER

Sayfa

ÖZET i
ABSTRACT iii
TEŞEKKÜR v
İÇİNDEKİLER vi
ÇİZELGELER ix
ŞEKİLLER x
SİMGELER VE KISALTMALAR xi
1. GİRİŞ 1
1.1. Motivasyon 1
1.2. Tezin Yapısı 3
2. YAPILAN ÇALIŞMALAR 4
2.1. Karıştırma Teknikleri 4
2.2. Evrimsel Hesaplama 6
2.3. Statik Analiz 10
3. ANDROİD PLATFORMU 13
3.1. Android Platformuna Giriş 13
3.2. Android Platformunda Güvenlik 15
3.2.1 Market Güvenliği 16
3.2.2 Platform Güvenliği 16
3.2.2.1 İzinler 16
3.2.2.2 Kum Havuzunda Çalıştırma 17
3.2.2.3 Uygulamalar Arası Etkileşimler 18
3.2.2.4 Linux Çekirdeği Güvenliği 18
3.3. Android Uygulamaları 18
3.3.1 Android Uygulamalarının Bileşenleri 19
3.3.1.1 Yayın Alıcılar (Broadcast Receiver) 19
3.3.1.2 İçerik Sağlayıcılar (Content Provider) 19

3.3.1.3 Servisler (Services) 20
3.3.1.4 Aktiviteler (Activities) 20
3.3.2 Android Uygulamalarının Derlenmesi 20
3.4. Tersine Mühendislik 21
4. EVRİMSEL HESAPLAMA 23
4.1. Genetik Programlama (GP) 24
4.1.1 Başlangıç Popülasyonu 29
4.1.2 Uygunluk Değerini Hesaplama 31
4.1.3 Genetik İşleçler 32
4.1.3.1 Seçme 32
4.1.3.2 Üreme 33
4.1.3.3 Çaprazlama 34
4.1.3.4 Mutasyon 35
4.1.4 Sonlandırma Kriteri 35
4.1.5 Kontrol Parametreleri 36
5. MODEL VE YÖNTEM 37
5.1. Bireylerin Gösterimi 39
5.2. Genetik İşleçler 40
5.2.1 Çaprazlama 40
5.2.1.1 Dinamik Analiz 41
5.2.2 Mutasyon 42
5.2.2.1 Karıştırma Teknikleri 43
5.3. Uygunluk Değeri 46
6. DENEY SONUÇLARI 48
6.1. Veri Seti 48
6.2. Sonuçlar 49
6.2.1 Evrimleştirilmiş Zararlı Yazılımların Değerlendirilmesi 49
6.2.1.1 Tekli Karıştırma Teknikleri ile Karşılaştırma 50
6.2.1.2 İkili Karıştırma Teknikleri ile Karşılaştırma 51
6.2.2 Statik Analiz Araçlarının Değerlendirilmesi 53

6.2.3 Genetik Programlamanın Avantajları 56
7. SONUÇ 58
KAYNAKLAR 60

ÇİZELGELER

Sayfa

5.1. Genetik programlamada kullanılan parametreler 42
5.2. Deneylerde kullanılan antivirüs sistemleri ve versiyonları 47

6.1. Kullanılan tekniklerin kısaltmaları 49
6.2. Karıştırma teknikleri tek tek uygulandığında antivirüs sistemlerinin başarısı .. 50
6.3. Karıştırma teknikleri ikili gruplar halinde uygulandığında antivirüs sistemlerinin başarısı 52
6.4. Antivirüs sistemlerinin genel başarısı 54

ŞEKİLLER

Sayfa

3.1. Android mimarisi 14
3.2. MalGenome veri setinde bulunan zararlı yazılımların en çok kullandığı 20 izin 17
3.3. Bir Android uygulamasının tersine derlenmiş hali 22

4.1. Genetik programlamanın kavramsal şeması 26
4.2. *grow* metodu ile oluşturulan ağaç örneği 30
4.3. *full* metodu ile oluşturulan ağaç örneği 31
4.4. Çaprazlama uygulamadan önce ağaçların durumu 34
4.5. Çaprazlama uygulandıktan sonra ağaçların durumu 35
4.6. Mutasyon uygulamadan önce ve uygulandıktan sonra ağaç durumu 36

5.1. Kavramsal şema 37
5.2. Örnek kaynak kodu 39
5.3. Şekil 5.2.'de gösterilen kaynak kodu için kontrol akış çizgesi 40
5.4. Kod parçası 43
5.5. Yerel değişkenleri yeniden adlandırma tekniği uygulanmış kod parçası 44
5.6. Gereksiz kod ekleme tekniği uygulanmış kod parçası 44
5.7. Veri şifreleme tekniği uygulanmış kod parçası 45
5.8. İkili kod sırası değiştirme tekniği uygulanmış kod parçası 45
5.9. Üçlü kod sırası değiştirme tekniği uygulanmış kod parçası 46

SİMGELER VE KISALTMALAR

Simgeler

Kısaltmalar

OR	Orijinal
JK	Gereksiz Kod Ekleme
DE	Veri Şifreleme
CR2	İkili Kod Değiştirme
CR3	Üçlü Kod Değiştirme
RI	Yerel Değişkenleri Yeniden Adlandırma
MU	Mutasyon
XO	Çaprazlama
GP	Genetik Programlama

Bölüm 1.

GİRİŞ

1.1. Motivasyon

Son yıllarda mobil cihazların kullanımı önemli ölçüde artmaktadır. Bu cihazlar içinde en çok kullanılan cihazlar tabletler ve akıllı telefonlardır. Bu akıllı cihazlar e-posta yazma ve okuma, internette gezinme, video konferans gibi pek çok işlevselliklerinden dolayı günlük hayatımızın kaçınılmaz bir parçası olmuşlardır. Fakat bu popülerlik ve yüksek kullanım, mobil cihazlara zarar vermek için zararlı yazılım geliştiricilerin de dikkatini çekmiştir. Zararlı yazılımlar; bilgisayarın normal işleyişini bozma, kullanıcıya ait hassas bilgileri izinsiz toplama ve bilgisayar sistemlerine izinsiz olarak erişim hakkı kazanma gibi işlevleri olan bilgisayar yazılımlarına denir. Kaspersky'nin 2014 Ocak ayı güvenlik raporuna göre [1] 2013 yılında 145.000 tane yeni zararlı yazılım ortaya çıkmıştır. Bu rakam 2012 yılında ortaya çıkan zararlı yazılım sayısının 3 katına denk gelmektedir. Bu 145.000 zararlı yazılımın %98.1'i Android platformunu hedef almaktadır. Ayrıca bu zararlı yazılımların bir çoğu zaten varolan zararlı yazılımların üzerinde değişiklikler yapılarak oluşturulmaktadır. Bu sebeple cihazlarımızı bu gibi tehlikelerden korumak ve daha etkili ve verimli antivirüs çözümleri üretmek için araştırmacılar ve güvenlik şirketleri yoğun olarak çalışmaktadırlar.

iOS uygulamalarını içeren App Store ve Android uygulamalarını içeren Google Play gibi uygulama marketleri, kullanıcılara milyonlarca ücretli ve ücretsiz uygulama sunmaktadır. 31 Temmuz 2014 tarihi itibariyle Google Play'de 1.100.000 tanesi ücretsiz, 200.000 tanesi ücretli olmak üzere yaklaşık 1.300.000 tane uygulama bulunmaktadır [2]. Ayrıca Android

platformunda Google Play'den başka Amazon AppStore [3] ve AppBrain [4] gibi onlarca uygulama market bulunmaktadır. Bu marketlerde bulunan yüzbinlerce uygulama Android cihazlara yüklenebilmektedir. 2013 yılında Google Play'den indirilen uygulama sayısı 50 milyarı aşmıştır [5]. Bu yaygın kullanım, Android cihazların güvenliğinin ne kadar önemli olduğunu ortaya koymaktadır.

Zararlı yazılım analiz ve tespit etme işlemi için farklı güçlü ve zayıf noktaları bulunan çeşitli teknikler vardır. Kodun nasıl analiz edildiğine göre, statik analiz ve dinamik analiz olmak üzere 2 tane yaygın zararlı yazılım tespit etme tekniği vardır. Statik analiz yönteminde, yazılımlar tersine mühendislik ile kaynak kod ve dosyalarına dönüştürülür ve bu kaynak kod ve dosyaları üzerinde çeşitli yöntemler uygulanarak yazılımın zararlı olup olmadığına karar verilir. Bu teknikte, kodun çalıştırılmasına gerek yoktur. Dinamik analiz yönteminde ise yazılım bir emulatör veya gerçek makine üzerinde çalıştırılır ve çalışma zamanında yazılım ile ilgili bilgiler toplanarak sonuca varılır. Dinamik analiz yönteminin mobil cihazlardaki güç tüketimi kısıtlamalarından dolayı bazı mobil cihazlar için kullanımı mümkün olmamaktadır. Bu yüzden literatürdeki çoğu çalışma statik analiz yöntemine odaklanmaktadır. Fakat bu yöntem bilinen atakların yeni çeşitlerini tespit edememekte ve kod karıştırma tekniklerine karşı zayıf kalmaktadır. Buna rağmen statik analiz, literatürde mobil cihazlar için en çok önerilen yöntemdir. Bu yöntemin bilinen ataklar, bilinen atakların çeşidi ve bilinmeyen ataklar karşısında ne kadar etkili olduğunun araştırılması gerekmektedir. Bu nedenle bu tezde, mobil cihazlar için statik güvenlik çözümlerinin değerlendirilmesi için varolan zararlı yazılımlardan yeni zararlı yazılımların üretilmesi amaçlanmaktadır.

Bu tezde, varolan mobil statik analiz araçlarının performansını ölçmek için evrimsel hesaplama algoritmalarından biri olan genetik programlama kullanılarak bilinen mobil zararlı yazılımlardan yeni mobil zararlı yazılımların evrimleştirilmesi araştırılmaktadır. Yeni yazılımların otomatik olarak evrimleştirilmesi, bu zararlı yazılımlara karşı güvenlik çözümlerinin performansının ölçülmesine ve bu güvenlik çözümlerinin de otomatik olarak geliştirilmesine olanak sağlayacaktır. Varolan statik analiz araçları, her yeni ya da bilinmeyen zararlı yazılım ile karşılaştığında imza veritabanını güncellerler. Bu nedenle, yeni zararlı yazılımların otomatik olarak üretilmesi, varolan güvenlik çözümlerinin daha zararlı yazılım piyasaya çıkmadan imzasını elde edileceğinden(ettiğinden) dolayı sıfırıncı gün ataklara karşı dirençli hale gelmesine ve varolan güvenlik çözümlerinin sürekli kendini geliştirilebilmesine olanak sağlar. Şimdilik önerilen yöntem sadece yeni ve bilinmeyen ataklar geliştirirken, ileride otomatik olarak yeni çözümler geliştiren, eş evrim mekanizmasına

dayanan bir çatı olarak genişletilmesi planlanmaktadır. Böylelikle, virüs-antivirüs eş evrimi ile hem zararlı yazılımlar, hem de bu zararlı yazılımları tespit eden sistemlerin eş zamanlı olarak geliştirilmesi hedeflenmektedir.

Genetik programlama, literatürdeki bazı çalışmalarda yeni atak ve zararlı yazılımların üretilmesi için kullanılmıştır. Fakat bu yöntemlerin birçoğu tamamen otomatik değildir ve sadece belirli tür ataklar için önerilmiştir. Bu nedenle, bir güvenlik uzmanı gözle zararlı yazılımların kodlarını analiz etmeli ve bir zararlı yazılımın değişik varyasyonlarında hangi parametrelerin değiştiğini ortaya çıkarmalıdır. Bu işlem oldukça zaman gerektirdiğinden, bu tezde varolan zararlı yazılımlara genetik programlama ve bazı karıştırma tekniklerinin uygulanması ile tamamen otomatik bir zararlı yazılım geliştirme sisteminin tasarlanması araştırılmıştır. Aldığımız sonuçlar, genetik programlama ile mobil güvenlik çözümleri içerisinde [6, 7] en başarılarından olan 8 antivirüs sisteminden kaçabilecek etkili zararlı yazılımlar geliştirilebildiğini göstermiştir. Ayrıca genetik programlamanın literatürde yaygın olarak kullanılan karıştırma teknikleri uygulayan yaklaşımlardan da daha iyi sonuçlar ürettiği görülmüştür. Bu çalışma, genetik programlama kullanarak zararlı yazılım evrimleştirilmesi konusunda mobil cihazlar için yapılan ilk çalışmadır.

1.2. Tezin Yapısı

Bölüm 2'de literatürde yer alan ilgili çalışmalar özetlenecektir. Bölüm 3'te Android platformunun yapısı, Android uygulamaların nasıl geliştirildiği ve Android uygulamalarına tersine mühendisliğin nasıl uygulandığı anlatılacaktır. Bölüm 4'te evrimsel hesaplama, evrimsel hesaplamanın alt dalı olan genetik programlama hakkında bilgiler verilecektir. Bölüm 5'te yeni zararlı yazılımlar evrimleştirmek için kullandığımız model ve mutasyon işlecinde kullanılan karıştırma teknikleri anlatılacaktır. Bölüm 6'da önerilen metodun performansı tartışılacaktır. Ve son olarak Bölüm 7 sonuç cümlelerine ve gelecekte yapılabilecek çalışma ve iyileştirmelere ayrılmıştır.

Bölüm 2.

YAPILAN ÇALIŞMALAR

Mobil zararlı yazılım analiz ve tespit etme alanındaki araştırmalar son bir kaç yıldır artarak devam etmektedir. Araştırmacılar mobil cihazlar için uygun güvenlik çözümleri üzerinde çalışmaktadırlar. Birçok güvenlik şirketi mobil güvenlik için kendi çözümlerini sunmuşlardır. Bu çözümlerin büyük çoğunluğu mobil cihazlardaki güç kısıtlamalarından dolayı dinamik analizden çok statik analize dayanmaktadır. Ayrıca araştırmacılar, çeşitli karıştırma teknikleri kullanarak zararlı yazılımların yeni çeşitlerini üreterek antivirüs sistemlerinden kaçmaya çalışmakta ve antivirüs sistemlerinin güvenilirliğini test etmektedirler. Antivirüs sistemlerini değerlendirmek için ayrıca evrimsel hesaplama algoritmaları da kullanılmaktadır. Bu yüzden bu bölümde karıştırma tekniklerini araştıran ve kullanan, evrimsel hesaplama kullanan ve statik analiz yöntemine dayanan çalışmalar anlatılacaktır.

2.1. Karıştırma Teknikleri

Literatürde, mobil antivirüs sistemlerinin karıştırma teknikleri karşısındaki başarısını ölçen iki önemli çalışma vardır. [8]'de, "ADAM" olarak adlandırılan Android antivirüs sistemlerini test eden otomatik ve genişletilebilir bir sistem tasarlanmıştır. Bu çalışmada, özgün, zararlı işlevini yitirmeden çeşitli karıştırma teknikleri uygulanarak yeniden üretilen zararlı yazılımların karşısında antivirüs sistemlerinin etkinliği değerlendirilmiştir. 222 Android

zararlı yazılımın yeni çeşitleri üretilmiştir. Üretilen zararlı yazılımları, şüpheli dosyaları inceleyen ve 50'nin üzerinde antivirüs kullanarak zararlı yazılımları hızlı bir şekilde tespit edebilen ücretsiz bir ağ hizmeti olan VirusTotal'e [9] göndererek değerlendirmişlerdir. Ayrıca bilgisayarda Linux işletim sistemi üzerinde çalışan ve mobil platformlardaki çözümlerle aynı tespit etme yöntemini kullanan Antiy [10] antivirüs sistemi üzerinde de test etmişlerdir.

Rastogi ve arkadaşları [11] DroidChameleon adını verdikleri [8] çalışmasındaki karıştırma tekniklerinin gelişmiş şeklini kullanarak ürettikleri zararlı yazılımları Android antivirüs sistemlerini değerlendirmek için kullanan bir sistem geliştirmişlerdir. Özgün, zararlı işlevini koruyacak şekilde çeşitli metamorfik ve polimorfik teknikler kullanarak Android uygulamalarını otomatik olarak değiştirmişlerdir. Contagio Minidump zararlı yazılım veri setinden [12] alınan farklı zararlı yazılım ailelerinden sadece 6 farklı zararlı yazılım kullanmışlardır ve bu zararlı yazılımları karıştırılmış haline değiştirerek antivirüs sistemlerinin etkinliklerini ölçmüşlerdir. 10 iyi tanınan Android antivirüs sistemi üzerinde karıştırılmış zararlı yazılımları incelemişlerdir ve kullandıkları tüm antivirüs sistemlerinin karıştırma tekniklerine karşı zaafları olduğunu ortaya çıkarmışlardır.

Christodorescu ve Jha [13] masaüstü zararlı yazılım tespit etme sistemleri için kaynak kodları üzerinde değişikliklere dayanan test uygulamaları oluşturan bir sistem geliştirmişlerdir. Önerdikleri yöntem, zararlı yazılım tespit etme sistemlerinin çeşitli karıştırma teknikleri karşısındaki etkinliğini değerlendirmektedir. Sonuçlarında çoğu zararlı yazılım tespit etme sistemlerinin kod karıştırma teknikleri karşısında zayıf olduğunu göstermişlerdir.

Morales ve arkadaşları [14] bilinen zararlı yazılımlar karşısında antivirüs sistemlerinin mobil cihazları nasıl koruduğunu analiz etmişlerdir. Test ortamları Microsoft Windows Mobile platformu üzerinde çalışmaktadır. Microsoft Windows Mobile platformunda bulunan 2 bilinen zararlı yazılım kullanarak 4 antivirüs sistemini değerlendirmişlerdir. 5 farklı karıştırma tekniği kullanarak test örnekleri oluşturan bir formal model sunmuşlardır. Sonuçlarında test ettikleri bütün antivirüs sistemlerinin yüksek yanlış negatif değeri ürettikleri görülmüştür. Bu yüksek yanlış negatif değerinin antivirüs sistemlerinin basit imza tabanlı tespit etme teknikleri kullandıklarından dolayı gerçekleştirdiği sonucuna varmışlardır.

Moser ve arkadaşları [15] tespit için model kontrolü kullanan ve anlamsal (semantik) imzalara dayanan bir zararlı yazılım tespit sistemi geliştirmişlerdir. Basit kod karıştırma teknikleriyle kolayca atlatılabilen örüntü eşleştirme tespit sistemlerinin zayıflıklarını hedef almışlardır. Bazı karıştırma tekniklerini kullanarak iyi bilinen 3 solucandan yeni zararlı

yazılımlar üretmişlerdir ve bu zararlı yazılımları kullanarak kendi tespit sistemlerini ve ticari virüs tarayıcılarını test etmişlerdir. Statik analiz yönteminin tek başına zararlı yazılım tespitinde yeterli olmadığını, dinamik analiz yöntemleriyle de desteklenmesi gerektiğini söylemişlerdir.

Ilsun ve arkadaşları [16] virüs geliştiricileri tarafından sıklıkla kullanılan karıştırma tekniklerini analiz etmişlerdir. Amaçları bu tekniklerin virüs geliştiricileri tarafından nasıl kullandıklarını anlamaktır.

Christodorescu ve arkadaşları [17] karıştırma teknikleri uygulanan zararlı yazılımları tekrar orijinal haline getirmeye yarayan bir normalize sistemi geliştirmişlerdir. Amaçları virüs tarayıcılarının tespit oranlarını arttıracak bir yardımcı sistem oluşturmaktır. Kod sırası değiştirme, paketleme, gereksiz kod ekleme olmak üzere 3 karıştırma tekniği kullanmışlardır. 4 antivirüs sisteminde normalize edilmiş zararlı yazılımları değerlendirmişlerdir ve tespit oranını %100'e çıkardıklarını söylemişlerdir.

2.2. Evrimsel Hesaplama

Evrimsel hesaplama teknikleri son yıllarda literatürde zararlı yazılım evrimleştirmek için kullanılmaktadır. Bu bölümde evrimsel hesaplama kullanılarak zararlı yazılım evrimleştirmesi yapan çalışmalardan bahsedilecektir.

Wu L. ve arkadaşları [18], biyolojik bağışıklık sisteminden esinlenerek bilgisayar virüsü evrim modeli geliştirmişlerdir. Bilgisayar virüslerini, gen modeline dayanarak formal olarak tanımlamışlardır. Modellerinde evrimsel operatörlerden olan seçme, çaprazlama, mutasyon ve bağışıklık operatörlerini kullanmışlardır. Bilgisayar virüslerini bulaşma işaretleyicisi (infected marker), başlatma birimi (initialization module), bulaşma birimi (infetion Module) ve performans birimi (Performance Module) olmak üzere 4 parçadan oluştuklarını söylemişlerdir. Deneylerinde, WildList [19] veri setinden seçtikleri 10 Windows betik virüsü kullanmışlardır. Bu 10 Windows betik virüslerinin bulaşma ve performans birimlerini ayıklamışlardır ve bu iki birimi kullanarak bütün virüslerin genlerini içeren bir gen havuzu oluşturmuşlardır. Bu gen havuzundaki genleri kullanarak yeni virüsler üretmişlerdir. Bu gen havuzunu 10 gruba bölmüşlerdir. İlk grupta rastgele seçilen tek bir virüsün genleri, ikinci grupta rastgele seçilen 2 virüsün genleri ve böyle devam ederek onuncu ve son

grupta tüm 10 virüsün genleri bulunmaktadır. Toplam 100 tane virüs üretmişlerdir. Deney sonuçlarına göre virüs gen ölçeği arttığında yeni üretilen virüslerin de yaşama şansının arttığı görülmüştür. Önerdikleri modelin bilinmeyen ve değişikliğe uğrayan virüsleri tespit etmekte antivirüs sistemlerinin gelişmesine yardımcı olacağını ve yaptıkları deneyleri sonucunda biyolojik virüsler ile bilgisayar virüslerinin benzer özelliklere sahip olduklarını belirtmişlerdir.

Noreen ve arkadaşları [20] Beagle virüsüne genetik işleçler uygulayarak yeni çeşitlerini üretmişlerdir. İlk olarak Beagle virüsünden, virüs tarafından açılan port numarası, virüsün kullandığı eklerin isimleri gibi Beagle virüsünü karakterize eden bazı statik öznitelikler çıkarmışlar ve böylece virüsün soyut bir temsilini oluşturmuşlardır. Sonra bu öznitelikler üzerinde genetik işleçler uygulamışlardır. Genetik işleçleri uyguladıktan sonra soyut temsilden tekrar makine koduna dönüşüm yaparak yeni virüs oluşturmuşlardır. En son bu üretilen virüsün, ticari antivirüs sistemleri üzerinde girdi virüsün bir çeşidi olup olmadığını test etmişlerdir. Ayrıca virüsü gerçek ve sanal bilgisayarlar üzerinde çalıştırmışlardır. Uygunluk değeri olarak orijinal virüse benzerliğini kullanmışlardır. Üretilen virüsleri AVG ve Symantec antivirüs sistemleri üzerinde denemişlerdir. AVG antivirüs sistemi üretilen yeni virüslerin %98.98'ini tespit etmiştir. Yeni virüslerin %1.02'sini ise tespit edememiştir. Symantec antivirüs sistemi ise yeni üretilen virüslerin %58.9'unu tespit edememiştir ve kötü bir tespit oranı sergilemiştir. Symantec, yeni virüslerin %41.1'ini ise tespit etmiştir.

Noreen ve arkadaşları, [20]'deki yaklaşımı ARM-Linux tabanlı sistemlerini tehdit eden virüslere de uygulamışlar ve genetik işleçlerde Markov modeli kullanarak virüslerin çeşitlerini üretmişlerdir [21]. Singleton ve Existere diye adlandırdıkları 2 tür veri seti kullanmışlardır. Bu çalışmada 2 birimden oluşan hem statik analiz hem de dinamik analiz yapan melez bir model geliştirmişlerdir. İlk birim ELF Yapısal İzleyici (ELF Structural Tracer - EST), ARM-Linux tabanlı sistemlerde sıfırıncı gün zararlı yazılımlarını tespit etmek için dosyalardan yapısal izler çıkartmaktadır. İkinci birim olan PCB Çalışma Zamanı İzleyici (PCB Runtime Tracer - PRT) ise, çalışma zamanında zararlı işlemleri ve bu işlemlerin polimorfik çeşitlerini tespit etmektedir. EST için 422 zararsız ve 430 zararlı yazılım kullanmışlardır. PRT için ise 100 zararsız, 100 zararlı yazılım kullanmışlardır. EST'de ilk olarak yazılımların özniteliklerini çıkarmışlardır. Toplam 383 tane öznitelik çıkartmışlardır. Daha sonra Gereksiz Öznitelik Çıkarma (Redundant Feature Removal - RFR) metodu kullanarak sabit değer veya yüksek sapma içeren öznitelikleri ayıklamışlardır. Daha sonra sınıflandırma işleminde ek yükü azaltmak için Ayrık Wavelet Dönüşümü (Discrete Wavelet Transform - DWT) tekniği kullanarak boyut indirgeme yapmışlardır. En son

olarak boyut indirgenen öznitelikleri JRip, J48, Örnek tabanlı öğrenme (IBK) ve Naive Bayes makine öğrenmesi teknikleri ile sınıflandırmışlardır. Singleton veri setinde %99, Existere veri setinde ise %100 tespit etme oranı sağlamışlardır. PRT'de 118 tane parametre belirlemiş ve bu 118 parametreyi 10 ms. aralıklarla kayıt altına alarak öznitelikleri çıkarmışlardır. Bu özniteliklere iki aşamalı önişleme uygulamışlardır. İlkinde makine öğrenmesi sınıflandırıcılarını yanıltabilecek gereksiz öznitelikler belirlenerek çıkartılmıştır. Bu aşamada 83 parametre elimine edilmiştir ve geriye 35 parametre kalmıştır. İkinci aşamada zararlı ve zararsız yazılımları birbirinden ayıran parametreleri belirlemek için bu 35 parametrenin zaman serisi ortalaması hesaplanmıştır. Ayrıca zaman serisi sapmaları da hesaplanarak yüksek sapmaya sahip parametreleri çıkarmışlardır. Her iki veri seti için de bu işlemleri yapmışlardır ve EST yöntemiyle aynı makine öğrenmesi teknikleriyle sınıflandırma işlemini yapmışlardır. Singleton veri setinde %96 ve Existere veri setinde %100 tespit oranı yakalamışlardır.

Noreen ve arkadaşları ayrıca [22]'deki çalışmada, zararlı yazılımları anlamsal olarak değiştirip yeni çeşitlerini oluşturan bir yöntem sunmuşlardır. Zararlı yazılımların soyut gösterimini (abstract representation) çıkarmışlardır ve sonra bu gösterimleri kullanarak genetik algoritma yardımı ile yeni zararlı yazılım evrimleştirmişlerdir. Bu yöntemde sadece COM infektörünü kullanmışlardır ve test ettikleri antivirüs sistemlerinin evrimleştirilmiş zararlı yazılımların %7'sini tespit edemediklerini göstermişlerdir.

Kayacık ve arkadaşları [23], daha iyi tespit etme sistemleri geliştirebilmek amacıyla genetik programlama kullanarak tespit etme sistemlerinin açıklarını önceden belirleyebilecek yeni ataklar üretmeyi amaçlamışlardır. İmza tabanlı tespit sistemlerinden kaçabilecek bellek taşması ataklarını evrimleştiren bir yöntem önermişlerdir. Evrimleştirme işlemini yapmak için genetik programlamanın önemli bir özelliği olan kod şişirme özelliğini kullanmışlardır. Böylece zararlı atakların ana görevleri gizlenerek, tespit sistemlerinden kaçabilmeleri sağlanmıştır. Geliştirilen atakların gerçek bilgisayarlarda çalıştırılması bilgisayara zarar verip sonuç dönememesine neden olacağından, ataklar sanal makine üzerine benzetimi yapılarak çalıştırılmıştır. Geliştirilen atak, esasen *execve* fonksiyonu aracılığıyla yapılan bir ataktır. *execve* Linux işletim sisteminde programların çalıştırılmasını sağlayan bir sistem çağrısıdır. *execve* sistem çağrısı üç parametre ile çağrılmaktadır: *int execve(const char *path, char *const argv[], char *const envp[])*. İlk parametre EBX, ikinci parametre ECX ve son parametre de EDX yazmacında bulunmalıdır. *execve* komutunun kendisi

de EAX yazmacında bulunur. Bu işlemleri gerçekleştirmek için 11 tane makine dili komutu gerekmektedir (Son komut kontrolü *execve* komutuna bırakan kesmedir). 10 komut çalıştırıldığında EAX, EBX, ECX ve EDX yazmaçları düzgün tanımlanmış olmalıdır. Eğer düzgün tanımlanmadıysa sırayı düzeltmek için gerekli komut sayısı hesaplanır ve bu komutlar yığına eklenir. Uygunluk değeri olarak atağı gerçekleştirmek için kullanılan komut sayısı kullanılır. Fakat yukarıda anlatıldığı gibi eklenen her komut uygunluk değerinden çıkartılır. En yüksek uygunluk değeri bütün şartlar sağlandıysa 10 olur. Eğer atağı gerçekleştirmek için komut eklenmesi gerekiyorsa eklenen komut sayısı 10 değerinden çıkartılır ve bu sayı o atak için uygunluk değeri olur. Önerilen çalışmada lineer genetik programlama kullanılmıştır. Üç tür deney yapmışlardır. İlkinde komut seti kümesini 6 komutla sınırlamışlardır. İkincisinde 12, üçüncüsünde ise 15 komut kullanmışlardır. Evrimleştirilen atakları Snort saldırı tespit sistemi üzerinde denemişlerdir ve bütün yeni geliştirilen atakların Snort [24] saldırı tespit sisteminden kaçtığını belirtmişlerdir. Genetik programlamanın kod şişirme özelliğinin atağın gerçek niyetini gizlemeyi başardığını belirtmişlerdir.

Kayacık ve arkadaşları [25]'deki çalışmalarında, atak yapandan önce muhtemel güvenlik açıklarını bulma amacıyla genetik programlama kullanarak bellek taşması atakları geliştirmişlerdir. Kendisini zararsız yazılım olarak gösteren ve tespit sistemlerinden kaçabilen ataklar oluşturmayı amaçlamışlardır. Yazılımları sistem çağrısı dizisi olarak temsil etmişlerdir. Oluşturulan atakları Linux işletim sistemi üzerinde çalışan ve güvenlik açığı bulunan 4 yazılım için 4 anomali tespit eden sistem üzerinde denemişlerdir. Hedefledikleri atak 3 adımdan oluşmaktadır: (1) UIX şifre dosyasını (/etc/password) açma, (2) atak yapanın şifre girmeden giriş yapabileceği süper kullanıcı hesabı satırı ekleme ve (3) Dosyayı kapatma. Eğer atak bu 3 adımı sırası ile birlikte doğru bir şekilde taşıyorsa en yüksek uygunluk değeri olan 5 değerini almaktadır. Aynı zamanda genetik programlama anomali yüzdesini de düşürmeyi amaçlamaktadır. Bu anomali yüzdesini ise kullandıkları 4 anomali tespit sisteminden almaktadırlar. Genetik programlama birden çok sonuç elde ettiği için bu bireyler arasında en iyi bireyi seçmek için Pareto sıralaması kullanılmaktadır. Atakların orijinal anomali yüzdeleri ile genetik programlama sonucu oluşan atakların anomali yüzdelerini karşılaştırmışlardır ve genetik programlama sonucu oluşan atakların anomali yüzdelerinin oldukça düşük olduğu ve anomali tespit sistemlerinden daha kolay kaçabildikleri görülmüştür. Bu çalışmada da bir önceki çalışmalarda olduğu gibi lineer genetik programlama kullanmışlardır.

Kayacık ve arkadaşları [26]'deki çalışmada [25]'deki çalışmayı 1 anomali tespit eden sistem ve kaçınma saldırılarını oluşturmak için gecikme parametresi ekleyerek genişletmişler. Güvenlik açıkları bulunan *traceroute*, *restore*, *samba* ve *ftpd* yazılımları üzerinde genetik programlama uygulamışlardır. Bu yazılımlarda en çok yer alan 20 sistem çağrısı çıkarmışlardır ve genetik programlamada bu sistem çağrılarını kullanmışlardır. Anomali tespit sistemi olarak Stide, Process Homeostasis, şema maskeli Process Homeostasis, Markov Modele dayanan ve otomatik ilişkisel sinir ağlarına dayanan sistemleri kullanmışlardır. Yine orjinal ve yeni üretilen atakların anomali yüzdelerini karşılaştırmışlardır. Ayrıca üretilen atakların tespit sistemlerinden nasıl kaçtığını da incelemişlerdir. Her atak için en düşük anomali yüzdesine sahip atakları seçmişlerdir. Bu atakların her bir anomali tespit sistemi üzerindeki etkilerini incelemişlerdir.

Kayacık ve arkadaşları [27]'de tespit edilebilme yüzdesi düşük olan kaçınma atakları geliştirmeye odaklanmışlardır. Atakların normal davranışlarını bozmadan, bu davranışları genetik programlama ile tespit sistemlerinden kaçınacak şekilde evrimleştirmeye çalışmışlardır. Önceki çalışmalarında geliştirdikleri, tespit sistemlerinin iç bilgisini kullanarak işlem yapan ve beyaz kutu adını verdikleri yönteme ilaveten siyah kutu adını verdikleri yeni bir yöntem sunmuşlardır. Siyah kutu yönteminde, tespit sisteminden dönen sonuç kullanılmaktadır. [26]'de kullandıkları uygunluk fonksiyonunu, güvenlik açığına sahip yazılımlar ve anomali tespit sistemlerini kullanmışlardır. Deneylerinde beyaz kutu ve siyah kutu yöntemlerini karşılaştırmışlardır. Sonuçlarda beyaz kutu yönteminin siyah kutu yöntemine göre daha düşük anomali yüzdeleri verdiği görülmüştür. Fakat beyaz kutu yönteminde tespit sisteminin iç bilgisinin ek bir yük getirdiğinden bahsetmişlerdir. Ayrıca bu iki yöntemi gecikme parametresi açısından da incelemişlerdir.

2.3. Statik Analiz

Literatürde, statik ve dinamik olmak üzere 2 tür analiz yöntemi çoklukla kullanılmaktadır. Fakat mobil cihazlardaki güç kısıtlamalarından dolayı dinamik analiz yönteminden ziyade statik analiz yöntemine araştırmacılar yoğunlaşmışlardır. Bu yüzden, bu bölümde Android platformundaki uygulamalarını incelemek için statik analiz yöntemi uygulayan çalışmalar anlatılacaktır.

Suarez-Tangil ve arkadaşları [28]'de zararlı yazılımları tespit etmek amacıyla veri madenciliği ve bilgi erişim tekniklerine dayalı *Dendroid* adını verdikleri bir sistem geliştirmişlerdir. *Dendroid*, Android platformundaki zararlı yazılımlardaki kod yapılarını statik olarak inceleyerek sonuca varır. Vektör Uzay Modeli kullanarak zararlı yazılımlar arasındaki benzerliği ölçer ve bu benzerlik değerlerine göre sınıflandırma yapar. Kod yapılarını çıkartmak için önce yazılımları tersine çevirme uygulayarak Dalvik komutlarını elde etmişlerdir. Sonra *Androguard* [29] aracını kullanarak kontrol akış çizgelerini çıkartmışlardır. Cesare ve arkadaşlarının çalışmasındaki [30] dilbilgisi yapısını kullanarak bu kontrol akış çizgelerini karakter dizisine çevirerek bir vektör haline getirmişlerdir. Daha sonra yazılımlardan çıkardıkları bu karakter dizilerini Vektör Uzay Modeli kullanarak sınıflandırma yapmak için kullanmışlardır. Bu yöntemi 33 zararlı yazılım ailesinden 610 zararlı yazılım üzerinde denemişler ve 35 zararlı yazılımı yanlış sınıflandırmışlardır. Yanlış sınıflandırılan zararlı yazılımlar sadece 6 ailedeki zararlı yazılımlar olmuştur. Geri kalan 27 ailedeki zararlı yazılımları ise düzgün sınıflandırmışlardır.

Grace ve arkadaşları [31]'de sıfırıncı gün Android zararlı yazılımlarını tespit etmek için proaktif (önleyici) *RiskRanker* adını verdikleri bir sistem önermişlerdir. Bu sistemde incelenen uygulamaların potansiyel tehlikeli risk içerip içermedikleri incelenmiştir. Tehlikeli riskleri yüksek, orta ve düşük risk olmak üzere 3 gruba ayırmışlardır. Yüksek riskli uygulamalar kullanıcının izni olmadan Android yüklü cihazın ve yüklü uygulamaların işleyişine müdahale edebilen, orta riskli uygulamalar hassas bilgileri elde eden ve finansal hasar veren ve düşük riskli uygulamalar kolay erişilebilir bilgileri elde edebilen uygulamalardır. İki aşamalı risk analizi yapmışlardır. İlk aşamada uygulamaların yüksek veya orta riskli olup olmadıklarına bakmışlardır. Eğer yüksek veya orta riskli ise tehlikeli davranış içerip içermediklerini kontrol etmişlerdir. Şifrelenmiş kod parçaları ilk analizden kaçabildiği için ikinci analizi yapmaktadırlar. Toplam 118,318 uygulama üzerinde bu yöntemi test etmişlerdir ve 3281 risk içeren uygulama bulmuşlardır. Bu 3281 uygulamanın 322 tanesinin sıfırıncı gün zararlı yazılımı olduğunu bulmuşlardır.

Schmidt [32]'te Android ve Symbian işletim sistemlerinde zararlı yazılım tespit etmek için statik ve dinamik analizi beraber kullanan bir yöntem sunmuştur. Statik analiz yönteminde önce metot çağrılarını çıkartmıştır. Sonra karar ağaçları kullanarak bu metot çağrılarını işlemiştir. En son sınıflandırıcılar kullanarak zararlı ve zararsız yazılımları ayırt etmiştir.

Elish ve arkadaşları [33]'te kullanıcının izni olmadan tetiklenen işlemleri belirleyerek tespit işlemi yapan bir sistem geliştirmişlerdir. Önce uygulama java baytkodlara çevrilir. Sonra

metot çağrıları ve kullanıcı tarafından tetiklenen işlemler çıkartılır. Daha sonra uygulamanın kontrol akış çizgesi çıkartılır ve bu kontrol akış çizgesi kullanılarak metot çağrıları ve kullanıcı tarafından tetiklenen işlemler arasında yollar bulunmaya çalışılır. Eğer metot çağrısı ile kullanıcı tarafından tetiklenen işlem arasında bir yol yoksa bu metot çağrısı risk içeren bir çağrı olarak etiketlenir. Kullanıcı tarafından tetiklenen metot çağrılarının tüm metot çağrılarına oranı güven yüzdesini verir. Bu yöntemi 708 zararsız, 482 zararlı yazılım üzerinde denemişlerdir. Zararlı yazılımların 313 tanesi %0 güven yüzdesine sahip olmuştur. Bunun nedeni bu uygulamaların hiç bir zaman kullanıcının tetiklemesine gerek olmayan metot çağrıları yapmalarıdır. Geriye kalan zararlı yazılımların güven yüzdesi ise oldukça düşük çıkmıştır. Sadece *FakeNetFlix* zararlı yazılımının güven yüzdesi %100 çıkmıştır. Bunun nedeni de bu yazılımın kullanıcıya sahte bir arayüz sağlamasından dolayıdır. Zararsız yazılımların ise %98'i %80 üstü güven yüzdesi sergilemiştir.

Bölüm 3.

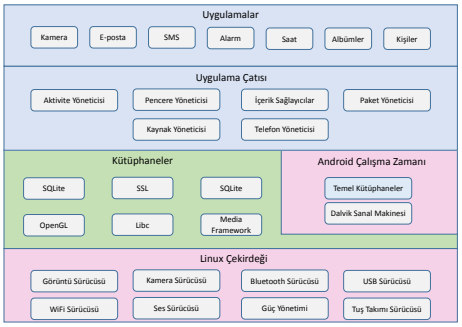
ANDROİD PLATFORMU

3.1. Android Platformuna Giriş

Android işletim sistemi, *Linux* çekirdeği üzerine yazılmış mobil cihazlarda kullanılan açık kaynaklı bir işletim sistemidir. Donanımsal olarak her türlü işlemci üzerinde çalışacak şekilde geliştirilmiştir. Akıllı telefonlar, tabletler ve akıllı gözlükler gibi her türlü mobil cihazlarda çalıştırılabilir. Java baytkodlara benzeyen dalvik baytkodları çalıştırır. Android işletim sistemi, kamera işlevleri, GPS verisi, ağ ve telefon işlevleri gibi kaynaklara erişmeyi sağlar. Sağladığı zengin kütüphaneler ile mobil cihazların ihtiyacı olan uygulamaların geliştirilmesini sağlar.

Android işletim sistemi mimarisi, beş bileşen ve dört katmandan oluşmaktadır. Android işletim sisteminin mimarisi Şekil 3.1.'de görülmektedir [34]. İlk katman olan linux çekirdeği işlem, bellek ve usb, wifi ve bluetooth gibi cihaz yönetimi işlevlerini sağlayarak cihaz kaynaklarına erişimi sağlar. İkinci katmanda kütüphaneler ve Android çalışma zamanı olmak üzere iki bileşen bulunur. Kütüphaneler, uygulamaların kullandığı harici kütüphaneleri ifade eder. Android çalışma zamanı ise Android uygulamaların çalışacağı Dalvik sanal makinesi ile temel Android kütüphanelerini sağlar. Dalvik sanal makinesi Android için özelleştirilmiş Java sanal makinesi benzeri bir bileşendir. Her uygulamanın kendine ait bir dalvik sanal makinesi bulunur. Böylece Android, her uygulama için bir kum havuzu sağlamış olur ve uygulamaların birbirleriyle etkileşmesini önleyerek güvenliği arttırmış olur. Üçüncü katmanda ise uygulama çatısı bulunur. Uygulama çatısı, uygulamaların kullanabileceği

servisleri sağlar. En son katmanda ise mesaj göndermek, telefon görüşmesi yapmak, anlık mesajlaşma yapmak, internette gezinmek ve e-mail göndermek gibi amaçlarla kullanılan Android uygulamaları bulunur.



Şekil 3.1.: Android mimarisi

Dalvik sanal makinesi, Java sanal makinesine benzemekle beraber Enck ve arkadaşları Dalvik sanal makinesi ve Java sanal makinesi arasındaki farkları şöyle özetlemişlerdir [35] :

- Dalvik sanal makinesinde aynı Java sanal makinesi gibi java kodları çalışır. Fakat çalışma zamanında java baytkod ve dalvik baytkodlar birbirinden farklıdır. Java uygulamalarında Java sanal makinesi çalışma zamanında birden çok sınıf dosyalarından oluştuğundan hangi sınıf dosyasından hangi fonksiyon çağrısı yapılıyorsa onu yükler. Dalvik sanal makinesi ise bütün sınıf dosyalarını içeren tek bir dex dosyasından oluşur.

- Java sanal makinesi yığın tabanlı olmasına rağmen Dalvik sanal makinesi yazmaç tabanlıdır. Bundan dolayı Dalvik sanak makinesi Java sanal makinesine göre daha hızlı işlem yapar, fakat bellekte kapladığı alan daha büyüktür.

- Komut setleri birbirinden farklıdır. Dalvik sanal makinesinde 218 tane işlem kodu bulunmasına karşın, Java sanal makinesinde 200 tane işlem kodu bulunur. Dalvik komutları kaynak ve hedef yazmaçları içerdiğinden Java komutlarına göre daha uzundur. Bu nedenle Dalvik baytkodlu uygulamalar, Java baytkodlu uygulamalara göre daha az komuttan oluşur. Dalvik baytkodlu uygulamalar, Java baytkodlu uygulamalara göre %30 daha az komut içermesine rağmen kod boyutu %35 daha fazladır [36].

- Java baytkodlarında ilkel tipler birbirinden ayırt edilebilirken, Dalvik baytkodlarda bu mümkün değildir. *int* ve *float* ilkel tipleri için aynı işlem kodlar kullanıldığından dolayı bir değişkenin *int* veya *float* olduğu ayırt edilemez. Farklı tiplerde tanımlanmış diziler için de bu geçerlidir.

- Dalvik baytkodlarda Java baytkodlarda olduğu gibi *null* tipi yoktur. Bunun yerine sabit sıfır değeri kullanılır.
- Java baytkodda nesneler karşılaştırılırken nesne referansı ve *null* karşılaştırılması yapılırken Dalvik baytkodda nesneler *int* tipine dönüştürülür ve *int* ve sıfır karşılaştırılması yapılır.
- Java baytkodlar yapı olarak kaynak kodlara çok benzerken Dalvik baytkodlar benzemez.
- Java uygulamalarında her bir sınıfın ayrı ayrı sabitler havuzu varken Dalvik uygulamalarında sınıflar tek bir *dex* dosyasında bulunduğu için ortak sabitler havuzu vardır.

Android uygulamaları, işletim sisteminin eriştiği kaynakları son kullanıcının kullanımına sunar. Android işletim sisteminde uygulamaların sadece kendi dizinindeki dosyalara erişim hakkı vardır ve düşük öncelikli işlem olarak çalıştırılırlar. Her bir program için bir kum havuzu oluşturulur ve uygulamalar kendilerine ait bu kum havuzu içinde çalışır. Böylece diğer uygulamalar ile etkileşim sağlaması engellenmiş olur. Android uygulamalarını kurmadan önce uygulamaların hangi izinleri kullandığı bir liste halinde kullanıcıya gösterilir. Böylece zararlı yazılımların diğer uygulama ve dosyalara erişimlerini engellemiş olur. Fakat eğer kullanıcı gerekli izinleri verirse uygulamalar birbiriyle etkileşebilir. Android uygulamaları resmi olarak Google Play'den [37] indirilmektedir. Ayrıca Android resmi marketi dışındaki marketlerden de uygulama indirip kurulmasına izin verilmektedir.

Android platformuna uygulamalar herhangi bir marketten de indirilebildiği için uygulamaların ne kadar güvenli olduğu önem taşımaktadır. Android, resmi olmayan marketlerden uygulama indirilmesini önermemekte, fakat bunu kullanıcının inisiyatifine bırakmaktadır. Bu tür bir marketten uygulama indirilmeye çalışıldığında kullanıcıya "zararlı bir aktivite içerebilir" benzeri bir uyarı vermektedir.

3.2. Android Platformunda Güvenlik

Android platformunda zararlı yazılımların engellenmesi için iki temel güvenlik mekanizması bulunmaktadır: market güvenliği ve platform güvenliği.

3.2.1 Market Güvenliği

Market güvenliği, uygulamaların cihaza yüklenmeden önce yüklendiği market tarafından yapılan analiz ve tespitleri içermektedir. İki türlü yöntem vardır. Birincisi uygulamaları statik ve dinamik analiz yöntemleriyle analiz ederek zararlı olup olmadığını belirlemedir. Android resmi marketinde bu işi Google Bouncer adını verdiği bir servisle yapmaktadır [38]. Google Bouncer, resmi Android marketinde varolan ve yeni eklenen uygulamalar üzerinde analiz yaparak uygulamaların zararlı olup olmadığına karar verir. İlk önce markete yüklenen uygulamanın bilinen zararlı yazılımlara benzeyip benzemediğine bakar. Ayrıca davranışına bakarak daha önceden analiz edilmiş uygulamalar ile karşılaştırır ve muhtemel alarmları çıkarır. Böylece hem statik hem de dinamik olarak uygulamaları analiz eder. Fakat Google Bouncer servisine rağmen resmi Android marketinin tam güvenli olduğu söylenemez. Çünkü Android bütün kaynak kodlarını tamamen uygulama geliştiricilere açtığı için uygulamaların, güvenlik mekanizmalarını aşması daha kolay olmaktadır. Ayrıca iOS gibi bazı platformlar, resmi market dışındaki marketlerden uygulama indirilmesine izin vermeyerek ek market güvenliği sağlamaktadır. Fakat Android işletim sistemine resmi olmayan marketlerden uygulama ve üçüncü parti yazılımlar yüklenebilmektedir. Bu da cihazı saldırılara karşı daha açık yapmaktadır.

Diğer yöntem ise uygulamaları, geliştiricisine ait özel imza ile imzalamaktır. Böylece uygulamanın kim tarafından yayımlandığı belli olacak ve uygulamanın bütünlüğünü denetlemek mümkün olacaktır. Fakat imzalama yönteminin kolayca bertaraf edilebileceği literatürde gösterilmiştir [32].

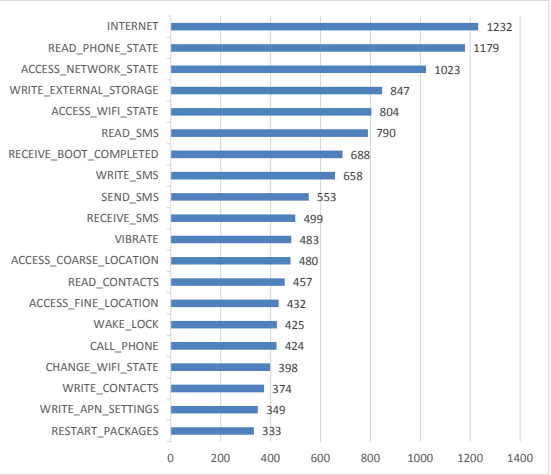
3.2.2 Platform Güvenliği

Uygulamalara, mobil cihaza yüklenme durumunda ve yüklendikten sonra platform tarafından bazı güvenlik mekanizmaları uygulanır. Bu mekanizmalar aşağıda anlatılmıştır:

3.2.2.1 İzinler

Uygulamaların işletim sisteminin hangi kaynaklarını kullanabileceğini belirlerler. Uygulama, mobil cihaza indirilmeye çalışıldığında uygulamanın kullanacağı izinler liste halinde kullanıcıya gösterilir. Böylece hem bir uygulamanın kullanıcının izni olmadan istem dışı bir

iş yapması engellenmiş olur hem de kullanıcının indirip kurduğu uygulamanın hangi kaynaklara eriştiği hakkında bilgisi olmuş olur. Anroid işletim sisteminde toplam 145 tane, sistem kaynaklarına erişmek için gerekli izin bulunmaktadır. MalGenome veri setinde bulunan zararlı yazılımların en çok kullandığı 20 izin Çizelge 3.2.'de gösterilmektedir [39]. Felt ve arkadaşları [40] uygulamaların, geliştirme aşamasında yapılan yanlışlıklardan ve kötü dokümantasyonu gibi bazı sebeplerden dolayı gereğinden fazla izin istediklerini ortaya çıkarmışlardır. Fakat bu mekanizma tamamen kullanıcıya bağlı olduğundan dolayı, yani kullanıcı eğer bir uygulamanın hangi kaynaklara erişeceğini bildiren metni kabul etmişse, tek başına bir güvenlik sağlamamaktadır.



Şekil 3.2.: MalGenome veri setinde bulunan zararlı yazılımların en çok kullandığı 20 izin

3.2.2.2 Kum Havuzunda Çalıştırma

Android işletim sisteminde her bir uygulamaya benzersiz bir kullanıcı tanımlama numarası verilir ve her bir uygulama ayrı işlemlerde çalıştırılır [34]. Bu, uygulamaların çekirdek seviyesinde kum havuzunda çalıştırılmasını sağlar. Bu kum havuzu, uygulamanın diğer uygulamaların verilerini okumasını ve izinsiz olarak başka bir işlemi yapmasını önler.

3.2.2.3 Uygulamalar Arası Etkileşimler

Android uygulamalarının yazıldığı Java programlama dili uygulamalar arası kaynak paylaşımı için iş parçacıkları, soketler ve paylaşımlı bellek gibi yapılar kullanmaktadır. Fakat bu işlemler hataya açık ve yönetmesi zor olduğundan dolayı Android işletim sisteminde işlemler arası haberleşmeyi sağlayan bir mekanizma geliştirilmiştir. Ayrıca Android işletim sistemi farklı uygulamalarda aynı işlevi gerçekleştiren işlerin çalıştırılmasını önleyecek şekilde tasarlanmıştır. İşlemler arası haberleşme (IPC) aracılığıyla bir uygulama başka bir uygulamanın çalıştırdığı işi çalıştırabilmektedir.

3.2.2.4 Linux Çekirdeği Güvenliği

Linux çekirdeği ilk ortaya çıktığından itibaren birçok kişi tarafından kullanılmış ve geliştirilmiştir. Bundan dolayı içerdiği hatalar ve açıklar birçok kişi tarafından araştırılmış ve bu hata ve açıkları düzelten geliştirmeler yapılmıştır. Linux çekirdeğinin temel özelliklerinden birisi tüm kullanıcıların birbirinden soyutlanmasıdır. Yani bir kullanıcının kullandığı kaynak ve dosyaları diğer kullanıcı kullanamaz. Android işletim sistemi de Linux çekirdeği üzerine yazıldığından dolayı bu güvenlik özelliklerini miras olarak almıştır. Linux çekirdeğinin Android işletim sistemine sağladığı bazı özellikler şunlardır: kullanıcıya dayalı izin modeli, işlemlerin birbirinden soyutlanması ve işlemler arası gelişmiş haberleşme mekanizması [34].

3.3. Android Uygulamaları

Android uygulamaları, java programlama dili ile yazılır ve kodlar *class* dosyalarında bulunur. Masaüstü uygulamalardan farklı olarak arayüzler *xml* dosyaları ile hazırlanır. Hangi aktivitelerin veya servislerin hangi izinleri kullanacağı, programın başlangıç noktasının hangi aktivite olacağı gibi bilgileri içeren bir manifest dosyasına sahiptir. Tüm bu dosyalar resim ve diğer kaynaklar gibi diğer dosyalar ile birlikte *dex* adı verilen bir dosyanın içinde tutulur. Tüm bu dosyalar derlenerek *apk* dosyası oluşturulur. *apk* dosyası Android işletim sisteminin çalıştırılabilir dosyasına verilen isimdir.

Android uygulamaları, çalıştırma zamanında masaüstü java uygulamaları gibi main fonksiyonu ya da başlangıç noktası içermezler. Manifest dosyasında hangi bileşen ana bileşen olarak belirtilmişse o bileşen uygulamanın başlangıcı olmuş olur. Manifest dosyası, uygulama ile ilgili önemli bilgileri barındıran bir dosyadır. Android işletim sistemi, bu dosyaya bakarak uygulamanın hangi kaynaklara erişebileceği ve başlangıç aktivitesinin ne olacağı gibi bilgileri okur ve bu bilgilere göre uygulamayı çalıştırır.

3.3.1 Android Uygulamalarının Bileşenleri

Android uygulamaları 4 çeşit bileşenden oluşur: *yayın alıcılar*, *içerik sağlayıcılar*, *servisler* ve *aktiviteler* [41].

3.3.1.1 Yayın Alıcılar (Broadcast Receiver)

Bir uygulamanın sistem ya da uygulama olayına bağlanmasını sağlar. Olay tetiklendiğinde Android işletim sistemi bağlı olan uygulamayı bilgilendirir. Böylece uygulamaların telefona çağrı geldiğinden, cihaza usb cihazı bağlandığından ve diğer olaylardan haberi olur. Örneğin; ACTION_UMS_CONNECTED olayı tetiklendiğinde Android işletim sistemi bu olaya yayın alıcılar ile bağlanan uygulamaları bilgilendirir ve bu uygulamalar da ona göre işlemlerini yaparlar. Hangi nesnelerin yayın alıcı olacağı manifest dosyasında belirtilir.

3.3.1.2 İçerik Sağlayıcılar (Content Provider)

Uygulamaların veri tabanındaki verilere erişmesi için kullanılır. Ayrıca bir uygulamanın veri tabanını diğer uygulamalarla paylaşabilmesi için kullanılır. Böylece tek bir içerik birden çok uygulamaya dağıtılmış olur. Her bir sağlayıcının içerdiği bilgiye göre kendine özgü sembolik bir adı (authority) vardır ve diğer uygulamalar bu adı kullanarak içerik sağlayıcıya bağlanıp işlerini gerçekleştirirler. Manifest dosyasında içerik sağlayıcı bilgileri belirtilir.

3.3.1.3 Servisler (Services)

Arkaplanda uzun süreli çalışacak işlemleri gerçekleştirmek için kullanılır ve kullanıcı arayüzü içermezler [42]. Bir bileşende çalışan işlem, bileşen kapandıktan sonra da devam etmesi için servisler aracılığıyla işlemini sürdürür. Böylece uzun süreli bir işlem çalıştırıldığında cihazın o uygulamada takılı kalmasını önler ve diğer uygulamaların da çalıştırılmasını sağlar. Diğer bileşenler uzak prosedür çağrısı arayüzü aracılığıyla servisleri kullanırlar. Böylece diğer bileşenler tarafından çağrılabilir.

3.3.1.4 Aktiviteler (Activities)

Kullanıcının arama yapma, fotoğraf çekme gibi sistem kaynaklarını kullanabilmesine olanak sağlayan kullanıcı arayüzlerini tanımlamak için kullanılır [43]. Bir uygulama genellikle birden çok aktiviteden oluşur. Bir anda sadece tek bir aktivite aktif olur. Manifest dosyasında bir aktivite ana aktivite olarak belirlenir ve uygulamanın başlangıcı bu aktivite olur. Bir aktivite diğer aktiviteleri çağrılabilir. Her yeni bir aktivite çalıştığında eskisi durdurulur. Fakat sistem eski aktiviteyi yığına alır; böylece tekrar o aktiviteye dönüldüğünde uygulama kaldığı yerden devam eder.

3.3.2 Android Uygulamalarının Derlenmesi

apk dosyası oluşturmak için bazı adımlar vardır. Bunlar şöyle özetlenebilir [44]:

1. *aapt (Android Asset Packaging Tool)* aracı projede yer alan bütün dosyaları derlemek için kullanır.
2. *Android Interface Definition Language (aidl)* istemci ve sunucu arasında işlemler arası haberleşme (IPC) kullanarak haberleşmeyi sağlayan java arabirimlerini uygun formata (*aidl*) çevirir.
3. Java derleyicisi tüm *java* ve *aidl* dosyalarını derler ve *class* dosyalarını oluşturur.
4. *class* dosyaları üçüncü parti yazılımlarla beraber *dex* aracı ile dalvik baytkodlara çevrilir ve *dex* dosyasını oluşturur.

5. *apkbuilder* aracı, *dex* dosyasını, derlenmiş diğer kaynakları ve resim gibi derlenmemiş diğer kaynakları tek bir *apk* dosyası içine paketler.

6. Oluşturulan *apk* dosyası gerçek cihaz veya emülatöre yüklenmeden önce imzalanır.

7. Eğer *apk* dosyası yayım (release) modunda imzalanmışsa *zipalign* [45] aracı kullanılır.

3.4. Tersine Mühendislik

Genetik programlama kullanarak antivirüs sistemlerini değerlendirmek için varolan zararlı yazılımların kaynak kodlarını kullanmamız gerekmektedir. Bu sebeple bu bölümde Android uygulamaları üzerinde nasıl tersine mühendislik yapılacağı anlatılacaktır.

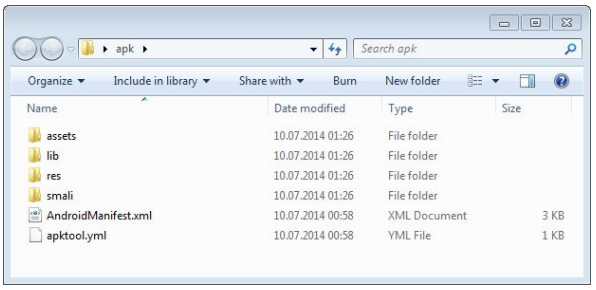
Android uygulamalarında tersine mühendislik yapmak için 2 yaklaşım vardır. Bunlar aşağıdaki gibidir:

1. **Kaynak Koda Dönüştürme (Decompiling).** Bu yaklaşımda *apk* dosyasından *java* dosyaları elde edilmeye çalışılır. İlk önce *apk* dosyası dex2jar [46] aracı kullanılarak *jar* dosyasına dönüştürülür. Daha sonra *jar* dosyasından *class* dosyaları çıkartılır. En son aşamada ise java tersine derleme aracı [47] kullanılarak *class* dosyaları *java* dosyalarına dönüştürülür.

2. **Tersine Çevirme (Disassembling).** Bu yaklaşımın amacı dalvik baytkodları içeren *dex* dosyası içindeki *smali* dosyalarını çıkarmaktır. Bu işlem apktool [48] aracı kullanarak yapılır.

Jad [47] ve Jeb [49] gibi tersine derleyici bazı araçlar kullanarak Android uygulamalarından Java kaynak kodlarına erişebilmemize rağmen, paketleme işleminde karşılaşılan sorunlardan dolayı tekrar *apk* dosyasına dönmek imkansız olmaktadır. Bu karşılaşılan sorunların otomatik olarak çözülmesi ise oldukça zordur. Bu nedenle bu tezde genetik programlama kullanarak zararlı yazılım evrimleştirmek için makine kodlarına benzer dalvik komut seti kullanılmıştır. Dalvik baytkodlara *apk* dosyası içinde yer alan *dex* dosyası içinden erişilebilir. Dalvik baytkodlar anlamlı şekilde birleşerek *smali* dosyalarını oluştururlar ve apktool [48]

aracı kullanılarak bu *smali* dosyalarına erişilebilir. *smali* dosyaları üzerinde değişiklik yaparak tekrar *apk* dosyasına dönüşüm yapılabilir. Tersine derleme yapılmış bir Android uygulamasında yer alan dosyalar Şekil 3.3.'de görülmektedir. Görüldüğü gibi bir Android uygulaması genel olarak *smali* kodlarından (smali), resim, video ve xml dosyaları gibi kaynak dosyalardan (res), üçüncü parti kütüphanelerden (lib), manifest dosyasından (AndroidManifest.xml) ve diğer dosyalardan (assets) oluşmaktadır.



Şekil 3.3.: Bir Android uygulamasının tersine derlenmiş hali

Kaynak Koda Dönüştürme yaklaşımında üretilen *java* dosyalarından tekrar geri *apk* dosyası üretmekteki sıkıntılardan dolayı bu tezde Tersine Çevirme yaklaşımı kullanılarak *smali* dosyaları üzerinden zararlı yazılım evrimleştirme işlemi yapılmıştır.

Bölüm 4.

EVRİMSEL HESAPLAMA

Evrimsel hesaplama doğal evrim mekanizmasından esinlenerek ortaya çıkmış bir optimizasyon tekniğidir. Darwin'in evrim teorisinde bahsedilen popülasyondaki en iyi bireylerin hayatta kalması prensibine dayanmaktadır. Bir popülasyondaki birey sayısı arttıkça popülasyonun yaşadığı çevrenin direnci azalacaktır. Bunun sonucunda çevre tepki göstererek bazı bireylerin elenmesini sağlayacaktır. Bu elenme korkusu dolayısıyla, popülasyondaki bireyler hayatta kalmak ve nesillerini sürdürebilmek amacıyla birbirleriyle yarış haline düşerler. Bu yarış, popülasyondaki bireylerin yeteneklerini geliştirmesini, yeni yetenekler kazanmalarını ve kendini geliştiremeyen bireylerin çevreden elenmesini sağlar. Böylece gelecek nesillerin önceki nesillerden daha nitelikli olması sağlanır. Evrimsel hesaplama bu bahsedilen özellikleri kullanarak belli bir nesil sayısı veya durdurma kriterine kadar popülasyon içinde kendisine lazım olan en iyi bireylerin seçiminin benzetimini sağlayan bir hesaplama yöntemidir. Populasyondaki bireyler aday çözümleri ve içinde yaşanılan çevre ise probleme özgü uygunluk fonksiyonunu temsil eder. Evrimsel hesaplama algoritmasının genel adımları Algoritma 1'de görülmektedir.

Evrimsel hesaplama, arama uzayının çok büyük olduğu durumlarda hem daha hızlı sonuca gitmeyi sağlar hem de deterministik yöntemden ziyade rastlantısal ve sezgisel bir yöntem olduğu için tüm arama uzayını daha iyi tarar. Bu da yerel minimumlardan daha kolay kaçarak küresel minimumu bulmayı kolaylaştırır. Ayrıca gradyan tabanlı tekniklerin tersine aday çözümleri eş zamanlı olarak değerlendirir.

Evrimsel hesaplama biyolojik evrimin sahip olduğu üreme, mutasyon, çaprazlama ve doğal seçilim mekanizmalarını kullanır. Evrimsel hesaplama, bu mekanizmaları içeren ve popülasyon tabanlı olan bütün algoritmaların genel adıdır denebilir. Bu algoritmalar;

- Evrimsel Programlama
- Evrimsel Stratejiler
- Genetik Algoritma
- Genetik Programlama

olarak 4'e ayrılır. Bu algoritmalar gösterim çeşidine, kullandıkları işleçlere göre çeşitlilik gösterirler. Bu çalışmada, literatürde en yaygın kullanılan algoritmalardan biri olan genetik programlama (GP) kullanılmaktadır. Bu nedenle bir sonraki bölümde genetik programlamanın ayrıntıları verilmektedir.

Algoritma 1 : Evrimsel Hesaplama

```
1: başlangıç populasyonunu oluştur
2: while şart sağlanıncaya kadar do
3:     her bir bireyin uygunluk fonksiyonunu hesapla
4:     uygunluk fonksiyonlarını normalize et
5:     seçilim işlecini uygula
6:     elit birey seçimi yap
7:     çaprazlama opearatörünü uygula
8:     mutasyon işlecini uygula
9:     yeni populasyonu oluştur
10: end while
11: en iyi bireyi döndür
```

4.1. Genetik Programlama (GP)

Genetik Programlama, 1992 yılında Koza [50] tarafından bulunan evrimsel hesaplama tekniklerinden en fazla kullanılanlarından biridir. Diğer evrimsel hesaplama teknikleri gibi evrim teorisinden esinlenilmiştir. Darwin'in hayatta daima en güçlüler kalır prensibine dayanmaktadır. Bireyler hayatta kalmak ve nesillerini sürdürmek için kıyasıya rekabet halindedirler ve bu rekabet bireylerin sürekli kendilerini ve gelecek nesilleri geliştirmek için

yeni özellikler öğrenmelerine olanak sağlar. Bu öğrenme nesiller boyunca süren toplu deneyimlere dayanır. Darwin'e göre evrimin var olması için 4 ana sebep bulunmaktadır [51]:

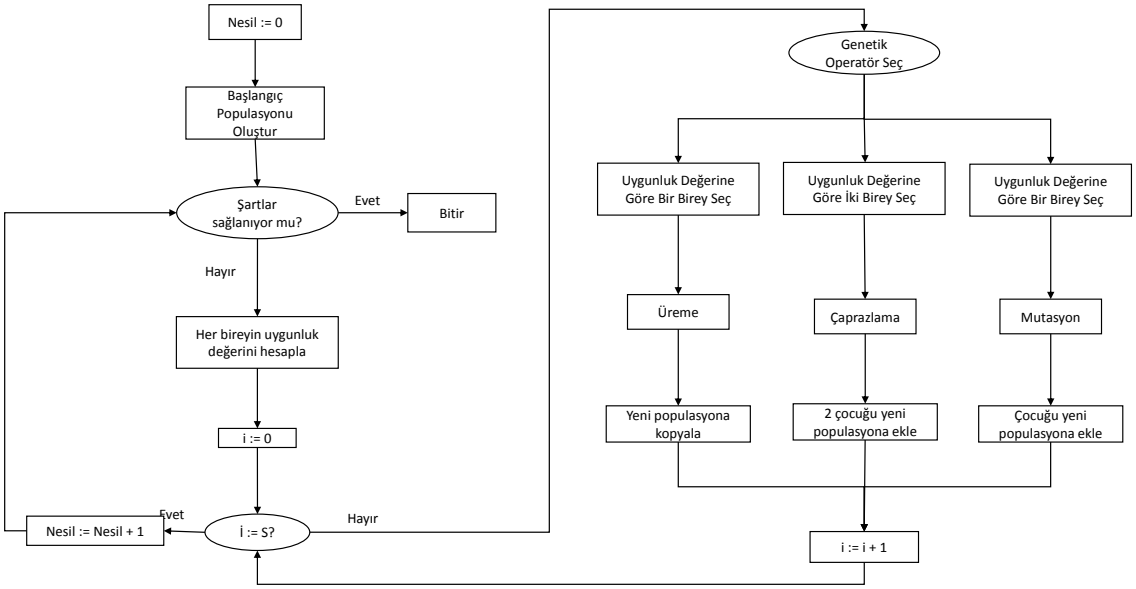
1. Popülasyondaki bireylerin üremesi
2. Bireylerin hayatta kalma şanslarını artıracak değişiklikler olması (varyasyonlar)
3. Kalıtımın olması
4. Rekabete yol açacak kısıtlı kaynakların olması

Darwin'in iddia ettiği bu sebepler zamanla bir çevredeki popülasyonun karakteristiğini değiştirir. Evrimin olması için çevrenin çok basit bir sistem olması bile yeterlidir. Önemli olan yukarıdaki şartları sağlayıp sağlamadığıdır.

Genetik programlama başlangıç programlarından yola çıkarak istenilen problem için uygun olan en iyi bilgisayar programına evrilmeyi amaçlar. Ayrıca bilgisayar programı dışında denklem vb. gibi bireyleri ifade etmede de kullanılabilir. Genetik programlama 1992 yılında ortaya çıkmasına rağmen bilgisayar programlarının evrimi fikri çok daha önceden ortaya çıkmıştır. Bilgisayar bilimcisi Turing 1950'lerin ilk yıllarında bu fikirden bahsetmiştir. Friedberg ise 1958 yılında ve arkadaşları ile yaptığı 1959 yılındaki çalışmalarında [52, 53] genetik programlamanın ilk adımları sayılabilecek fikirleri öne sürmüştür.

Genetik programlama genelde üreme, mutasyon, çaprazlama ve doğal seçilim mekanizmalarının hepsini içerir. Şekil 4.1. genetik programlamanın alt adımlarını göstermektedir. Genetik programlama, bireyler üzerinde bu mekanizmaları kullanarak ve kalite kriterine bağlı olarak nesiller boyunca sürekli bir geliştirmeyi amaçlar. Birey burada evrimleşecek tek bir bilgisayar programına, nesil iterasyona, kalite kriteri ise uygunluk fonksiyonuna karşılık gelmektedir. Bireylerin seçiminde rastgelelik olduğu için arama uzayında yerel minimumlara yakınsanamaz ve daha geniş bir arama kabiliyeti sağlayarak küresel minimuma ulaşmayı hızlandırır. Her seferinde global minimuma ulaşmayı garanti etmez ancak küresel minimuma en yakın sonuçları bulmayı garanti eder.

Genetik programlamaya olan ilgi, genetik algoritmanın verilerin gösterim kısıtlamasından dolayı daha karmaşık problemleri çözmede yetersiz kalmasından dolayı artmıştır. Genetik programlama, genetik algoritma gibi 0 ve 1'lerden oluşan dizi gösterimi yerine boyut ve şekli değişiklik gösteren hiyerarşik yapılar kullanır. Bireylerin gösterimi genelde ağaç yapısı ile



Şekil 4.1.: Genetik programlamanın kavramsal şeması

gösterilir. Her birey birden çok veri yapısı ile temsil edilebilir ve her bir veri yapısının boyutu da değişkenlik gösterebilir. Fakat kullanılacak veri yapısının bütün bireyler için aynı olması gerekmektedir.

Genetik programlama probleme özgü fonksiyonlar ve sonlandırıcılardan oluşan rastgele veya önceden hazırlanmış bilgisayar programlarından oluşan başlangıç popülasyonu ile başlar. Bu fonksiyonlar aritmetik, matematik, mantıksal veya alana özgü fonksiyonlar olabilir. Sonlandırıcılar ise gerçel değer alan değişkenlerdir. Her bir programın çözülecek probleme ne kadar uyduğu uygunluk fonksiyonu ile ölçülür. Uygunluk değeri fonksiyonu probleme özgü olarak değişkenlik gösterir. Genelde başlangıçta bir değer belirlenir ve programların bu değere ne kadar yakınsadığı ölçülerek, bir nevi hatası bulunarak uygunluk fonksiyonu hesaplanır. Uygunluk değerinin genelde sıfıra yaklaşması istenir. Sıfır, en iyi programın bulunduğunu işaret eder ve genetik programlamanın çalıştırılması durdurulur. Bazen de uygunluk fonksiyonunun değerinin artması istenebilir. Örneğin; örüntü tanıma uygulamalarında programın uygunluk fonksiyonu kaç tane örüntüyü düzgün olarak tanıdığı olabilir [50]. Problemin minimizasyon problemi ya da maksimizasyon problemi olup olmadığına göre değişiklik gösterir.

Darwin'in üreme ve hayatta kalma prensipleri ile genetik bir operasyon olan çaprazlama mekanizmaları kullanılarak mevcut bilgisayar programından yeni çocuk programlar elde

edilir. Üreme, uygunluk değeri iyi olan bireylerin seçimine dayanır. Böylece daha iyi uygunluk değerine sahip birey sonraki nesillere aktarılarak sonraki nesillerin daha kaliteli olması sağlanır. Çaprazlama mekanizmasında, iki bilgisayar programı aralarında bilgi alışverişi yaparak yeni çocuk program oluşturmak üzere seçilir. Bu iki program da yine uygunluk değerleri yüksek olanlardan seçilir. Ata program ile çocuk programın aynı boyut ve şekilde olmasına gerek yoktur. Aralarında değiştirilecek bilgi rastgele olarak seçilir. Mutasyon mekanizmasında ise tek bir bilgisayar programı seçilir ve fonksiyonlarından veya sonlandırıcılarından biri değiştirilir. Yeni oluşan programlar eskilerinin yerine geçer. Mutasyon işleci, popülasyondaki çeşitliliği sağlar. Gerekli şartlar sağlanıncaya kadar bu işlemler nesiller boyunca devam eder. En iyi bilgisayar programı sadece son iterasyonda bulunmayabilir, herhangi bir iterasyonda bulunabilir ve bu programa *best-so-far* birey denir.

Genetik programlama, klasik arama metotlarından farklı olarak tek bir nokta üzerinde işlem yapmak yerine arama uzayındaki yüzlerce nokta üzerinde eşzamanlı olarak işlem yapar. Her bir nokta bir bireyi temsil eder ve noktaların kümesine popülasyon denir. Popülasyondaki her bir bilgisayar programı hiyerarşik olarak yapılandırılmıştır. Boyutu, şekli ve içeriği birbirlerinden farklılık gösterir ve çalışma zamanında dinamik olarak değiştirilebilir.

Genetik programlamada genelde ağaç, lineer ve çizge yapıları olmak üzere üç gösterim kullanılır. Genelde bu üç gösterimden en çok ağaç yapısı kullanılır. Fakat bu yapılar sanal yapılardır [51]. Aslında bir bilgisayar programı terminal, fonksiyon ve bunların nasıl ve ne şekilde bir araya geleceklerini belirten kurallardan oluşur. Program, komut dizilerinden oluşur ve her programın bilgisayarda çalıştırılabilmesi ve uygunluk fonksiyonlarının hesaplanabilmesi için düğümlerinin uygun çalıştırma sırasına getirilmesi gerekmektedir.

Bu tezde genetik programlamadaki her bir programı ağaç yapısında kullandığımızdan dolayı bundan sonra genetik programlama dendiğinde ağaç yapılı genetik programlamadan bahsedilecektir.

Her bir bilgisayar programı farklı sayıda ve çeşitte fonksiyon ve sonlandırıcılardan oluşur. Sonlandırıcılar sistemde kullanılacak giriş değerleri, sistemde kullanılacak sabitler ve parametre almayan fonksiyonlardan oluşur. Sonlandırıcılar ağaç yapısında her zaman en son dalda bulunurlar. Fonksiyonlar ise sonlandırıcıları veya başka değişkenleri alarak işlem yapan ve sonuç dönen bileşenlerdir. Genetik programlama da her bir sonlandırıcı ve fonksiyona düğüm denir. Kullanılabilecek fonksiyonlar [50];

- matematik fonksiyonlar (tan, log, sqrt, ...)
- aritmetik fonksiyonlar (-, +, *, ...)
- mantıksal fonksiyonlar (AND, OR, NOT, ...)
- şart işleçleri (if-else)
- döngü işleçleri (for, while, ...) ve
- diğer probleme özgü fonksiyonlardan oluşabilir.

Bir problem için uygun fonksiyon ve terminal kümesinin seçilmesi önemlidir. Eğer kümedeki eleman sayısı düşük tutulursa problem yeterli olarak temsil edilemeyeceğinden dolayı yeterli çözüme ulaşılamaz. Eğer küme büyük seçilirse bu sefer de arama uzayı çok büyüyeceğinden dolayı çözüm bulmak zorlaşır.

Genetik programlamada kullanılacak fonksiyon ve terminallere örnek vermek gerekirse [50];

$$F = \{+, \ -, \ sqrt, \ log\}$$

fonksiyon kümesi ve

$$T = \{T0, \ T1\}$$

terminal kümesi olsun. Burada $T0$ ve $T1$, F fonksiyon kümesinde bulunan fonksiyonlar için gerçel sayılı giriş değerleridir. Bilgisayar programı oluşturmak için kullanılacak yani düğümlerde yer alacak birimlerin kümesi

$$C = F \cup T = \{ \ +, \ -, \ sqrt, \ log, \ T0, \ T1\}$$

olur. Bu birimleri kullanan örnek bir kontrol akış çizgeleri Şekil 4.2. ve Şekil 4.3.'te görülmektedir.

Genetik programlamada kullanılacak her bir fonksiyonun *kapalılık* özelliği olmak zorundadır. Kapalılık özelliği; fonksiyon kümesindeki her bir fonksiyonun parametre olarak aldığı

değeri kabul etmesi ve hata vermeden işleyebilmesi demektir. Örneğin [50]; bölme işleminde bölen kısmı sıfır değerini alırsa *sıfıra bölünme hatası alınır* ve hatadan dolayı genetik programlama çalıştırmasını durdurur. Buna izin vermemek için bölme fonksiyonunda bölen kısmının değerinin sıfır gelmesi durumunun işlenmesi gerekmektedir. Problemin doğasına göre eğer bölme kısmına sıfır gelirse bölme işleminin değerine sıfır, eksi sayı veya çok büyük bir sayı verilebilir. Sıfırın logaritmasını alma, negatif sayının karekökünü alma vb. gibi işlemlere de dikkat edilmelidir.

Şekil 4.1. genetik programlamanın tüm adımlarını göstermektedir. Bu şekil üzerinden genetik programlama adımları anlatılacaktır.

4.1.1 Başlangıç Popülasyonu

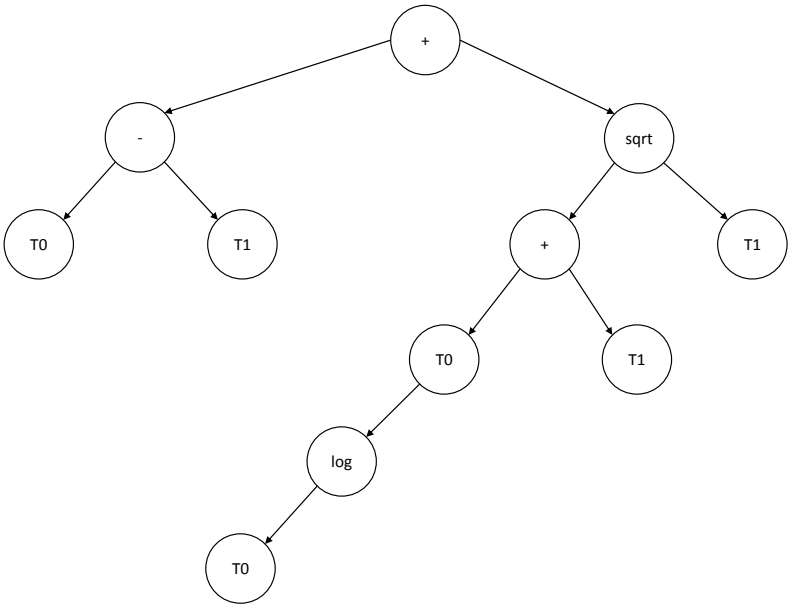
Genetik programlamanın ilk adımı evrimleştirmek için gerekli çeşitli programlardan oluşan başlangıç popülasyonunun oluşturulmasıdır. Programları oluştururken maksimum program boyutu parametresi dikkate alınır. Bu parametre ya bir programda yer alan toplam düğüm sayısını ya da ağacın derinliğini ifade eder. Genelde bu parametre ağacın derinliğini belirtmek için kullanılır. Ağacın derinliği kök düğüm ile en alttaki terminal düğüm arasındaki derinliktir. Tüm düğümleri 3 çocuk ihtiva eden bir yapıda toplam $3^{derinlik}$ kadar düğüm bulunur.

Ağaç yapısını oluşturmaya bir örnek vermek için yukarıdaki C kümesini ele alalım.

$$C = F \cup T = \{\ +,\ -,\ sqrt,\ log,\ T0,\ T1\}$$

Koza'nın [50] kitabında belirttiği 2 tip ağaç oluşturma metodu vardır: *full* ve *grow*. *grow* metotunda düğümler için fonksiyon ve terminal rastgele bir şekilde seçilir ve bu da ağacın dengesiz bir hal almasına sebep olur. Yani ağacın bir tarafı diğer tarafından daha büyük olabilir; bunun için bir kısıtlama yoktur. Bir dalın düğümü terminal olmuşsa ağacın derinliğine bakılmadan büyüme o dal için durur. Şekil 4.2. *grow* metodu ile oluşturulmuş bir ağaç örneğini göstermektedir. Bu şekilde ağaç 5 derinliktedir ve düğümlere rastgele fonksiyon ve terminal seçildiğinden dolayı kök düğümün sağ dalı, sol dalından daha fazla büyümüştür.

Şekil 4.3. ise *full* metodu ile oluşturulmuş ağacı göstermektedir. Bu metotta maksimum derinliğe ulaşana kadar düğümler için fonksiyon seçilir ve maksimum derinlikte ise sadece

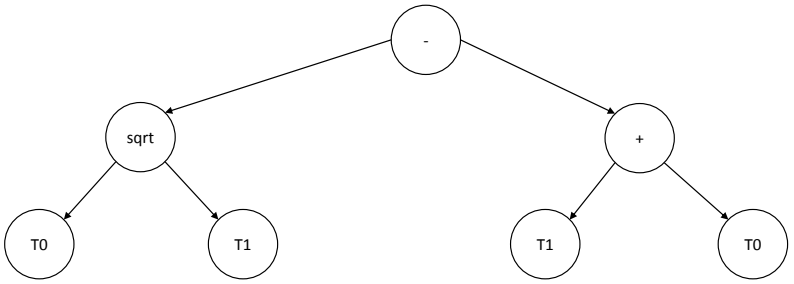


Şekil 4.2.: *grow* metodu ile oluşturulan ağaç örneği

terminal düğümler yer alır. Eğer maksimum program boyutu bir programda yer alan toplam düğüm sayısını temel alıyorsa o sayıya gelince ağacın oluşturulması durdurulur.

grow ve *full* metotları başlangıç popülasyonunu tekdüze olarak oluşturur. Yani popülasyondaki her bireyin şekli ve boyutu benzer olur. Fakat genetik programlamanın önemli özelliklerinden birisi çok çeşitli olmasıdır. Yani her bireyin boyutu ve şekli birbirinden farklı olabilir. Bunu sağlamak için *ramped-half-and-half* metodu geliştirilmiştir [50]. Bu metotta ağaç 2 ile maksimum derinlik arasındaki bütün sayıların derinliğinde olur. Örnek vermek gerekirse; maksimum derinliği 5 olarak ele alalım. Ağacı oluştururken 2, 3, 4 ve 5 derinlikli alt ağaçlar oluşturulabilir. Daha sonra bu dört derinlik eşit sayıda olacak şekilde ağaç oluşturulur. Yani her bir derinlik %25 şansa sahiptir. Ve her bir derinliğin yarısı *grow* metodu ile diğer yarısı ise *full* metodu ile oluşturulur.

Başlangıç popülasyonunu oluştururken programların birbiriyle aynı olması pek istenmez. Bunun için her birey oluşturulduğunda çeşitliliği sağlamak için kontrol yapılır ve başlangıç popülasyonunun birbirinden farklı olması amaçlanır. Ancak ileriki nesillerde genetik işleçler aracılığı ile birbirine benzeyen bireyler oluşabilir.



Şekil 4.3.: *full* metodu ile oluşturulan ağaç örneği

4.1.2 Uygunluk Değerini Hesaplama

Uygunluk değeri, bir popülasyondaki birey için hayatta kalma şansının yüksekliği olarak söylenebilir. Probleme ne kadar uygun bir uygunluk değeri varsa bir bireyin bir sonraki nesile taşınma şansı bir o kadar artar. Üreme, çaprazlama ve mutasyon işleçleri için birey seçiminde uygunluk değerine bakılarak seçim yapılır. Uygunluk değeri fonksiyonu problemden probleme farklılık gösterir. Uygunluk değerinin amacı öğrenme algoritmasına popülasyondaki bireylerle ilgili geri dönüş sağlamaktır [51].

Literatürde 4 farklı uygunluk fonksiyonu vardır [51]:

- Ham uygunluk fonksiyonu; Problemin kendi içinde tanımladığı uygunluk fonksiyonunun kendisidir. En iyi birey yüksek uygunluk değerine sahip olabildiği gibi düşük uygunluk değerine de sahip olabilir.

$$r(i,t) = \sum_{j=1}^{N_e} |S(i,j) - C(i,j)|$$

  M boyutlu bir popülasyonun t.nci nesil i.nci bireyinin uygunluk değeri yukarıdaki

gibidir. Burada $S(i,j)$ j.nci durum için uygunluk değeri, $C(i,j)$ j.nci durum için olması gereken değer ve N_e ise bütün durumların kümesidir.

- Standartlaştırılmış uygunluk fonksiyonu: Problemin tipine bakmadan olması gereken en iyi bireyin uygunluk değeri daima 1 değerini alır. Uygunluk değerinin alabileceği en büyük değer ile ham uygunluk değerinin birbirinden çıkarılmasıyla hesaplanır [50].

$$s(i,t) = r_{max} - r(i,t)$$

- Ayarlanmış uygunluk fonksiyonu: Uygunluk değeri daima 0 ve 1 arasında olur. Popülasyondaki en iyi bireyin daha büyük uygunluk değeri olur [50].

$$a(i,t) = \frac{1}{1 + s(i,t)}$$

- Normalize uygunluk fonksiyonu; Ayarlanmış uygunluk fonksiyonu gibi 0 ve 1 arasında değer alır, daha iyi birey daha yüksek uygunluk değerine sahip olur ve normalize edilmiş uygunluk fonksiyonu değerlerinin toplamı 1'e eşit olur

$$n(i,t) = \frac{a(i,t)}{\sum_{k=1}^{M} a(k,t)}$$

4.1.3 Genetik İşleçler

Bu bölümde anlatılacak işleçler yeni bireylerin oluşturulması ve popülasyondaki genetik çeşitliliği sağlamak için kullanılmaktadır.

4.1.3.1 Seçme

Seçme işlemi genetik programlamanın en önemli adımlarından biridir. Genelde uygunluk değerine bağlı olarak seçme işlemi yapılır. Seçme işlemi sonucunda seçilen birey

veya bireyler üreme, çaprazlama veya mutasyon işleçlerine gönderilerek yeni nesil popülasyonunda yer almaya hak kazanır. Seçme işleminde uygunluk değeri yüksek olan bireylerin seçimi sağlanır ki, böylece gelecek nesillerin daha iyi sonuç vermeleri hedeflenir. Rulet tekerleği, turnuva, kesme ve sıra seçimi gibi seçme algoritmaları vardır. Bu tezde turnuva seçme algoritmasını kullandığımız için sadece onunla ilgili bilgi vereceğiz.

4.1.3.1.1 Turnuva Seçimi

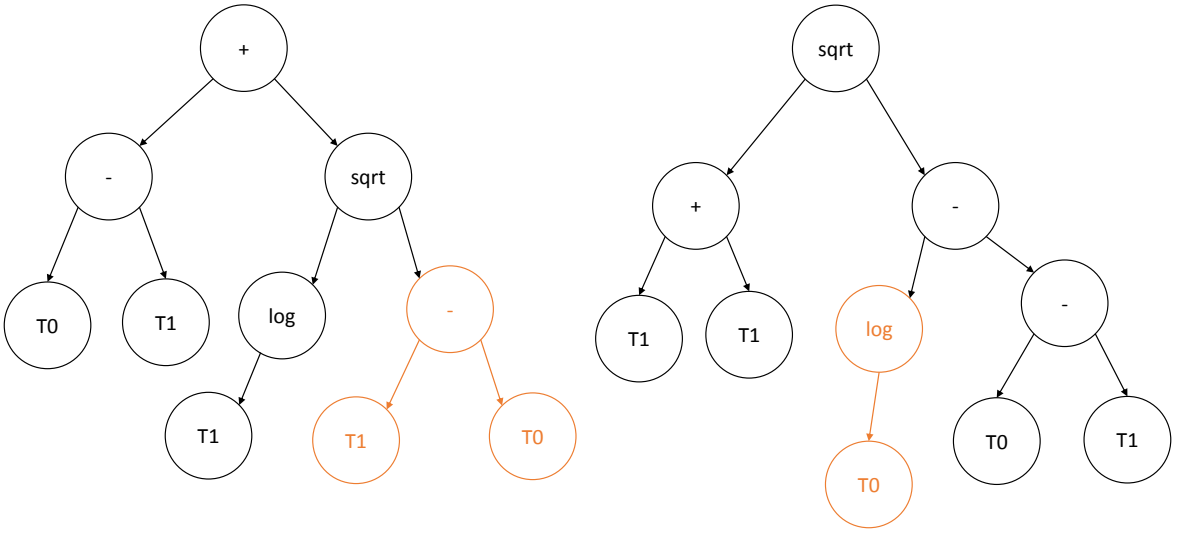
Turnuva seçimi algoritmasında bütün popülasyon kullanılmaz. Onun yerine popülasyondan bir birey seçilir ve uygunluk fonksiyonu hesaplanır. Sonra sırayla turnuva boyutu kadar eleman ile bu ilk birey uygunluk değerlerine göre kıyaslamaya tabi tutulur. Problemin maksimizasyon veya minimizasyon problemi olup olmadığına göre kıyaslama işlemi değişiklik gösterir. Eğer minimizasyon problemi ise yeni seçilen bireyin uygunluk değeri ilk bireyden küçükse bir sonraki nesile aktarılacak birey olur. Eğer büyükse ilk seçilen birey bir sonraki nesile aktarılır. Maksimizasyon problemi ise tam tersi bir kıyaslama yapılır Bireyler rastgele biçimde seçilir. Algoritma 2, turnuva seçiminin adımlarını göstermektedir.

Algoritma 2 : Turnuva Seçimi Algoritması

turnuva boyutu := k
i := 0
$F(Best) \leftarrow$ rastgele populasyondan bir birey seç ve uygunluk fonksiyonunu hesapla
while i $<$ k **do**
    $F(B_i) \leftarrow$ rastgele populasyondan yeni birey seç ve uygunluk fonksiyonunu hesapla
    ilk seçilen birey $Best$ ile yeni seçilen bireyin B_i uygunluk fonksiyonlarını karşılaştır
    if $F(B_i) < F(Best)$ **then**
        $Best = B_i$
        $F(Best) = F(B_i)$
    end if
    $i \leftarrow i + 1$
end while

4.1.3.2 Üreme

Uygunluk değerlerine bakarak seçme işlemi ile tek bir birey seçilir. Bu birey üzerinde hiçbir değişiklik yapmadan yeni popülasyona kopyalanır.



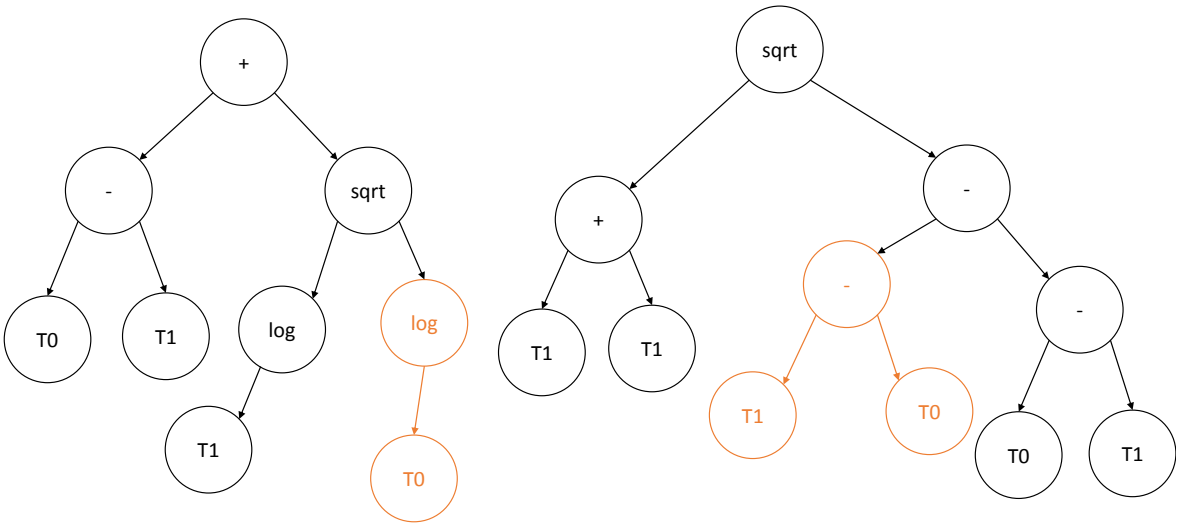
Şekil 4.4.: Çaprazlama uygulamadan önce ağaçların durumu

4.1.3.3 Çaprazlama

Çaprazlama, popülasyondan iki birey seçerek bunlar arasında bilgi alışverişi yapar ve sonunda iki farklı birey oluşturup yeni popülasyona ekleyerek popülasyondaki çeşitliliği artırmaya yarar. Genellikle çaprazlama olasılığı yüksek verilmektedir. Çaprazlamanın adımları aşağıdaki gibidir [51]:

- Eşleme havuzundan uygunluk değerlerine göre iki tane ata birey seçilir.
- Her bir ata bireyden rastgele olarak bir alt ağaç seçilir. Terminallere ve fonksiyonlara ayrı olasılık değeri verilebilir.
- Seçilen alt ağaçlar birbirleri ile yer değiştirerek iki tane yeni birey oluşturulur.

Çaprazlama için bir örnek Şekil 4.4. ve 4.5.'de verilmiştir. Kapalılık özelliğinden dolayı alt ağaçlar, birbirleriyle değiştirildikten sonra da programları bozmayacak şekilde seçilir. Ayrıca ağaçların düzensiz bir şekilde büyümesini önlemek için maksimum derinliği geçmeyecek şekilde alt ağaç seçimi yapılır.



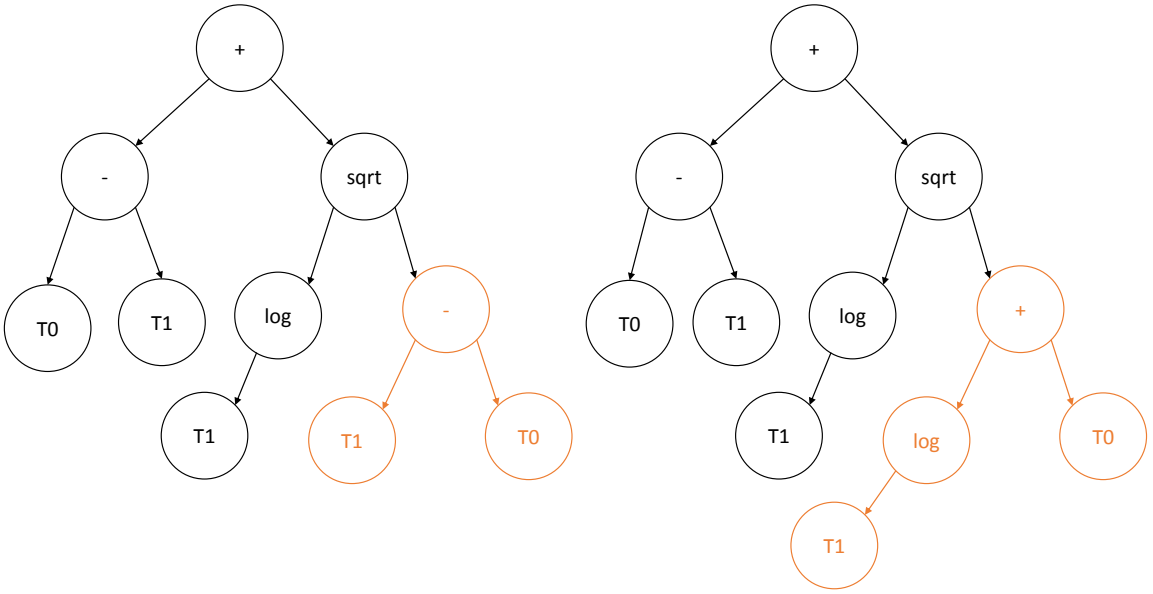
Şekil 4.5.: Çaprazlama uygulandıktan sonra ağaçların durumu

4.1.3.4 Mutasyon

Mutasyon işlecinde tek bir birey seçilir, bu bireyden rastgele bir düğüm seçilir ve bu düğüm alt ağaçlarıyla beraber yeniden oluşturulan düğüm ve onun alt ağacı ile değiştirilir. Yeni düğüm, başlangıç popülasyonu oluşturulurken kullanılan yöntemle oluşturulur ve maksimum derinliğe dikkat edilir. Seçilen düğüm fonksiyon veya terminal düğüm olabilir. Çocuk birey, ata birey yerine popülasyona dahil edilir. Şekil 4.6. bir düğüm için mutasyon örneğini göstermektedir.

4.1.4 Sonlandırma Kriteri

Genetik programlama aynı esinlendiği doğa gibi hiç bitmeyen bir işlemdir [50]. Bu yüzden genetik programlamayı sonlandırmak için bir sonlandırma kriteri belirlemek gerekir. Genelde ya genetik programlamanın koşacağı nesil sayısı ile ya da önceden belirlenmiş uygunluk değerine eşit veya küçük uygunluk değerine sahip herhangi bir nesilden bir birey bulunduğunda sonlanır. Fakat genetik programlamada genellikle en uygun çözümü bulmak zor olduğundan sonlandırma kriteri olarak nesil sayısı kullanılmaktadır.



Şekil 4.6.: Mutasyon uygulamadan önce ve uygulandıktan sonra ağaç durumu

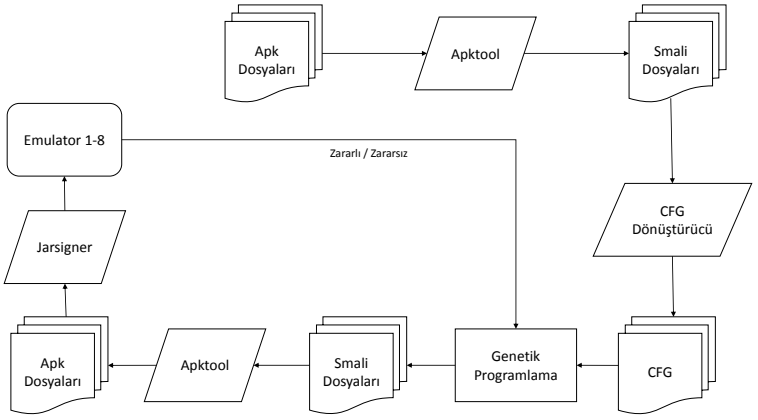
4.1.5 Kontrol Parametreleri

Genetik programlamanın çalışması için gerekli kontrol parametreleri vardır. Popülasyon boyutu, maksimum birey sayısı, çaprazlama, mutasyon ve seçme olasılıkları, maksimum nesil sayısı, ağaç oluşturma metodu, ağacın maksimum derinliği, elit birey sayısı, turnuva boyutu bunlardan bazılarıdır.

Bölüm 5.

MODEL VE YÖNTEM

Bu bölümde, bu çalışmada önerilen model açıklanacaktır. Önerilen modelin kavramsal şeması Şekil 5.1.'de görülmektedir.



Şekil 5.1.: Kavramsal şema

Önceden de belirttiğimiz gibi kaynak kod olarak tekrar *apk* dosyası oluşturmaktaki sorunlardan dolayı java kodu yerine smali kodu kullanılmaktadır. Bu yüzden, ilk önce apktool [48] aracı kullanılarak Android uygulamalarının kaynak kodu çıkartılır. Daha sonra çıkarılan smali kodlarından her fonksiyon için ayrı ayrı kontrol akış çizgeleri çıkarılmaktadır. Kontrol akış çizgeleri bir programın yürütülürken içinden geçebileceği bütün yolların çizgesel olarak gösterimidir. Örnek bir kontrol akış çizgesi Şekil 5.3.'te görülmektedir.

Bu kontrol akış çizgeleri, genetik programlama için girdi olmakta ve başlangıç popülasyonunu oluşturmaktadır. Genetik programlama bireylerin gösterimi için ağaç yapısını kullandığından, kontrol akış çizgelerinin genetik programlama için uygun hale dönüştürülmesi kolaylık sağlamaktadır. Böylece her fonksiyon genetik programlamada bir ağaç olarak temsil edilecektir. Bazı fonksiyonlar özyinelemeli fonksiyonlar çağırdıklarından dolayı, bu fonksiyonların kontrol akış çizgeleri ağaç yapısı yerine çizge yapısında olacaktır. Bu yüzden özyinelemeli fonksiyon çağıran uygulamalar deneylerde kullanılmamıştır. Genetik programlamada bir Android uygulamasını temsil eden birey sahip olduğu fonksiyon sayısı kadar ağaçtan oluşmaktadır. Genetik programlamada bu kontrol akış çizgeleri üzerinde karıştırma teknikleri ile birlikte çaprazlama ve mutasyon gibi genetik işleçler uygulanarak yeni bir zararlı yazılım evrimleştirmesi gerçekleştirilir. Daha sonra tekrardan kontrol akış çizgelerinden *smali* kodları oluşturulmakta ve bu *smali* kodları yeni *apk* dosyası oluşturmak için kullanılmaktadır. Yeni üretilen *apk* dosyası, uygunluk fonksiyonunun hesaplanması için emulatörlere gönderilmeden önce imzalanmaktadır. İmzalamak için önce keytool [54] programı kullanılarak rastgele bir anahtar üretilmektedir. Daha sonra bu anahtar kullanılarak jarsigner [55] programı ile uygulama imzalanmaktadır. Bu çalışmada statik analiz tekniklerini uygulayan 8 tane antivirüs kullanılmaktadır. Uygulamalar imzalandıktan sonra bu 8 antivirüsü içeren 8 farklı resmi Android emülatörü üzerinde çalıştırılır ve her programın uygunluk fonksiyonu hesaplanır. Genetik programlama sonlandırma kriterine kadar aynı işlemleri tekrar eder. Burada sonlandırma kriteri için, ideal sonucu elde etmek zor olduğundan dolayı nesil sayısı kullanılmaktadır. Her birine ayrı antivirüs sistemi yüklenen emulatörler eğer yeni üretilen dosyayı zararlı yazılım olarak tanıyorlarsa 1 değerini, tanımıyorlarsa veya hiç bir kanıya varmıyorlarsa 0 değerini dönmektedir. Buradaki amacımız emulatörlerin tespit etme oranını düşürmek yani sıfıra yaklaştırmaktır.

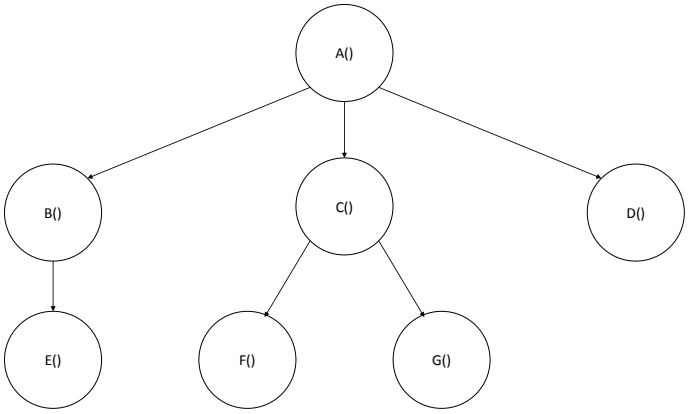
Genetik programlamayı gerçekleştirmek için evrimsel hesaplama algoritmalarını içeren Java tabanlı bir araç olan *ecj* v.20 [56] aracı kullanılmıştır. *ecj* aracı, gerekli bütün ayarları ve sınıfları, çalışma zamanında kullanıcı tanımlı parametre dosyasından alan oldukça esnek bir yapıya sahiptir.

Şekil 5.2.: Örnek kaynak kodu

```
.method public A(Landroid/widget/AdapterView;Landroid/view/View;IJ)V
  .
  .
  invoke-direct {p0}, Lcom/keepwired/utility/controls/FileChooser;->B()V
  .
  .
  invoke-direct {p0}, Lcom/keepwired/utility/controls/FileChooser;->C()V
  .
  .
  invoke-virtual {p0, p0, v3}, Lcom/keepwired/utility/controls/FileChooser;->D(Landroid/content/Context;Ljava/lang/String
      ;)V
  .
  .
.end method


.method private B()V
  .
  .
  invoke-direct {v3, p0}, Lcom/keepwired/utility/controls/FileChooser$2;->E(Lcom/keepwired/utility/controls/FileChooser;)V
  .
  .
.end method


.method private C()V
  .
  .
  invoke-direct {v4, p0}, Lcom/keepwired/utility/controls/FileChooser$3;->F(Lcom/keepwired/utility/controls/FileChooser;)V
  .
  .
  invoke-direct {v3, p0}, Lcom/keepwired/utility/controls/FileChooser$4;->G(Lcom/keepwired/utility/controls/FileChooser;)V
  .
  .
.end method


.method public D(Landroid/content/Context;Ljava/lang/String;)V
  .
  .
.end method
```

5.1. Bireylerin Gösterimi

Genetik programlamada *apk* dosyalarını belirtmek için Android uygulamalarındaki her sınıfın her metodunun kontrol akış çizgesi çıkartılmıştır. Bir kontrol akış çizgesinde, her metot bir düğüm olarak temsil edilir ve kenarlar ise düğümler arası ilişkiyi belirtir. Bir düğüme gelen ok; düğümün o okun geldiği düğüm tarafından temsil edilen metodun çağrılan bir metodu olduğunu belirtir. Şekil 5.3., Şekil 5.2.'de gösterilen kaynak kodun kontrol akış çizgesini göstermektedir. Koyu renkli kısımlar metotların isimlerini göstermektedir. Bu şekilde A metodunun B, C ve D olmak üzere 3 tane metot çağırdığı görülmektedir. B metodu E metodunu, C metodu ise F ve G metotlarını çağırmaktadır. D, E, F ve G metotları ise herhangi bir metot çağrısı yapmamaktadır ve kontrol akış çizgesinde sonlandırıcı konumundadır. Her Android uygulamasının farklı sayıda metodu bulunduğundan dolayı genetik programlamada her birey farklı sayıda ağaçtan oluşur. Çaprazlama ve mutasyon işleçleri, bu çıkarılan kontrol akış çizgelerindeki ağaçlara uygulanır. Bu genetik işleçlerin ayrıntıları bir sonraki bölümde detaylandırılacaktır.



Şekil 5.3.: Şekil 5.2.'de gösterilen kaynak kodu için kontrol akış çizgesi

5.2. Genetik İşleçler

5.2.1 Çaprazlama

Çaprazlama işleci, bireyler arasındaki değişimi alt ağaçlar üzerinden yapmaktadır. Genetik programlama, çaprazlama işlecine uygulayacağı bireyleri seçtikten sonra, bireylerin yer değiştirecek alt ağaçlarını seçer. Burada dikkat edilen önemli bir husus, seçilen alt ağaçların birbiri ile uyumlu olmasıdır. Burada uyumluluk, seçilen alt ağaçların ata düğümünün aynı sayı ve türde parametre alması ve aynı türde değer geri döndürmesidir. Bu kısıtlamayı yapmamızın nedeni derlenebilir ve çalıştırılabilir bireylerin türetilmesidir. Genelde genetik programlama kullanılırken çaprazlama oranı 0.9 gibi yüksek bir değer olarak verilir. Fakat yapılan deneyler sonucu çaprazlama oranının artmasının derlenemeyen ve buna bağlı olarak çalıştırılamayan yeni zararlı yazılımlar oluşturduğu görülmüştür. Bu yüzden deneylerde ya hiç çaprazlama kullanılmamıştır ya da 0.1 gibi düşük bir oran verilerek kontrollü bir yer değişimi yapılması sağlanmıştır. Literatürde yapılan ayrıntılı bir karşılaştırma [57], çaprazlama işlecinin, mutasyon işlecine çok büyük bir üstünlüğünün olmadığını göstermiştir.

Çaprazlama işleci, zararlı yazılımların benzer zararlı işlevlerinin, birbiri arasında yer değiştirmesini veya bir arada toplanmasını sağlayarak zararlı yazılımların imzalarının değişmesine ve daha güçlü zararlı yazılımlar geliştirilebilmesine olanak sağlar. Bu sayede değişik imzalara sahip zararlı yazılımların üretilmesi beklenmektedir.

5.2.1.1 Dinamik Analiz

Çaprazlama işleci kullandığımız zaman yeni üretilen bireylerin düzgün bir şekilde çalıştığını araştırmamız gerekmektedir. Bu işlemi yapabilmek için bir dinamik analiz aracı geliştirilmiştir [58]. Bu araç, Android uygulamalarından çalışma zamanında bazı öznitelikler çıkartır ve bu öznitelikleri makine öğrenmesi yöntemleriyle kullanarak uygulamaların zararlı olup olmadığını tespit etmeye çalışır. Bu araca verilen uygulamalar eğer çalıştırılabilir durumda değilse dinamik analiz sonucu bize hiç bir değer dönmemektedir. Bu tür bireylere uygunluk değerleri yüksek verilmiş ve bir sonraki popülasyonlarda yer almaları engellenmiştir. Eğer çalıştırılan uygulamalar için dinamik analiz aracı sonucu dönerse, bu uygulamaların çalıştırılabilir olduğunu ve çaprazlama işleminin düzgün bir şekilde gerçekleştiğini göstermektedir.

Geliştirdiğimiz dinamik analiz aracı, Android uygulamalarını Android emülatörü üzerinde çalıştırarak uygulama ile ilgili öznitelikleri çıkartmakta ve bu öznitelikleri makine öğrenmesi yöntemleriyle işlemekte ve zararlı olup olmadığını sınıflandırmaktadır. İlk aşamada uygulamaların dinamik olarak analizi için açık kaynak kodlu bir dinamik analiz aracı olan DroidBox [59] kullanılmıştır. Her bir uygulama tek tek DroidBox üzerinde 60 saniye boyunca çalıştırılmış ve DroidBox'un elde ettiği veriler *json* formatında kaydedilmiştir. Ayrıca bu ve-riler, üzerlerinde gerekli analizleri yapabilmek adına MySql [60] veri tabanına kaydedilmiştir. İkinci aşamada veri tabanına atılan tüm öznitelikler incelenerek uygulamaların zararlı veya zararsız olup olmadığını ayırt eden gerekli öznitelikler çıkartılmıştır. Bunun için eldeki ve-riler üzerinden istatistiki bir çalışma yapılmıştır. Yapılan çalışmalar sonucu 44 adet öznitelik belirlenmiştir. Her bir uygulama için bu 44 öznitelik özellik ilişkili dosya formatında (*arff*) kaydedilmiştir. En son aşamada ise bu *arff* dosyaları açık kaynaklı veri madenciliği uygulaması olan *Weka* [61] aracına verilerek işlenmekte ve uygulamaların zararlı olup olmadığı belirlenmektedir. Eldeki verilerin %90'ı model öğrenimi, geri kalan %10'u ise modelin test edilmesi için kullanılmıştır. Veriler birçok algoritma üzerinde çalıştırılmıştır ve çoğunda yüksek doğruluk değerleri elde edilmiştir. Bu algoritmalar arasında en yüksek doğruluk değerine sahip algoritma rastgele orman algoritması [62] olmuştur. Bu nedenle bu tez için kullanılan dinamik analiz aracında rastgele orman algoritması kullanılmıştır.

5.2.2 Mutasyon

Mutasyon işleci genetik programlamada çeşitliliği sağlamaktadır. Genetik programlama, seçtiği bireyin ağaçlarından bir düğüm seçer ve bu düğüm üzerinden değişikliği yapar. Uygunluk değerine göre seçilen bireylerin kontrol akış çizgelerinden biri ve bu çizgenin bir düğümü rastgele seçilir. Düğümün temsil ettiği metodun kaynak kodları üzerine rastgele seçilen karıştırma tekniklerinden biri uygulanır. Bir düğüm bir sonraki nesilde yeniden seçilebilir, fakat önceden uygulanan bir karıştırma tekniği çıkartılarak diğer karıştırma tekniklerinden biri uygulanır. Çaprazlama işlecindeki kısıtlamalardan dolayı mutasyon oranına, klasik genetik programlamada kullanılan oranlardan daha yüksek olan 0.9 değeri verilmiştir. Buradaki amaç, kodu mümkün olduğunca karıştırarak, kötü amaçlı kodun tespit edilmesini zorlaştırmaktır. Ayrıca çaprazlama değeri çok düşük seçildiğinden, koddaki değişiklikler daha çok mutasyon işleci ile gerçekleştirilmektedir. Genetik programlamada kullanılan parametreler Çizelge 5.1.'de verilmiştir. Burada bahsedilmeyen genetik programlamaya ait diğer kontrol parametreleri *ecj* aracının varsayılan parametreleri olarak seçilmiştir. Bu varsayılan parametrelerin bir tanesi ağaç boyutudur. Burada ağaç boyutu 17 olarak alınmaktadır. Genelde genetik programlamanın kod şişirme özelliğinden dolayı, araştırmacılar ağaç boyutunu küçültmektedirler. Ama burada ağacın boyutunun büyütülmesi daha karışık programların üretilmesini sağlamaktadır. Bu da literatürde belirtildiği gibi saldırı ve zararlı yazılım üretilmesinde pozitif bir etki sağlamaktadır [23].

Mutasyon işleci ise aşağıda anlatılan karıştırma tekniklerini kullanmaktadır. Kontrol akış çizgelerinden rastgele seçilen bir ağacın rastgele seçilen bir düğümü üzerinde işlem yapar.

Çizelge 5.1.: Genetik programlamada kullanılan parametreler

Parametre	Değer
Populasyon boyutu	50
Nesil sayısı	50
Çaprazlama oranı	0.1
Mutasyon oranı	0.9
Turnuva boyutu	7

5.2.2.1 Karıştırma Teknikleri

Bu bölümde mutasyon işleci tarafından kullanılan karıştırma tekniklerinden bahsedilecektir. Bu teknikler, zararlı yazılımların özgün zararlı işlevlerinin korunarak farklı çeşitlerinin üretilmesini sağlamaktadır. Bu teknikleri uygulamak için bazı adımlar gerekmektedir. Bu adımlar şöyle özetlenebilir:

- Girdi olarak *apk* dosyası alınır ve *apktool* [48] aracı kullanılarak smali kodları çıkartılır.
- Smali kodları üzerine karıştırma teknikleri uygulanır.
- Smali kodları *apktool* [48] kullanılarak tekrar *apk* dosyasına paketlenir.
- Son olarak cihaz veya emulatöre yüklenmeden önce imzalanır.

Genetik programlamanın mutasyon işlecinde 5 farklı karıştırma tekniği kullanılmıştır. Aşağıda FakeNetflix virüsünde yer alan bir kod parçası görülmektedir. Karıştırma teknikleri bu kod parçası üzerinde anlatılacaktır. Mutasyon işlecinde kullanılan teknikler aşağıda açıklanmıştır:

Şekil 5.4.: Kod parçası

```
new-instance v0, Landroid/content/Intent;
const-class v1, Lcom/android/MonitorService;
invoke-direct {v0, p1, v1}, Landroid/content/Intent;-><init>(Landroid/content/Context;Ljava/lang/Class;)V
.local v0, timer:Landroid/content/Intent;
const-string v1, "com.android.MonitorService"
invoke-virtual {v0, v1}, Landroid/content/Intent;->setAction(Ljava/lang/String;)Landroid/content/Intent;
const/4 v1, 0x2
invoke-virtual {v0, v1}, Landroid/content/Intent;->setFlags(I)Landroid/content/Intent;
invoke-virtual {p1, v0}, Landroid/content/Context;->startService(Landroid/content/Intent;)Landroid/content/ComponentName;
return-void
```

5.2.2.1.1 Yerel Değişkenleri Yeniden Adlandırma (Rename Local Identifier - RI)

Bu teknikte, metotlarda kullanılan yerel değişkenlerin isimleri değiştirilmektedir. Bunun için rastgele karakter dizisi üreteci kullanılmaktadır. Bu nedenle, değişken isimleri anlamlı değerler olarak üretilmemektedir. Mutasyon işlecinde 10 uzunluklu karakter dizileri kullanılmış ve bu karakter dizileri metodun kodlarında yer alan yerel değişkenler ile değiştirilmiştir. Şekil 5.5.'te koyu renkli satırda orijinal adı *timer* olan yerel değişkenin *eEMYYuSmnY* olarak değiştirildiği görülmektedir.

Şekil 5.5.: Yerel değişkenleri yeniden adlandırma tekniği uygulanmış kod parçası

```
1  new-instance v0, Landroid/content/Intent;
2  const-class v1, Lcom/android/MonitorService;
3  invoke-direct {v0, p1, v1}, Landroid/content/Intent;-><init>(Landroid/content/Context;Ljava/lang/Class;)V
4  .local v0, eEMYYuSmnY:Landroid/content/Intent;
5  const-string v1, "com.android.MonitorService"
6  invoke-virtual {v0, v1}, Landroid/content/Intent;->setAction(Ljava/lang/String;)Landroid/content/Intent;
7  const/4 v1, 0x2
8  invoke-virtual {v0, v1}, Landroid/content/Intent;->setFlags(I)Landroid/content/Intent;
9  invoke-virtual {p1, v0}, Landroid/content/Context;->startService(Landroid/content/Intent;)Landroid/content/ComponentName;
10 return-void
```

5.2.2.1.2 Gereksiz kod ekleme (Junk Code Insertion - JK)

Bu teknikte, uygulamanın metotlarına uygulamanın işleyişini değiştirmeyecek şekilde kodlar eklenir. Bir metodun rastgele yerlerine üç komut satır eklenmiştir. İlk olarak metoda string, int ve double gibi ilkel tiplerde yerel değişken eklenmiştir. Sonra bu değişkeni konsola yazdırmak için gerekli olan *PrintStream* nesnesi eklenmiş ve son olarakta *PrintStream* nesnesinin *println* komutu çağrılmıştır. Şekil 5.6.'da eklenen 3 satır kod koyu renkle gösterilmektedir.

Şekil 5.6.: Gereksiz kod ekleme tekniği uygulanmış kod parçası

```
1  new-instance v0, Landroid/content/Intent;
2  const-class v1, Lcom/android/MonitorService;
3  invoke-direct {v0, p1, v1}, Landroid/content/Intent;-><init>(Landroid/content/Context;Ljava/lang/Class;)V
4  .local v0, timer:Landroid/content/Intent;
5  const-string v1, "com.android.MonitorService"
6  const-string v2, "xjoia7863y4hb"
7  sget-object v3, Ljava/lang/System;->out:Ljava/io/PrintStream;
8  invoke-virtual v3, v2, Ljava/io/PrintStream;->println(Ljava/lang/String;)V
9  invoke-virtual {v0, v1}, Landroid/content/Intent;->setAction(Ljava/lang/String;)Landroid/content/Intent;
10 const/4 v1, 0x2
11 invoke-virtual {v0, v1}, Landroid/content/Intent;->setFlags(I)Landroid/content/Intent;
12 invoke-virtual {p1, v0}, Landroid/content/Context;->startService(Landroid/content/Intent;)Landroid/content/ComponentName;
13 return-void
```

5.2.2.1.3 Veri şifreleme (Data Encryption - DE)

Şifreleme, zararlı yazılımların zararlı aktivitelerini saklamak için kullanılan bir yöntemdir. Bu teknikte, metotlarda yer alan bütün karakter dizileri şifrelenir. İlk olarak rastgele olarak oluşturulan anahtar ile karakter dizisi şifrelenir. Sonra kod kısmında bu şifrelenmiş karakter dizisi özgün karakter dizisi ile yer değiştirir. Bu kodun altında rastgele oluşturulan anahtar, tekrar özgün karakter dizisine dönüşüm sağlayabilmek için yerel değişken olarak eklenir. En son kendi oluşturduğumuz şifreleme ve deşifreleme metotlarını içeren *SimpleCrypto* sınıfının *decrypt* metodu çağrılır. Bu nedenle, *SimpleCrypto* sınıfı uygulamanın içine dahil edilir. Şekil 5.7.'de koyu olarak gösterilen kısımlar veri şifrelemek için yerleştirilmiş komutlardır. 6. satırdaki *"fev3v56uni8o"* anahtarı 5. satırdaki orijinali *"com.android.MonitorService"*

olan değişkenin değerini şifrelemek için kullanılmıştır. Şifreleme sonucu değişken *"7B10845DBDDE74190D55AA5CFEC3A2A7CD554ADC9FF120F909358E003D5FA198"* değerini almıştır. Daha sonra değişken *SimpleCrypto* sınıfının *decrypt* metodu kullanılarak tekrar *"com.android.MonitorService"* değerini alır ve işleyiş bu şekilde devam eder.

Şekil 5.7.: Veri şifreleme tekniği uygulanmış kod parçası

```
new-instance v0, Landroid/content/Intent;
const-class v1, Lcom/android/MonitorService;
invoke-direct {v0, p1, v1}, Landroid/content/Intent;-><init>(Landroid/content/Context;Ljava/lang/Class;)V
.local v0, timer:Landroid/content/Intent;
const-string v3, "7B10845DBDDE74190D55AA5CFEC3A2A7CD554ADC9FF120F909358E003D5FA198"
const-string v2, "fev3v56uni8o"
.local v2, seed:Ljava/lang/String;
invoke-static/range v2 ..  v3, LCrypto/SimpleCrypto;->decrypt(Ljava/lang/String;Ljava/lang/String;)Ljava/lang/String;
move-result-object v1
invoke-virtual {v0, v1}, Landroid/content/Intent;->setAction(Ljava/lang/String;)Landroid/content/Intent;
const/4 v1, 0x2
invoke-virtual {v0, v1}, Landroid/content/Intent;->setFlags(I)Landroid/content/Intent;
invoke-virtual {p1, v0}, Landroid/content/Context;->startService(Landroid/content/Intent;)Landroid/content/ComponentName;
return-void
```

5.2.2.1.4 İkili kod sırası değiştirme (Two-fold Code Reordering - CR2)

Kod sırası değiştirme; bir uygulamanın metotlarının işlevini değiştirmeyecek şekilde metotların komutlarının yerlerinin değiştirilmesidir. İkili kod değiştirme tekniğinde her bir metodun başlangıcı ve sonuna *goto* deyimi konulur. Uygulama o metoda girdiğinde, ilk *goto* deyiminde işleyiş metodun sonuna atlar. Metodun sonundaki ikinci *goto* deyimi de işleyişi yine kodun başına döndürür ve normal işleyiş buradan devam eder. Şekil 5.8.'de eklenen *goto* deyimleri görülmektedir.

Şekil 5.8.: İkili kod sırası değiştirme tekniği uygulanmış kod parçası

```
goto :g_2
:g_1
new-instance v0, Landroid/content/Intent;
const-class v1, Lcom/android/MonitorService;
invoke-direct {v0, p1, v1}, Landroid/content/Intent;-><init>(Landroid/content/Context;Ljava/lang/Class;)V
.local v0, timer:Landroid/content/Intent;
const-string v1, "com.android.MonitorService"
invoke-virtual {v0, v1}, Landroid/content/Intent;->setAction(Ljava/lang/String;)Landroid/content/Intent;
const/4 v1, 0x2
invoke-virtual {v0, v1}, Landroid/content/Intent;->setFlags(I)Landroid/content/Intent;
invoke-virtual {p1, v0}, Landroid/content/Context;->startService(Landroid/content/Intent;)Landroid/content/ComponentName;
return-void
:g_2
goto :g_1
```

5.2.2.1.5 Üçlü kod sırası değiştirme (Three-fold Code Reordering - CR3)

Üçlü kod sırası değiştirme tekniği, ikili kod sırası değiştirme tekniği ile aynı mantığa sahiptir, fakat burada farklı olarak fark üç *goto* deyimi kullanılmaktadır. Böylece metodun kodunun

sırasını değiştirmemize rağmen çalışma zamanında kod sırasının korunması sağlanmış olur. Şekil 5.9.'da *goto* deyimlerinin nasıl yerleştirildiği görülmektedir. Önce **goto :g_1** satırı ile işleyiş **g_1** etiketine atlar ve altındaki 3 satır kodu işletir. Sonra **goto :g_2** satırı ile işleyiş **g_1** etiketine atlar. En son olarak **goto :g_3** satırı ile işleyiş metodun sonuna atlar ve metot işleyişini bitirir.

Şekil 5.9.: Üçlü kod sırası değiştirme tekniği uygulanmış kod parçası

```
goto :g_1
:g_2
.local v0, timer:Landroid/content/Intent;
const-string v1, "com.android.MonitorService"
invoke-virtual {v0, v1}, Landroid/content/Intent;->setAction(Ljava/lang/String;)Landroid/content/Intent;
goto :g_3
:g_1
new-instance v0, Landroid/content/Intent;
const-class v1, Lcom/android/MonitorService;
invoke-direct {v0, p1, v1}, Landroid/content/Intent;-><init>(Landroid/content/Context;Ljava/lang/Class;)V
goto :g_2
:g_3
const/4 v1, 0x2
invoke-virtual {v0, v1}, Landroid/content/Intent;->setFlags(I)Landroid/content/Intent;
invoke-virtual {p1, v0}, Landroid/content/Context;->startService(Landroid/content/Intent;)Landroid/content/ComponentName;
return-void
```

5.3. Uygunluk Değeri

Uygunluk değeri, bireylerin problemin çözümüne ne kadar yaklaştıkları hakkında bilgi veren önemli bir parametredir. Bu nedenle, genetik programlama uygulamalarında iyi seçilmiş uygunluk fonksiyonu oldukça önemlidir.

Bu tezdeki deneylerde uygunluk fonksiyonu olarak mobil güvenlik sistemlerinden en başarıları arasından seçilen 8 antivirüs sisteminin sonuçları kullanılmıştır. Antivirüs sistemleri koruma puanlarına göre AV-TEST [6] ve AV-COMPARATİVES [7] sitelerinde bulunan virüs sistemleri arasından seçilmiştir. Antivirüs sistemlerini seçmemizdeki ikinci kriter ise antivirüs sistemlerinin emulator üzerinde çalışırken bize geri sonuç döndürebilmeleri olmuştur. Çünkü bir antivirüs sisteminin ürettiği kayıt dosyalarına bakılarak bir uygulamanın verdiği tepki alınmaktadır. Fakat bazı antivirüs sistemleri kayıt dosyalarına, taradıkları dosyaların sonuçlarını kaydetmemektedir ve bu da bizim sonuç alamamamıza ve buna bağlı olarak o antivirüsü kullanamamamıza neden olmaktadır. Çizelge 5.2.'de deneylerde kullanılan antivirüsler ve AV-TEST [6] sitesinde yer alan koruma puanları gösterilmektedir. Koruma puanları 6 üzerinden verilmiştir. Evrimleştirilen zararlı yazılımları çalıştırmak ve antivirüs sistemleri karşısındaki sonuçlarını almak için her bir antivirüs, farklı resmi Android emülatörleri [63] üzerine yüklenmiş ve çalıştırılmıştır. Her bir antivirüs,

evrimleştirilmiş zararlı yazılımı virüs olarak görüyorsa 1, virüs olarak görmüyorsa veya hiç bir şekilde tanımlayamıyorsa 0 sonucunu uygunluk değeri olarak dönmektedir. Her bir emülatörden dönen sonuçlar toplanır ve toplam antivirüs sayısı olan 8 değerine bölünerek uygunluk fonksiyonu hesaplanır. Uygunluk fonksiyonunun hesaplanması Denklem 1'de gösterilmektedir. Bu denkleme göre uygunluk fonksiyonu 0 ile 1 arasında değerler almaktadır. Genetik programlama ile uygunluk fonksiyonunun 0 değerine yaklaştırılması amaçlanmaktadır.

$$\text{Uygunluk değeri} = \frac{\text{bir bireyi tespit eden antivirüs sistemi sayısı}}{\text{toplam antivirüs sistemi sayısı}} \quad (1)$$

Emülatörlerdeki antivirüs sistemlerinin belirli bir zamanda sonuç üretebilmesi için emülatörler 1 dakika süre ile çalıştırılmaktadır. Bu süre, yaptığımız deneyler sonucu belirlenmiştir. Genelde antivirüs sistemleri 1 dakikadan kısa bir süre içerisinde cevap dönmektedir. Eğer bu süre içerisinde yanıt alamazsak genetik programlamaya 0 değeri döndürülmektedir. Genelde büyük boyutlu yazılımların analizinin uzun sürdüğü gözlemlenmiştir.

Çizelge 5.2.: Deneylerde kullanılan antivirüs sistemleri ve versiyonları

Üretici	Ürün	Versiyon	Koruma Puanı
Eset	Eset Mobile Security	3.0.882.0-16	6
GData	Gdata Internet Security	25.0.0	6
Ikarus	Ikarus mobile.security	1.7.20	4.5
Kaspersky	Kaspersky Internet Security	11.1.3.10	6
Avast	Avast Mobile Security	3.0.7550	5.5
Trend Micro	Trend Micro Mobile Security	5.0.0.1225	6
BitDefender	BitDefender Mobile Security	2.18.119	6
Norton	Norton Mobile Security	3.8.6.1533	5.5

Bölüm 6.

DENEY SONUÇLARI

6.1. Veri Seti

Önerilen modelde *Android Malware Genome* projesinde üretilen zararlı yazılım veri seti kullanılmıştır [39]. Literatürdeki ilk zararlı yazılım veri setidir. Birçok çalışma, diğer yaklaşımlarla kendi sonuçlarını karşılaştırmak için bu veri setini kullanmıştır. Bu çalışmada [39] araştırmacılar, Ağustos 2010 ve Ekim 2011 tarihleri arasında bir yıldan fazla bir sürede 49 zararlı yazılım ailesinden 1,260 Android zararlı yazılımı toplamışlardır. 5 zararlı yazılım ailesinden zararlı yazılımlar sadece resmi Android markette, 35 zararlı yazılım ailesinden zararlı yazılımlar sadece alternatif Android marketlerde ve geri kalan 9 zararlı yazılım ailesinden zararlı yazılımlar ise hem resmi hem de alternatif Android marketlerde bulunmuştur. Bu yazılımların 1083 tanesinin varolan zararlı yazılımların yeniden paketlenmiş hali olduğu belirtilmiştir.

Çalışmamızda, bu veri setinden rastgele zararlı yazılımları seçtik ve bunların kontrol akış çizgelerini oluşturarak genetik programlamaya giriş olarak verdik. Zararlı yazılımları seçmemizdeki tek kriter zararlı yazılımların Çizelge 5.2.'de verilen antivirüs sistemleri tarafından tespit edilebilmesi olmuştur. Zararlı yazılımların kaç antivirüs tarafından tespit edildiği dikkate alınmamıştır. Bu çalışmada, bu veri setinde rastgele olarak seçilen zararlı yazılımlar kullanılmış ve bu yazılımların kontrol akış çizgeleri genetik programlamaya girdi olarak verilmiştir.

Çizelge 6.1.: Kullanılan tekniklerin kısaltmaları

Teknik	**Kısaltma**
Orjinal	OR
Gereksiz Kod Ekleme	JK
Veri Şifreleme	DE
İkili Kod Değiştirme	CR2
Üçlü Kod Değiştirme	CR3
Yerel Değişkenleri Yeniden Adlandırma	RI
Mutasyon	MU
Çaprazlama + Mutasyon	XO + MU

6.2. Sonuçlar

Genetik programlama farklı zararlı yazılımlarla birçok kez çalıştırılmıştır. Bazı çalışmalarda genetik programlama *smali* dosyalarını değiştirirken karşılaşılan sorunlar nedeniyle derlenemeyen ve çalışamayan çıktılar üretmiştir. Bu çıktılar, sonuçlarda dikkate alınmamıştır.

6.2.1 Evrimleştirilmiş Zararlı Yazılımların Değerlendirilmesi

Bu bölümde, genetik programlama sonucu üretilen yeni zararlı yazılımları değerlendirmek için yapılan deney ve deney sonuçları anlatılacaktır. Karıştırma teknikleri zararlı yazılımlara tekli ve ikili şekilde olmak üzere 2 farklı şekilde uygulanmıştır. Karıştırma teknikleri uygulanan zararlı yazılımlar, bu çalışma kapsamında evrimleştirilen zararlı yazılımlar ile karşılaştırılmıştır. Sonuçlar Çizelge 6.2.'de ve Çizelge 6.3.'de görülmektedir. Çizelge 6.2.'de, karıştırma teknikleri her bir zararlı yazılıma tek tek uygulanmıştır. Çizelge 6.3.'de ise karıştırma teknikleri ikili gruplar şeklinde zararlı yazılımlara uygulanmıştır. Ayrıca önerilen yöntem sadece mutasyon işleci kullanılarak ve hem mutasyon hem çaprazlama işleci kullanılarak 2 farklı şekilde uygulanmış ve sonuçlarda değerlendirilmiştir. *OR*; orijinal zararlı yazılımı, *JK*; gereksiz kod ekleme tekniği uygulanan zararlı yazılımı, *DE*; veri şifreleme tekniği uygulanan zararlı yazılımı, *CR2*; ikili kod değiştirme tekniği uygulanan zararlı yazılımı, *CR3*; üçlü kod değiştirme tekniği uygulanan zararlı yazılımı, *RI*; yerel değişkenleri yeniden adlandırma tekniği uygulanan zararlı yazılımı, *MU*; sadece mutasyon işleci kullanılarak evrimleştirilmiş zararlı yazılımı ve *XO + MU*; çaprazlama ve mutasyon işleçleri beraber kullanılarak evrimleştirilmiş zararlı yazılımı temsil etmektedir. Tüm bu yöntemlerin kısaltmaları Çizelge 6.1.'de verilmiştir.

6.2.1.1 Tekli Karıştırma Teknikleri ile Karşılaştırma

Bu bölümde karıştırma tekniklerinin zararlı yazılımlara tek tek uygulandığında elde edilen sonuçları ile önerilen yöntemin sonuçları karşılaştırılacaktır. Çizelge 6.2.'de, karıştırma tekniklerinin çoğunun orijinal zararlı yazılımın tespit oranını düşürdüğü görülmektedir. Fakat zararlı yazılıma sadece mutasyon işleci kullanan genetik programlama uygulanırsa sonuçların karıştırma tekniklerine göre daha da düştüğü *MU* kolonunda görülmektedir. Bazı sonuçlarda sadece mutasyon işleci kullanan genetik programlamanın karıştırma tekniklerinden üstün olmadığı görülmektedir. Fakat bu sonuçlardan hiçbirinde karıştırma tekniklerinden daha kötü bir sonuç elde edilmemiştir, eşit sayıda tespit oranına ulaşılmıştır. Genetik programlamada çaprazlama işleci kullanıldığında ise sonuçların karıştırma tekniklerine göre daha iyi veya eşit, sadece mutasyon işleci kullanan sonuçlara göre ise eşit veya daha kötü olduğu görülmektedir. Karıştırma teknikleri ile aynı sayıda tespit oranına ulaşan birçok sonuç elde edilmiştir, fakat burada sadece en iyi performans gösteren sonuçları göstermek istediğimiz için bu sonuçlar Çizelge 6.2.'de ve Çizelge 6.3.'de gösterilmemiştir.

Çizelge 6.2.: Karıştırma teknikleri tek tek uygulandığında antivirüs sistemlerinin başarısı

	OR	*JK*	*DE*	*CR2*	*CR3*	*RI*	*MU*	*XO + MU*
NickySpy_1ce27fa92a313da39f1e31e97d3ac05a8d6ffe78	8	7	8	7	7	8	5	7
NickySpy_63e642f0d859e096342321c9e03baca7cd1210fa	8	8	8	7	7	8	6	6
Asroot_0c059ad62b9dbccf8943fe4697f2a6b0cb917548	7	7	6	7	7	7	6	6
GPSSMSSpy_0eb4b7737df1b8b52213599e405d71c9be8a68ac	6	5	5	6	6	6	5	5
GPSSMSSpy_4d43d7771e480de34dbf748867152406b91a0de8	6	5	5	6	6	6	5	5
HippoSMS_bd7e85f5a0c39a9aeecc05dbc99a9e5c52150ba6	8	7	5	8	7	7	4	7
FakeNetflix_0936b366cbc39a9a60e254a05671088c84bd847e	6	6	4	6	6	6	3	3
DroidKungFu2_8bb6106b7c1160e8812788bbd16b563f5a00080a	7	7	7	7	7	7	6	6
GPSSMSSpy_af727f5e23e69bfe2321f5d556c63f741dae8283	6	5	5	6	6	6	4	5
GPSSMSSpy_73c1657ddf52cc82b57c2db80554c59927e7970a	6	5	5	6	6	6	4	5
GPSSMSSpy_94b56252ff610126135c568b1cc7b92405b9e608	6	5	5	6	6	6	4	5
GPSSMSSpy_5900250af412b71477647068 47cf1dbc54cd6e0e	6	5	5	6	6	6	4	5
DroidKungFu1_02d2e109d16d160f77a645f44314fedcdbcd6e18	8	8	8	8	8	8	7	7
RogueLemon_08a21de6b70f584ceddbe803ae12d79a33d33b50	6	5	4	6	6	6	4	5

Gereksiz kod değiştirme tekniği 9 zararlı yazılım üzerine etki etmiştir. Bu zararlı yazılımların tespit oranının hepsinde 1 düşme sağlamıştır. Diğer 5 zararlı yazılım üzerinde ise hiç bir etki göstermemiş ve tespit sayıları orijinal zararlı yazılım ile aynı olmuştur.

En iyi iyileştirmeyi sağlayan teknik veri şifreleme tekniği olmuştur. Bu teknik, 14 zararlı yazılım içinden sadece 4 tanesinde hiç bir değişikliğe sebep olmamıştır. Özellikle HippoSms ailesinden olan bir zararlı yazılım karşısında tespit oranını 8'den 5'e düşerek en iyi gelişimi sağlamıştır. RogueLemon ve FakeNetFlix ailesinden zararlı yazılımlarda tespit oranı 2 düşerken, diğer 7 zararlı yazılımda tespit oranı 1 düşme sağlamıştır.

İkili kod değiştirme tekniği sadece NickSpy ailesinden 2 zararlı yazılım üzerinde daha iyi sonuç vermiştir. Bu zararlı yazılımların tespit oranını 1 azaltmıştır. Diğer zararlı yazılımlar üzerinde hiç bir etkinlik gösterememiştir.

Üçlü kod değiştirme tekniği, ikili kod değiştirme tekniğine göre kodu daha fazla karıştırmasından dolayı tespit oranında ikili kod değiştirmeden daha iyi sonuç vermiştir. Fakat bu iyileştirme beklenildiği gibi olmamış ve ikili kod değiştirme tekniğine göre fazladan sadece 1 zararlı yazılım üzerinde etki göstermiştir. Yine aynı ikili kod değiştirme tekniğindeki gibi NickSpy ailesindeki 2 zararlı yazılımın ve HippoSms ailesinden zararlı yazılımın tespit oranını 1 düşürmüştür.

Çizelge 6.2.'de görüldüğü gibi yerel değişkenleri yeniden adlandırma tekniği antivirüs sistemleri karşısında pek bir etki yaratmamıştır. Orijinal zararlı yazılımdan sadece 1 eksik sayıda tespit etme oranına ulaşmıştır. Bir tek HippoSms ailesindeki bir zararlı yazılım karşısında daha iyi sonuç göstermiştir.

6.2.1.2 İkili Karıştırma Teknikleri ile Karşılaştırma

Bu bölümde karıştırma teknikleri zararlı yazılımlara ikili gruplar halinde uygulandığında elde edilen zararlı yazılımlar ile önerilen yöntem ile evrimleştirilmiş zararlı yazılımlar karşılaştırılacaktır.

Literatürde karıştırma tekniklerini zararlı yazılımlara tek tek uygulayan çalışmalar bulunmaktadır [8, 11], fakat karıştırma tekniklerini bir arada kullanan bir çalışma bulunmamaktadır. Evrimsel hesaplama, birden çok karıştırma tekniğini bir düğüm üzerinde uygulayabildiğinden dolayı, bu çalışmada karıştırma teknikleri zararlı yazılımlara ikili gruplar halinde de uygulanmış ve sonuçlar alınmıştır. İkili gruplar halinde uygulanan karıştırma tekniklerine ilişkin sonuçlar Çizelge 6.3.'de görülmektedir. Sadece üçlü kod değiştirme ve ikili kod değiştirme (CR3 + CR2) teknikleri beraber uygulandığında derlenebilir sonuçlar üretilemediği için Çizelge 6.3.'de gösterilmemiştir. Ayrıca karıştırma teknikleri üçlü ve dörtlü gruplar halinde de zararlı yazılımlara uygulanmıştır, fakat karıştırma teknikleri birbirini etkilediğinden dolayı derlenme oranı çok düşük olmuş ve sonuçlarda gösterilmemiştir. Evrimsel hesaplama, doğası gereği derlenemeyen ve çalıştırılamayan zararlı yazılımları eleyerek bu tür sorunların oluşmasını engellemiştir. Çizelge 6.3.'de görülen "-" boş kısımlar, derlenen fakat emülatöre yüklenmeye çalışıldığında çeşitli hatalar veren zararlı yazılımların

olduğunu göstermektedir. Ayrıca karıştırma tekniği uygulanarak elde edilen sonuçların çok az bir kısmı otomatik olarak üretildiğinde hata verdiğinden dolayı, el ile küçük değişiklikler yapılarak düzeltilmiştir. Fakat önerilen yönteme el ile hiç bir müdahale yapılmamıştır ve önerilen yöntem tamamen otomatik bir şekilde zararlı yazılımların evrimini sağlamaktadır.

Çizelge 6.3.: Karıştırma teknikleri ikili gruplar halinde uygulandığında antivirüs sistemlerinin başarısı

	JK-DE	*CR2-DE*	*DE-RI*	*CR3-DE*	*JK-CR3*	*JK-RI*	*CR2-RI*	*JK-CR2*	*CR3-RI*	*MU*	*XO + MU*
NickySpy_1ce27fa92a313da39f1e31e97d3ac05a8d6ffe78	7	7	8	7	7	7	7	7	7	5	7
NickySpy_63e642f0d859e096342321c9e03baca7cd1210fa	7	7	8	7	8	7	7	8	7	6	6
Asroot_0c059ad62b9dbccf8943fe4697f2a6b0cb917548	6	6	6	-	-	7	7	7	-	6	6
GPSSMSSpy_0eb4b7737df1b8b52213599e405d71c9be8a68ac	5	5	5	-	-	6	6	6	-	5	5
GPSSMSSpy_4d43d7771e480de34dbf748867152406b91a0de8	5	5	5	-	-	5	6	5	-	5	5
HippoSMS_bd7e85f5a0c39a9aeecc05dbc99a9e5c52150ba6	5	5	5	5	6	7	7	7	6	4	7
FakeNetflix_0936b366cbc39a9a60e254a05671088c84bd847e	4	4	4	4	6	6	6	6	6	3	3
DroidKungFu2_8bb6106b7c1160e8812788bbd16b563f5a00080a	7	7	7	-	-	7	7	7	-	6	6
GPSSMSSpy_af727f5e23e69bfe2321f5d556c63f741dae8283	5	5	5	-	-	5	6	5	-	4	5
GPSSMSSpy_73c1657ddf52cc82b57c2db80554c59927e7970a	5	5	5	-	-	5	6	5	-	4	5
GPSSMSSpy_94b56252ff610126135c568b1cc7b92405b9e608	5	5	5	-	5	5	6	6	-	4	5
GPSSMSSpy_5900250af412b7147764706847cf1dbc54cd6e0e	5	5	5	-	-	5	6	6	-	4	5
DroidKungFu1_02d2e109d16d160f77a645f44314fedcdbcd6e18	8	8	8	-	-	8	8	8	-	7	7
RogueLemon_08a21de6b70f584ceddbe803ae12d79a33d33b50	4	4	4	-	-	6	6	6	-	4	5

Zararlı yazılımlara ikili gruplar halinde uygulanan karıştırma tekniklerinden alınan sonuçların, karıştırma tekniklerinin tek tek uygulandığındaki sonuçlarından çok daha iyi olmadığı Çizelge 6.2. ve Çizelge 6.3.'ye bakılarak söylenebilir.

Sadece mutasyon işleci kullanan genetik programlama yönteminin sonuçları Çizelge 6.2. ve Çizelge 6.3.'in *MU* kolonunda görülmektedir. Görüldüğü üzere 5 karıştırma tekniğine ve bunların ikili gruplarına göre genelde daha iyi sonuçlar üretmiştir. Sadece mutasyon işleci kullanan genetik programlama yöntemi, bazı zararlı yazılımlarda ise karıştırma tekniklerine göre eşit sayıda tespit oranına sahip olmuştur.

XO + MU kolonunda çaprazlama ve mutasyon işleçlerini beraber kullanan genetik programlama yönteminin sonuçları görülmektedir. Çizelge 6.2. ve Çizelge 6.3.'den de anlaşılacağı gibi sadece mutasyon işleci kullanan genetik programlamanın çıktılarına göre bazı zararlı yazılımlarda daha kötü bir performans göstermektedir. Bunun nedeni ise çaprazlama işlecinin, *smali* kodlarında kaynaklanan problemlerden dolayı istenilen şekilde uygulamalara etki edememesidir. Çaprazlama oranı 0.1 gibi düşük bir oran seçilmesine rağmen çaprazlama sonucu oluşan bireylerin büyük bir kısmı çalıştırılabilir olmamaktadır. Çaprazlama işleci sonucu çalıştırılabilir olan uygulamalar ise statik analiz araçlarından kaçabilecek büyüklükte bir kod değişimine sebep olmamaktadır. Çaprazlama ve mutasyon işleçlerini beraber kullanan genetik programlama yöntemi bazı zararlı yazılımlarda ise mutasyon kullanan genetik programlama yöntemiyle aynı sonuçları vermiştir, fakat zararlı yazılımların yarısında

mutasyon kullanan genetik programlama yöntemine göre sonuçları daha kötü olmaktadır. Çaprazlama ve mutasyon işleci kullanan genetik programlama yöntemini karıştırma teknikleriyle karşılaştırdığımızda ise, HippoSms zararlı yazılımı hariç diğer tüm zararlı yazılımlar karşısında daha iyi ya da eşit bir performans gösterdiği görülmektedir. Ayrıca çaprazlama işleci, özellikle Kaspersky antivirüs sisteminden kaçınabilen zararlı yazılımlar üretmiştir.

Çizelge 6.2. ve Çizelge 6.3.'deki 14 zararlı yazılım, sadece mutasyon kullanan genetik programlama yönteminin sonuçlarının içindeki en iyi sonuçlardan seçilmiştir. Çaprazlama ve mutasyon işleçlerini beraber kullanan genetik programlama yöntemi, bu 14 zararlı yazılım üzerinde sonradan uygulanmış ve sonuçlar alınmıştır. Ayrıca, çaprazlama ve mutasyon işleçlerini beraber kullanan genetik programlama yöntemi, dinamik analiz aracının uzun süreli işlem yapmasından dolayı bu 14 zararlı yazılım için sadece 1 defa çalıştırılmıştır. Bilindiği gibi, genetik programlama genelde birden çok kez çalıştırılır ve bu çalıştırmalar içerisindeki en iyi sonuçlar alınır. Bu nedenle, Çizelge 6.2. ve Çizelge 6.3.'deki 14 zararlı yazılım, çaprazlama ve mutasyon işleçlerini beraber kullanan genetik programlama yöntemi ile birden çok kez çalıştırıldığında daha iyi sonuçlar alınabileceği öngörülebilir.

6.2.2 Statik Analiz Araçlarının Değerlendirilmesi

Bu tezde kullanılan çoğu antivirüs sistemi klasik imza tabanlı tespit yöntemini kullanmaktadır. Bunun en önemli nedeni, imza tabanlı tekniklerin, hem oldukça hızlı bir şekilde tespit etme işlemi yapması hem de diğer tekniklere göre daha az güç harcamasıdır. Antivirüs sistemleri bu tekniğin yanında sezgisel ve bulut tabanlı yöntemler de kullanmaktadır. Sezgisel yöntemler hem statik hem de dinamik analiz ile kullanılabilir. [64]. Statik analize dayalı sezgisel yöntemlerde yazılımlar kaynak kodlarına dönüştürülür ve bu kaynak kodların içinde tehlikeli olabilecek kodlar bulunmaya çalışılır. Dinamik analize dayalı sezgisel yöntemlerde ise yazılımlar sanal bir kum havuzu içinde çalıştırılır ve bu yazılımın gerçek bir makinede çalıştırıldığında nasıl bir etki yarattığı incelenir. Genelde antivirüs sistemleri tarama birimi ve imza veri tabanlarını istemci cihazlarda bulundurur. Her bir istemci cihaz, karşılaştığı yeni zararlı yazılımların imzasını kendi veri tabanına ekler. Bulut tabanlı yöntemde ise antivirüs yüklü istemci cihazlar, kendi imza veri tabanlarını diğer cihazlar ile bulut üzerinden paylaşarak diğer cihazların daha önceden karşılaşmadığı zararlı yazılımlar karşısında direnç kazanmasını sağlar. İstemci tarafında oldukça az kaynak kullanımına sebep

olur ve tespit oranını arttırır, fakat bu işlem için ağ bağlantısı gerektiğinden çevrimdışı durumunda işlevsiz kalır. Ayrıca istemci cihazdaki her tarama yapılan dosyanın bilgisini diğer cihazlara açtığından güvenlik problemlerine de neden olabilir [65].

Bu çalışmada önerilen yöntem, tek bir çalışmada birden çok zararlı yazılım üretmektedir ve 5 karıştırma tekniğine göre antivirüs sistemlerinden daha çok kaçabilen yeni zararlı yazılımlar evrimleştirmektedir. Bu bize Çizelge 6.4.'teki gibi antivirüs sistemlerinin bilinmeyen zararlı yazılımlar karşısındaki genel başarısını ölçmemizi sağlar.

Çizelge 6.4.: Antivirüs sistemlerinin genel başarısı

	OR	***JK***	***DE***	***CR2***	***CR3***	***RI***	***MU***	***MU + XO***
Eset	8	8	6	8	8	8	4	6
GData	6	6	6	6	6	6	5	5
Ikarus	13	8	6	12	11	13	7	12
Kaspersky	13	12	12	13	13	13	10	5
Avast	14	14	14	14	14	14	14	13
TrendMicro	13	12	11	12	12	12	9	12
BitDefender	14	14	14	14	14	14	13	13
Norton	13	11	11	13	13	13	5	11
Toplam	94	85	80	92	91	93	67	77

Çizelge 6.4.'te görüldüğü gibi Avast Mobile Security yeni ataklara karşı en dirençli antivirüs sistemi olmuştur. Hem 5 karıştırma tekniğinden hem de önerdiğimiz yöntem sonucu elde edilen bütün zararlı yazılımların hepsini başarıyla tespit etmiştir. Çaprazlama ve mutasyon kullanılan genetik programlama sonucunda oluşan tek bir zararlı uygulamayı ise tespit edememiştir.

BitDefender antivirüs sistemi de oldukça başarılı bir tespit oranı sergilemiştir. Yalnız hem sadece mutasyon işlecini kullanan genetik programlama yönteminde hem de çaprazlama ve mutasyon işlecini kullanan genetik programlama yönteminde birer zararlı yazılım karşısında etkisiz kalmıştır.

Kaspersky antivirüs sistemi karıştırma teknikleri karşısında başarılı sonuçlar vermiştir. Sadece gereksiz kod ekleme ve veri şifreleme tekniklerinde 1 zararlı yazılım karşısında etkisiz kalmıştır. Sadece mutasyon kullanan genetik programlama yöntemiyle evrimleştirilen

zararlı yazılımlardan üç zararlı yazılım karşısında başarısız olmuştur. Çaprazlama kullanıldığında ise zararlı yazılımları tespit etmekte zorlandığı görülmektedir ve sadece 5 zararlı yazılımı tespit etmektedir.

TrendMicro antivirüs sistemi karıştırma teknikleri karşısında iyi sonuçlar veren bir diğer antivirüs sistemidir. Gereksiz kod eklemede 1, veri şifrelemede ise 2 zararlı yazılımı tespit etmede başarısız olmuştur. Sadece mutasyon kullanan genetik programlama yönteminde 4 zararlı yazılımı tespit edemediği görülmektedir. Çaprazlama kullanıldığında ise tespit oranı artmaktadır.

Tespit etme oranı oldukça düşük olmasına rağmen (6/14), GData bütün karıştırma tekniklerinden elde edilen zararlı yazılımlar karşısında başarılı olmuştur. Hem sadece mutasyon işlecini kullanan genetik programlama yönteminde, hem de çaprazlama ve mutasyon işlecini kullanan genetik programlama yönteminde ise birer zararlı yazılım karşısında başarısız olmuştur.

Norton antivirüs sistemi sadece mutasyon işlecini kullanan genetik programlama yöntem karşısında en kötü sonucu gösteren sistem olmuştur. 5 karıştırma tekniğinde oldukça başarılı sonuçlar vermesine rağmen sadece mutasyon işlecini kullanan genetik programlama yöntem karşısında tespit oranı 13'ten 5'e düşmüştür. Çaprazlama kullanıldığında ise tespit oranını diğer yönteme göre oldukça arttırmaktadır. Yine de çaprazlama kullanıldığında tekli ve ikili karıştırma tekniklerine göre daha iyi sonuçlar vermiştir. Veri şifreleme ve gereksiz kod ekleme tekniklerinde ise iki zararlı yazılımı tespit edememiştir.

Önerilen yöntem karşısında etkisiz kalan bir diğer antivirüs sistemi ise Eset antivirüs sistemidir. Veri şifreleme tekniği ile karıştırılan iki zararlı yazılımı tespit edememiştir. Diğer tekniklerle karıştırılan tüm zararlı yazılımları başarıyla tespit etmiştir. Sadece mutasyon işlecini kullanan genetik programlama yöntemde dört, hem çaprazlama hem mutasyon işlecini kullanan genetik programlama yöntemde ise altı zararlı yazılım karşısında başarılı olabilmiştir.

Ikarus antivirüs sistemi de iyi sonuçlar vermesine rağmen, veri şifreleme tekniği sadece mutasyon işlecini kullanan genetik programlama yönteminden bir fazla zararlı yazılımdan kaçmıştır. Çaprazlama kullanılan yöntemde ise daha çok sayıda zararlı yazılımı tespit etmiştir. Ayrıca gereksiz kod ekleme yönteminde ise diğer tekniklere göre daha başarısız olmuştur.

Çizelge 6.4.'te antivirüs sistemlerinin yerel değişkenleri yeniden adlandırma, ikili kod değiştirme ve üçlü kod değiştirme tekniklerine karşı oldukça dirençli olduğu görülmektedir. Ayrıca gereksiz kod ekleme ve veri şifreleme tekniklerine karşı ise zayıflığı olduğu görülmektedir. Antivirüs sistemleri, mutasyon ve çaprazlama işleçlerinin beraber kullanıldığı genetik programlama yöntemine karşı veri şifrelemeye karşı gösterdikleri sonuca yakın bir sonuç göstermişlerdir. Yine de mutasyon ve çaprazlama işleçlerinin beraber kullanıldığı genetik programlama yöntemi, veri şifrelemeye göre daha iyi tespit oranına sahip olmuştur. Sadece mutasyon işleci kullanan genetik programlama yöntemi ise antivirüs sistemlerini en çok zorlayan yöntem olmuştur. Antivirüs sistemleri, sadece mutasyon işleci kullanan yöntem ile üretilen zararlı yazılımların yaklaşık %33'ünü tespit edememektedir.

Önerilen yöntemde çaprazlama işleci kullanıldığında sadece mutasyon kullanılan yönteme göre sonuçların beklenildiği gibi iyi olmadığı görülmektedir. Aslında çaprazlama işleci kullanıldığında sonuçların sadece mutasyon kullanan yönteme göre düşmesi beklenmektedir. Fakat *smali* kodlarından kaynaklanan problemlerden dolayı çaprazlama işleci etkisini zararlı yazılımlar üzerinde hissettirememiştir. Çaprazlama işleci zararlı yazılımlara etki ettiğinde ise istenilen gelişmeyi sağlayacak kod değişimleri sağlanamamıştır. Yine de çaprazlama işleci kullanan genetik programlama yöntemi karıştırma tekniklerine göre daha kötü bir sonuç vermemiştir. Zararlı yazılımların yarısında karıştırma tekniklerine göre daha iyi sonuç verdiği Çizelge 6.4.'te görülmektedir.

6.2.3 Genetik Programlamanın Avantajları

Genetik programlama kullanarak yeni, bilinmeyen virüsler üretmenin üç önemli avantajı bulunmaktadır. Bunlardan biri, arama uzayını küçülterek virüs üzerinde en etkili olan karıştırma tekniklerinin hızlıca bulunmasına yardımcı olmasıdır. Karıştırma tekniği sayısı artırıldığında evrimsel hesaplamanın da önemi daha iyi görülecektir. Bu tezde, 5 tane karıştırma tekniği kullanılmasına rağmen literatürde yer alan başka karıştırma teknikleri de eklenerek daha iyi sonuçlar üretmek mümkün olabilir. Bu tekniklerden bazıları şunlardır:

- Fonksiyon çağrısı yönlendirme
- Fonksiyonları parçalara bölme
- Fonksiyonları birleştirme

- Fonskiyon isimlerini değiştirme
- Eş komutlarla değiştirme
- Sonucunu bozmayacak şekilde değişkenleri yeniden düzenleme
- Kontrol değişkenlerini değiştirme

İkinci avantaj, ilerde yapılması planlanan virüs-antivirüs eş evrimi için bir alt yapı oluşturmasıdır. Statik analiz tespit sistemleri, karşılaştığı zararlı yazılımlardan eşsiz bir imza oluşturur ve bu imzayı kendi veri tabanına ekler. Bu çalışmada hızlı ve otomatik şekilde zararlı yazılım evrimleştirmek, ileride çıkabilecek muhtemel zararlı yazılımların önceden üretilmesini sağlar. Bu evrimleştirilmiş zararlı yazılımların imzalarının çıkarılıp statik analiz tespit sistemlerine öğretmek statik analiz tespit sistemlerinin, sıfırıncı gün saldırılarına karşı dirençli hale gelmesini sağlar. Ayrıca, bir zararlı yazılımın değişik çeşitlerini tespit edebilecek genel imzaların üretilmesine olanak sağlar.

Genetik programlama kullanarak yeni, bilinmeyen virüsler üretmenin son avantajı ise kod şişmesidir (bloat). Kod şişmesi, evrimsel hesaplamada kodun kontrol edilemez bir şekilde büyümesi anlamına gelmektedir. Genelde genetik programlama uygulamalarında kod şişmesi istenmez. Bu nedenle, ağaçların maksimum derinliğine bir sınır konulur. Fakat bizim uygulamamızda kod şişmesi olması daha karmaşık programlar üretilmesine neden olur. Böylece statik analiz tespit sistemlerinden daha kolay kaçabilecek zararlı yazılımlar evrimleştirebilmemizi sağlar.

Özetlemek gerekirse, bu çalışmada kullanılan 8 antivirüs sisteminin de literatürde sıklıkla kullanılan karıştırma tekniklerine karşı zayıf kaldığı görülmüştür. Genetik programlama kullanarak bu zafiyeti daha da artırmanın mümkün olduğu sonuçlarla gösterilmiştir. Özellikle bu sonuçların çaprazlama işleci olmadan elde edilmesinden dolayı çaprazlama işleci kullanıldığında daha da iyi sonuçlar alınabileceği varsayılabilir. Ayrıca bu yöntem, ilerde yapılması planlanan virüs-antivirüs eş evriminin bir parçası olacağı düşünülerek geliştirilmiştir.

Bölüm 7.

SONUÇ

Yapılan bu çalışmada, mobil statik zararlı yazılım tespit etme araçlarının performansını ölçmek için literatürde sıklıkla kullanılan karıştırma teknikleri ve evrimsel hesaplama algoritmalarından olan genetik programlama kullanılarak varolan zararlı yazılımlardan yeni ve bilinmeyen zararlı yazılımlar evrimleştirmek amaçlanmıştır. Önerilen yöntem Android platformu üzerinde çalışmaktadır. Android uygulamaları üzerinde tersine mühendislik metotları uygulanarak elde edilen kaynak kodlar üzerinde karıştırma teknikleri uygulanarak işlem yapılmıştır. Mobil antivirüs çözümleri içinden [6] en başarılı 8 antivirüs sistemi seçilmiş ve evrimleştirilen zararlı yazılımlar karşısındaki performansları ölçülmüştür. Ayrıca karşılaştırma yapmak için kullanılan 5 karıştırma teknikleri tek tek ve ikili şekilde olmak üzere evrimleştirilecek zararlı yazılımlara uygulanmış ve hepsi ayrı ayrı 8 antivirüs sistemine verilerek sonuçlar toplanmıştır. 5 karıştırma tekniği ve önerdiğimiz yöntemin karşılaştırılması yapılmıştır. Önerdiğimiz yöntem sadece mutasyon işleci kullanarak ve çaprazlama ve mutasyon işleci beraber kullanılarak olmak üzere 2 şekilde test edilmiştir. Mutasyon işlecinde değişiklikleri yapabilmek için 5 karıştırma tekniği de kullanılmıştır.

Sonuçlar, kullanılan bütün mobil antivirüs sistemlerinin karıştırma teknikleri karşısında zayıf kaldığını göstermiş ve genetik programlama kullanarak bu tespit etme sonuçlarının daha da düşürülebildiğini göstermiştir. Fakat genetik programlamada çaprazlama işleci kullanıldığında, *smali* kodlarından kaynaklanan problemlerden dolayı çaprazlama işlecinin bireyler üzerinde yeterli bir şekilde etki edemediği ve sadece mutasyon kullanan genetik programlamaya göre daha kötü sonuçlar verdiği gözlenmiştir. Yine de çaprazlama işleci kullanan genetik programlama yöntemi karıştırma tekniklerine göre daha iyi ya da eşit sonuçlar

vermiştir. Sonuçlarda önerilen yöntemin karıştırma tekniklerine göre sadece daha iyi olan sonuçlarıyla beraber bazı eşit sayıda tespite sahip sonuçlarda gösterilmiştir. Aslında eşit sayıda tespit eden sonuçlar daha fazla olmasına rağmen sonuçlarda gösterilmemiştir.

Antivirüs sistemlerinin, karıştırma teknikleri ve önerilen yöntem karşısındaki sonuçları tartışılmıştır. Antivirüs sistemleri, yerel değişkenleri yeniden adlandırma, ikili kod değiştirme ve üçlü kod değiştirme tekniklerine karşı iyi sonuçlar vermelerine rağmen gereksiz kod ekleme ve veri şifreleme tekniklerine karşı zayıf kalmaktadırlar. Önerilen yöntem karşısında ise bu zayıflık daha da artmaktadır. Sadece mutasyon kullanan genetik programlama yönteminde zararlı yazılımların %33'ü antivirüs sistemlerinden kaçabilmektedir.

Önerilen yöntem tamamen otomatik bir şekilde genetik programlama ve karıştırma teknikleri kullanarak zararlı yazılım evrimleştirmesi yapmaktadır. Ayrıca mobil platformlarda zararlı yazılım evrimleştirmek için evrimsel hesaplama kullanan ilk çalışmadır. Zararlı yazılım evrimleştirmek için evrimsel hesaplama kullanmanın 3 avantajı vardır. Birinci avantaj arama uzayını küçülterek, zararlı yazılım üzerinde en iyi geliştirmeyi sağlayacak karıştırma tekniklerini hızlıca bulmasıdır. İkinci avantajı, virüs-antivirüs eş evrimi için bir alt yapı oluşturmasıdır. Üçüncü ve son avantajı ise kod şişmesidir. Kod şişmesinin, daha karmaşık zararlı yazılımların evrimleştirilmesine yardımcı olduğu sonuçlarda gözlemlenmiştir.

İleriki çalışmalarda önerilen yönteme daha fazla karıştırma tekniği eklenmesi planlanmaktadır. *smali* kodları üzerinde daha derinlemesine bir analiz yapılarak çaprazlama işlecinde kaynaklanan problemlerin çözümüne yönelik çalışmalar yapılacaktır. Çaprazlama işleci kullanan genetik programlama yöntemi, daha fazla sayıda çalıştırılacaktır. Ayrıca aynı virüs ailesinden zararlı yazılımlar üzerinde çaprazlama işlemi yapılarak, çaprazlama işleminin başarısının arttırılması araştırılacaktır. Son olarak eş zamanlı olarak hem zararlı yazılım hem de zararlı yazılım tespit etme sistemi evrimleştirilmesi diğer amaçlarımızdan biridir. Bu tezde geliştirdiğimiz sistem ileride yapmayı düşündüğümüz eş evrim sisteminin bir parçası olarak tasarlanmıştır.

KAYNAKLAR

[1] Kaspersky Lab. Mobile malware evolution: 3 infection attempts per user in 2013, **Ağustos 2014**. http://www.kaspersky.com/about/news/virus/2014/Mobile-malware-evolution-3-infection-attempts-per-user\-in-2013.

[2] AppBrain Stats. Number of android applications, **Ağustos 2014**. http://www.appbrain.com/stats/free-and-paid-android-applications.

[3] Amazon. Amazon appstore for android, **Ağustos 2014**. http://www.amazon.com/mobile-apps/b?node=2350149011.

[4] AppBrain. Top android apps and games in the android market — appbrain.com, **Ağustos 2014**. http://www.appbrain.com/.

[5] BGR. Android activations, app downloads: 1.5m devices per day, 50b apps — bgr, **Ağustos 2014**. http://bgr.com/2013/07/20/android-activations-app-downloads/.

[6] AV-TEST. The independent it-security institute, **Ağustos 2014**. http://www.av-test.org/en/home/.

[7] AV-COMPARATIVES. Independent tests of anti-virus software, **Ağustos 2014**. http://www.av-comparatives.org/.

[8] M. Zheng, P. P. C. Lee, J. C. S. Lui. Adam: An automatic and extensible platform to stress test android anti-virus systems. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, volume 7591 of *Lecture Notes in Computer Science*, pages 82–101. Springer Berlin Heidelberg, **2013**.

[9] VirusTotal, **Ağustos 2014**. https://www.virustotal.com/.

[10] Antiy Labs, **Ağustos 2014**. http://www.antiy.net/.

[11] V. Rastogi, Y. Chen, X. Jiang. Droidchameleon: Evaluating android anti-malware against transformation attacks. In *Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security*, ASIA CCS '13, pages 329–334. ACM, New York, NY, USA, **2013**.

[12] Contagio Malware Dump, **Ağustos 2014**. http://contagiodump.blogspot.com.tr/.

[13] M. Christodorescu S. Jha. Testing malware detectors. In *Proceedings of the 2004 ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2004)*, pages 34–44. ACM Press, Boston, MA, USA, **2004**.

[14] J. A. Morales, P. J. Clarke, Y. Deng, B.M. Golam Kibria. Testing and evaluating virus detectors for handheld devices. *Journal in Computer Virology*, 2(2):135–147, **2006**.

[15] A. Moser, C. Kruegel, E. Kirda. Limits of static analysis for malware detection. In *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*, pages 421–430. **2007**.

[16] I. You K. Yim. Malware obfuscation techniques: A brief survey. In *Broadband, Wireless Computing, Communication and Applications (BWCCA), 2010 International Conference on*, pages 297–300. **2010**.

[17] M. Christodorescu, J. Kinder, S. Jha, Katzenbeisser S., Veith H., Technische Universität München. Malware normalization. Technical report, **2005**.

[18] L. Wu Y. Zhang. Research of the computer virus evolution model based on immune genetic algorithm. In *Proceedings of the 2011 10th IEEE/ACIS International Conference on Computer and Information Science*, ICIS '11, pages 9–13. IEEE Computer Society, Washington, DC, USA, **2011**.

[19] WildList. The wildlist organization international, **Ağustos 2014**. http://www.wildlist.org/.

[20] S. Noreen, S. Murtaza, M. Z. Shafiq, M. Farooq. Evolvable malware. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, GECCO '09, pages 1569–1576. ACM, New York, NY, USA, **2009**.

[21] F. Shahzad, M. Saleem, M. Farooq. A hybrid framework for malware detection on smartphones using elf structural & pcb runtime traces. Technical report, Tech. Report TR-58 FAST-National University, Pakistan, **2012**.

[22] S. Noreen, S. Murtaza, M. Z. Shafiq, M. Farooq. Using formal grammar and genetic operators to evolve malware. In *Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection*, RAID '09, pages 374–375. Springer-Verlag, Berlin, Heidelberg, **2009**.

[23] H. G. Kayacık, M. Heywood, N. Zincir-Heywood. On evolving buffer overflow attacks using genetic programming. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, GECCO '06, pages 1667–1674. ACM, New York, NY, USA, **2006**.

[24] Snort. Snort.org, **Ağustos 2014**. `https://www.snort.org/`.

[25] H. G. Kayacık, A. N. Zincir-Heywood, M. I. Heywood, S. Burschka. Generating mimicry attacks using genetic programming: A benchmarking study.

[26] H. G. Kayacık, A. N. Zincir-Heywood, M. I. Heywood. Can a good offense be a good defense? vulnerability testing of anomaly detectors through an artificial arms race. *Appl. Soft Comput.*, 11(7):4366–4383, **2011**.

[27] H. G. Kayacık, A. N. Zincir-Heywood, M. I. Heywood. Evolutionary computation as an artificial attacker: generating evasion attacks for detector vulnerability testing. *Evolutionary Intelligence*, 4(4):243–266, **2011**.

[28] G. Suarez-Tangil, J. E. Tapiador, P. Peris-Lopez, J. Blasco. Dendroid: A text mining approach to analyzing and classifying code structures in android malware families. *Expert Systems with Applications*, 41(4, Part 1):1104 – 1117, **2014**.

[29] Androguard. androguard - reverse engineering, malware and goodware analysis of android applications ... and more (ninja !) - google project hosting, **Ağustos 2014**. `https://code.google.com/p/androguard/`.

[30] S. Cesare Y. Xiang. Classification of malware using structured control flow. In *Proceedings of the Eighth Australasian Symposium on Parallel and Distributed Computing - Volume 107*, AusPDC '10, pages 61–70. Australian Computer Society, Inc., Darlinghurst, Australia, Australia, **2010**.

[31] M. Grace, Y. Zhou, Q. Zhang, S. Zou, X. Jiang. Riskranker: Scalable and accurate zero-day android malware detection. In *Proceedings of the 10th International*

Conference on Mobile Systems, Applications, and Services, MobiSys '12, pages 281–294. ACM, New York, NY, USA, **2012**.

[32] Aubrey-Derrick Schmidt. *Detection of Smartphone Malware*. Ph.D. thesis, Technische Universität Berlin.

[33] Yao D. D. Ryder B. G. Jiang X. Elish, K. O. A static assurance analysis of android applications. *Technical Report TR-13-03, Computer Science, Virginia Tech*, **2013**.

[34] Android. Android security overview, **Ağustos 2014**. `https://source.android.com/devices/tech/security/`.

[35] W. Enck, D. Octeau, P. McDaniel, S. Chaudhuri. A Study of Android Application Security. In *Proceedings of the 20th USENIX Security Symposium*. **2011**.

[36] J. Burns. Developing secure mobile applications for android, **Ağustos 2014**. `https://www.isecpartners.com/media/11991/isec_securing_android_apps.pdf`.

[37] Google. Google play, **Ağustos 2014**. `https://play.google.com/store`.

[38] H. Lockheimer. Android and security, **Ağustos 2014**. `http://googlemobile.blogspot.com.tr/2012/02/android-and-security.html`.

[39] Y. Zhou X. Jiang. Dissecting android malware: Characterization and evolution. In *Proceedings of the 2012 IEEE Symposium on Security and Privacy*, SP '12, pages 95–109. IEEE Computer Society, Washington, DC, USA, **2012**.

[40] A. P. Felt, E. Chin, S. Hanna, D. Song, D. Wagner. Android permissions demystified. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, pages 627–638. ACM, New York, NY, USA, **2011**.

[41] W. Enck, M. Ongtang, P. McDaniel. Understanding android security. *Security Privacy, IEEE*, 7(1):50–57, **2009**.

[42] Android. Services, **Ağustos 2014**. `http://developer.android.com/guide/components/services.html`.

[43] Android. Activities, **Ağustos 2014**. http://developer.android.com/guide/components/activities.html.

[44] Android. Building and running, **Ağustos 2014**. http://developer.android.com/tools/building/index.html.

[45] Google. zipalign, **Ağustos 2014**. http://developer.android.com/tools/help/zipalign.html.

[46] dex2jar, **Ağustos 2014**. https://code.google.com/p/dex2jar/.

[47] JAD, **Ağustos 2014**. http://varaneckas.com/jad/.

[48] Apktool: A tool for reverse engineering Android apk files, **Ağustos 2014**. https://code.google.com/p/android-apktool/.

[49] JEB. The interactive android decompiler, **Ağustos 2014**. http://www.android-decompiler.com/.

[50] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, **1992**.

[51] W. Banzhaf, F. D. Francone, R. E. Keller, P. Nordin. *Genetic Programming: An Introduction: on the Automatic Evolution of Computer Programs and Its Applications*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, **1998**.

[52] R. M. Friedberg. A learning machine: Part i. *IBM Journal of Research and Development*, 2(1):2–13, **1958**.

[53] R. M. Friedberg, B. Dunham, J. H. North. A learning machine: Part ii. *IBM Journal of Research and Development*, 3(3):282–287, **1959**.

[54] Oracle. keytool - key and certificate management tool, **Ağustos 2014**. http://docs.oracle.com/javase/7/docs/technotes/tools/solaris/keytool.html.

[55] Oracle. jarsigner, **Ağustos 2014**. http://docs.oracle.com/javase/7/docs/technotes/tools/windows/jarsigner.html.

[56] ECJ. A java-based evolutionary computation research system, **Ağustos 2014**. http://cs.gmu.edu/~eclab/projects/ecj/.

[57] D. R. White S. Poulding. A rigorous evaluation of crossover and mutation in genetic programming.

[58] H. B. Ozkan, E. Aydogan, S. Sen. An ensemble learning approach to mobile malware detection. Technical report, Hacettepe University, Department of Computer Engineering, **2014**. http://eprints.cs.hacettepe.edu.tr/7/.

[59] DroidBox. droidbox - android application sandbox - google project hosting, **Ağustos 2014**. https://code.google.com/p/droidbox/.

[60] MySQL. Mysql :: The world's most popular open source database, **Ağustos 2014**. http://www.mysql.com/.

[61] University of Waikato. Weka 3 - data mining with open source machine learning software in java, **Ağustos 2014**. http://www.cs.waikato.ac.nz/ml/weka/.

[62] L. Breiman. Random forests. *Mach. Learn.*, 45(1):5–32, **2001**.

[63] Android. Android emulator, **Ağustos 2014**. http://developer.android.com/tools/help/emulator.html.

[64] Tech Data. Kaspersky lab core detection technologies - comprehensive protection from threats of today and tomorrow, **Ağustos 2014**. https://www.techdata.com/business/kaspersky/files/KasperskyLab_CoreDetectionTechnologies_WP0607.pdf.

[65] BitDefender. Tales from cloud nine, **Ağustos 2014**. http://labs.bitdefender.com/wp-content/uploads/2010/04/MChiriac-VB2009.pdf.

ÖZGEÇMİŞ

Kimlik Bilgileri

Adı Soyadı :	EMRE AYDOĞAN
Doğum Yeri :	ANKARA
Medeni Hali :	Bekar
E-posta :	emreaydogan@hacettepe.edu.tr
Adresi :	Ayvalı Mah. Seval Cad. Aydoğan Apt. 19 / 9 Etlik KEÇİÖREN / ANKARA

Eğitim

Lise :	Yahya Kemal Beyatlı Lisesi, Ankara, TÜRKİYE
Lisans :	Bilgisayar Mühendisliği ,Erciyes Üniversitesi, Kayseri, TÜRKİYE

Yabancı Dil ve Düzeyi

İngilizce - Orta

İş Deneyimi

Yazılım Mühendisi,	Dirisoft Bilgi ve İletişim Teknolojileri Ltd. Şti., Nisan 2012 - Ocak 2013

Tezden Üretilmiş Projeler ve Bütçesi

112E354 Tübitak Projesi,
Kötücül Yazılımların ve Anti-Kötücül Yazılım Sistemlerinin Eş Evrimi,
Bütçe : 144.235,00 TL

Tezden Üretilmiş Yayınlar

”An ensemble learning approach to mobile malware detection”,
Teknik Bildiri , Hacettepe Üniversitesi, Bilgisayar Mühendisliği Bölümü, 2014,
H. B. ÖZKAN, E. AYDOĞAN, S. ŞEN

Tez ile İlgili Diğer Yayınlar

”Zararlı Yazılımların Tespitinde Makine Öğrenmesi Yöntemlerinin Analizi”,
IEEE 22. Sinyal İşleme ve İletişim Uygulamaları Kurultayı (SİU-2014) , Nisan 2014,
E. AYDOĞAN, S. ŞEN

Yayınlar

”2-Opt Based Artificial Bee Colony Algorithm For Solving Traveling Salesman Problem”,
2nd World Conference on Information Technology (WCIT-2011) , Kasım 2011,
B. AKAY, E. AYDOĞAN, L. KARACAN