Mobil Cihazlarda Zararlı Yazılım Tespitinde Kullanılan Statik Analiz Araçları

Gülsüm KAYABAŞI, İbrahim Alper DOĞRU
Gazi Üniversitesi Teknoloji Fakültesi
Bilgisayar Mühendisliği Bölümü
glsmkayabasi@gmail.com, iadoğru@gazi.edu.tr

***Özet*—Günümüz teknolojisinde kullanılan akıllı telefonlar mobil işletim sistemlerine sahiptir. Bu mobil işletim sistemleri her kullanıcı ihtiyacına göre farklılık göstermektedir. Açık kaynak kodlu ve Linux tabanlı bir mobil işletim sistemi olan Android'in yapılan çeşitli araştırmalar sonucunda en popüler işletim sistemi olduğu görülmüştür [1]. Android'in popüler olması ve açık kaynak kod yapısı, kötücül saldırı tiplerinin ve kötü amaçlı saldırganların hedefi olmasına sebep olmuştur. Bu çalışmamızda, mobil cihazlarda zararlı yazılım tespiti yapan statik analiz yöntemlerinden bahsedilmiş ve bu yöntemlerin özellikleri karşılaştırılmıştır.**

Anahtar Kelimeler—Mobil işletim sistemi, Android, kötücül yazılım, mobil uygulama güvenliği, statik analiz metotları

I. GİRİŞ

İSTENMEYEN kötü amaçlı yazılım kapsamına giren virüs, solucan, Truva atları, casus ve reklam içerikli yazılımlar yıllardır kişisel veya kurumsal bilgisayar sistemlerine tehdit oluşturmuşlardır [2]. Akıllı telefon kullanıcılarının sayısı arttıkça, akıllı telefon platformlarının çeşitli tehditlere duyarlı hale gelmesi de bir gerçektir [3]. Bu tehditlerden masaüstü bilgisayarlardan bilindiği gibi, akıllı telefonlar için cep telefonları ve sofistike kompakt minibilgisayarlar da etkilenmektedir [4][5].

Geleneksel bilgisayar sistemlerine benzer şekilde, akıllı platformlarda zararlı yazılım tespiti için birçok çözüm imza tabanlı yaklaşımlara dayanmaktadır [2]. İmza tabanlı ikili dosya eşleştirme yaklaşımının kolay ve verimli olduğu bilinmektedir. Ancak bunun gibi çözümler her zaman zararlı yazılım tespit etmede verimli olmamaktadır çünkü imzaların kapsamı ve kalitesi değişiklik göstermektedir. Diğer bir y.anlış negatif koşulları icat etmiştir. Öte yandan, imzaların oluşumu statik ve dinamik analiz metotlarının oluşmasını gerektirmiştir. Bu metotlar zaman alıcı ve uzmanlık gerektiren interaktif süreçlerdir.

Piyasada bulunun akıllı telefonlardaki kötü amaçlı yazılımlara karşı alınan önlemler imzalama yaklaşımına sahip olduğundan beri zayıflığa maruz kalmıştır. Bu yaklaşım imza kullanılabilir oluncaya kadar kullanıcıları yeni bir zararlı yazılım ile karşı karşıya bırakmıştır. Yapılan çalışmalardan örnek verilecek olursa; Bulygin [6] en kötü durumda 700.000'den fazla cihaz kullanarak bir MMS solucanının rastgele hedeflediği bir telefon rehberindeki numaralara yaklaşık 3 saat içerisinde bulaşabileceğini göstermiştir. Buna ek olarak, Oberhide ve arkadaşları[7], bir imza tabanlı anti virüs motoru için yeni tehditleri tespit etmesi için gerekli ortalama sürenin 48 gün olduğunu değerlendirmiştir. Akıllı telefonlardaki kötücül yazılım için bu sayılar güvenlik önlemleri genişletildiğinde virüslü bir cihazda saniyeler içersinde ciddi bir şekilde zarar verebilir. Bu durumda yeni akıllı telefon işletim sistemlerinden biri olan Android, geliştiriciler arasında özel bir ilgi kazanmıştır. Açık kaynak kurmak için, güvenlik araçları çekirdek seviyesinde geliştirilebilir. Bu da Android telefonlar üzerine kurulan sadece mobil cihazlar için kaynak kısıtları ile sınırlı kapsamlı bir güvenlik mekanizması oluşumuna izin vermiştir [7].

Mobil cihazlardaki kaynak kısıtlamaları nedeniyle, Android cihazlarda zararlı yazılımın varlığının tespit edilmesi için statik ve hafif mekanizmalara odaklanılmıştır. Zararlı yazılım tespiti için kullanılan statik analiz yaklaşımı, çok kaynak tüketmeyen ve mobil ihtiyaçlara çok iyi uyum sağlayan basit sınıflandırıcıların kullanımına izin vermiştir. Önceki yaklaşımlar, mobil cihazın hesaplama yükünü azaltmak için çoğunlukla dış sunuculara bağlıdır. Bu durumda ise, tespit işlemi için sunucudan yararlanılmıştır fakat buna bağlı olmak zorunda değildir. Böylece, ağır öğrenme mekanizması işlemi için, uzak sunucunun entegrasyonundan yararlanılmıştır [8][9].

Bu çalışmada statik analiz yöntemi kullanan mimarilerden bahsedilmiştir. Çalışmanın geri kalan kısmı şu şekilde yapılandırılmıştır: 2.bölümde mobil cihazlarda kullanılan zararlı yazılım tespit yöntemleri özetlenmiştir, 3.bölümde statik analiz metodunu kullanan mimariler detaylı bir şekilde anlatılmıştır ve 4.bölümde ise sonuçlar ve bu mimarilerin karşılaştırmaları yapılmıştır.

II. MOBİL CİHAZLARDA ZARARLI YAZILIM TESPİTİNDE KULLANILAN METOTLAR

Modern bilgisayar ve iletişim altyapıları saldırılara karşı son derece duyarlıdır. Bu saldırıların başlamasının en önemli sebepleri; solucanlar, virüsler ve Truva atları vb. zararlı yazılımların yayılmasıdır. Bu yayılma ticari şirketlere, özel kullanıcılara ve devletlere ciddi hasarlar verebilir. Bu hasarların yayılmasının en önemli sebebi internet kullanımının yoğunlaşmış olmasıdır. Özellikle yüksek hızda internet

bağlantılarındaki son büyüme yeni zararlı yazılımların oluşumunda artışlara sebep olmuştur.

Zararlı yazılım tespiti için çeşitli analiz teknikleri önerilmiştir [10]. Genel olarak statik analiz ve dinamik analiz olmak üzere ikiye ayrılır. Dinamik Analiz (davranışsal tabanlı analiz olarak da bilinir.) yönteminde dosya ve hafiza değişiklikleri, ağ erişimi ve sistem çağrıları gibi işletim sisteminin çalışma zamanında (yani programın yürütülmesi sırasında) toplanan bilgileri içeren br tespit gerçekleştirilir. Örneğin, Panorama adındaki sistem; farklı tiplerdeki zararlı yazılımın ortak işlem davranışlarını (örneğin bilgi akış analizi) ve şüpheli bilgi erişim desenleriyle dinamik yöntemin fizibilitesini çıkarır ve gösterir. Bu tespitlerin dönüşümü mobil ortama yaklaşır ancak akıllı telefonların dayattığı kaynak kısıtlamaları(CPU,güç.hafıza) yüzünden düzgün değildir [10].

Statik Analiz yönteminde; program hakkında bilgi veya ikili/kaynak kodundaki açık veya üstü kapalı gözlemlerin içerdiği davranışların olduğu tahmin edilmektedir. Statik analiz teknikleri sınırlı olmasına ragmen hızlı ve etkilidir. Çeşitli bulanıklaştırma teknikleri statik analiz yöntemini kullanmaktan kaçınmaya sebebiyet verebilir ve böylece sınırlı polimorfik zararlı yazılımlarla başa çıkma yetenekleri kılar.

Dinamik analiz yaklaşımında, çeşitli bulanıklaştırma metotları problemler meydana gelir çünkü programın gerçek davranışı monitörlenir. Bununla birlikte bu yaklaşım diğer dezvantajlara da sahiptir. İlk olarak, programın zararlı fonksiyonları aktif olacağında(örneğin zararlı yazılımın sömürdüğü hassas uygulamalar) uygun durumların simülasyonu zordur. İkincisi, her kötü amaçlı yazılım için kötü niyetli faaliyet görünümünü gözlemleme ihtiyacının ne kadar süre gerektirdiği belirsizdir [10].

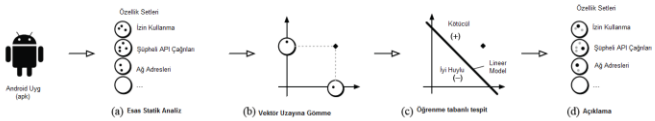
Statik analiz ve dinamik analiz çözümleri imza tabanlı ve sezgisel tabanlı iki metot kullanılarak uygulanmaktadır. İmza tabanlı metot anti virüs üreticileri tarafından sık kullanılan bir metottur ve zararlı yazılımı benzersiz bir imzayla tanımlamaya dayanmaktadır. Çok hassas olduğu durumlarda, imza tabanlı metotlar bilinmeyen bir kötücül yazılım koduna karşı kullanışlı değildir. Sezgisel tabanlı metot, uzmanlar tarafından veya makine öğrenmesi teknikleri ile tanımlanan kötücül veya iyi huylu yazılımlar için meydana gelmiş kurallara dayanmaktadır.

Son zamanlarda, sınıflandırma algoritmaları sezgisel tabanlı metot fikirlerini genişletmek ve otomatikleştirmek için kullanılmaya başlanmıştır. Bu metotlar, dosyanın ikili kodu veya uygulamanın çalışma esnasındaki davranışını temsil etmektedir. Sınıflandırıcılar uygulamaların iyi veya kötü huylu olduğunu sınıflandırarak desenlerin öğrenilmesi için kullanılmaktadır [10].

III. ZARARLI YAZILIM TESPİTİNDE KULLANILAN STATİK ANALİZ METOTLARI

Android işletim sistemi geliştikçe kötücül yazılım çeşitleri ve sayısı da artmaktadır. Bunlara yönelik koruma yöntemleri de birbirlerine paralel şekilde çoğalmaktadır. Android mobil işletim sistemine sahip cihazlardaki uygulamaların kurulumlarından önce uygulamanın bizlere verdiği bilgiler (veriler) sayesinde yapılması statik analiz metodunu açıklar. Statik analiz ile zararlı yazılım tespiti ve korunmasının en büyük avantajı kötücül yazılımın kullandığımız cihaza kurulmadan önce tespit edilmesidir. Böylece cihaz hiçbir şekilde kötücül yazılımın sebep olduğu etkilere maruz kalmaz.

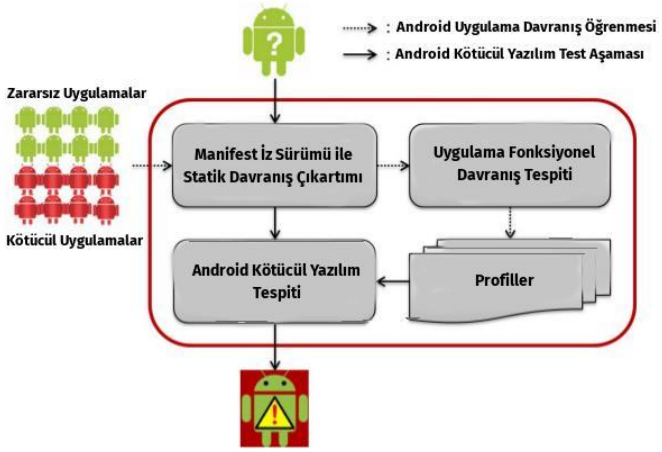
Statik analiz yaklaşımıyla kötücül yazılım tespiti yapan araçlar bulunmaktadır. Bu araçların birkaçından bahsedecek olursak Drebin[11], statik analiz yaklaşımına makine öğrenmesi yaklaşımı da ekleyerek Android'te kötücül yazılım tespitinde kullanılan bir araç olmuştur. Drebin, android uygulamasının kaynak kodlarını ve AndroidManifest dosyasını kullanarak uygulamaya ait izinleri, uygulama programlama arayüzü çağrılarını ve ağ adresleri gibi çeşitli özellikleri toplamaktadır. Bu özelliklerin tamamı bir araya gelerek bir vektör uzayına gömülmüştür ve çıkan desenler yardımıyla kötücül yazılım tespiti yapılmıştır. Drebin kötücül yazılım tespit aracı tarafından yapılan statik analiz adımlarının şematik yapısı Şekil-1'de gösterilmiştir.



Şekil-1 Drebin aracının statik analiz adımları

Şekil-1'de statik analiz yapılırken kullanılan özellik setleri oluşturulmuş ve bunlar vektör uzayının içerisine gömülmüştür. Daha sonra lineer bir model kullanılarak iyi ve kötü huylu uygulamalar meydana çıkarılmıştır.

DroidMat[12] aracı ise, AndroidManifest dosyasını ve onunla alakalı izinlere yönelik uygulama programlama arayüzü (API) çağrıları üzerinden kötücül yazılım tespiti yapar. DroidMat mimari yapısı Şekil-2'de gösterilmiştir.

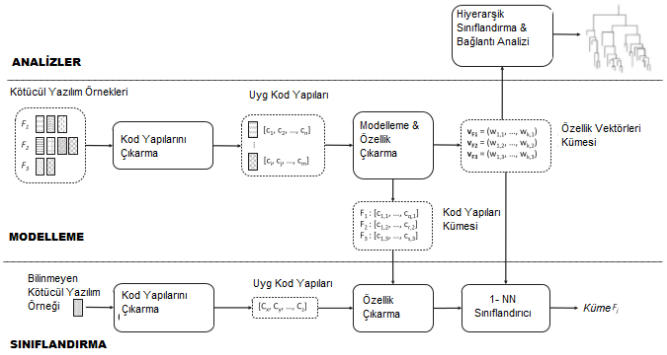


Şekil-2 Droid Mimari Yapısı [13]

Şekil-2'de görüldüğü üzere, manifest dosyası sayesinde statik analiz çıkarımı yapılmıştır. Analizler elde edildikten sonra fonksiyonel davranış tespiti safhasına geçilmiştir. Analizler sonucunda K-means adı verilen bir algoritma ile kötücül yazılım modelleri oluşturulmuştur. Burada oluşacak küme sayısı tekil değer ayrışımı (SVD-Singular Value Decomposition) algoritması ile belirlenmiştir. En sonunda ise k=1 olmak üzere kNN (k Nearest Neighbours) algoritması ile

android uygulamasının kötücül olup olmadığı tespit edilmiştir [12].

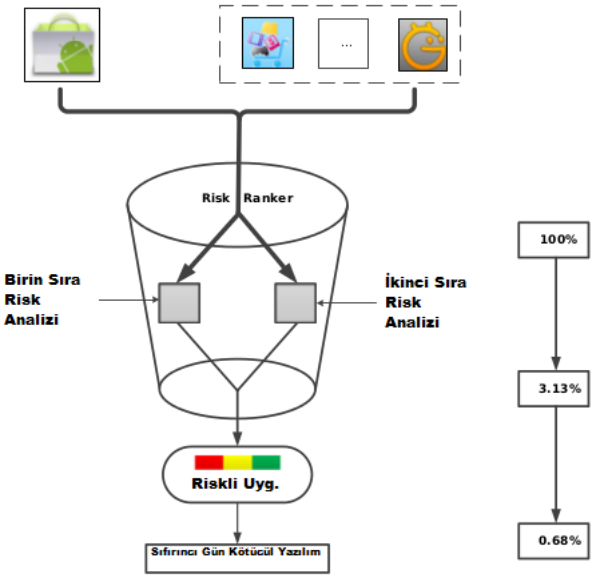
DENDROID, iki farklı yolla yeni çıkmış bir analiz aracıdır. Bir taraftan, Android mobil işletim sistemindeki daha önce ortaya çıkmamış kötücül yazılım ailelerinin karakteristiklerini belirlerken kod yapılarını kullanma hususunda en iyisidir. Kod metotlarının iç yapısına odaklanmanın komutların özel dizilerine göre önemli bir avantajı bulanıklaştırmaya karşı direnç geliştirmiş olmasıdır. Ayrıca bu tür yapılar, kodların yeniden kullanımına dayalı hızlı geliştirme metodolojilerinin yaygın olduğu yerlerde, akıllı telefonlardaki kötücül yazılımın durumu için kullanışlı olduğu bilhassa kanıtlanmıştır [14].



Şekil-3 DENDROID Mimari Yapısı

Öte yandan, örnekleri sınıflandırmak, kod bileşenlerini araştırmak veya kötücül yazılım ailelerinin evrimsel bağlantıları üzerine çalışma durumlarını otomatikleştirmek için metin madenciliği tekniklerini kullanma fikrini ortaya atmıştır. Ayrıca, metin madenciliği teknikleri büyük miktarda verilerle etkin bir şekilde başa çıkarak geliştirilmiş ve özellikler adreslenen problemler için çok elverişli bir şekilde oluşturulmuştur. DENDROİD mimarisi Şekil-3'te gösterilmiştir [14].

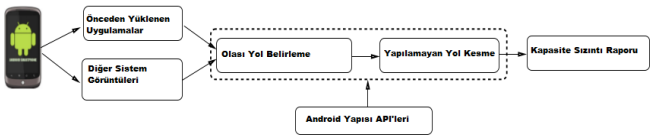
RiskRanker, ölçeklenebilir şekilde ve sıfırıncı gün kötücül yazılımı ortaya çıkarmak için Android marketteki bütün uygulamaları elemek için dizayn edilmiştir. Güvenilmeyen uygulamalardan potansiyel risk taşıyanları ön planda tutarak ve değerlendirerek, ölçeklenebilirlik, verimlilik ve doğruluk gereksinimlerini doğal olarak artıran yönetilebilir bir alan oluşturmak amaçlanmaktadır.Daha ayrıntılı olarak kaynak verimli olma durumuna bakarak, istenilen zamanda yüzlerce veya binlerce uygulamayı ele almak için ölçeklenebilir olmalıdır. Ayrıca sistemin yeterince kısa olan listeyi elle doğrulamak için girdileri verimli bir şekilde ayıklamaya ihtiyacı vardır ve kötücül uygulamaları tespit ettiğini yeterince doğrulamalıdır. RiskRanker mimarisi Şekil-4'te gösterildiği gibidir. RiskRanker'da tehlike içeren riskler düşük, orta ve yüksek seviyede risk olmak üzere 3'e ayrılmıştır. Risk için düşükten yükseğe doğru gidecek olursak uygulamalara erişim güçleştikçe risk seviyesinin azaldığı görülmüştür. Yani risk ile uygulamaya erişim arasında ters bir orantı bulunmaktadır. RiskRanker'da iki kademeli risk analizi yapılmıştır. İlk kademesinde uygulamaların ne derecede riskli olup olmadıklarına bakılmıştır. Alçak seviyede riske sahip olanlar haricindekiler için tehlike içeriği kontrol edilmiştir. 118,318 uygulama kullanılarak test edilmiş bunların 3281 tanesi risk içermiştir. 3281 tane uygulamanın da 322 tanesinin sıfırıncı gün zararlı yazılımı olduğu tespit edilmiştir [15].



Şekil-4 RiskRanker Mimarisi

AppProfiler yönteminde vurgulanmak istenen, kötücül yazılım tespit etmek değil, yasal uygulamalar hakkında izin verilen kullanıcılara bilgi sağlamaktır. Doğal kodlar veya uygulamalarla çalışmak analizleri önlemek için olağanüstü adımlar atılmasını gerektirir. Odaklanılan konu Android API'sinin nasıl kullanıldığının veya bu API'nin harici olup olmadığının gösterilmesidir. Korumaları yıkmaya çalışmak AppProfiler altyapısının kapsamı dışındadır. Kabul edilmeyen bir gizlilik ihlali olduğunu belirleme girişimi bulunmamaktadır. Bu kapsamda çoğu davranışın zararsız olduğu keşfedilmiştir. Örneğin, çalındığı zaman izlenen bir mobil uygulama casus yazılımdan ayırt edilebilir. Diğerleri kullanıcıdan kullanıcıya farklılık gösterebilir. Örneğin, konum alan uygulamalarla ilgili bütün kullanıcılar bilgi sahibi değildir. AppProfiler'ın amacı yüklenen uygulamalar hakkında kullanıcılara fikir vermektir [16].

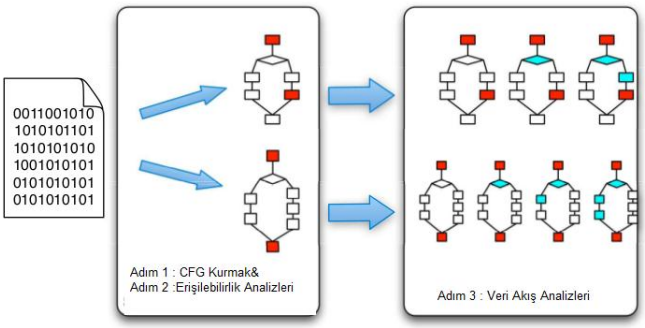
Kapasite sızıntılarının açığa çıkarılmasını kolaylaştırmak için Woodpecker[17] adında bir sistem geliştirilmiştir. Önceden yüklenmiş uygulamalar üzerinde veri akış analizini kullanarak, Woodpecker sistematik olarak açık veya savunmasız arayüzden tehlikeli bir izne erişimi telefonda her uygulama için analiz eder.



Şekil-5 Woodpecker Mimarisi

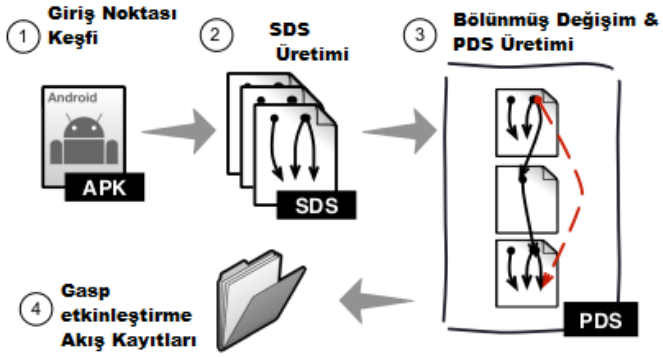
Olası kapasite sızıntılarını en iyi şekilde inceleyen sistemde iki farklı kategori bulunmaktadır. Açık kapasite sızıntıları, kendi kendine izin talep etmeyen servisler veya erişilebilir bazı arayüzlerin kesin izinlere başarılı bir şekilde erişmesine izin verir. Kapalı kapasite sızıntıları ise bazı açık arayüzler veya servisler dışındakilere izin verir. İmzalama anahtarının aynısıyla diğer uygulamanın aldığı izinleri aktarılır veya elde edilir. Sonuç olarak açık sızıntılar ciddi güvenlik hataları verir. Kapalı sızıntılar ise Android'te izin tabanlı güvenlik modelini yıkar ve uygulamaya yeteneklerini yanlış sunar. Woodpecker mimarisi Şekil-5'te gösterildiği gibidir [17].

PiOS (Privacy İOS), ilk olarak uygulamanın hassas veriye eriştiği ve sonradan ağ üzerinde verinin iletildiği yerde kod yollarının varlığı için uygulamaları kontrol edebilmek adına statik analiz yöntemini kullanmıştır. PiOS, binarilerde doğrudan analiz gerçekleştirmelidir çünkü kaynak kodu mevcut değildir. Statik analiz gibi ikili analiz yöntemi de zor bir yöntemdir. Bunun yanı sıra İOS uygulamalarının çoğu nesne tabanlı C programlama dili ile geliştirilmiş olup daha da karmaşıklaşmıştır. PiOS mimarisini Şekil-6'da görebiliriz [18].



Şekil-6 PiOS Mimarisi

Android uygulamalardaki bileşen kaçırma açıklıklarını tespit edebilmek için hataları otomatik olarak ayıklamak işlevini gerçekleştiren CHEX sistemi statik analiz metodu önerilmiştir. Bir veri akış analizi açısından bu açıklıkların modellenmesi, CHEX'in Android uygulamaları analiz etmesi ve sistemin bağımlı grafiklerinde erişilebilir testle yapması ile gerçekleşmektedir. Android'in özel programlama paradigması tarafından kullanıcıya zorla sunulan analiz zorluklarıyla mücadele etmek için uygulamada asenkron uygulamaların çoklu giriş noktaları modellenmiştir ve bu giriş noktası bileşenleri keşfetmek ve bütünlüğü korumak için yeni bir teknik kullanılmıştır [19].



Şekil-7 CHEX İş Akış Diyagramı

CHEX (Component Hijacking Examiner), Dalysis (Dalvik Bytecode Analysis) örnek alınarak geliştirilmiştir. Android uygulamasının baytkodunda analizlerin birçok tipini desteklemek için kurulan kapsamlı statik analiz çerçevesidir. CHEX'in çalışma prensibine göre elimizdeki Android uygulama alınarak önce veri akış özetine bölünür ve daha sonra bölünen parçalardan bitişik olanlar birbirlerine bağlanır. CHEX, 5486 Android uygulaması incelenmiş ve 254 tanesinde potansiyel bileşen kaçırma açıklığı bulunmuştur. CHEX'in bir uygulamada ortalama çalışma süresi 37.02 saniyedir. Bu yeterince hızlı olup test senaryoları ve uygulama incelemeleri için kullanılmaktadır. CHEX'in iş akışı Şekil-7'de görüldüğü gibidir [19].

IV. SİSTEM KARŞILAŞTIRMALARI

Sistem karşılaştırmaları sadece statik analiz metotu kullanan analiz metotları için monitörleme tipi (sistem çağrıları, ağ, olay günlüğü, talimatlar, izinler, program izleri, işlem kontrol blokları, API çağrıları, çekirdek seviyesi ve kullanıcı seviyesi), analiz tipi (uzman, makina öğrenmesi, sınıflandırma, bağımlı grafikler, istatistikler ve olasılık modelleri), teşhis tipi (anomali, suistimal ve spesifikasyon), monitörleme ve teşhiş yeri ve analiz yeri (yerel anahat, IRM, bulut, dağıtık, balküpü, bulut yineleme, sandbox, hibrit) olarak yapılacaktır. Bu yönemlerin karşılaştırmalı haline Tablo-1'de ulaşabilirsiniz.

Tablo-1 Statik Analiz Metotlarının Karşılaştırılması

Tespit Yaklaşımı	DENDROID	AppProfiler	RiskRanker	WoodPecker	CHEX	PiOS
Platform	Android	Android	Android	Android	Android	IOS
Monitörleme	Talimatlar	API Çağrıları, Program İzleri	Talimatlar, İzinler, API Çağrıları	Talimatlar, İzinler	Talimatlar	Yok
Analiz Tipi	İstatsitiksel, Bağımlı Grafikler, Sınıflandırma	Uzman	Bağımlı Grafikler	Bağımlı Grafikler	Bağımlı Grafikler	Bağımlı Grafikler
Teşhis Tipi	Yok	Suistimal	Suistimal	Yok	Yok	Yok
Monitörleme ve Teşhis Yeri	Bulut	Yerel Anahat	Bulut	Bulut	Bulut	Bulut
Analiz Yeri	Bulut	Bulut,Yerel Anahat	Bulut	Bulut	Bulut	Bulut

Tablo-1'de görüldüğü gibi, tespit yaklaşımlarına göre yapılan karşılaştırmada Android platformunu kullanmayan tek araç PiOS aracıdır ve monitörleme ve teşhiş yaklaşımlarını kullanmamaktadır. Monitörleme yaklaşımında AppProfiler'ın

Talimatlar tipini kullanmadığı, Analiz Tipi karşılaştırmasında AppProfiler dışındakilerin bağımlı grafiklerden faydalandığı, Teşhis Tipi yaklaşımını suistimal boyutunda AppProfiler ve RiskRanker'ın kullandığı, Monitörleme ve Teşhis Yeri karşılaştırmasında sadece AppProfiler'ın yerel anahat üzerinde çalıştığı diğerlerinin bulutta çalıştığı ve son olarak Analiz Yeri kıyaslanmasında ise hepsinin bulut kullandığı AppProfiler'da ek olarak yerel anahatta çalıştığı görülmektedir.

V.YAPILAN DEĞENLENDİRMELER VE SONUÇ

Bu çalışmada, dünya çapında kullanımı en yaygın olan mobil işletim sistemi Android için zararlı yazılım tespitinde kullanılan statik analiz metotları anlatılmış ve karşılaştırmalı olarak incelenmiştir. İnceleme sonucunda tespit yaklaşımlarına göre statik analiz yöntemiyle zararlı yazılım tespiti yapan araçların farklı yaklaşım yöntemleri olduğu kadar benzer yöntemlere sahip oldukları da görülmüştür.

Kıyaslama yapılan statik analiz araçları arasında makina öğrenmesi yaklaşımını kullanan bir metot bulunmamaktadır. İnceleme neticesinde herhangi bir Android uygulaması geliştirildikten sonra güvenlik açıklarının temelini oluşturan izin mekanizmaları uygulamanın kaynak kodları ile beraber analiz edilmesinin gerekli olduğu anlaşılmıştır. Ayrıca uygulamanın kurulumu öncesinde zararlı yazılım tespiti için uygulamanın kullanıcılarını yönlendirecek daha ayrıntılı zararlı yazılım tespit araçlarına da ihtiyaç olduğu yapılan araştırmalar sonucunda görülmüştür.

REFERENCES

[1] http://www.idc.com/prodserv/smartphone-os-market-share.jsp

[2] A. Schmidt, R. Bye, H. Schmidt, 1. Clausen, O. Kiraz, K. A. Yuksel, S. A. Camtepe, and S. Albayrak, "Static Analysis of Executables for Collaborative Malware Detection on Android", Proceedings of the 2009 IEEE international conference on Communications, 2009, pp. 63 1 - 63S.

[3] S. Seo, D. Lee, and K. Yim, "Analysis on maliciousness for mobile applications", Proceedings of 20 12 Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMlS), pp. 126-129.

[4] A. Shabtai, Y. Fledel, and Y. Elovici, "Automated Static Code Analysis for Classifying Android Applications Using Machine Learning", Proceedings of 20 I 0 International Conference on Computational Intelligence and Security, 20 10, pp. 329-333.

[5] A. Shabtai, "Malware Detection on Mobile Devices", Proceedings of 20 10 Eleventh International Conference on Mobile Data Management (MDM), 20 10, pp. 289 - 290.

[6] Y. Bulygin, "Epidemics of mobile worms," in Proceedings of the 26th IEEE International Performance Computing and Communications Conference, IPCCC 2007, April 11-13, 2007, New Orleans, Louisiana, USA. IEEE Computer Society, 2007, pp. 475–478.

[7] J. Oberheide, E. Cooke, and F. Jahanian, "Cloudav: N-version antivirus in the network cloud," in Proceedings of the 17th USENIX Security Symposium (Security'08), San Jose, CA, July 2008. *IEEE Criteria f or Class IE Electric Systems* (Standards style), IEEE Standard 308, 1969.

[8] D. Samfat and R. Molva, "IDAMN: An Intrusion Detection Architecture for Mobile Networks," IEEE Journal on Selected Areas in Communica-tions, vol. 15, no. 7, pp. 1373–1380, Sep. 1997.

[9] A. Bose, X. Hu, K. G. Shin, and T. Park, "Behavioral detection of malware on mobile handsets," in Proceeding of the 6th international conference on Mobile systems, applications, and services. Brecken-ridge, CO, USA: ACM, 2008, pp. 225–238.

[10] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, Y. Weiss, "Andromaly": a behavioral malware detection framework for android devices, J. Intell. Inf. Syst. 38 (1) (2012) 161–190.

[11] Arp D, Spreitzenbarth M, Malte H, Gascon H, Rieck K. Drebin: effective and explainable detection of Android malware in your pocket. In: Symposium on network and distributed system security (NDSS); 2014. p. 23e6.

[12] D.-J. Wu, C.-H. Mao, T.-E. Wei, H.-M. Lee, and K.-P. Wu, "DroidMat: Android Malware Detection through Manifest and API Calls Tracing," in 2012 Seventh Asia Joint Conference on Information Security, 2012, pp. 62–69.

[13] Abdullah Talha KABAKUŞ, İbrahim Alper DOĞRU, Aydın ÇETİN, "Android kötücül yazılım tespit ve koruma sistemleri," in Erciyes Üniversitei Fen Bilimleri Enstitüsü Dergisi, 31(1):9-16

[14] G. Suarez-Tangil, J. E. Tapiador, P. Peris-Lopez, and J. B. Alis, "Dendroid: A text mining approach to analyzing and classifying code structures in android malware families," Expert Systems with Applications, 2013, in Press.

[15] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang, "Riskranker: scalable and accurate zero-day android malware detection," in Proc. 10th int. conf. on Mobile systems, applications, and services. ACM, 2012, pp. 281–294.

[16] S. Rosen, Z. Qian, and Z. M. Mao, "Appprofiler: a flexible method of exposing privacy-related behavior in android applications to end users," in Proc. 3rd ACM conference on Data and application security and privacy. ACM, 2013, pp. 221–232.

[17] M. Grace, Y. Zhou, Z. Wang, and X. Jiang, "Systematic detection of capability leaks in stock android smartphones," in Proc. 19th Annu. Symp. on Network and Distributed System Security, 2012.

[18] M. Egele, C. Kruegel, E. Kirda, and G. Vigna, "Pios: Detecting privacy leaks in ios applications," in Proc. Network and Distributed System Security Symp., 2011.

[19] L. Lu, Z. Li, Z. Wu, W. Lee, and G. Jiang, "Chex: statically vetting android apps for component hijacking vulnerabilities," in Proc. 2012 ACM conf. on Computer and communications security. ACM, 2012,pp. 229–240.

Gülsüm KAYABAŞI, Ankara'da doğdu. İlk ve orta öğrenimini Ankara'da, yükseköğrenimini Eskşehir'de tamamladı. 2010 yılında Eskişehir Osmangazi Üniversitesi Bilgisayar Mühendisliği Bölümü'nden mezun oldu. 2011 yılından beri bir kamu kuruluşunda veritabanı yöneticisi olarak çalışmaktadır. 2013 yılında Gazi Üniversitesi Bilgisayar Mühendisliği Anabilim Dalı'nda yüksek lisans eğitimine başladı. İlgi alanları; mobil uygulamalar, veri güvenliği, web yazılımı..

İbrahim Alper DOĞRU, 2004 yılında Atılım Üniversitesi Bilgisayar Mühendisliği bölümünden mezun olmuştur. Yüksek lisansını 2007 yılında Gazi Üniversitesi Bilgisayar Mühendisliği bölümünde tamamlamıştır. 2012 yılında Gazi Üniversitesi Elektronik-Bilgisayar eğitimi bölümünde doktorasını tamamlamıştır. Halen Gazi Üniversitesi Bilgisayar Mühendisliği Anabilim Dalı'nda Yardımcı Doçent olarak görev yapmaktadır. İlgi Alanları; mobil ağ teknolojileri, mobil tasarsız ağlar, mobil güvenlik, ağlarda adli bilişim