Literatür Özeti

Geleneksel yazılım dağıtım mekanizmaları genellikle dağıtık olurken, mobil cihazların yaygınlaşması ile birlikte uygulamaların dağıtımı merkezileştirilmiş ve kullanıcılar, uygulama marketlerden ilgilendikleri uygulamaları indirebilmektedir. Bu tür marketlerde, uygulamanın tanımı, uygulamanın skoru, uygulama hakkında kullanıcı yorumları gibi bilgiler elde edinilebilmektedir. Son yıllarda, bu bilgilerden faydalanarak Anroid uygulamalarının analizi önem kazanmaya başlamıştır. Bu konudaki çalışma alanlarından bir tanesi uygulama tanımı ile uygulamanın izinlerinin eşleşmesinin (izin-tanım eşleştirmesi) kontrol edilmesidir.

İzin-tanım eşleştirmesindeki ilk çalışma, 2013 yıllında Pandita ve arkadaşları tarafından yapılmıştır. Whyper [5] denilen sistem, NLP yöntemlerinden faydalanarak, bazı izinlerin, uygulamanın tanımında geçip geçmediğine bakmış ve arama-tabanlı yaklaşım ile karşılaştırmıştır. Arama tabanlı yöntemde, bir sözcük, farklı cümleler içinde farklı anlamlara gelebilmektedir. Diğer yandan, Whyper anlamsal bir yaklaşım önerdiğinden, arama tabanlı yaklaşımdan çok daha iyi bir başarım göstermiştir. Whyper, tanım üzerinde temel önişleme işlemlerini (cümle sınırlarını bulma, kısaltmaları ele alma, vb. gibi) yaptıktan sonra, cümleleri First-Order Logic (FOL) gösterimlerine çevirir. Her bir izinin ise, API dökümanlarından anlamsal çizgelerini çıkarır ve FOL gösterimindeki cümleler ile eşleşip eşleşmediğine bakar. Sistemin başarısı, izinlerin anlamsal çizgelerinin yeterince kapsamlı olmasına bağlıdır. Autocog [6] da, Whyper'ın API dökümanlarından elde edilen sözcük ve bunların eşanlamlılarından oluşan sabit bir sözlük ile çalışmasının onu kısıtlandırdığını ve her bir izin için ilgili API dökümanlarının olmayabileceğini belirtmiştir. Dahası, Whyper tarafından önerilen yöntemlerin otomatikleştirilemeyeceğini ve Whyper'in yazarlarının da bunu doğruladığını belirtmişlerdir. Autocog, anlam çıkarımı için Wikipedia gibi geniş ölçekli veri tabanlarından faydalanan Belirgin Anlam Analizi (Explicit Semantic Analysis - ESA) yöntemini kullanmıştır. ESA'nın benzer sözlük tabanlı yaklaşımlardan daha iyi olduğu gösterildiği için bu yöntem tercih edilmiştir. ESA, iki metinin benzerliğini, bu iki metini gösteren vektörlerin birbirine olan cosine uzaklığı ile ölçmektedir. Bundan yola çıkarak, eğitim veri kümesindeki tüm isim tamlamalarının birbirlerine göre anlamsal benzerlik skor matrisi (semantic relatedness score matrix) üretilir. Yazarların da belirttiği gibi, bu matrisi oluşturma ölçeklenebilir değildir. Bu matris sonucu, birbirine anlamsal olarak yakın olan matrisler gruplandırılır. Sonraki aşamada, eğitim setindeki her bir izin ile bu isim tamlamaları arasındaki ilişkiyi tanımlamak için bir metrik çıkarılır. Bu metriğin çıkarılma nedeni, doğrudan bir isim tamlaması ile izinin eğitim kümesinde eşleşme sayısına bakmanın, bazı durumlarda (çok sık kullanılan isim tamlamaları, bazı isim tamlama-izin eşleşmelerinin kümede çok az sayıda durumda olması) hataya neden olabileceğidir. Bu metrik sayesinde, bir izin ile anlamsal olarak en benzer ilk n (=500) isim tamlaması seçilir. Son olarak, bir isim tamlamasının bir izine anlamsal olarak benzer olmasının, o izinin kullanımını açıklıyor olmama durumuna ilişkin bir filtreleme daha yapılır. Bu filtreleme için kullanılan eşik değeri, kullanılan veri setinden elde edilir. Sonuç olarak, Autocog, Whyper'dan ortalamada %7 oranında daha fazla doğruluk sağlamıştır.

Son olarak, 2016 yılında ACODE tanıtılmıştır [7]. Whyper'in aksine, anahtar sözcük tabanlı bir yaklaşım önerilmiş ve Whyper ile karşılaştırabilir sonuçlar ürettiğini göstermiştir. Burada amaç, daha hafif (lightweight) çözümler ile çok sayıda uygulamayı inceleyebilmek ve izin-uygulama eşleşmesine uymayan uygulamaları analiz edebilmektir. Öncelikle uygulamalar, statik kod analizinden geçirilerek, istenilen izinleri manifest dosyasından alan ve kodunda kullanan uygulamalar filtrelenir. Kodunda kullanmasını kontrol etmek için, izin-API, url-API eşleşmesi yapan Pscout [24] çalışmasından faydalanılmıştır. İlgili izini kullanan API'ler çıkarıldıktan sonra, ayrıca bu API'lerin çağrıldığına ilişkin bir kontrol daha yapılmaktadır. Bu sistem, çağrı ağacından derin öncelikli arama yaparak, programın giriş noktalarından en az birinden bu API'lere bir yol olup olmadığına bakar. Bu yolun olması, çalışma anında bu API çağrısının yapılacağını garantilemese de, bir fikir vermektedir. Bu çalışmada ilgilenilen izinler ve bu izinlerin kullanıldığı uygulamalar çıkarıldıktan sonra anahtar sözcük tabanlı metin analizine başlanır. Genel metin önişleme işlemleri yapıldıktan sora, bu alana ilişkin durak kelimeler tanımdan çıkarılır. Bu kelimelerin bulunması için Kullback-Leibler uzaklaşma ölçütünü kullanarak terim tabanlı örnekleme yapan bir yöntem kullanılmaktadır. Sonrasında, uygulama tanımları, kalan kelimeler ile bir vektör olarak gösterilir. Bu aşamada, birbirine çok yakın tanımlardan (cosine benzerlik ölçütüne göre) sadece biri veri setinde yer alır. Anahtar sözcükler, uygulama tanımlarından oluşan bu veri setindeki sıklıklarına göre belirlenen ilişkili ağırlıklarına (relevance weight) göre seçilirler. Burada, her izin için en iyi 3 (bu sayı deneysel olarak bulunmuştur) anahtar sözcüğün tanımda geçmesi ile, izin-tanım eşleştirmesi yapılır. Önerilen yaklaşımın, Whyper ile karşılaştırılabilir sonuçlar ürettiği görülmüştür. Ayrıca bu yaklaşım, hem İngilizce, hem de Çince tanımlar için önerilmiştir. Bu çalışmanın en büyük katkılarından birisi, çok sayıda uygulamayı inceleyerek, kullanılan ama tanımda geçmeyen izinlerin nedenlerini analiz etmeleridir. Bu analiz sonucu, dört temel neden ortaya çıkmıştır: uygulama oluşturucu çatıların gereğinden fazla izin istemesi, uygulamanın ikincil ya da opsiyonel işlevselliklerine ilişkin izinlerin tanımda belirtilmemesi, üretken geliştiriciler tarafından kötü amaçlı olarak ya da sonra kullanılmak amaçlı alınan izinlerin tanımda belirtilmemesi ve üçüncü parti kütüphanelerin kullanılması. Özellikle son neden, uygulama geliştiricilerin, kullandıkları kütüphanelerine ilişkin izinleri tanımda belirtmediklerini göstermekte ve bu proje kapsamında üretilecek olan tanımların bu probleme çözüm getireceği düşünülmektedir. Ayrıca bu çalışma, kullanıcıların etkileşimde bulunduğu izinlerin, tanımlarda çoğunlukla geçtiğini gözlemlemiştir.

NLP tabanlı yaklaşımlar, son yıllarda yaygınlaşmıştır. Bunlardan biri olan CHABADA [9], uygulama davranışlarının, tanımlarına uyup uymadığını bulmaya çalışmıştır. Bunun için öncelikle, Latent Dirichlet Allocation (LDA) yöntemi ile uygulama tanımlarına bakarak, her bir uygulamanın temel konusu bulunur. Daha sonra ilişkili konulu uygulamalar kümelendirilir (clustering). Bu kümelenmiş uygulamaların kullandığı hassas API (sensitive API)'ler üzerinden, bu gruplara uymayan uygulamalar bulunarak, tanımından farklı davranan uygulamalar bulunmaya

çalışılır. Burada özellikle konu bulma aşamasının, sistem üzerindeki başarısı görülmektedir. Benzer bir çalışma, uygulama tanımlarından çıkarılan konu ve hassas API'leri öznitelik olarak kullanarak, zararlı ve zararsız yazılımları ayırmayı hedeflemiştir. Zararsız uygulamaları da eğitime katarak, CHABADA [9]'dan daha iyi başarım sergilediklerini söylemişlerdir.

Son yıllarda, farklı amaçlarla kulllanıcı yorumlarını kullanan az sayıda yaklaşım önerilmiştir. Bunlardan güvenlik açısından, bizim proje konumuz ile en ilişkili olan Auoreb [8] çalışmasından bahsedeceğiz. Bu çalışmada, kullanıcı yorumları, güvenlik ile ilgili 4 kategoriye ayırılmaya çalışılır. Bu kategoriler, veri sızıntısı, ayrıcalıklı izin kullanımı, finansal sorunlar ve spam olarak belirlenmiş, ileride daha fazla kategoriye bölümlendirme yapılabileceği öngörülmüştür. Burada öncelikle, ilgili kategoriler ile ilgili öznitelikler belirlenmiş ve kullanıcı yorumlarının farklı kelime ve ifadeler ile gösterimi olabileceğinden öznitelik arttırımı yapılmıştır. Sparse SVM ile manuel olarak etiketlenmiş kullanıcı yorumlarının sınıflandırılması için bir model eğitilmiştir. Bu model, %94.05 doğruluk sağlamıştır. Anlamsal öznitelikler kullanılarak eğitilen model, kelime tabanlı sistemle karşılaştırıldığında büyük bir fark yarattığı gösterilmiştir. Tüm kullanıcı yorumlarının değerlendirildiği, kalabalık kaynaklı (crowdsourcing) bir sistem ile önerilen yaklaşım genişletilmiştir. Burada çoğunluk tabanlı (majority voting) değil, kullanıcıların güven değerlerine göre yorumlarının ağırlıklandırıldığı two-coin modeli kullanılmıştır. Bu çalışmada karşılaşılan en büyük güçlüklerden birisi, kullanıcı yorumlarının etiketlenmesi olarak belirtilmiştir. Üç uzman, iki ay boyunca bu etiketleme üzerine çalışmışlardır. Bu çalışma, kullanıcı yorumlarını güvenlik ile ilgili kategoriye ayırmaya çalışırken, biz kullanıcı yorumlarında aranılan izin ile ilgili bir yorum geçip geçmediğini ve bu yorumun pozitif/negatif ayırımının yapılmasını amaçlamaktayız.

Probleme farklı açıdan yaklaşan çalışmalar da bulunmaktadır. ASPG [11], tanımdan izinlere ilişkin bilgileri çıkarmaktadır. Bunun için anlamsal tabanlı bir analiz yaptığını söylese de, daha çok anahtar kelime tabanlı bir çalışmadır. İzinler ve bu izinleri ifade etmek için gerekli sözcükler çıkarılır. Tanımda bu sözcüklere ilişkin negatif bir yorum varsa, bu durum negatif kelimeler (not, no, cannot, vb. gibi) içeren cümlelerin filtrelenmesi şeklinde basitçe ele alınmıştır. Bu yöntem ile, tanımda belirtilmeyen izinler çıkarılır. Önerilen çalışma, ufak bir uygulama kümesi üzerinde test edilmiştir. Ayrıca, bu uygulamalardan gereksiz olduğu düşünülen bazı izinler çıkarıldığında uygulamalar çalıştırılamamıştır. Bu nedenle, hala insan etkileşimi gerektiren bir sistemdir. Anlamsal bir analiz yapmadığından gerçek dünya problemlerinde başarımının düşük olacağı beklenmektedir. Ayrıca, bu sistemin başarımı, uygulama tanımlarının kapsamlı olmasına bağlıdır.

Ayrıca, son yıllarda güvenlik amaçlı NLP tabanlı yöntemleri uygulayan birkaç çalışma önerilmiştir. Bunlardan bir tanesi, mobil uygulamalar için gri yazılımların sınıflandırıldığı çalışmadır [49]. Gri yazılımlar, zararlı yazılımlar gibi kötü niyetle yazılmayan, fakat beklenmedik (bilgi sızdırma, ekran görüntüsü vs. değiştirerek kullanıcıyı kızdırma, vb. gibi) davranışlar sergileyen uygulamalardır. Bu uygulamaların varlığı, zararlı yazılım analizi açısından zorluk yaratmaktadır. Bu çalışma, gri uygulamaları mobil açısından tanımlayan ve bir gri yazılım veri kümesi tanıtan ilk çalışma olması nedeniyle önemli bir katkı sunmaktadır. Ayrıca, bu yazılımların tespitinin, bilinen statik analiz teknikleri ile mümkün olmadığı belirtilmiştir. Bu nedenle, bu yazılımların tespiti için metin analizi tabanlı yöntemler önerilmiştir. Burada önerilen teknikler çok basit olmakla birlikte, çoğu saldırının tespitinin doğrulanması için manuel bir adım da gerektirmektedir. Bu proje kapsamında önerilen yöntemlerin, Yaygın Etki Tablosu'nda belirtildiği gibi, gri yazılımların tespiti için kolaylıkla kullanılabileceği, kod analizi ile birlikte genişletilebileceği düşünülmektedir. Son yıllarda, üstveri analizi, statik ve dinamik analizi tamamlayan bir yöntem olarak karşımıza çıkmaktadır. Üstveri kullanarak, makine öğrenmesi yöntemleri ile zararlı yazılım sınıflandırması yapan yakın zamanlı çalışmalar bulunmaktadır [50][51]. Ayrıca, üstverilerden faydalanılarak şüpheli uygulamaları, detaylı yazılım analizi için filtrelemeyi öneren yaklaşımlar da bulunmaktadır [59]. Bu çalışmalar da, üstveri kullanımının güvenlik alanına uygulanma potansiyelini göstermektedir.

Yukarıda, son yıllarda NLP tabanlı çalışmaların, özellikle Android alanındaki bazı önemli uygulamalarına değinilmiştir. Biz bu uygulamalardan özellikle izin-tanım eşleştirmesi üzerine yoğunlaşmaktayız. Burada yapılan üç önemli çalışma bulunmaktadır : Whyper [5], Autocog [6] ve ACODE [7]. Biz bu çalışmalardan farklı olarak, bu proje kapsamında yeni bir yöntem öneriyoruz. Bu projede, çok daha geniş bir veri kümesi kullanmayı amaçlıyoruz. İlk aşamada kullanacağımız veri kümesi, Wikipedia içeriklerinden elde edilmiş ve 220K sözcük türünü kapsayan word2vec veri kümesi olacaktır. (http://people.csail.mit.edu/karthikn/data/vectors_wiki.normalized.txt). Sonraki aşamalarda, veri kümesini uygulama tanımları ve API dökümanları ile genişleterek alana özgü sözcükleri de içeren bir veri kümesi elde etmeyi amaçlıyoruz. Alan spesifik word2vec oluşturulması, son yıllarda birçok alan için uygulanmıştır [54]. Bu listede önerilen App2Vec [55] çalışması, bizim yaklaşımımızdan oldukça farklıdır. Bu çalışmada, kullanıcıların uygulamaları kullanım alışkanlıkları vektörize edilerek, uygulama önerisi için bir öneri yapılmıştır. Biz ise uygulamaların, uygulamalardaki kelimelerin vektörel gösterimlerini üretmeyi, üstveri kullanan ve uygulamaların anlamsal ilişkileri üzerinde çalışacak araştırmacılar için de kullanılabilecek bir vec önermekteyiz. mob-app2vec ismini verdiğimiz bu küme, projenin yenilikçi yönlerinden ve önemli katkılarından birisi olacaktır. Bu veri kümesi ile uygulama dokümanlarındaki sözcüklerin anlamını bulmayı hedefliyor ve her bir sözcüğün anlamını, bu büyük veri kümelerinden faydalanarak derin öğrenme yöntemleri ile çok boyutlu uzayda boyutunu 100-200 gibi daha küçük boyutlarda göstermeyi planlıyoruz. Böylece anlamla ilgili daha fazla veriyi daha sıkıştırılmış bir gösterimle ve daha zengin bir biçimde ifade edebileceğimizi düşünüyoruz. Literatürdeki en iyi başarım, Autocog [6]'a aittir. Diğer yaklaşımların başarımı, basit anahtar sözcük tabanlı bir yöntem kullanması (ACODE [7]), ya da kısıtlı bir sözlük kullanmasından (Whyper[5]) dolayı daha düşüktür. Bu çalışmada kullanılan Belirgin Anlam Analizi (ESA) yöntemi de, Wikipedia metinlerini kullanarak anlam analizi yapmaya çalışmaktadır. Ancak ESA [52], 2007

yılında geliştirilmiş ve performansı nedeniyle derin öğrenmeye göre çok daha küçük veri kümelerinde çalışmaktadır. Çünkü TF-IDF matrisinde her bir sözcük bir vektör olarak ifade edilmektedir. Metin boyutu büyüdükçe matrisin boyutu da büyüdüğünden çok büyük verileri işlemek oldukça zor olmaktadır. Derin öğrenme ile ise çok daha büyük veri kümelerinin boyutları etkin bir şekilde küçültülerek daha ufak boyutlu uzaylarda ifade edilebildiği için hem daha fazla veri kullanılabilmekte, hem de bu, performans yitimine neden olmamaktadır. Ayrıca, biz bu çalışmalardan farklı olarak, izin-tanım eşleştirmesini özellikle tehlikeli izinler için yapmayı planlıyoruz. Böylelikle, zararlı yazılım analizi için statik ya da dinamik analizinden önce inceleme yapacak bir araç geliştirmeyi hedefliyoruz. Her izin için izin-tanım eşleştirmesi gerekli olmayabilirken (her izinin tanımda yer alması gerekmiyorken), tehlikeli izinlerin tanımda yer alması, tanımda yer almıyor ise bile uygulamayı kullanan kullanıcılar tarafından farkedilip kullanıcı yorumlarına yansımasını bekliyoruz. Bu amaçla, Whyper, Autocog ve ACODE'dan farklı olarak, kullanıcı yorumları da bu sistemde değerlendirilecektir.

Tanımı kısa olan bir uygulama için yukarıda bahsedilen çalışmaların başarımı düşmektedir. Ayrıca, kısa tanımlı uygulamalar, uygulamaların kategorilendirilmesi, vb. gibi sistemlerin başarımını da düşürmektedir [49]. Buna ek olarak, izin-tanım eşleştirmesinde oluşan büyük boşluğun, 3. parti kütüphanelerden kaynaklı olduğu düşünülmektedir [3]. Literatürde, bazı uygulamaların 20'den fazla 3. parti kütüphane kullanabildiği [1], uygulamanın kodunun ortalamada %60'dan fazlasının bu kütüphanalerden oluştuğu [2] gösterilmiştir. Bu da bizi, projemizin ikinci önemli amaçlarından birisi olan, otomatik olarak tanım yaratmaya getirmektedir. Bildiğimiz kadarıyla, literatürde benzer sadece bir çalışma bulunmaktadır. Bu çalışma [15] kod analizi yaparak Android uygulamaların güvenlik temelli tanımlarını otomatik olarak çıkarmaktadır. Öncelikle, statik olarak programların güvenlik davranış çizgeleri çıkarılmaktadır. Burada özellikle, programın giriş noktaları, API çağrı ve bağımlılıkları, koşulları (bir kodu tetikleyen durumlar) ve bazı sabitler üzerinden güvenlik temelli davranışları çıkarılmaktadır. Burada, tamamıyla bütün bir çizge çıkarmak ve tüm koşulları çözmek mümkün olmamakla birlikte, çok büyük bir çizge elde edilmektedir. Sonrasında bu çizge, çizge madenciliği ile sıkıştırılmaktadır. Son olarak, bu çizgeler, NLG yöntemleri ile tanımlara dönüştürülmektedir. Yapılan anket ve deneyler, kullanıcıların anlayabileceği tanımların otomatik olarak yaratılabildiğini göstermiştir. Bir uygulamanın işlenip tanımının yaratılması için gereken ortalama sürenin 25 dk. olduğu belirtilmiştir. Bu çalışma, özellikle güvenlik temelli bir tanım yarattığından, uygulamanın davranışını çıkarmak için (veri akışı, kontrol akışı, hassas API çağrıları için koşulların çıkarılması, vb. gibi) ayrıntılı bir analiz yapmakta, bu da zaman olarak bir maliyete neden olmaktadır. Bizim amacımız ise, daha çok 3. parti kütüphanelerin kullanımından dolayı izin-tanım eşleştirmesini zorlaştıran ya da kısa tanımlı uygulamalara, ek tanımların eklenmesidir. Bunu yaparken, kod üzerinde yapılacak bir ön analizden sonra, kütüphane dökümantasyonları kullanılarak tanımların oluşturulması hedeflenmektedir. Ayrıca, ilgili kütüphane versiyonlarının zayıflıklarına yönelik olarak, kullanıcılara karşılaşabilecekleri güvenlik risklerine ilişkin ek bilgiler verilebilecektir.

Bildiğimiz kadarıyla, bu proje kapsamına en yakın çalışma, yukarıda bahsettiğimiz güvenlik temelli tanımları yaratan çalışmadır [15]. Fakat, bu literatür özetinde, bize ilhan veren, konu ile ilgili diğer bazı benzer çalışmalardan da bahsedilecektir. Çok yakın zamanda, izinlerin ne amaçlı kullanıldığına ilişkin bir çalışma önerilmiştir [16]. Burada amaç, kullanıcının sadece kullanılan izine göre değil, bunun ne amaçla kullanıldığına göre bir karar verebilmesidir. Bu nedenle, bir izin (contacts) kullanım amaçlarına göre sınıflara (e-posta, arkadaşların bulunması, istenmeyen çağrıların bloklanması, vb. gibi) ayrılmıştır. Bu sınıflandırmaya göre, uygulamalar manuel olarak etiketlenmiştir. Burada çalışma, sadece uygulama yazarı tarafından geliştirilen kod üzerinde yoğunlaşmaktadır. Bu kod üzerinden iki farklı türde öznitelikler çıkarılmıştır : uygulama tabanlı ve metin tabanlı. Uygulama tabanlı özniteliklerde, bir izine ilişkin API çağrı sıklığı gibi bilgiler çıkarılır. Metin tabanlı öznitelikler ise, paket, sınıf, metod, değişken isimlerinden anahtar sözcüklerin çıkarılması ile elde edilir. Çıkarılan her bir kelimeye, TF-IDF ile bir skor verilir. Bu öznitelikler kullanılarak, C4.5 karar ağacı algoritması ile bir sınıflandırıcı modeli yaratılır. Sistem, %85 oranında bir doğruluk sağlar. Metin tabanlı özniteliklerin, daha önemli olduğu gösterilmiştir. Bu çalışma, bir tanım yaratmamakta, uygulamadaki izinleri, kullanım amaçlarına göre sınıflandırmaktadır. Her bir izin için bu sınıflandırmanın manuel olarak yapılması gereklidir. Ayrıca, uygulama geliştiricilerin, uygulamanın paket, sınıf, metod ve değişkenlerini, anlamlı bir şekilde isimlendireceğine dayanmaktadır. Özellikle, zararlı yazılımlarda bu durum sağlanmayabilir, diğer bir yandan anlamsız isimlerin kullanılması zararlı yazılımın farkedilmesine neden olabilir.

Yakın zamanda, zararlı yazılımların davranışlarının otomatik olarak çıkarılmasına ilişkin bir çalışma sunulmuştur [12]. Bu çalışma, Android tabanlı zararlı yazılımlar hakkındaki literatürde varolan makaleleri veri kaynağı olacak şekilde kullanarak, NLP teknikleriyle zararlı yazılımlara özgü öznitelikleri toplamaktadır. 1068 akademik makaleden, otomatik olarak 195 öznitelik elde edilmiştir. Bu öznitelikler, literatüre bir zararlı yazılım veri kümesi ve manuel olarak tespit edilmiş 545.334 öznitelik kullanan, statik analiz tabanlı bir tespit sistemi öneren Drebin [13] çalışması ile karşılaştırılmıştır. FeatureSmith çalışması [12], Drebin'den çok daha az sayıda öznitelik içermesine rağmen, benzer bir başarım göstermektedir. Ayrıca, Drebin'in tespit edemediği bazı zararlı yazılım ailelerini (Gappusin) tespit edebilmektedir. Bu sonuçlar, NLP yöntemlerinin otomatik olarak seçtiği öznitelikler ile başarılı bir sınıflandırma yapılabileceğini göstermiştir.

Zararlı yazılımlar için benzer diğer bir çalışma [14] uygulamalardaki istenmeyen davranışların açıklamalarını otomatik oluşturan bir çalışmadır. Güvenlik uzmanları tarafından daha önceden tespit edilen zararlı yazılımlara ait görüşleri veri kaynağı olarak kullanıp, yeni tespit edilen benzer zararlı yazılımlara ait açıklamaları oluşturmak için makine öğrenmesi ve metin işleme yöntemlerini birlikte kullanmıştır. Öncelikle zararlı yazılımları, zararsızlardan ayırmak için izinler, API çağrıları ve aksiyonlardan oluşan bir öznitelik kümesi hazırlanmıştır. Zararlı yazılım tespiti için değişik makine öğrenmesi teknikleri denenmiştir. Buradaki amaç, sadece en iyi başarımı elde eden algoritmayı

bulmak değil, daha az sayıda öznitelik ile bunu başaranı bulmaktır. Bu öznitelikler ayırdetme özelliklerine göre sıralanır ve ilk n tanesi seçilir. Bu n sözcük ile uygulama tanımının ne kadar eşleştiği kontrol edilmiştir. Sonrasında, bu özniteliklere ilişkin bazı anahtar kelimeler Android dökümanlarından elde edilmiştir. Bu anahtar kelimelerin derecelendirilmesi için TF-IDF kullanılmıştır. Bu anahtar sözcükler ile uygulama tanımının eşleşmesine bakıldığında, doğruluğun iki kat arttığı gösterilmiştir. Son olarak, bu anahtar sözcüklere en yakın cümle ile, manuel oluşturulmuş zararlı yazılım dökümantasyonlarından elde edilen, yeni bulunan zararlı yazılımın davranışı için bir tanım oluşturulmuştur. Bu tanımların, ne kadar kapsamlı ve doğru olduğuna ilişkin bir deney yapılmamıştır, ileriki çalışmalara bırakılmıştır. Ayrıca, tek bir öznitelik ile açıklanamayan bazı zararlı yazılımlara ilişkin davranışların (root erişimi kazanma, DoS saldırısı yapma, gelen mesajları alma, vb. gibi) bu yaklaşım ile açıklanamayacağı belirtilmiştir.

Biz bu çalışmada 3. parti kütüphanelere ilişkin dökümantasyonlardan ve API tanımlarından, uygulamalara ilişkin tanımları otomatik olarak oluşturmayı amaçlıyoruz. Bildiğimiz kadarıyla, bu amaçla yapılmış başka bir çalışma yoktur. Fakat, literatürde API tanımlarının farklı amaçlar ile kullanılabildiğini gördük. Bunlardan önemli olanlarından bazıları, API tanımlarından kaynak [17] ya da metod spesifikasyonlarının [18] çıkarılması, basit kod modellerinin oluşturulması [19], vb. gibidir. Bu proje kapsamında yapılacak olan çalışma, bu uygulama alanlarından farklıdır.