HAKEMLİ Denetişim
Ortak Aklın Harmanı
Kamu İç Denetçileri Derneği Meşrutiyet Caddesi Konur Sokak No: 36/6 Kızılay - ANKARA
www.kidder.org.tr/denetisim/ • denetisim@kidder.org.tr
ISSN 1308-8335
Yıl: 14, Sayı: 28, 53-74, 2023

Araştırma Makalesi

ÜÇÜNCÜ TARAF YAZILIM BİLEŞENLERİNDEN KAYNAKLANAN ZAYIFLIKLARIN TESPİTİ VE YÖNETİMİNE İLİŞKİN BİR UYGULAMA

(A PRACTICE FOR DETECTION AND MANAGEMENT OF WEAKNESSES FROM THIRD-PARTY SOFTWARE COMPONENTS)

Önder ŞAHİNASLAN[1], Ender ŞAHİNASLAN[2], Erdi KÜÇÜKALİ[3]

ÖZ

Üçüncü taraf yazılım bileşenlerinin kullanımı her geçen gün artmaktadır. Bu yazılım bileşenleri, şirketlerin hızlı, esnek ve pratik çözümler geliştirmesine yardımcı olurken, birçok güvenlik açığını da bünyesinde barındırabilmektedir. Üçüncü taraf yazılımlara olan talebin artmasına rağmen bu yazılımların kapalı kaynak kod içermesi sebebiyle güvenlik kontrol ve testleri yeterince yapılamamaktadır. Yeterli güvenlik testleri ve kontrolleri yapılmadan bu yazılımların sistemlere entegre edilmesi büyük risk oluşturmaktadır. Bunun sonucunda kurumlar telafisi mümkün olmayan kayıplarla karşı karşıya kalabilmektedir. Bu risk ve kayıpların önüne geçebilmek için üçüncü taraf yazılımların güncel tutulması ve güvenlik açıklarının hızlı bir şekilde tespit edilmesi büyük önem taşımaktadır. Bu kurumların aşması gereken önemli bir sorundur. Bu çalışmada yazılım bileşenlerinden kaynaklanan zafiyetlerin tespiti, analizi, yönetimi, kontrolü, raporlanması ve kurumsal yazılımlarla entegrasyonu ele alınmıştır. Black Duck güvenlik açığı kontrol aracı üzerinde uygulamalı çalışmalar yapılmıştır. Sonuç olarak, uygulama güvenlik araçlarının kullanılması denetim ve kontrol faaliyetlerinin daha hızlı, daha şeffaf ve güvenilir sonuçlara ulaşmasına katkı sağlamaktadır ve uygulama açıklarından kaynaklanan güvenlik risklerinin etkin yönetimine, kontrolüne ve denetimine yardımcı olmaktadır. Aynı zamanda güvenlik ve uyumluluğa da katkıda bulunur, kaynakların etkin kullanımı ile işletmelere çeviklik, verimlilik, karlılık ve rekabet gibi alanlarda avantajlar sunmaktadır. Bu çalışma bilgi teknolojileri, iç kontrol, risk ve denetim profesyonelleri için bir rehber niteliği taşımakta ve literatüre katkı sağlamaktadır.

Anahtar Kelimeler: Teknoloji ve Yenilik, Üçüncü Taraf Yazılım, Zafiyet, Uygulama Güvenliği, Denetim ve Kontrol.

JEL Kodları: M15, C88, L86, M42

ABSTRACT

The use of third-party software components is increasing day by day. While these software components help companies develop fast, flexible and practical solutions, they can also contain many security vulnerabilities. Despite the increasing demand for third-party software, security controls and tests cannot be done adequately due to the closed source code of these software. Integrating these software into systems without adequate security tests and controls poses a great risk. As a result, institutions may face irreparable losses. In order to prevent these risks and losses, it is of great importance to keep third-party software up-to-date and to detect security vulnerabilities quickly. This is an important problem that institutions must overcome. In this study, the detection, analysis, management, control, reporting and integration with corporate software of vulnerabilities arising from software components are discussed. Hands-on work has been done on the Black Duck vulnerability check tool. As a result, the use of application security tools contributes to faster, more transparent and reliable results of audit and control activities and helps to effectively manage, control and audit security risks arising from application vulnerabilities. It also contributes to security and

[1]Dr. Öğr. Üyesi, Maltepe Üniversitesi, Bilişim Bölümü, Orcid Id:0000-0003-2695-5078, ondersahinaslan@maltepe.edu.tr, Sorumlu Yazar

[2]Dr. Öğr. Üyesi, Mudanya Üniversitesi, Bilgisayar Mühendisliği, Orcid Id:0000-0001-8519-7612, ender.sahinaslan@mudanya.edu.tr

[3] Yüksek Lisans, Maltepe Üniversitesi, Bilgisayar Mühendisliği, Orcid Id:0000-0001-7803-1163, erdikucukali@gmail.com

compliance, and offers advantages to businesses in areas such as agility, efficiency, profitability and competition with the effective use of resources. This study serves as a guide for information technology, internal control, risk and audit professionals and contributes to the literature.

Keywords: *Technology and Innovation, 3rd Party Software, Vulnerability, Application Security, Audit and Control.*

JEL Classification: *M15, C88, L86, M42*

1. GİRİŞ

Dijital servis ve yazılım uygulamalarına olan talep her geçen gün artmaktadır. Teknolojik yenilikler, ilerlemeler ve kullanım alanlarının yaygınlaşması bu talep artışına sebep olan ana unsurlardır. Bilgi teknolojisi ekipleri, bu kadar yüksek bir yazılım talebini karşılamak için var gücüyle çalışmaktadır. Ancak dağıtık yapıda farklı tür ve özelliklere sahip teknolojik yapılarda bu talepleri zamanında eksiksiz bir şekilde karşılamak her zaman mümkün olamamaktadır. Bu durum insan kaynaklarının yetersizliği, güncel bilgi ve deneyime ihtiyaç duyulması gibi birçok nedene bağlıdır. Ayrıca her kurum kendi bünyesinde yazılım geliştirme ekibi bulundurmak zorunda değildir. Diğer taraftan yazılım geliştirme alanında kullanılan teknoloji ve uygulamalarda sürekli yeni gelişmeler yaşanmaktadır. Farklı sistemlere özgü farklı çözüm önerileri sunulmaktadır. Kaliteli ve verimli bir yazılım geliştirmede, en uygun yazılım geliştirme araç ve yönteminin kullanılması dikkate alınması gereken bir zorunluluktur (Keskinkılıç & Özmen, 2018). Kurumların bu yazılım ihtiyaçlarının zamanında ve tek elden karşılanması genelde zor ve maliyetli bir yaklaşımdır. Yazılım geliştirmedeki yüksek maliyet ve teknik güçlükler, teknolojik yenilikler ve hızlı çözüm talepleri üçüncü taraf uygulamalara olan talebi artırmaktadır (Şahinaslan vd., 2023). Kurumlar yazılım ekiplerine sahip olsunlar ya da olmasınlar üçüncü taraf yazılım bileşenlerini bir şekilde kullanmaktadırlar. Bu yaklaşım bir takım sorunların çözümüne yardımcı olmakla birlikte bu ürünlerden kaynaklı güvenlik başta olmak üzere başka sorun ve risklerin oluşmasına sebebiyet verebilmektedir. Temel faaliyetlerinde bilgi teknolojilerine dayalı sistemleri kullanan kuruluşlar, bu sistemlere yönelik risklerin kapsam ve doğasının bilincinde olmalıdır (Arslan & Özbilger, 2022). Günümüzde yazılımdan kaynaklı bir risk aynı zamanda bir güvenlik riskidir. Üçüncü taraf ürünlerin edinim ve kurulumundan önce bu ürünlerden kaynaklı herhangi bir güvenlik riskine sebep olabilecek bir zafiyet veya açıklığının olup olmadığı sıkı bir şekilde kontrol edilmelidir. Yazılıma bağlı güvenlik açıklığı yazılımın geliştirme veya yapılandırması esnasında oluşan, ortaya çıkması durumunda ise kurum güvenlik politikasının ihlaline neden olan unsurlardır (Ozment, 2007). Hızla gelişen teknolojilerin kontrolsüz kullanımından kaynaklı bir zafiyet güvenlik, uyum ve risk sorunlarına sebep olur (Sahinaslan, 2019). Yazılımlardaki güvenlik sorunu kurumlar için büyük bir tehdittir. Bu sorun, edinilen bir yazılım uygulamasından kaynaklanabileceği gibi bu uygulamanın mevcut uygulama ve sistemlerle bütünleştirilmesinden kaynaklı yeni bir problem olarak da ortaya çıkabilir. Yazılım güvenliği aynı zamanda iş süreçleri sürekliliğinin sağlanmasında da önem arz etmektedir (Şahinaslan vd., 2011). Tüm bu sorun, zafiyet ve bunu kullanan tehditlerin bir zarara dönüşmesini önlemek için bu zafiyet ve sorunların önceden tespit edilerek gerekli önlemlerin alınması şarttır (Sahinaslan & Sahinaslan, 2019). Bu sorunların tespit edilmesi amacıyla gerçekleştirilecek test ve kontrollerde bir riskin söz konusu olması durumunda risk önceliklendirme ve yönetimi etkin bir şekilde gerçekleştirilmeli, gerekli önlem ve iyileştirmeleri sağlanmalıdır. Aksi durumda telafisi güç maddi, manevi pek çok kayıp yaşatacağı bir gerçektir. Üçüncü taraf çözümlerin devreye alınmasından önce bir zafiyet unsuru olup olmadığı kontrol edilmelidir. Bu yazılım bileşenlerinin zafiyetlerinin kontrolünde başvurulabilecek güvenilir zafiyet veri tabanları vardır. Ortak güvenlik zafiyetleri (CVE, 2022), Exploit Database (EDB, 2022), Rapid 7 zafiyet ve tehdit veritabanı (Rapid, 2022), Amerika ulusal güvenlik zafiyet veri tabanı (NVD, 2022), Çin ulusal zafiyet veri tabanı (CNVD, 2022) ve Japonya güvenlik açığı (JVN, 2022) yaygın kullanılan ve bilinen zafiyet veri tabanlarıdır. Diğer taraftan yazılım güvenlik zafiyetleri uzman kişiler tarafından el yordamıyla tespit edilmeye çalışılarak sınıflandırma ve puanlandırma yapılmaktadır (Kekül vd., 2021). Günümüzde kurumsal firmalarda üçüncü taraf her bir bileşen ve bu bileşenlerin sürümlerine ait her bir zafiyetin tespiti, kontrolü ve yönetiminin herhangi bir yardımcı uygulama olmadan etkin bir şekilde yapılabilmesi oldukça zordur.

Literatürde yer alan çalışmalara bakıldığında, açık kaynak kodlu üçüncü taraf yazılımların yaygın şekilde kullanımı saldırganların ilgisini çeker, güvenlik açıklarından kaynaklı potansiyel riskleri artırır. Büyük yazılım organizasyonlarında otomatik kod analiz programları mevcut olsa bile güvenlik analistleri genellikle savunmasız açık kaynak kodlu yazılım araçlarının kullanımına ilişkin genel bir bakış elde etmeden yoksundur (Dennig vd., 2021). Üçüncü taraf bileşenler üzerinde bilinen güvenlik zafiyetlerinden kaynaklı risklerin varlığı saldırganlar için önemli bir anahtardır. Bu risklere rağmen yazılım geliştiricileri halen herkesin kullanımına açık havuzlarda yer alan savunmasız yazılım bileşenlerini kullanmaktadır (Cadariu vd., 2015). Bu risklerin önüne geçebilmek adına yazılım projelerinde bilinen güvenlik açıklarını izlemek için süreç tabanlı bir güvenlik açığı uyarı sistemine ihtiyaç vardır. Bu amaçla bilgi ve sistem altyapısında konumlandırılan varlıklar üzerinde oluşabilecek zafiyetlerin tespit edilmesine yönelik sızma test yöntemleri incelenmiş ve buna dair örnek bir kılavuz oluşturulmuştur. Bilgi sistem mimarisinde yer alan varlıkların listesi, bu varlıklara ait zafiyetlerin ortak güvenlik zafiyet veri tabanlarında yayınlanan açıklıkların kontrol ve takibinin yapıldığı, bunlara karşı alınacak önlemlerin bir arada tanımlanıp yönetildiği bir siber önlem sistemi geliştirmiştir (Şahinaslan, 2013). Günümüzde keşfedilen güvenlik açıklarının sayısı artmaktadır, bunları izlemek ve özellikleri hakkında ayrıntılı bilgi sahibi olmak önemlidir. Kamuya açık veri tabanlarında yayınlanan bilgiler, satıcılar olarak adlandırılan yazılım üreticileri tarafından

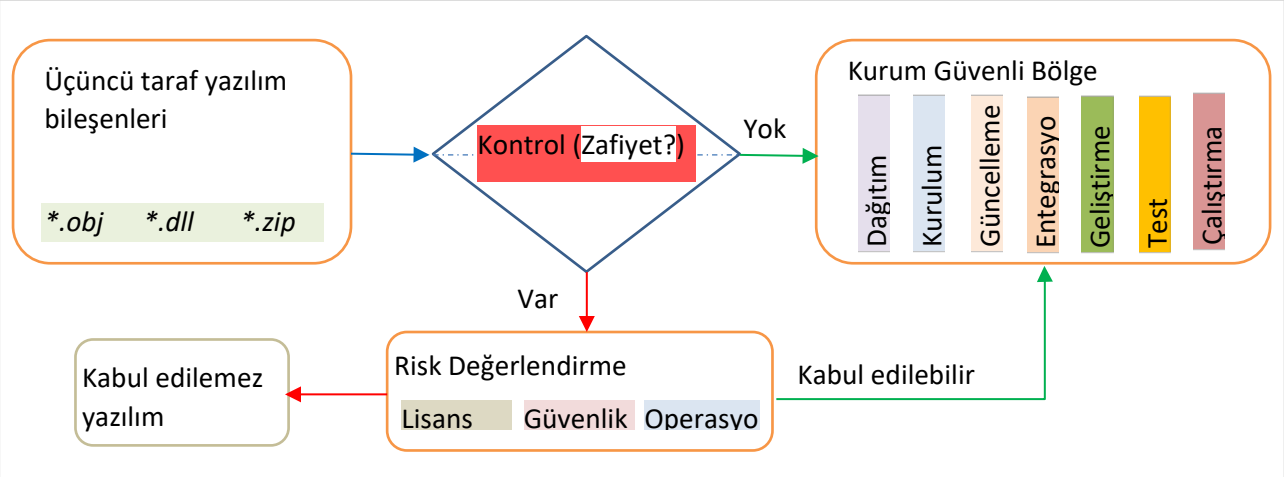
sunulan kaynaklardan toplanabilir (Rebrošová, 2018). Bu çalışma güvenlik açıkları hakkında bilgi vermenin yanında güvenlik açıklık verilerinin toplandığı (NVD, 2022) zafiyet veri tabanı verileri ile yazılım sağlayıcılar tarafından sunulan zafiyetleri karşılaştıran bir analize de yer verilmiştir. Cobleigh ve arkadaşları çalışmalarında, yazılımdaki güvenlik açıklarını belirleme, önceliklendirme ve değerlendirme için bir araç sunmuşlardır. Bu araç ile üçüncü taraf yazılım bileşenlerinin bakımını daha verimli ve sağlam hale getirerek ürünlerde güvenliği artırmak amaçlanmıştır. Yazılım bileşeni yayın sürümleri ile açık kaynaklardan gelen güvenlik açığı bilgilerini eşleştirme sonuçlarının zamana bağlı değişiminin izlenebildiği vurgulanmıştır (Cobleigh vd., 2018). Nesnelerin interneti (IoT)'nin benimsenmesi önündeki en büyük engelin güvenlik kaygısı olduğu, bunu ağa bağlı cihaz ve bildirilen güvenlik zafiyetlerinin artışının tetiklediği görülmüştür. Yazılım güvenliği için üçüncü taraf bileşenlerindeki güvenlik açıklarının ele alındığı, 21 uygulama üzerinde, sektörde görüşmeler sonucunda toplam altı yetenek alanında olgunlaşan değerlendirmelerin sunulduğu, üreticiden temin edilen yazılımların güçlü ve zayıf yönlerinin değerlendirilmesinde yararlı olan bir model sunulmaktadır (Nikbakht Bideh vd., 2018).

Mobil cihazlar için geliştirilen Android uygulamalarının hızlı biçimde geliştirilmesinde üçüncü taraf yazılım kütüphanelerinin kullanımı yaygındır. Bu yazılım paketlerinin içerisine gizlenebilen kötü amaçlı yazılım kodların tespit edilebilmesinde çeşitli zorluklar vardır. Bu sorunu çözmek amacıyla hassas uygulamaların içinde kullanılan üçüncü taraf kütüphane sürümlerini kesin olarak belirleyebilen ve güvenlik açıkları ile bu kütüphane paketleri hakkında ayrıntılı bilgi sağlayan ATVHunter adlı bir sistem önerilmektedir (Zhan vd., 2021). Günümüzde ticari yazılımlar çok fazla olup kamu ve özel sektör tarafından yaygın olarak kullanılmaktadır. Bu yazılımlara ait güvenlik açıklarının tespit edilmesi ve raporlanması için yama üretimi, yama yönetimi gibi faaliyetler önem kazanmaktadır. Zafiyet açıklıklarına ilişkin bilgilerin derlenmesi, tasnifi ve bir veri tabanı üzerinde belli bir düzende tutularak kullanıma sunulması, siber güvenliği artırmakta ve açıkların istismarını zorlaştırmaktadır (Bozoklu & Çil, 2017). Bu çalışmada yazılım güvenlik açığı ve standartları üzerine bir literatür taraması yapılarak mevcut durum değerlendirilmiş ve bu yapıya özgü bir model önerisi sunulmuştur.

2. ÜÇÜNCÜ TARAF YAZILIM BİLEŞENİ VE KONTROLÜ

Üçüncü taraf yazılım bileşeni, bir yazılımın geliştirme evresinde kişiler veya firmalar tarafından pazarlanan veya dağıtımının gerçekleştirildiği yeniden kullanabilen yazılım bileşenlerdir. Bu bileşenler işletim sistemi hariç makro, betik, bot, obje(*.obj*), kütüphane(*.dll*), çalışabilir kod(*.exe*) gibi ikili kod (*binary code*) dosyalardan oluşmaktadır. Üçüncü taraf yazılım bileşeni içerisinde başka kurumlara ait yazılım bileşenlerinin varlığı da söz konusu olabilir. Bu yazılım bileşenlerinden kaynaklı bir kusur ya da zafiyet olup olmadığının kontrolü manuel veya zafiyet veri tabanlarını kullanan otomatik araçlarla gerçekleştirilebilir. Şekil 1'de üçüncü taraf yazılım bileşenlerinin kontrol ve değerlendirmesine ilişkin örnek bir akış şemasına yer verilmiştir.

Şekil 1. Üçüncü Taraf Yazılım Bileşenlerinin Kontrolü



(Yazarlar tarafından oluşturulmuştur)

Kontrol sonucunda herhangi bir zafiyete rastlanmayan yazılım bileşeni kullanılmak üzere kurumun güvenli sayılan uygulama bölgesine alınır. Kusurlu veya şüpheli bulunan bir yazılım bileşeni ise tehdit ve zafiyetler bakımından bir risk değerlendirmesine tabii tutulur. Değerlendirme sonucunda risk derecesi kabul edilebilir risk seviyenin altında bulunan yazılım bileşenleri kontrollü olarak güvenli bölgeye alınır. Riski kabul edilerek alınan yazılım bileşenleri, kurum risk yönetimi kapsamında risk bütünüyle ortadan kalkana kadar belli bir kontrol ve gözetim altında tutulur. Risk

değerlendirmesi sonucunda riski kabul edilemez bulunan yazılım bileşenleri hiçbir şekilde kurum içerisine alınmaz, herhangi bir şekilde kullanılmasına müsaade edilmez.

2.1. Zafiyet Analizi

Zafiyet, tehdit karşısında varlığının korunmasını sağlayan sistemlerdeki bir eksiklik, açıklık, kontrol zayıflığı olarak adlandırılabilir (Şahinaslan, 2010). Zafiyet analizi, bilgi sistemleri, uygulama ve ağ altyapılarının güvenlik açıklarını tespit, tanımlama, sınıflandırma ve önceliklendirme sürecidir. Zafiyet değerlendirme, bir kuruluş ve çevresine yönelik tehditleri anlamak ve bunlara tepki vermek için gerekli olan bilgi, farkındalık ve risk arka planı sağlar (Rosencrance, 2021). Yazılım güvenlik zafiyeti, bir saldırganın bir sistemin kontrolünü ele geçirmesine izin verebilecek yazılımdaki bir kusurdur (JFrog, 2022). Bu zafiyet, yazılımın tasarımından kaynaklı olabileceği gibi kodlama biçimi, üçüncü taraf şirketlerden edinilen yazılım veya bileşenlerinden kaynaklı bir kusur da olabilmektedir. Bir açıklığın/zafiyetin bir tehdit unsuru tarafından kullanılması, kurumlara itibar kaybı, finansal kayıp ve yasal yaptırım gibi telafisi güç kayıplar yaşatabilme potansiyeline sahiptir. Bu tür kayıpları en aza indirmek veya hiç kayıp yaşamamak için bunların zamanında tespit edilmesi, buna ilişkin etkin kontrol ve önlemlerin alınması ve riskin doğru yönetilebilmesi önemlidir.

Klasik risk analizi temel olarak tasarım ve operasyonlarını çevreleyen riskleri araştırmakla ilgilidir. Bu analizler, incelenen nesnenin neden ve sonuçlarına odaklanma eğilimindedir. Zafiyet analizi ise hem nesnenin kendisi için etkileyeceği sonuçlara hem de çevredeki birincil ve ikincil sonuçlara odaklanır. Ayrıca olumsuz sonuçları azaltma ve gelecekteki olayları yönetme kapasitesini geliştirmeyle de ilgilenir (Ritchey vd., 2004).

Yazılım bileşenlerinde zafiyet analizi yöntemleri aşağıda belirtilen üç temel başlıkta incelenebilir.

2.1.1. İkili Kod Analizi

Varlık, tehdit, saldırı olasılığı ve buna karşı alınan önlemler güvenlik risk değerlendirmesinde kullanılan bileşenlerdir (Şahinaslan, 2010). Günümüz güvenlik tehditleri artık ağ düzeyindeki saldırılar seviyesinden uygulama katmanı seviyesine gelmiştir. Uygulama içerisinde kullanılan bileşenlerin çeşitliliği ve yazılan kaynak dillerin farklılığı sebebiyle kod seviyesindeki güvenlik açıklarını görmek için ikili kod analizine ihtiyaç vardır. Bu analiz, üçüncü taraf üreticinin müdahalesi olmadan yazılım bileşenlerindeki zafiyetin tespiti ve lisans inceleme gibi alanlarda kullanılır. İkili kod (*binary code*) bir bilgisayar tarafından doğrudan yorumlanan programlama verilerinin temel biçimidir (Synopsis, 2022). Çalıştırılacak işlemci mimarisine göre bir programlama dilinde geliştirilen kaynak kodların derlenmesi gerekir. Diğer bir deyişle insan tarafından yazılan ve okunabilen kaynak kodların bilgisayarın anlayabileceği hale getirilmelidir. İkili kod analizi ise, ikili koddan oluşan dosyaların içerik ve yapılarını değerlendiren, bunu yaparken kaynak koda erişme ihtiyacı duymayan bir incelemedir. Bazı ikili kod analiz araçları, dosyanın içeriğini anlayabilmek için dosyanın içindekiler tablosunu okur. Analiz için kaynak kodun mevcut olmadığı durumlarda, özellikle üçüncü taraf yazılım bileşenlerinin zafiyetlerinin kontrol edilmesinde genelde '*ikili kod analizi*' kullanılır.

2.1.2. Bağımlılık Analizi

Bir yazılım projesinde üçüncü taraf bileşenlerin kullanılması, aynı zamanda o projede kullanılan diğer yazılım bileşenlerine bir bağımlılık (*dependency*) oluşturur. Üçüncü taraf yazılım bileşenlerinin fazla kullanıldığı bir yapıda bu bağımlılık daha da artar. Günümüzde kanıtlanmış kullanıma hazır araçlar yazılım geliştiriciler tarafından da yaygın olarak kullanılmaktadır. Bu durum üçüncü taraf yazılım bileşenlerin birbirlerine olan bağımlılıklarını önemli ölçüde artırmıştır. Üçüncü taraf yazılım bileşenlerinin kendisi bir zafiyet içermemesine rağmen, birlikte çalışmak zorunda kaldığı veya referans aldığı bir bileşenden kaynaklı açıklıkta söz konusu olabilir. Bu nedenle projede kullanılan üçüncü taraf bileşenleri ile birlikte bu bileşenlere bağımlı kurum içi uygulamaların tamamı tespit edilerek bir bütünlük içerisinde bağımlılık analizi gerçekleştirilmelidir.

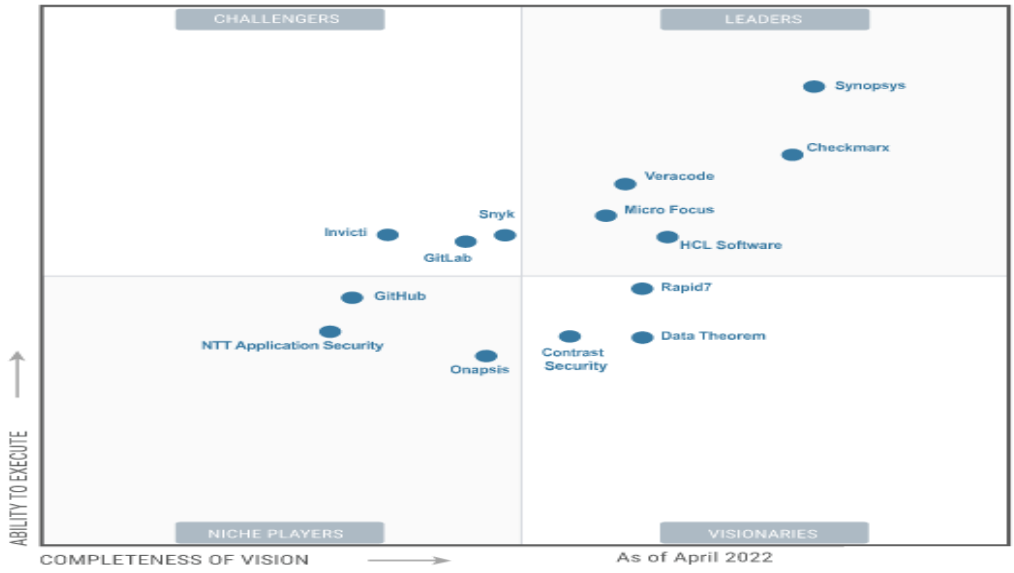
2.1.3. Kod Parçacık Analizi

Makasla kesilmiş parça anlamına gelen '*snippet*', tek başına çalışmayan ancak yazılım içerisinde kısa yollar aracılığıyla kullanılan kaynak kod parçacığı anlamındadır, genişletilebilir işaret dili olan XML ile yazılır, '*for*' ve '*foreach*' komutları örnek olarak verilebilir (ITU, 2022). Kod parça analizi (*snippet anaylsis*), bir kod bloğunun tamamının ya da bir kısmının üretilen yazılım içinde kullanılması sonucunda ortaya çıkması muhtemel bir zafiyetin tespit edilmesidir. Günümüzde genel kabul görmüş bazı kod bloklarının yazılım projelerinde doğrudan kullanımı tercih edilen bir yöntemdir. Kod parçası içerisindeki bir zafiyet sadece bulunduğu kod bloğunu değil, o yazılımın tamamının risk altına bırakacağı bir gerçektir.

2.2. Uygulama Güvenlik Test Araçları

Üçüncü taraf yazılım bileşenleri yazılım geliştirmede yaygın olarak kullanılmaktadır. Kaynak koduna sahip olunmayan bir yazılım bileşenin içerisinde yer alan/alabilecek zafiyetin manuel olarak tespiti oldukça zordur. Tespit edilemeyen bir zafiyetin doğurabileceği riski öngörebilmek de zorlaşır. Gün geçtikçe genişleyen ve karmaşık hale gelen sistem ve alt yapılarda, kullanılan bir yazılım bileşeninden kaynaklı zafiyetin etkileşimde bulunduğu diğer sistem ve uygulamaları da riskli hale getirir. Bu sebeple üçüncü taraf bileşen zafiyetlerinin tespit edilmesi bir yazılımın yaşam döngüsünde oldukça önemlidir. Günümüzde bunların manuel yapılabilmesi oldukça güçtür. Bu test ve kontrollerin sistemler tarafından otomatik bir şekilde gerçekleştirilmesi için bilinen zafiyet veri tabanlarını da kullanan çeşitli uygulama güvenlik test araçları geliştirilmiştir. Bu tür araçlarda zafiyet kontrolü kaynak koda ihtiyaç olmadan ikili kodlar üzerinden de gerçekleştirilebilir. Bu alanda yaygın kullanılan uygulamalara ilişkin Gartner tarafından 2022 yılında yayınlanan rapor Şekil 2'de gösterilmektedir.

Şekil 2. Uygulama Güvenliği Test Araçları



(Gartner, 2022)

Bu uygulamalar çeşitli yönleriyle öne çıkmaktadırlar, genelde yazılım geliştirme ve üretiminde, açık kaynaklı veya üçüncü taraflardan temin edilen yazılım bileşenlerine ait risklerini algılar ve yönetirler. Ürün bazlı incelendiğinde; konteyner görüntülerinde (*container images*) ve ikili (*binary*) dosyalarda bulunan açık kaynağı benzersiz bir şekilde tanımlar (Synopsys, 2022). Güvenli yazılım kodu geliştirmede, uygulamaların testinde ve güvenli dağıtımında bir araç olma niteliğindedir (Checkmarx, 2022). Uygulama güvenlik risklerini azaltan, güvenli ve hızlı kod yazmaya yardımcı, düzenleyici kurallarıyla uyumlu yazılım geliştirici ve güvenlik uzmanlarına yardımcı bir uygulamadır (Veracode, 2020). Güvenli ürün geliştirme ve testinde güvenlik ekiplerine ve geliştiricilere yazılım yaşam döngüsünün her evresinde destek olarak kullanılır (Micro Focus, 2022). Geliştirilen veya tedarik edilen yazılıma ait güvenlik açıklıklarını tespit eden ve bunun iyileştirilmesine katkı sunan bir araçtır (HLC Software, 2022). İnternet uygulamalarındaki riskin azaltılması için güvenlik zafiyetlerinin tespitine yönelik kara kutu güvenlik testi gerçekleştirir. Dinamik uygulamaların güvenliğini sağlamada yazılım yaşam döngüsünün geliştirme, test ve dağıtımında geliştirme ortamlarıyla bütünleşik çalışır (Rapid7, 2022). Yazılım geliştiricilerinin kod yazarken güvenlik açıklıklarını bulmasına ve düzeltilmesine yardımcı olur, güvenlik ekiplerine eksiksiz görünürlük ve kapsamlı kontroller sağlarken, geliştiricilerin güvenli yazılım geliştirmesine imkân sunar (Snyk, 2022). Yazılım kodunun incelenmesi, güvenlik açıklık ve sorunların bulunması, açık kaynak kodların güvenli bir şekilde kullanımına yardımcı olan bir uygulamadır (GitHub, 2022). Sürüm kontrolü, sorun izleme, kod inceleme, sürekli bütünleşme ve sürekli dağıtım özelliklere sahip açık kaynaklı, uçtan uca bir yazılım geliştirme platformudur (GitLab, 2022). Yazılım yaşam döngüsü boyunca güvenli geliştirme ve uygulamayı korumaya dönük birleşik bir güvenlik platformuna sahiptir. Üçüncü taraf bağımlılıklara ait güvenlik açıklarını bulma ve bildirmeye yarayan Contrast SCA aracına sahiptir (Contrast-Security, 2022). *Data Theorem*, modern uygulama güvenliğinin lider sağlayıcısıdır. Temel görevi, herhangi bir modern uygulamayı her zaman, her yerde analiz etmek ve güvenliğini sağlamaktır (DataTheorem, 2022). Kullanılan teknoloji ve dilden bağımsız her türden web uygulamasını, web hizmetini ve web API'sini tarayan ve güvenlik zafiyetlerini bulabilen bir uygulamadır. Kurumsal internet uygulama güvenlik ürünü Java, .NET, Node.js genel dil desteği sınırlıdır (Invicti, 2022). SAP uygulamaları için özel olarak tasarlanmış otomatikleştirilmiş güvenlik testlerine sahiptir. Bu ürünle yazılım geliştirme yaşam döngüsünde kurum içi veya üçüncü taraf yazılım kodlarının analiz ve kodun kullanılabilirliği test edilmektedir (Onapsis, 2022). Hem geliştiriciler hem de yazılım dağıtıcıları tarafından kullanılabilen, uygulamalardaki güvenlik açıklarını tespit eden ve düzeltilmesine katkı

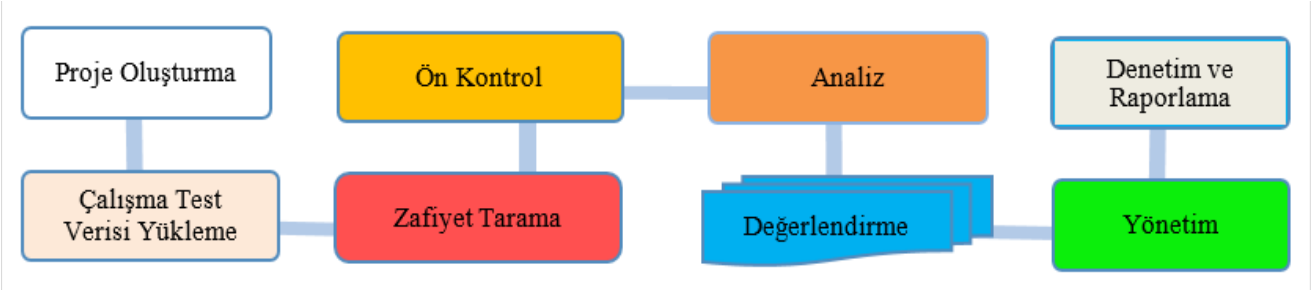
sunan bir uygulamadır (NTT, 2022). Üçüncü taraf yazılım bileşenlerindeki zafiyetlerin tespitinde kullanılan uygulamalar genelde ikili kodların analizi şeklinde gerçekleştirilir. Bu amaçla her biri kendine özgü yeteneklere sahip bu uygulamalar, benzer özelliklerin yanı sıra birbirinden ayrışan pek çok farklılığa da sahiptir. 2022 Gartner raporuna göre liderlik ve vizyon bakımından *Synopsys* firmasının satın aldığı *Black Duck* ürünü öne çıkmaktadır.

Bu çalışmada üçüncü taraf yazılıma ait güvenlik zafiyetlerin tespitine yönelik çalışma örnek bir uygulama üzerinden sunulmaktadır. Ayrıca üçüncü taraf bileşen zafiyet ve risklerine dikkat çekilerek bu zafiyetlerin tespitinde kullanılan uygulama araçlarından *Black Duck* yazılımı (Blackduck, 2022) üzerinden analiz ve uygulama pratiği gerçekleştirilmiştir. Çalışmadan elde edinilen deneyim ve bulguların literatüre kazandırılması amaçlanmıştır.

3. MATERYAL VE METOT

Bu çalışmada üçüncü taraf yazılımlardan kaynaklı zafiyetler, bunların tespitinde kullanılabilecek yöntem ve uygulamalar ele alınmıştır. Yazılım bileşenlerinden kaynaklı zafiyetlerin tespiti ve risklerin etkin yönetebilmenin yolları araştırılmış, bu amaçla kullanılan yazılım araçları incelenmiş ve seçilen bir uygulama üzerinden çalışma pratiği gerçekleştirilmiştir. Çalışma pratiğinde seçilen bir üçüncü taraf yazılım bileşeni üzerinden sırayla zafiyet tarama, yazılım bileşen kontrolü, analiz, değerlendirme, zafiyet yönetimi, denetim ve raporlama süreç aşamaları çalışılmıştır. Pratik çalışmada uygulanan yöntem ve ana süreç aşamaları Şekil 3'de gösterilmektedir.

Şekil 3. Pratik Çalışma Süreç Aşamaları



(Yazarlar tarafından oluşturulmuştur)

3.1. Tanımlar

Çalışmada kullanılan yazılım ve entegrasyona dair temel terim ve tanımlar şu şekilde açıklanabilir.

3.1.1. Üçüncü taraf yazılım bileşeni (*third party software component*), yazılım geliştirme sürecinde geliştirilen ana uygulamanın geliştiricisi dışında biri ya da birileri tarafından geliştirilen ve dağıtılan yazılım bileşenlerine verilen addır.

3.1.2. Açık kaynak kodlu yazılım (*open source software*), yazılım kaynak kodu sahibi tarafından yazılımı üçüncü taraflara kullanma, inceleme, değiştirme ve dağıtma gibi haklarıyla birlikte veren bir lisanslama türüdür (Laurent, 2004).

3.1.3. Sürekli entegrasyon (*continous integration*), geliştirilen yazılım bileşenlerinin ana yapıya entegre edilmesidir. Her entegrasyon sonrası hata oluşma durumunu tespit etmek için otomatik bir yapı tarafından doğrulama gerçekleştirilir. Bu durum entegrasyon sorunlarını azaltır ve yazılımın geliştirme hızını artırır (Fowler, 2006).

3.1.4. Kaynak kod deposu (*source code repository*), üzerinde çalışılan yazılım kodun bir arşividir. Bu bir kodun ötesinde, belge, not, web sayfası ve diğer öğelerden de oluşabilir. Herhangi bir başarılı yazılım geliştirme projesi için kod deposu gereklidir (Pawlan, 2021).

3.1.5. İnşa (*build*), yazılan bir kodun derlenmesi, paketlenmesi, test edilmesi ve kod analizinin yapılma sürecidir.

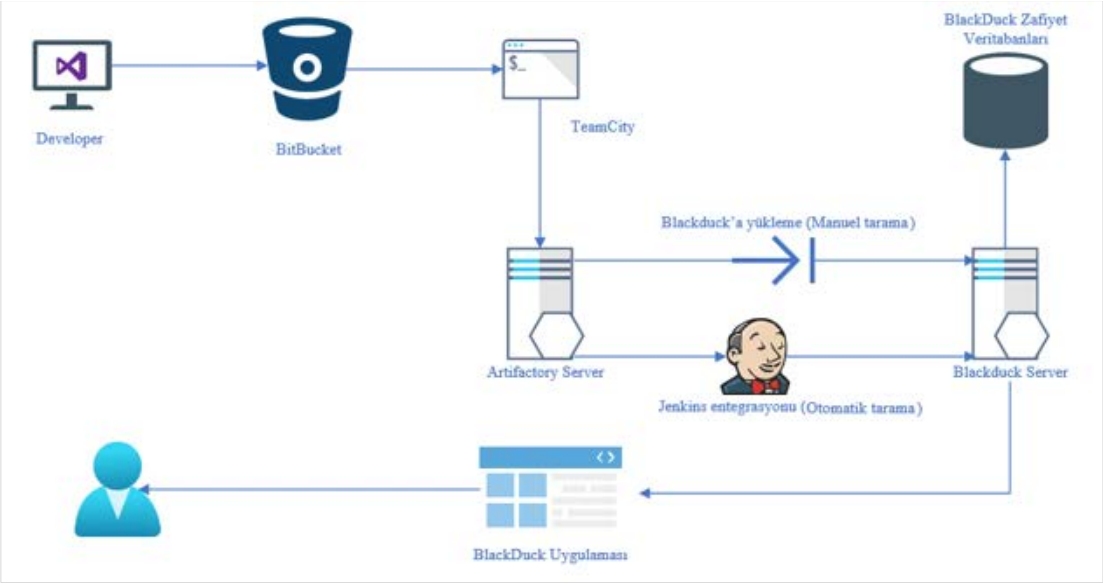
3.2. Çalışma Ortamı

Üçüncü taraf bileşen zafiyet tespitinde yaygın kullanılan araçlar çalışma içerisinde yer almaktadır. Çalışma kapsamında bu uygulamaların bütünü üzerinde uygulamalı çalışabilmek mümkün olamamıştır. Bu nedenle 2022 Gartner raporunda lider konumda olan *Synopsis Black Duck* uygulaması seçilmiş ve uygulamalı çalışma bu yazılım aracı üzerinden gerçekleştirilmiştir. Yazılım bileşeni olarak NVD'de yer alan zafiyete sahip üçüncü taraf CKEditor yazılım bileşeninin 4.12.1 sürümü seçilmiştir. *Black Duck* ürününün minimum ihtiyacı olan 24 GB yüklü bellek (RAM) ve 6 çekirdekli işlemcili, en az 500 GB depolama alanına sahip bir sunucuya ihtiyaç duyulmuştur. Uygulama kodlama ve test işlemlerinde Microsoft Visual Studio 2019 Enterprise Edition editörü ve C# 64 bit programlama dili kullanılmıştır.

Bu çalışmada yazılım projelerinde kaynak kod deposu (*source code repository*) olarak *Bitbucket*, kaynak kodların inşasında araç olarak *TeamCity* uygulaması, inşa sonrası çıkan ikili dosyaların saklanması için *Artifactory* ve manuel tarama için *Jenkins* kullanılmıştır. Jfrog Artifactory birçok yazılım paket tipini destekleyen ve otomatik sürekli entegrasyonu (*automated continuous integration*) sağlayan bir depo yöneticisidir (*repository manager*). *Black Duck*'a gönderilen tarama verilerinin öncelikle *JFrog Artifactory*'ye gönderileceği ve oradan *Black Duc*k uygulamasına aktarımı planlanmıştır. Çalışma ortam ve ilişkileri bakımından, uygulama kaynak kodları Bitbucket sunucuları üzerinde kullanılabilir şekilde hazırlandı. Derleme aşamasında kullanılan *TeamCity*, .Net platformu uygulama sunucusu üzerinde yüklendi, dağıtım çıktılarının gönderileceği ve saklanacağı *Jfrog* Enterprise *Artifactory* platformunun çalışır şekilde hazırlandı.

Çalışma ortam ve ilişkileri Şekil 4'te sunulmaktadır.

Şekil 4. Çalışma Ortam ve İlişkileri



(Yazarlar tarafından oluşturulmuştur)

Çalışma sırasında bir yazılım geliştiricisinin ihtiyaç duyulabileceği yetki tanımları temin edildi.

3.3. Varsayım ve Sınırlılıklar

Bu çalışmada ihtiyaç duyulan çalışma ortamının *Bitbucket*, *TeamCity*, *Artifactory* ve *Black Duck* sunucu, uygulama ve zafiyet veri tabanı kurulum, konfigürasyon, bağlantı ve entegrasyonlarının kurulu ve erişim izinlerinin eksiksiz bir şekilde verilmiş olduğu varsayılmıştır.

Çalışmada kullanılan zafiyet tarama uygulaması, hali hazırda belli başlı programlama dillerinde yazılmış üçüncü taraf bileşenleri test edebilmektedir. Bunlar: *C, C++, C#, PHP, .Net, JavaScript, Objective C, Java, Kotlin, Groovy, Python, Node.js, Scala, Perl, Swift, Ruby vb.* Çalışma ve elde edilen bulgular çalışmanın yürütülen zamanı, ortam koşullarıyla ve çalışılan uygulamanın sunduğu özelliklerle sınırlıdır.

4. UYGULAMA

Bu bölümde *Black Duck* yazılım bileşen analiz uygulaması üzerinde gerçekleştirilen internet tabanlı bir uygulamanın yazılım bileşen testi deneyimlenmiştir. Çalışmanın uygulama aşamaları sunulmaktadır.

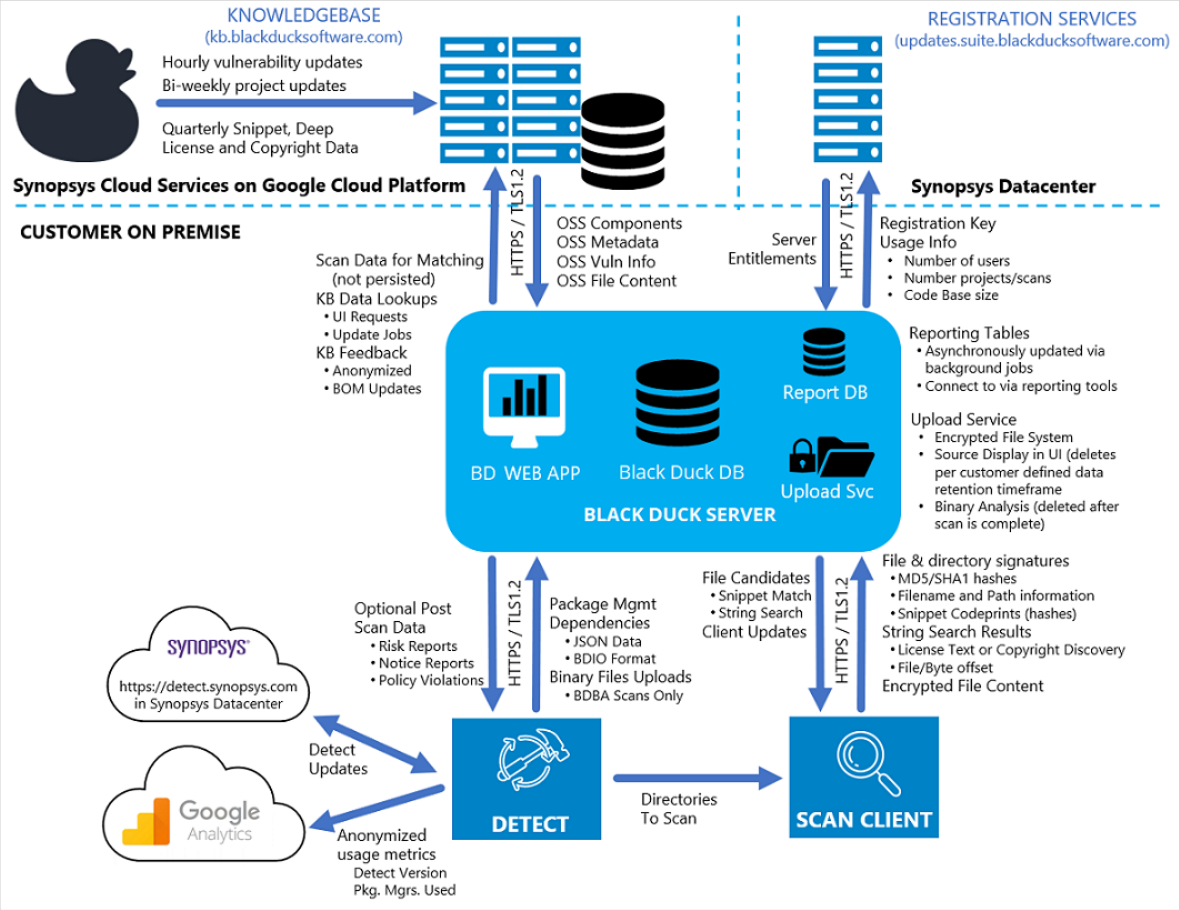
4.1. Uygulama ve Mimarisi

Üçüncü taraf bileşenlerin güvenlik zafiyetlerinin test ve kontrolü *Black Duck* uygulamasına entegre ikili kod analiz eklentisiyle gerçekleştirilir. Bu araç uygulama içerisindeki yazılım kütüphanelerini, çalıştırılabilir dosyaları, açık kaynak kodlarını, yazılım bileşenleri ve bu bileşenlerindeki zafiyetler ile güvenlik, lisans, kalite ve uyum riskini sunar. Uygulama güvenlik açıklık ve riskleri bir bütün olarak görme ve değerlendirme imkânı verir. Ayrıca çeşitli üreticilere ait yazılımı formatları, dosya sistemleri/disk görüntüleri, kurulum formatları ve çeşitli sıkıştırma ve arşiv formatları dâhil pek çok dosya tipini destekler. İkili analiz ile temelde şu işlemler yapılır (Blacduck Binary Analysis, 2022):

- Kodun kaynağına erişme ihtiyaç duymadan derlenmiş yazılımları, yerleşik yazılımları, mobil uygulamaları ve birden çok yükleyiciye ait yazılımı analiz eder.
- İkili yürütülebilir dosyalar ve kitaplıklar içindeki gömülü açık kaynak kullanımını ve riskleri belirler.
- İkili bağımlılıklar içinde kod bozulmasını yönetir ve yazılım kalitesini iyileştirir.
- Önceden taranan ikili dosyalardaki yeni güvenlik açıkları izlenebilir.
- Güvenlik ve geliştirme ekiplerinin uygulama portföylerindeki açık kaynak kodlarıyla ilgili riskleri belirleyerek bunların giderilmesine yardımcı olur.

Black Duck tarafından sunulan uygulama mimari ve ilişkileri Şekil 5'de gösterilmektedir.

Şekil 5. Kurum içi uygulama mimarisi



(Black Duck Documentation, 2022)

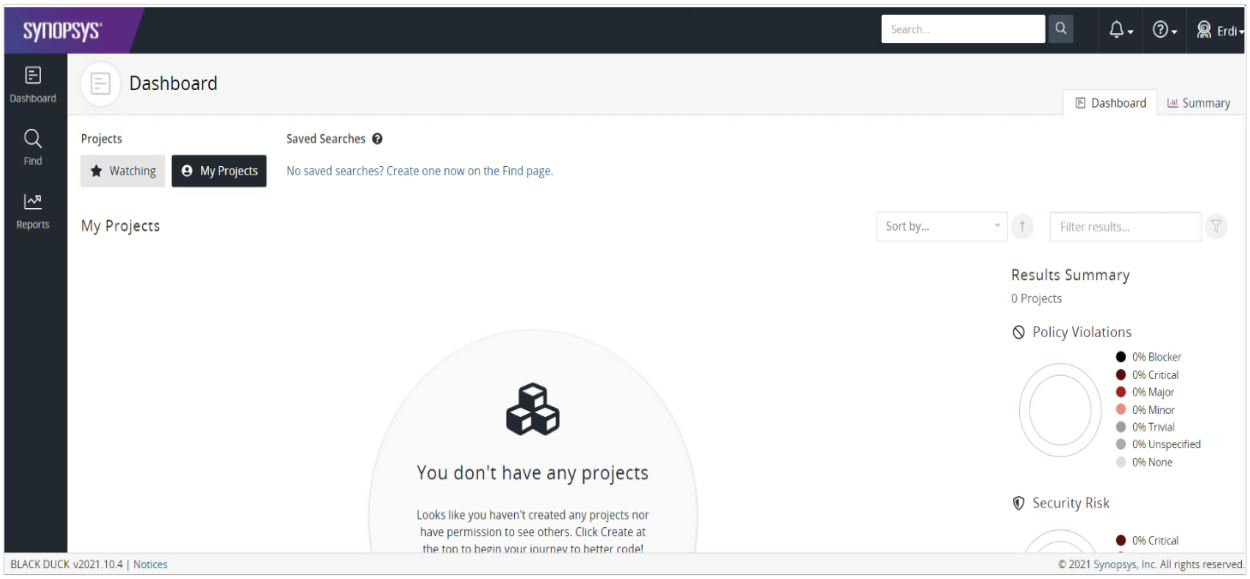
Ana fonksiyon ve bileşenleri olarak tespit etme (*detect*), tarama istemcisi (*scan client*), *Black Duck* sunucu, kayıt servisi (*registration services*) ve bilgi tabanı (*knowledgebase*) öne çıkmaktadır. Tespit etme, doğrudan veya dolaylı bağımlılıkları belirleme, toplama ve açık kaynak yazılım (*OSS*) eşleştirmesi için istemcinin *Black Duck* sunucusuna BDIO formatında JSON verileri gönderme görevini yerine getirir. Bağımlılıklar belirlendikten sonra, tarama için '*Black Duck Scan*' istemcisini başlatabilir. Diğer tarama yöntemleriyle birlikte *Black Duck Binary Analysis* (*BDBA*) taraması yapabilir. Genelde çevrimiçi çalışır, *https:\\detect.synopsys.com* adresinden bir curl komutuyla indirilir. Ağ erişimi olmadığı durumlarda çevrimdışı çalışma olanağı da sunmaktadır. *Synopsys Detect* anonim kullanıcı verisini toplamada *Google Analytics* kullanır. Dış ağlara erişimin sınırlı olması durumunda bu mekanizma çalışmayabilir ve ancak bu tarama için bir sorun oluşturmaz. Tarama istemcisi, kurum yerelinde taranacak dosyaların bulunduğu her yerde çalışır. Dosyaları incelerken, taradığı dosya ve dizin imzasını oluşturur. Dosya yolu ve dosya/dizin boyutları gibi diğer meta verileri toplar. Bu verileri analiz edilecek yazılım bileşen ve sürümünü belirlemede zafiyet veri tabanı (*Black Duck KnowledgeBase)* verileriyle eşleştirmek için kullanılır. Tarama sonuçlarını, şifreleme, dosya isim, dizin ve imzalarını kullanıcı kurumun *Black Duck* sunucusuna gönderir. *Tarama İstemcisi* kurumun *Black Duck* sunucusu ile güvenli (*https / TLS1.2*) bir şekilde iletişim kurar. Mevcut sürümünü o sunucu kontrol eder ve gerekirse kendini günceller. Tarama istemcisini çalıştırma

seçenekleri olarak; *Synopsys Detect* aracılığıyla kullanılabilir veya *CI* araçlarına (örneğin, *Jenkins*) entegre edilebilir, bağımsız (bir *CLI* veya *GUI* ile) veya diğer kapları taramak için bir *Docker* kapsayıcısı olarak kullanılabilir. İstemci, bileşen düzeyinde bir imza taraması çalıştırmanın yanı sıra isteğe bağlı olarak diğer tarama işlevlerini de gerçekleştirebilir. Son kullanıcı tarafından istenirse, lisans metni, telif hakkı beyanı, metin tabanlı dosyalarda bir dizi arama analizi çalıştırılabilir. Dize arama eşleşmelerinde, eşleşme kategorisi, dosya adı ve dosya içindeki konumu döndürür. Telif hakkı eşleşmeleri için telif hakkı beyanı yükler. Bileşen analizinin tamamlanmasını bekler. Eşleşmeyen dosyalarda OSS snippet'ler için tarama yapar. *Snippet tarama*, kayan pencere yöntemi aracılığıyla dosyaların parçaları üzerinde kod baskıları (tek yönlü kriptografik hash) oluşturur ve bunları daha sonra OSS ile eşleştirmek için kullanılır. Analiz için herhangi bir metin dosyası içeriği yakalamaz, depolamaz veya *Black Duck* bilgi tabanı (*Knowledge Base*)'e göndermez. Tüm kodlar şirket güvenlik duvarının arkasında kalır, yalnızca dosyadan elde edilen hash verisi aktarılır. Kod parçacığını yan yana karşılaştırmak veya *Black Duck* kullanıcı arabiriminde lisans ve telif hakkı görüntülemek için taranan kaynağı şifreler ve müşterinin *Black Duck* sunucusuna yüklenir. Kod yükleme, sunucu ve tarama düzeylerinde etkinleştirilmesi isteğe bağlı bir özelliktir. Sunucu kaynak yükleme için etkinleştirilmesi durumunda şirkete özel bir şifreleme anahtarını içerir. Kullanıcı kaynak dosyaları yüklemek istediğini belirtirse, istemci, açık kaynak kod yazılımı şifreler ve *Black Duck* sunucusuna bu şekilde yükler. Bir tarama işleminden sonra, tarama istemcisi imzaları *Black Duck* web uygulaması(*BD WEB APP*)'na gönderir. Bu uygulama, tarama sonucunda elde edilen bilgileri yönetmek ve görüntülemek için merkezi bir araçtır. *Black Duck*'ın web uygulaması kullanıcı bağlantıları güvenli bağlantı şeklinde gerçekleşir. *KB* servisi, *Black Duck* tarafından izlenen tüm açık kaynak projelerine ilişkin meta verileri (sürümler, imzalar, lisanslar, güvenlik açığı verileri vb.) içerir. Yeni bileşen veya sürüm, güvenlik açıklarına dair gerçek zamanlı güncellemeleri sağlar. Son kullanıcılar, bu sonuçları gözden geçirip düzenleyebilir, risk ve diğer faktörlere göre filtreleyebilir ve güvenlik açıkları, lisanslar, dosyalar ve ürün ağacındaki bileşenlerin biriyle ilişkileri hakkında daha ayrıntılı bilgi alabilir. *Black Duck* uygulaması tarama, güvenlik zafiyetinin durumunu izleme, raporlama gibi özellikleri de içerir. Uygulama, Apache Tomcat web sunucusu, *PostgreSQL* veritabanı vb. gibi çeşitli açık kaynaklı bileşenlerin üzerine inşa edilmiştir. *Black Duck* uygulaması, bir dizi *Docker* kapsayıcısı olarak dağıtılır ve yönetilir. Web uygulaması, hem *Black Duck KB* hem de *Black Duck* kayıt servislerine istek gönderebilir. Sistemin tam çalışabilmesi için bu hizmetlere sunucu tarafında erişim izni verilmelidir. *Black Duck* yükleme hizmeti (*Upload Svc*) ise tarama sırasında, tarama istemcisi kaynak dosya içeriklerini *SSL/TLS* korumalı uç nokta(lar) aracılığıyla ve uygun yetkilendirme belirteciyle *Black Duck*'a gönderir. Bir karşıya yükleme uç noktası, kaynak dosya içeriğini *HTTPS* aracılığıyla alır. Uç noktaya erişim doğrulanan *JSON Web Simgesi* (*JWT*) aracılığıyla güvence altına alınır. Kaynak dosyalar, dosya isimleriyle yerine ilişkili dosya imzalarıyla saklanır, yükleme sırasında ana anahtarla şifrelenir. *BD WEB APP*'a yüklenen dosyalar müşteri tarafından tanımlanan veri saklama politikasına uygun şekilde silinir.

4.2. Uygulama Genel Görünümü

Uygulama çalıştırıldığında kullanıcı/şifre doğrulama işlemi istenir. Kullanıcı kontrolleri sonrasında uygulamaya giriş yapıldığında Şekil 6'da sunulan gösterge(*dashboard*) ekranı ile karşılaşılır.

Şekil 6. Uygulama Gösterge Paneli



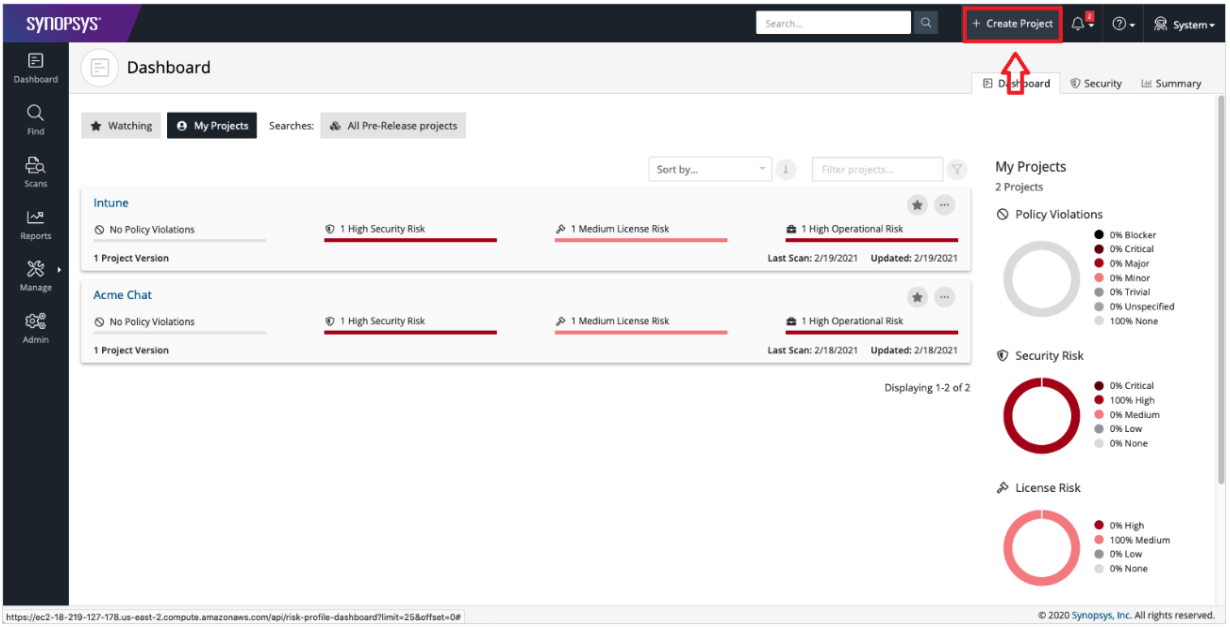
Herhangi bir proje ve buna bağlı tarama olmadığı durumlarda ekranda yer alan proje bölümü boş görünmektedir. Ekranın solunda bulunan menü nesneleri ile gösterge paneli, arama ve rapor ekranları arasında geçiş yapılabilmektedir. Ekranın

sağ üst bölümünde, *Black Duck* uygulamasına giriş yapmış kullanıcı ve kullanıcıya ait bildirimler yer alır. Sonuç özeti (*result summary*) kısmında, görüntülenen projelere ait politika ihlali (*policy violation*), güvenlik riski (*security risk*), lisans riski (*license risk*) ve operasyonel risk (*operational risk*) başlıkları listelenmektedir. Yine bu projelere ait zafiyetlerin tip ve düzeyini gösteren grafiklerde yer almaktadır.

4.3. Proje Oluşturma

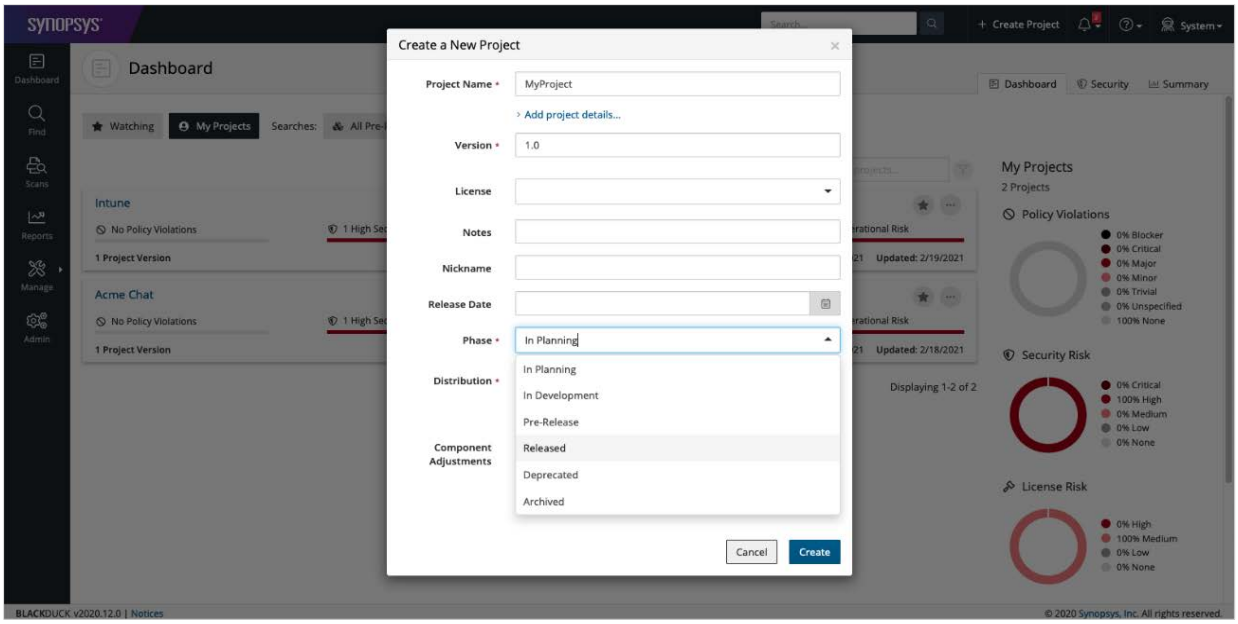
Bir yazılım projesindeki bileşenlerin zafiyet taramalarının gerçekleştirilebilmesi için bu projenin *Black Duck* üzerinde de yaratılması gerekmektedir. Böylece zafiyet taramasına tabi tutulacak projenin manuel ve otomatik taramaları için gereken entegrasyonlar kolayca sağlanabilir. Yeni bir proje oluşturmaya ait örnek ekran Şekil 7'de sunulmaktadır.

Şekil 7. Yeni Proje Oluşturma



Black Duck ana ekranı üzerinde '*Create Project*' ikonu kullanılarak yeni bir proje oluşturma işlemine başlanır. Ardından Şekil 9'da sunulan ekran üzerindeki alanlar oluşturulacak projeye uygun doldurulur ve '*Create*' düğmesine basılarak yeni bir proje oluşturulur.

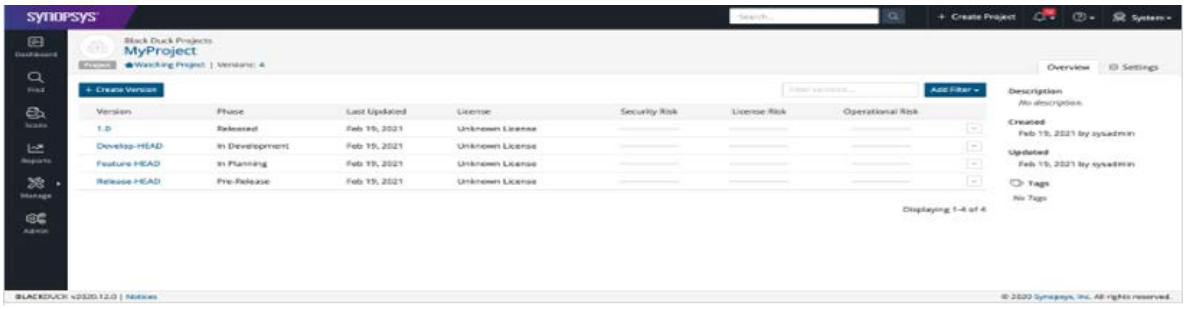
Şekil 8. Proje Oluşturmada İstenen Alanlar



Proje adı, sürümü, faz ve dağıtıma ait alanların doldurulması zorunludur. Proje adı projenin entegrasyon süreçlerinde kullanılacağı için her bir proje için tekil olmak zorundadır. Sürüm numarası alanına girilecek bilgilerin takip açısından doğru ve isabetli girilmesi gerekir. Bu ekran üzerinde taranacak projenin planlama, geliştirme gibi faz bilgisi ile '*Internal*', '*External*', '*SaaS*' ya da '*Open Source*' dağıtım türü seçeneklerinden biri seçilir.

Yazılım bileşen zafiyetlerinin taranmasında uygulanan en iyi yöntemlerden biri, projenin geliştirme, test ve canlı ortam için birer kopyasının oluşturulup her bir aşamada bu tarama süreci yeniden gerçekleştirilebilir. Bu amaçla her bir ortam için ayrı birer tarama örneği (*instance*) oluşturulur. Buna ilişkin örnek ekran Şekil 9'da gösterilmektedir.

Şekil 9. Farklı Ortamlara Göre Oluşturulan Tarama Örnekleri



Bu yöntemin izlenmesindeki temel amaç kullanılan üçüncü taraf bileşenlerinin başvurduğu bir kısmının çevrimiçi çalışması nedeniyle fazlar arasında herhangi bir zafiyetin var olmasından kaynaklanır. Aynı zamanda geliştirme ya da test fazında tespit edilememiş bir zafiyetin uygulamanın canlı ortama taşınması sırasında tespit edilebilmesi de mümkündür. Bu aşamalarda tespit edilebilecek bir zafiyetin giderilmesi bakımından kıymetlidir.

4.4. Test Verisi Yükleme

Çalışmaya konu yazılımın kodu, *BitBucket* adlı kaynak kod deposu üzerinde saklanır ve derlenir. Başarılı bir derleme sonrasında derlenmiş kod referansı yazılımın inşası için *TeamCity* uygulamasından referans alınır. *TeamCity*, *Bitbucket*'dan alınan kodun inşa edilmesini sağlar, bunu takiben inşa sonrasında ortaya çıkan tüm çıktılar sıkıştırılmış dosya halinde veya bir klasör halinde inşa çıktılarının saklandığı *Artifactory*'ye yüklenir. Bu yükleme sonucunda uygulama *API*'si kullanılarak veya isteğe bağlı kullanıcı arayüzü üzerinden zafiyet test aşamasına geçilir.

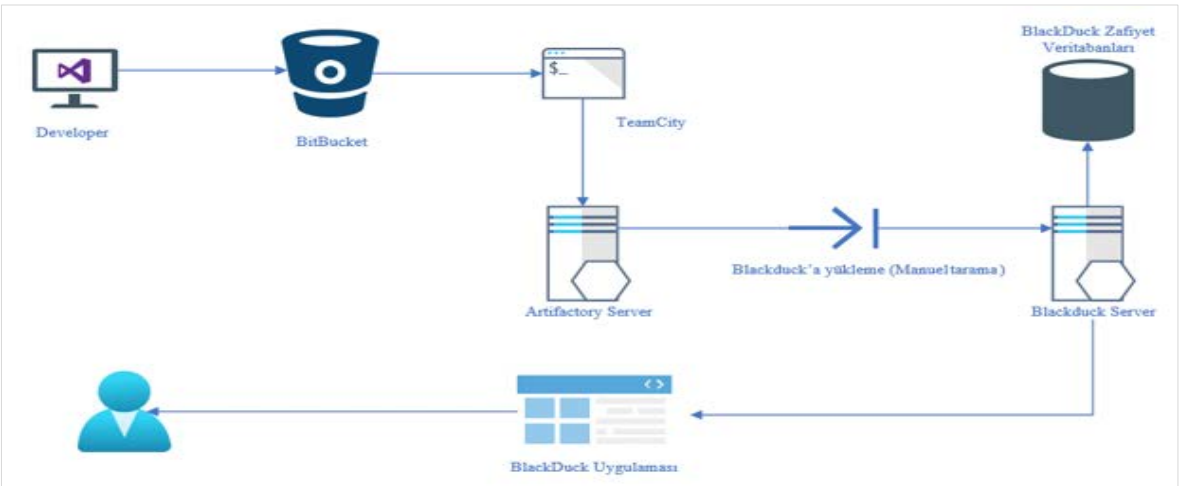
4.5. Zafiyet Tarama

Zafiyet tarama manuel ve otomatik olarak iki ayrı yöntemle yapılabilir.

4.5.1. Manuel Tarama

Manuel tarama yönteminde kullanılacak uygulama mimarisi Şekil 10'de gösterilmektedir.

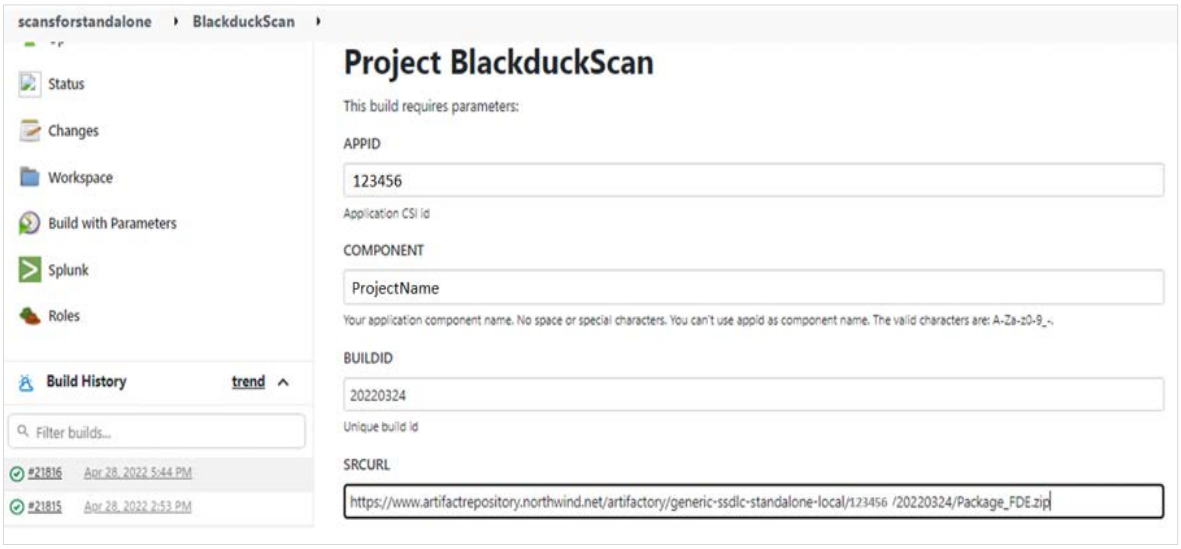
Şekil 10. Manuel Tarama Mimarisi



(Yazarlar tarafından oluşturulmuştur)

Manuel tarama süreci kullanıcı tarafından başlatılır. Kullanıcının süreci başlatabilmek için *Jenkins* üzerinden talep kaydı oluşturması gerekir. Bu aşamada kullanılan ekran ve ilgili alanlar Şekil 11'de gösterilmektedir.

Şekil 11. Manuel Zafiyet Tarama Ekranı

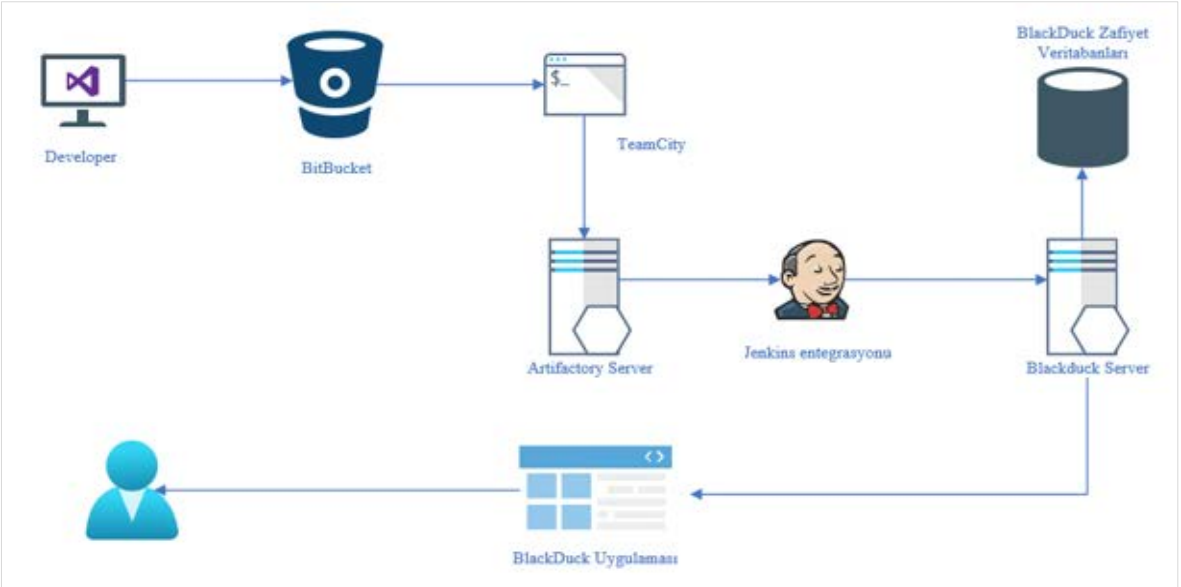


TeamCity'den *Artifactory*'e aktarılan ikili dosyaların adresi, proje adı, uygulama numarası ve inşa numarası(*BUILDID*)'nin Jenkins'e tanıtılması verilmesi gerekmektedir. *APPID* kısmına uygulamaya ait tekil uygulama numarası (örn:123456) girilmelidir. *COMPONENTID* bölümüne ise uygulamanın taranacak bileşen adı girilir. *BUILDID* kısmına *TeamCity* uygulamasından alınacak *Build* numarası eklenmelidir. *SRCURL* kısmında ise *TeamCity*'de *Build* edilmiş uygulama kodlarının *JFrog Artifactory*'deki sıkıştırılmış halinin tam adresi tanımlanır.

4.5.2. Otomatik Tarama

Manuel tarama yönteminde kullanılacak uygulama mimarisi Şekil 12'de gösterilmektedir. Otomatik zafiyet tarama süreci herhangi bir kullanıcı müdahalesi olmaksızın otomatik olarak başlatılır.

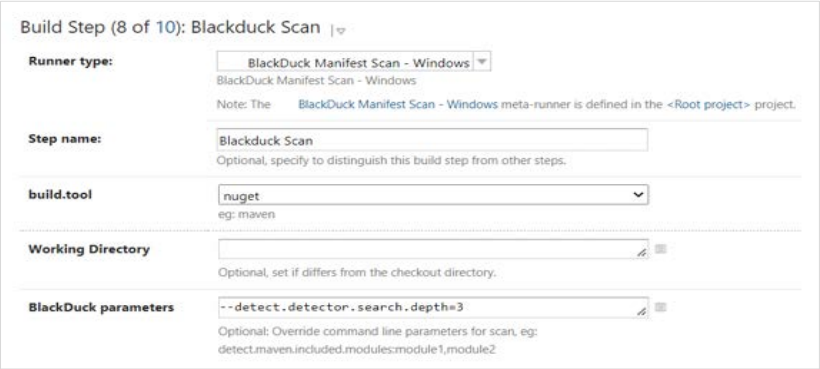
Şekil 12. Otomatik Tarama Mimarisi



(Yazarlar tarafından oluşturulmuştur)

İnşa edilen kod *Artifactory*'e yükleme adımı da tamamlandıktan sonra Şekil 13'de gösterilen inşa adımı ile *Black Duck API*'si kullanılarak otomatik olarak *TeamCity*'deki *Black Duck* uygulamasına tarama amaçlı gönderilir.

Şekil 13. Otomatik Zafiyet Tarama Ekranı

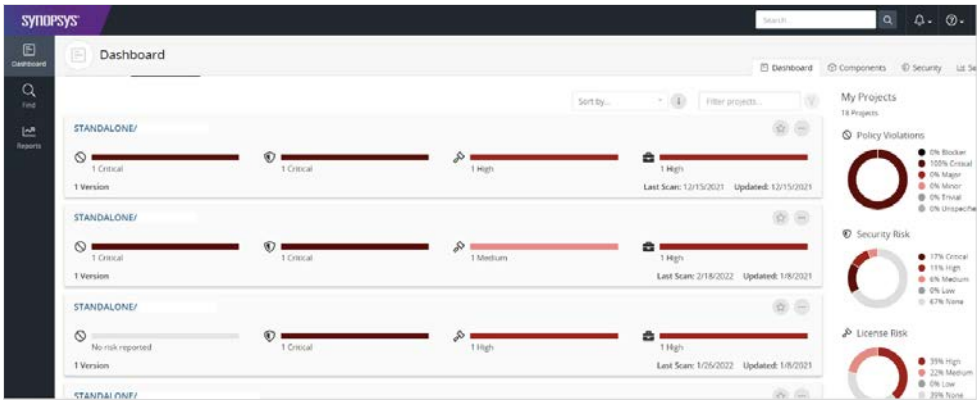


'*Runner Type*' kısmında '*BlackDuck Manifest Scan–Windows*' adı verilen manifesto dosyası seçildi. Bu manifesto içerisinde *Black Duck*'a *TeamCity*'deki hangi dosyanın nereden alınacağına dair *Jfrog Artifactory* dizin yolu, dosya adı bilgileri ile uygulama dosyalarının taramasında gerekli olan parametre verileri bulunmaktadır.

4.6. Ön Kontrol

Zafiyet taraması ile elde edilen bulgu ve risk düzeyleri uygulama kullanıcı arayüzü üzerinden kontrol edilir. Buna ilişkin örnek ekran görüntüsü Şekil 14'de gösterilmektedir.

Şekil 14. Black Duck Dashboard



Taramanın birden fazla proje üzerinde gerçekleştirilmesi durumunda her bir projeye ait sonuçlar özet olarak gösterilmektedir. Bu veriler bir ön kontrol ve değerlendirme için özet bilgiler içermektedir.

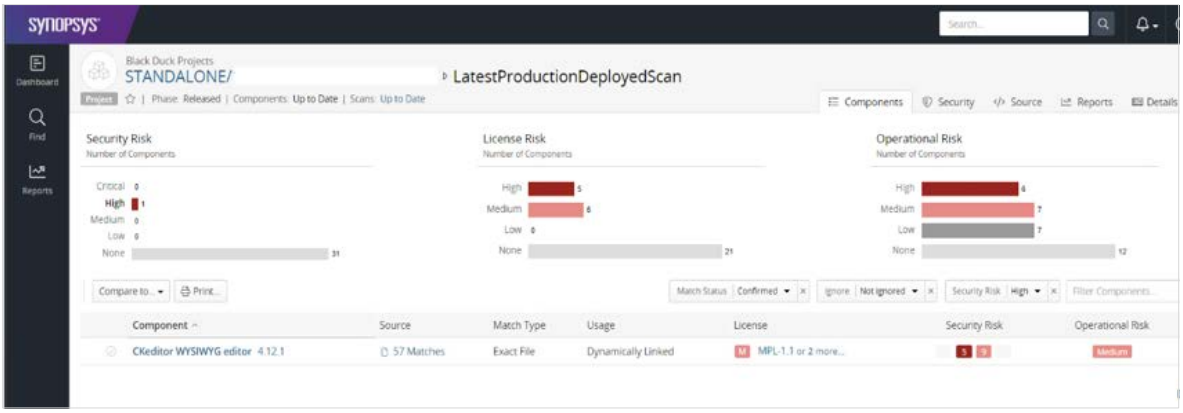
4.7. Analiz

Projeye ait politika ihlali, güvenlik riski, lisans riski ve operasyonel risk göstergeleri dört başlıkta sunulmaktadır.

- **Politika İhlali** (*Policy Violation*): *Black Duck* kendi içerisinde aşağıdaki üç risk tipinin de kullanılabildiği ve ağırlıklarının belirlendiği politikalar belirlenmesini sağlar.
- **Güvenlik Riski** (*Security Risk*): NIST tarafından sağlanan ulusal güvenlik açıkları veritabanı (*NVD*) veya *Black Duck* Güvenlik Önerileri (*BDSA*) veri tabanından bildirilen *CVE* numaralarıyla bileşene bağlanır. Risk olasılığı kritik (*critical*), yüksek (*high*), orta (*medium*), düşük (*low*) olarak değerlendirilir.
- **Lisans Riski** (*License Risk*): Yazılım bileşenlerinden kaynaklı lisans riskleri belirler. Riski derecesi yüksek, orta, düşük risk kategorilerinde değerlendirilir.
- **Operasyonel Risk** (*Operational Risk*): Katılımcı sayısı, kod geliştirme seviyesi, bileşen topluluğunun gücü ve bileşenin şu anda kullanımda olandan daha yeni sürümlerinin sayısı ile hesaplanır.

Tarama sonrasında yazılım bileşeni üzerinde tespit edilen güvenlik, lisans ve operasyonel risk tiplerine göre dağılımına ilişkin bileşen bazlı risk analiz örneği Şekil 15'de sunulmaktadır.

Şekil 15. Bileşen Bazlı Risk Analizi

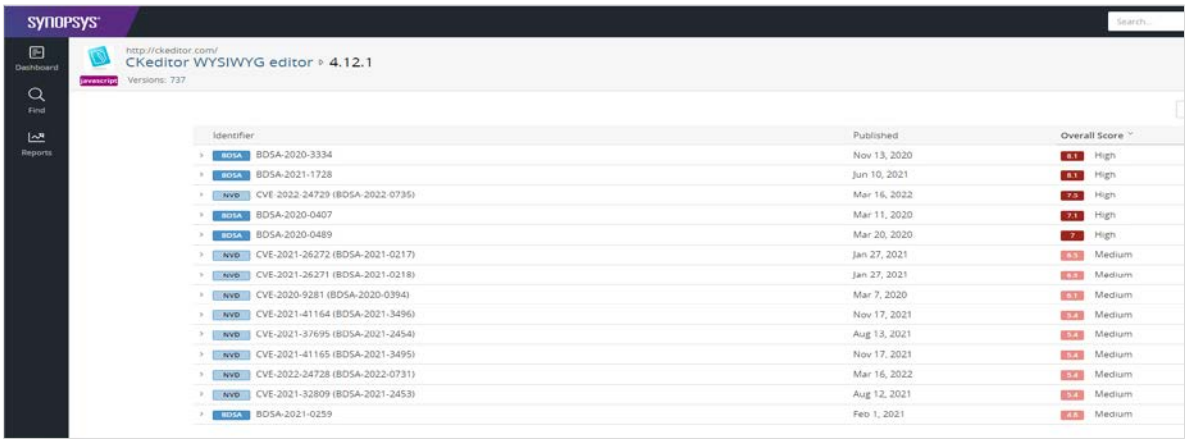


Dashboard ekranında görülen projenin detaylarını görebilmek için projenin adına tıklama yapılarak proje özelinde risk tiplerine göre detay veriler edinilir. Bu ekranda zafiyet sayıları ve bu zafiyetlerin hangi risk başlıkları altında ve hangi seviyede olduğu görülmektedir. İlk bakışta taranan proje hakkında genel bir görünüm vermesi sağlanır. Projede zafiyet olduğu tespit edilen her bir bileşen için; bileşen (*component*) kolonunda bileşeni adı, kaynak (*source*) sütununda zafiyete konu bileşenin proje içerisinde yer alan kaç kaynakta tespit edildiği bilgisi yer almaktadır. Karşılaşma tipi (*match type*) sütununda zafiyetin karşılaşma şekli yer alır. Karşılaşma tipi, zafiyet konusunun dosyanın tamamı ile mi yoksa dosya içerisinde belirli bir kod parçası ile mi ilişkili olduğuna dair bulgulardır.

4.8. Değerlendirme (Skorlama)

Test edilen bir yazılım bileşeninde birden fazla zafiyet bulunabilir. '*CKEditor WYSIWYG editor*' yazılımı 4.12.1 sürümüne ait farklı zafiyet veri tabanlarından elde edilen zafiyet verilerinin *Black Duck* üzerinde gösterimi Şekil 16'da sunulmaktadır.

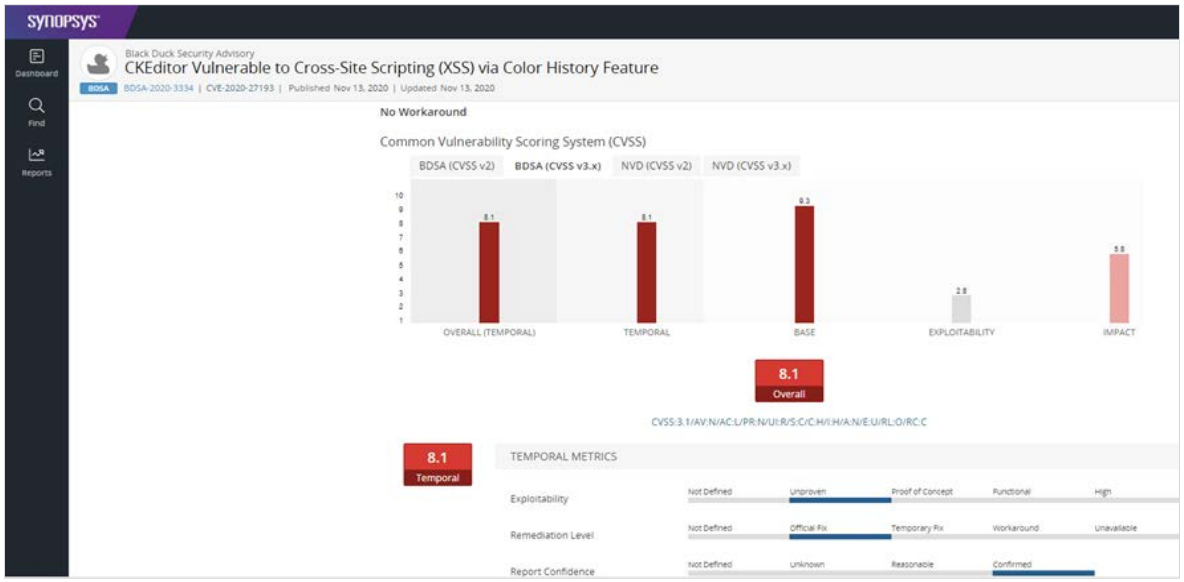
Şekil 16. Yazılım Bileşeni Zafiyet Liste Örneği



Tanımlayıcı (*Identifier*) sütununda zafiyetin elde edildiği veri tabanı bilgisi ve bu zafiyete ait tekil bir tanımlayıcı kodu yer alır. Yayınlanma (*Published*) adlı sütunda ise bu zafiyetin ilgili veri tabanında ilk yayınlanma tarihi yer alır. Listelemede, genel skor (*Overall Score*) adı verilen ve zafiyetin risklerinin değerlendirilmesi sonucunda elde edilen zafiyet riski üzerinden yüksekten düşüğe doğru sıralanmaktadır. Bu alanda yer alan risk değeri ulusal zafiyet veri tabanı (*NVD*) ve *Black Duck* güvenlik danışmanları (*BDSA*) tarafından zafiyet verileri dikkate alınarak yapılan değerlendirme sonucunda oluşmaktadır. Bununla birlikte bileşenler ve zafiyetler yaşam döngülerine devam eden yapılardır. Bu sebeple burada örneği verilmiş bir bileşen ve buna ait bir zafiyet, bileşen yaşam döngüsü içerisinde sonradan düzeltilmiş ve zafiyet ortadan kaldırılmış olabilir. Bu sebeple örneklemde kullanılan bileşenler ve zafiyet durumları dikkate alınırken çalışmanın yürütüldüğü tarih ve çalışma ortam ve koşulları göz önünde tutulmalıdır.

Tanımlayıcı sütununda yer alan her bir zafiyetin üzerinde tıklanarak ilgili zafiyete ait diğer detay bilgilere ulaşılması da mümkündür. Buna ilişkin örnek ekran görüntüsü Şekil 17'de gösterilmektedir.

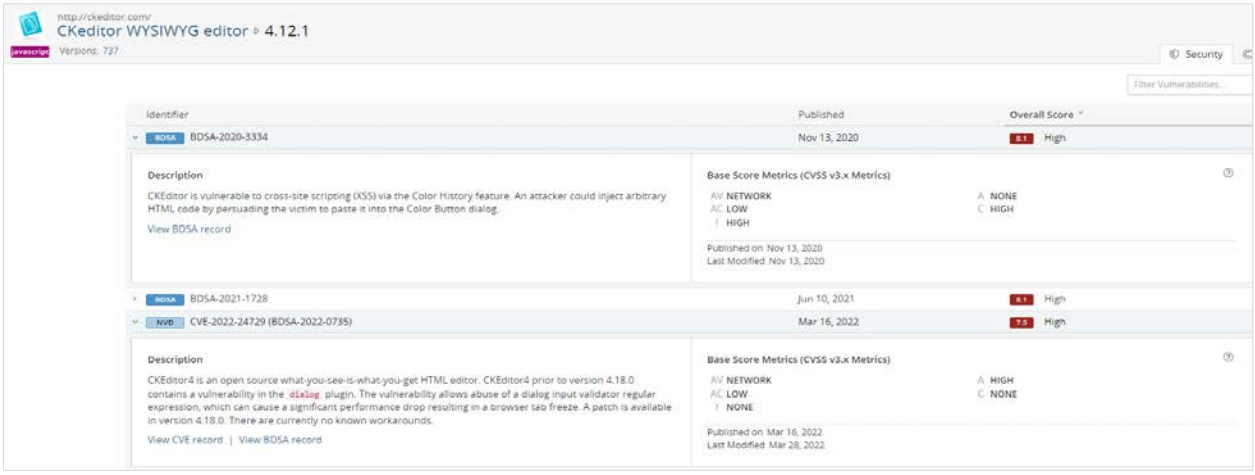
Şekil 17. Zafiyet Skor Ekran Örneği



'*No workaround*' ifadesi bu zafiyetin giderilmesi adına herhangi bir yama üretilmeden, zafiyetin etrafından dolaşarak atlatmanın mümkün olmadığı anlamına gelmektedir. Zafiyet skorlama sistemi (*Common Vulnerability Scanning System – CVSS*), *BDSA* ve *NVD* skorlarının belli bir ağırlığa göre ortalamasıdır. Geçici (*Temporal*) puan, bir güvenlik açığının teknik ayrıntılarının onayını, herhangi bir yama veya geçici çözümün varlığını ve yararlanma kodu veya tekniklerin kullanılabilirliğini dikkate alan güvenlik açığının zamana bağlı niteliklerini temsil eder. Temel(*Base*) skor ise bir güvenlik açığının zaman ve kullanıcı ortamlarında sabit kalan genel temel özelliklerini yansıtır. Sömürülebilme (*Exploitability*) puanı, erişim vektörü, karmaşıklık ve kimlik doğrulamayı dikkate alarak güvenlik açığına nasıl erişildiğini ve bu güvenlik açığından yararlanmak için ek koşulların gerekip gerekmediğini ölçer. Etki (*Impact*) puanı ise, gizlilik, bütünlük, erişilebilirlik niteliklerine olan etkisini yansıtır. Temel ve geçici skor metriklerinin hesaplanmasında CVSS'in genel kabul görmüş formüllerinden faydalanılır.

Zafiyet üzerinde tıklandığında bu zafiyet ile ilgili güvenlik detay bilgilerine ulaşmak mümkündür. Buna ilişkin örnek ekran görüntüsü Şekil 18'de sunulmaktadır.

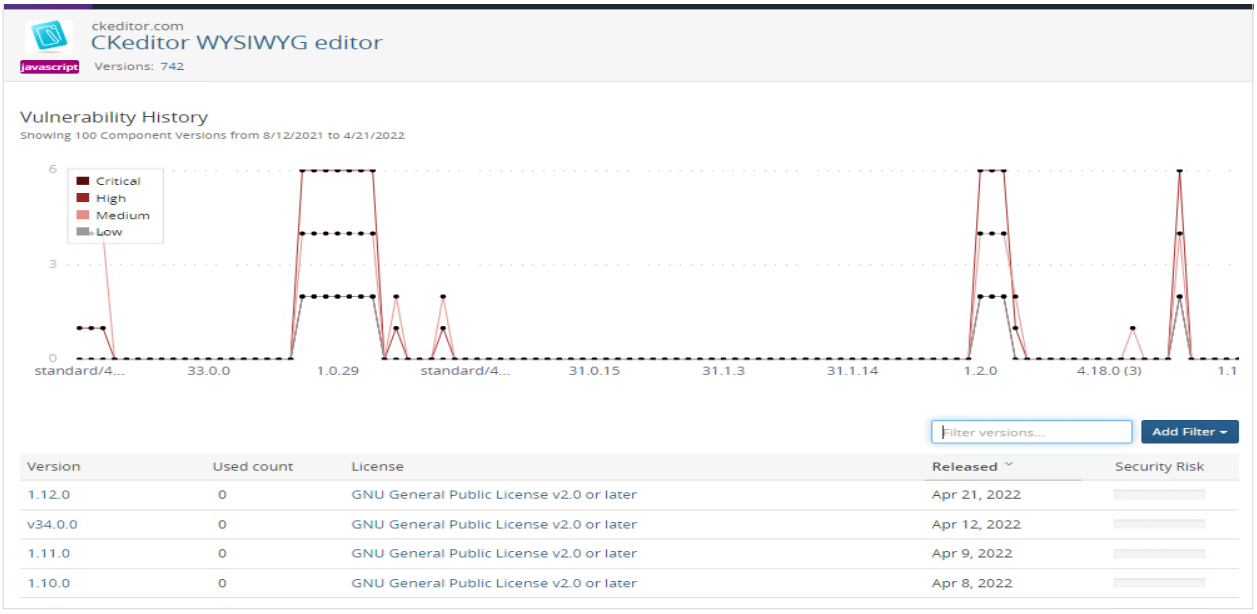
Şekil 18. Zafiyet Detay Bilgi Ekranı



Zafiyetin elde edildiği veritabanı, tekil kod, yayınlanma tarihi, genel skor değeri ve temel skor metrikleri yer almaktadır. Temel skor metrik bölümünde bileşen, yayınlanma ve son güncelleme tarih bilgileri yer almaktadır.

Uygulamada, tespit edilen bir zafiyete ait geçmişin detaylı bir şekilde sunulduğu ekran da vardır. Buna ilişkin örnek ekran görüntüsü Şekil 19'da sunulmaktadır.

Şekil 19. Zafiyet Geçmişi

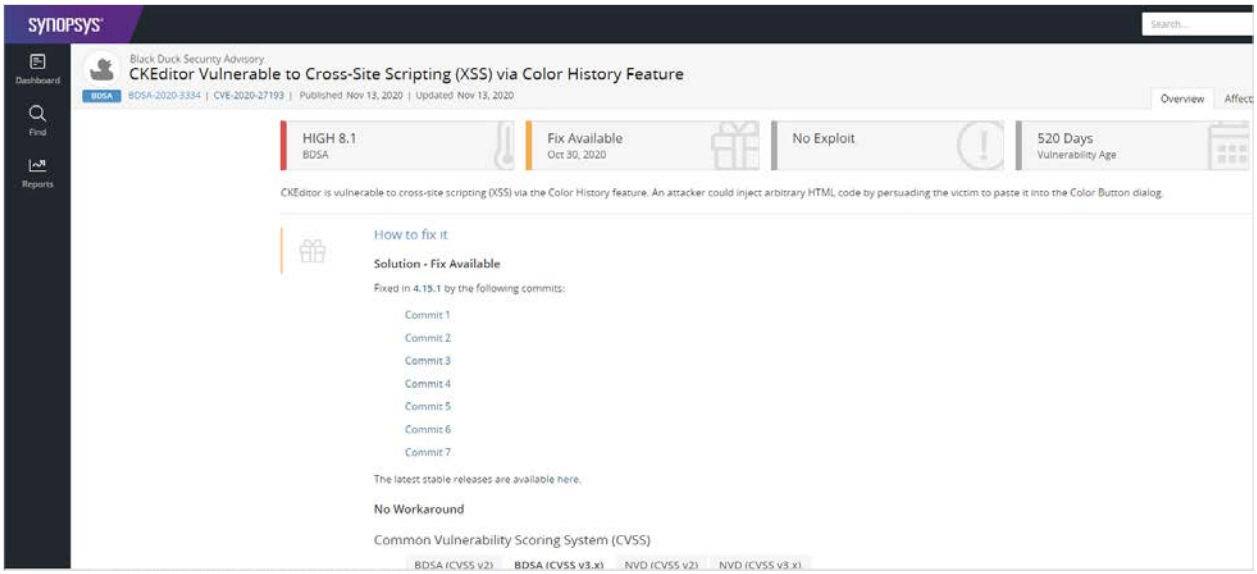


Kullanılan yazılım bileşenin piyasaya sunulma tarihçesi, zafiyetlerin hangi sürümlerinde hangi seviyede yer aldığına dair bilgi sunulmaktadır. Bu grafiğinde altındaki listede ise sürüm numarası, lisans bilgisi, sunum tarihi ve güvenlik risk bilgileri yer almaktadır.

4.9. Zafiyet Yönetimi

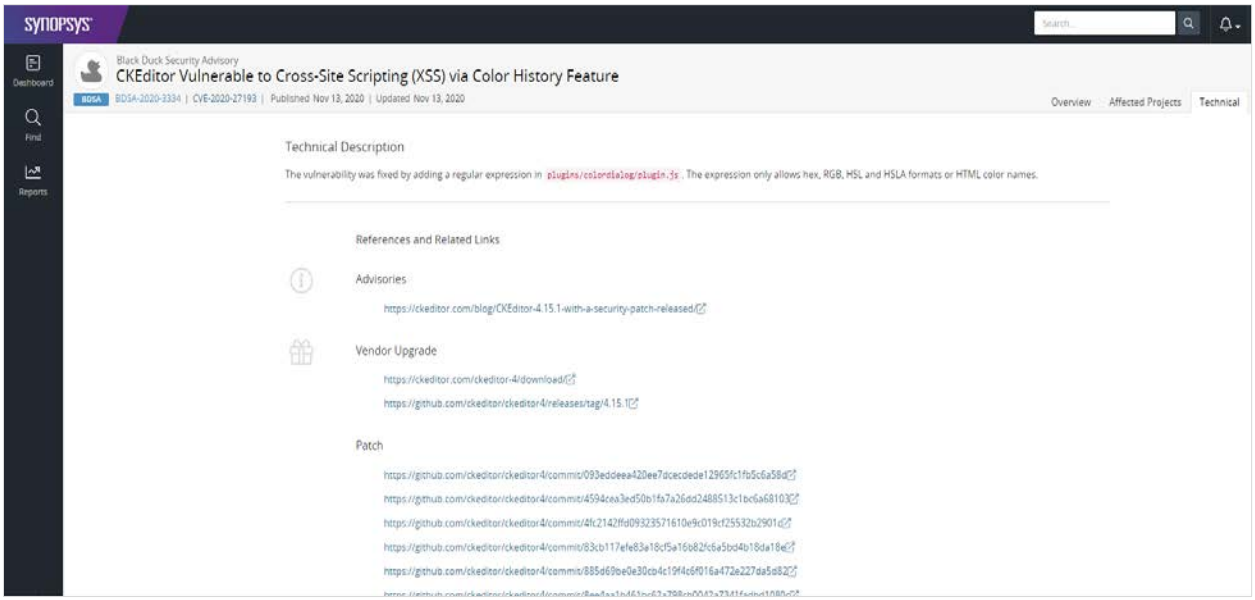
Uygulama üzerinde bileşen üzerinde yer alan her bir zafiyetin elde edildiği veri tabanı, tarihi, yaşı, genel skoru, sömürülme(*exploit*) durumu gibi veriler yanında bu zafiyetin çözümüne dair teknik detaylara ulaşmak mümkündür. Uygulamaya ait ana ekran görüntüsü Şekil 20'deki gibidir.

Şekil 20. Zafiyet Çözümü Genel Bilgi Ekran



'Nasıl düzeltilir' başlıklı kısımda zafiyete ait bir çözümünün olup olmadığı, çözüm var ise hangi sürümde nasıl çözüldüğüne dair bilgi paylaşılmaktadır. Bu çalışmadaki örnekte CKeditor 4.12.1 sürümünde yer alan bir zafiyetin 4.15.1 sürümünde düzeltilmiş olduğu, bu düzeltmenin hangi kod (*commit x*) ile yapıldığına dair detaylar liste halinde sunulmaktadır. Çözüme ilişkin detay bilgilere teknik sekmesinden ulaşılmaktadır. Buna ilişkin örnek ekran görüntüsü Şekil 21'de sunulmaktadır.

Şekil 21. Zafiyet Çözümü Teknik Detay Bilgi Ekranı

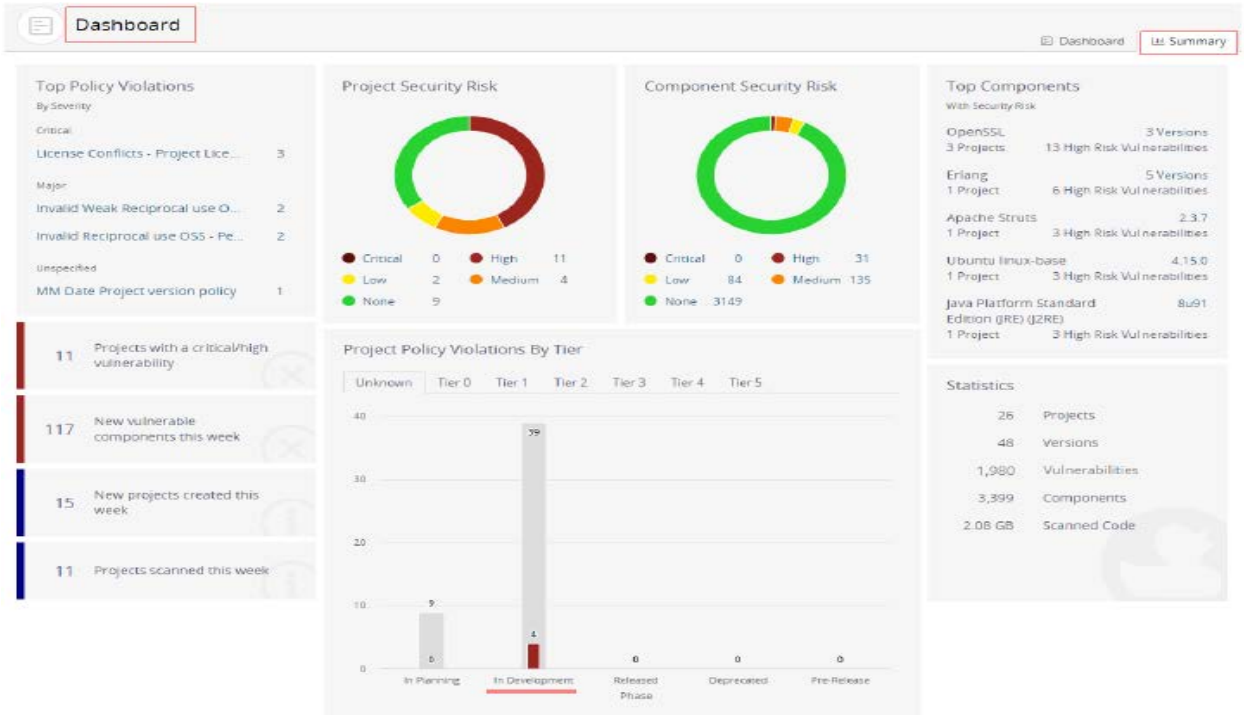


Çalışmada yer alan zafiyet incelendiğinde, teknik olarak zafiyetin '*plugins/colordialog/plugin.js*' dosyasına düzenli ifade (*regular expression*) ekleyerek çözüldüğü ve bu ifadenin sadece '*hex, RGB, HSL, ve HSLA*' formatlarına ya da HTML renk kodlarına izin verilerek giderilebileceği belirtilmektedir. Bu bileşenin sağlayıcısı tarafındaki güncellemeler için internet adresleri ile yama referans adreslerine yer verilmektedir. Bu veriler kullanılarak projede yer alan bileşen ve referansları önerilen veya varsa bir zafiyet tespit edilmemiş bir sürüme güncellenmesi sağlanabilir.

4.10. Denetim ve Raporlama

Uygulama üzerinden belirlenen zafiyetlerin geliştirme, test ve üretim gibi ortamlarda tespit edilmesi ve bunların raporlanmasına dair ekranlar yer almaktadır. Buna ilişkin örnek bir gösterge panel raporu Şekil 22'de sunulmaktadır.

Şekil 22. Özet Rapor Ekranı



Uygulamada bu tür raporlar yanında tespit edilen bir zafiyete dair uyarı, sorun, risk gibi kuruma ait başka uygulamalar üzerinde otomatik kayıt açmaya yarayan uygulama programlama arabirim (*API*) desteği sunulmaktadır. Bu yapı sayesinde elde edilen bir zafiyete dair risk kayıtlarının kurumlarda kullanılan risk yönetimi, iç kontrol ve denetim gibi bölümlerde kullanılan diğer uygulamalarla entegrasyona imkân sunulur. Böylece herhangi bir hata veya soruna ait güvenlik politika ihlal ve uyarılarına ilişkin sorun kayıtlarının oluşturulması, bunların izlenmesi ve yönetimi birlikte gerçekleştirilebilir. API desteği genel, rapor, bildirim ve eklenti türüne sahiptir. *Genel API*'ler; proje, bileşen, lisans, zafiyet, kullanıcı, rol/grup, tarama özetleri, zafiyetli bileşenler ve eşleşen dosyalar gibi alt başlıklara ayrılmıştır. *Rapor API*'leri, bir proje sürümüne ait bileşen, lisans, dosya ve zafiyet raporlarına erişime izin verir. *Bildirim API*'leri ile taramalar sonucunda tespit edilen zafiyet, politika ihlali ve lisans limiti gibi bilgiler sunulmaktadır. Özellikle bildirim API'lerinin kullanımıyla bir zafiyete ilişkin takibin kurumların kendi iç sistemleri üzerinden yürütülmesine imkân verir. Zafiyete konu olan bileşenin izlenmesine, risk ve kontrol kaydının oluşturulmasına, iç sistemler üzerinde yönetilen diğer risklerle bütüncül olarak ele alınmasına, görevler ayrılığı ilkesine uyumlu bir işletim, yönetim, denetim ve raporlanmaya katkı sunar. Eklenti API'leri ile dış sistemlerin *Black Duck*'a entegrasyonu ile üçüncü taraf uygulamalara uygulamadan veriler iletilebilir. Tarama sonuçlarına ait veriler elektronik posta bildirimi eklenti API'si üzerinden gerçekleşir.

5. BULGU VE TARTIŞMA

Günümüzde her işletme kullandığı uygulamalarla adeta bir yazılım işletmesine dönüşmektedir. İşletmeler gelen taleplere yetişmek için elindeki her türlü teknolojiyi kullanma çabası içerisindedir. Yeni teknolojilerin kullanımı işletmelere rekabet, karlılık, çeviklik gibi birçok alanda güç vermektedir. Bu kazanımların yanı sıra kullanılan uygulamalardaki zafiyetlerin kötüye kullanılması durumunda ise birçok kayıplar yaşanmaktadır. Üçüncü taraf veya açık kaynak kodlu edinilen herhangi bir uygulamadaki zafiyet, çalışan diğer uygulama ve sistemlere zarar verebilmekte, iş sürekliliği ve yasal uyum gibi sorunlara yol açabilmektedir. Bu yüzden üçüncü taraf bir yazılım bileşeni yeterli kontrol ve denetimlere tabi tutulmadan şirketin güvenli sayılan alanlarına kabul edilmemelidir. Başta sıkı kontrol edilse bile sonradan tespit edilebilecek bir güvenlik zafiyetinin oluşup olmadığı da sürekli kontrol ve gözetim altında tutulmalıdır. Dışarıdan alınan ikili koda sahip bir uygulamada yer alan güvenlik zafiyetinin tespiti oldukça zordur. Güvenlik sağlamaya çalışan taraf kadar güvenlik açıklıklarını kullanarak güvenliği aşmaya çalışan tehlikeli oluşumlarda mevcuttur. Bu açıdan var olan veya sonradan oluşabilecek bir zafiyetin zamanında tespiti ve buna karşı alınacak önlemler riski önlemde kıymetlidir. Zafiyet kontrolünün manuel yapılması, saldırganların motivasyon ve kullandıkları yeni teknolojiler karşısında zayıf kalabilir. Otomatik araçlarla tarama manuel taramalara göre zafiyetlerin belirlenmesinde daha etkilidir. Güvenlik riskleri tespit ve önlemede kullanılacak teknolojik araçlar günün koşullarına uygun olmalıdır. Uygulama güvenliğine katkı sunan araçlar zafiyet ve risk tespitinde yararlıdır. Bu amaçla çalışmada ele alınan uygulama veya amaca uygun benzer bir araç kullanılabilir. Zafiyet veya güvenlik tarama testlerinin sadece yılın belli dönemlerde yapılması yeni bir taramaya kadar geçen süre içinde ortaya çıkan zafiyetler karşısında uygulama güvenliğini zaafa uğratabilir. Diğer taraftan güvenlik zafiyet ve açıklıklarının açık bir şekilde yayınlandığı birçok zafiyet veri tabanı olmasına rağmen bunların bir arada etkin bir şekilde kullanılmasında birtakım sorunlar vardır. Oysa bu zafiyetlerden kaynaklı yaşanmış risk ve ihlaller hakkında ise veri yok denecek kadar azdır. Dijital dünyanın bu denli hızlı geliştiği ve genişlediği bir ortamda kurumlar çok ciddi kayıplar yaşamaktadır. İşletmelerde kullanılan yazılımların büyük çoğunluğu üçüncü taraf veya açık kaynaklı yazılımlardan oluşmaktadır. Bunların birçoğu çalışır halde edinilen uygulamalardır. Bu uygulamaların zararlı herhangi bir bileşen barındırıp barındırmadığının manuel olarak bulunması oldukça güçtür, tam olarak tespiti mümkün değildir. Edinilen veya kullanılan kod kütüphanelerinde bir zafiyet olup olmadığının zamanında tespit edilebilmesi riskin önlenmesi ve güvenliğin sağlanması bakımından kritik öneme sahiptir.

Uygulama güvenlik araçları asgari ölçüde belli yeteneklere sahip olmalıdır. Yazılım bileşenlerini analiz ederken tescilli güvenlik zafiyet veri tabanlarını çevrimiçi kullanan, gelişen tehditlere karşı sürekli güncel tutan, makine öğrenme, yapay zekâ gibi güncel teknolojileri barındırmalıdır. Zafiyet analiz sonucunda elde edilen bulgulara ilişkin akıllı öneriler sunabilen, risk, güvenlik, denetim gibi tarafların temel ihtiyacını karşılayabilen raporlama, bildirim gibi yeteneklere sahip olmalıdır. Şirket ölçeğinde kullanılan bilgi güvenliği yönetim sistemi, risk yönetim sistemi, yönetişim gibi kurumsal uygulamalarla entegre çalışabilmeli, bu sitemlere tespit edilen bir zafiyeti çevrimiçi veri olarak sunulmalıdır. Bu çalışmada detaylı olarak incelenen ve uygulama pratikleri gerçekleştirilen *Black Duck* uygulaması üzerinde bir projede kullanılan yazılım bileşenine ait zafiyet taraması gerçekleştirilmiştir. Yürütülen çalışmanın aşamaları sunulmuştur. Bu tür bir uygulama kullanması, yazılım bileşenlerinden kaynaklı zafiyet ve risklerin erken teşhisi, riskin önlenmesi veya en az kayıpla atlatılabilmesi bakımından faydalı olacaktır. Yazılım geliştirme yaşam döngüsünde güvenlik zafiyet tarama süreçlerini otomatikleştirerek SQL açıklıkları ve siteler arası komut dosyası çalıştırma (XSS) gibi zafiyetlerin erken tespit edilmesi sağlanabilir. Böylece erken tespit edilen bir zafiyetin erken düzeltilmesi hem uygulama güvenliğine hem de yazılım süreçlerinin daha hızlı işletimine katkı sunabilir. Çalışmada incelenen uygulama güvenlik araçlarının birçoğu çalışılan uygulamayla benzer özellikleri taşımaktadır. Kurumlar uygulama seçimini kendi amaç, hedef, bütçelerine uygun herhangi bir uygulama güvenlik aracını tercih edebilirler.

Diğer taraftan kurumlar, birçok yasal düzenleme yanında uluslararası bilgi güvenliği yönetim standardı (ISO/IEC 27001), risk yönetim standardı (ISO/IEC 31000), iş sürekliliği yönetim standardı (ISO/IEC 22301), BT yönetişim ve denetim

(COBIT), CMMI, SPICE gibi kalite standartlarına ait sıkı kural ve gereksinimlere tabidir. Bağımsız otoriteler tarafından bu kurallara uyuma ilişkin yıl içinde pek çok kontrol ve denetimler gerçekleştirilmektedir. Bu kontrol ve denetimlerde sürece ilişkin detaylı veri ve iz kayıtlarına ihtiyaç duyulmaktadır. Bu tür isteklerin karşılanmasında, uygulama güvenlik ve denetimine yönelik uygulamaların kullanılması denetlenen tarafın denetim esnasında sunumunu kolaylaştıracak ve denetim süresini kısaltacaktır. Kontrollerin sistem üzerinden uçtan uça yapılabilir olması denetleyen taraflar içinde kolaylık ve güvence sağlayacaktır. Manuel gerçekleşen süreçler, genellikle zafiyetlerin nasıl tespit edildiği, buna karşın risk kaydının tutulup tutulmadığı, kontrol ve önlem alınıp alınmadığı, zafiyetin bir olaya sebebiyet verip vermediği, kimin neyi, ne zaman, nerede, ne yapıp/yapmadığına kanıt teşkil edecek güvenilir veri ve delillerden yoksundur. Süreçte herhangi bir kayıt yapısı bulunmaz, bu nedenle gerçekleştirilen aktivitenin kayıt altında tutulması için ekran görüntülerinin manuel alınıp güvenli saklanması bile istenilebilmektedir. Bu tür manuel gerçekleştirilen süreçlerde denetlenen ve denetleyen taraf için fazladan harcanan işgücü, çaba ve emek demektir. Kontrol ve denetimlerin etkin, verimli ve şeffaf yürütülebilmesi için yenilikçi teknoloji ve uygulamalar günümüzde olmazsa olmaz yardımcı araçlardır.

6. SONUÇ VE ÖNERİLER

Bu çalışmada, üçüncü taraf yazılım bileşen zafiyetlerine dikkat çekilerek bunlardan kaynaklı riskleri gidermenin önemi ve yöntemi vurgulanmıştır. Uygulamalarda yer alan zafiyetlerin tespit, analiz, kontrol ve yönetiminde bilinen güncel uygulama güvenlik test araçları incelenerek özetlenmiştir. Uygulama güvenlik test araçlarını kullanmanın sağladığı faydalara değinilmiş, manuel yapılmasının zorlukları vurgulanmıştır. Çalışmanın uygulama aşamasında, seçilen örnek bir yazılım bileşeninin Gartner 2022 araştırma raporunda lider olan *Black Duck* uygulaması üzerinde gerçekleştirilmiştir. Bu pratik çalışmadan elde edilen deneyim ve gözleme dair sonuçların literatüre kazandırılması amaçlanmıştır.

Yazılım bileşen zafiyetlerinin tespitinde kullanılan araçlar bileşen yaşam döngüsünde son derece önemli bir rol oynamaktadır. Üçüncü taraf yazılım bileşen zafiyetlerinin tespitinde kullanılan uygulamalar, uygulama amaç ve kapsamı, tarama türü, çalışma ortamı, desteklediği programlama dilleri, kullanımı, sunduğu rapor ve yetkinlikleri, ürün maliyetleri gibi farklı niteliklere sahiptir. Bu çalışmada seçilen uygulama detaylı olarak incelenmiş, mimarisi, çalışma şekli, uygulama üzerinden proje oluşturma, veri yükleme, tarama, analiz, yorumlama, risk yönetimi, raporlama ve diğer sistemlerle entegrasyon yetenekleri uygulamalı olarak çalışılmıştır. Bu yönüyle benzer amaçla kullanılan diğer uygulamalar içinde bir örnek teşkil etmektedir. Kurumlar, üçüncü taraf yazılım bileşenlerinin testinde bu tür uygulama araçlarını kullanarak yazılımdan kaynaklı zafiyetlerini etkin biçimde tespit edebilirler. Kurumların ürettikleri veya kullandıkları yazılımlardaki güvenlik zafiyetlerini biliyor olmaları, bunlardan kaynaklı risklerin önlenmesine, güvenlik, risk yönetimi, iç kontrol ve denetim faaliyetlerinin etkin yürütülebilmesine, yasal uyum kontrol faaliyetlerinin daha hızlı ve sorunsuz gerçekleşmesine pozitif katkı sunacaktır. Ayrıca yazılım geliştiricilere, kullanmayı düşündükleri ancak içerdiği kodu tam bilemedikleri bir üçüncü taraf yazılım bileşeninin güvenli olup olmadığına dair belli bir güvence sunacaktır. Bu tür uygulamaların kullanımı yazılım üretim ve dağıtım süreçlerinin daha hızlı ve güvenli bir şekilde işletilmesine olanak sağlayacaktır. Her geçen gün teknoloji ve üzerindeki yazılım bileşenlerine daha bağımlı hale gelen şirketlerde bu tür uygulamaların kullanımı, güvenlik yönetimi, risk yönetimi, denetim ve kontrol birimlerin çalışma verimliliğine de katkı sunacaktır. Nihayetinde işletmelerin yoğun kullanılan üçüncü taraf ürün zafiyetinden kaynaklı risklere karşı etkin bir kontrol mekanizması kurmalarına yardımcı olacaktır. Bu kontrol ve önlemler sonucunda işletmelerin güven, itibar gibi birçok alanda rekabet üstünlüğü elde ederek daha kazançlı hale geleceği öngörülmektedir.

Telekom, banka, sigorta, enerji, savunma gibi sektörler düzenleyici kurumlar tarafından sıkı kural ve denetime tabidir. Bu kurumlarda iç kontrol, iç denetim, bağımsız dış denetim, kalite denetimi gibi birçok kontrol ve denetim faaliyeti yürütülmektedir. Sık gerçekleşen bu denetim ve kontrol faaliyetleri bilgi teknolojileri tarafında ciddi iş yüküne sebep olmaktadır. Yöneticiler bu tür süreç tabanlı otomatik dağıtım araçlarını kullanarak denetimde harcanan süreyi kısaltmış olurlar. Böylece denetim süreleri kısalmış ve bu işlemlerde kullanılan insan kaynağı kendi uzmanlık alanlarına daha fazla zaman ayırmış olur. Özetle, bu uygulamaların varlığı, denetimlerin daha hızlı, şeffaf ve güvenilir sonuçlar üretmesine, uygulama zafiyetinden kaynaklı güvenlik risklerinin etkin yönetimine katkı sunar. Bu işletmelere güven, çeviklik, hız, rekabet ve karlılık gibi alanlarda birtakım üstünlükler kazandır.

Bu çalışma kurumsal işletmelerde, yazılım geliştirme ve operasyon ekiplerine, güvenlik, risk, denetim ve yönetişim ekiplerine, bu alanlara yönelmek veya uzmanlaşmak isteyen teknik çalışanlara, öğrencilere, araştırmacılara yardımcı bir kaynak niteliğindedir. Aynı zamanda üçüncü taraf bileşenlerden kaynaklı güvenlik zafiyetleri ve uygulamalarına ilişkin literatüre katkı sağlamaktadır.

Kaynakça

Arslan, Y. & Özbilger, H. İ. (2022). Ulusal mevzuat perspektifinde bilgi işlem birimlerinin iç denetiminde bir kontrol listesi önerisi. *Denetişim Dergisi*, 1-12. Retrieved from https://dergipark.org.tr/tr/pub/denetisim/issue/73424/1126790

Bozoklu, O. & Çil, C.Z. (2017). Yazılım güvenlik açığı ekosistemi ve Türkiye'deki durum değerlendirmesi. *Uluslararası Bilgi Güvenliği Mühendisliği Dergisi*, 3 (1) , 6-26. Doi: 10.18640/ubgmd.303598

Cadariu, M., Bouwers, E., Visser, J., Deursen, A. (2015). Tracking known security vulnerabilities in proprietary software systems. *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering, SANER 2015* - Proceedings. 516-519. 10.1109/SANER.2015.7081868.

Cobleigh, A., Hell, M., Karlsson, L., Reimer, O., Sönnerup, J., & Wisenhoff, D. (2018). Identifying, prioritizing and evaluating vulnerabilities in third party code. *2018 IEEE 22nd International Enterprise Distributed Object Computing Workshop (EDOCW)*, 208-211.

Dennig, F., Cakmak, E., Plate, H., Keim, D.A. (2021). Vulnex: exploring open-source software vulnerabilities in large development organizations to understand risk exposure. *Software Engineering*, https://doi.org/10.48550/arXiv.2108.06259.

Fowler, M. (2006). Continuous integration. https://martinfowler.com/articles/continuousIntegration.html adresinden alındı. (Erişim Tarihi, 06 Kasım 2022).

Kekül, H., Ergen, B. & Arslan, H. (2021). Yazılım güvenlik açığı veri tabanları. *Avrupa Bilim ve Teknoloji Dergisi*, Ejosat Özel Sayı 2021 (ICAENS), 1008-1012 . Doi: 10.31590/ejosat.1012410

Keskinkılıç, M. & Özmen, E. (2018). Yazılım projelerinde yazılım geliştiricilerin yazılım süreç modellerini kullanım farkındalıkları. *Akademi Sosyal Bilimler Dergisi*, 5(15), 61-78, https://dergipark.org.tr/tr/pub/asbider/issue/41362/500096.

Laurent, A.M. (2004). *Understanding open source and free software licensing*. O'Reilly Media, Inc

Nikbakht Bideh, P., Höst, M., & Hell, M. (2018). HAVOSS: A maturity model for handling vulnerabilities in third party oss components. *In Product-Focused Software Process Improvement* (pp. 81-97). (Lecture Notes in Computer Science; Vol. 11271). Springer. https://doi.org/10.1007/978-3-030-03673-7_6

Ozment, A.(2007). *Vulnerability discovery & software security*. Doctoral thesis. University of Cambridge

Pawlan, D. (2021). What is a code repository? https://aloa.co/blog/what-is-code-repository-best-code-repo-hosting-2021#:~:text=A%20code%20repository%20is%20an,any%20successful%20software%20development%20project. (Erişim Tarihi, 15 Kasım 2022).

Rebrošová, P. (2018). Gathering Vulnerability Information Published by Software Manufacturers. *Bachelor's Thesis*, Faculty of Informatics, Masaryk University.

Ritchey, T., Lövkvist-Andersen, A., Olsson, R., Stenström, M. (2004). *Modelling society's capacity to manage extraordinary events developing a generic design basis (GDB) model for extraordinary societal events using computer-aided morphological analysis.*

Sahinaslan, E. (2019). On the internet of things: Security, threat and control. *AIP Conference Proceedings* 2086, 030035; https://doi.org/10.1063/1.5095120.

Sahinaslan, O., & Sahinaslan, E. (2019, 04 02). Cross-object information security: A study on new generation encryption. *AIP Conference Proceedings* 2086, 030034; https://doi.org/10.1063/1.5095119

Şahinaslan, E. , Arpacıoğlu, N. & Şahinaslan, Ö. (2023). Yazılım Dağıtım Sürecinin Otomatikleştirilmesine İlişkin Uygulamalı Bir Çalışma. *Uluslararası Yönetim Bilişim Sistemleri ve Bilgisayar Bilimleri Dergisi*, 7 (1) , 41-67. Doi: 10.33461/uybisbbd.1206484

Şahinaslan, E. (2010). Bankacılık sektörüne yönelik ISO 27001 temelli bir bilgi güvenliği risk analiz ve ölçümleme metodunun modellenmesi ve uygulama yazılımının gerçekleştirilmesi. Doktora Tezi, *Fen Bilimleri Enstitüsü*, Trakya Üniversitesi.

Şahinaslan, Ö., Borandağ, E., & Aksoy, Ş. (2011). Web tabanlı uygulamalarda performansı etkileyen unsurlar. *XIII. Akademik Bilişim Konferansı* (s. 599-604). İnönü Üniversitesi.

Şahinaslan, Ö. (2013). Siber saldırılara karşı kurumsal ağlarda oluşan güvenlik sorunu ve çözümü üzerine bir çalışma. Doktora Tezi, *Fen Bilimleri Enstitüsü*, Trakya Üniversitesi.

Zhan, X., Fan, L., Chen, S., Wu, F., Liu, T., Luo, X., & Liu, Y. (2021). ATVHunter: Reliable Version Detection of Third-Party Libraries for Vulnerability Identification in Android Applications. *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, 1695-1707.

İnternet Kaynakları

Blackduck (2022). Black duck software composition analysis, https://www.synopsys.com/software-integrity/security-testing/software-composition-analysis.html. (Erişim Tarihi, 24 Kasım 2022).

Black Duck Documentation (2022), Black Duck on-premise architecture overview, https://community.synopsys.com/s/document-item?bundleId=bd-hub&topicId=Network_Communications/ ArchitectureOverview.html&_LANG=enus. (Erişim Tarihi, 24 Kasım 2022).

Checkmarx (2022). Why checkmarx. https://checkmarx.com/. (Erişim Tarihi, 12 Eylül 2022).

Contrast-Security (2022). https://www.contrastsecurity.com/. (Erişim Tarihi, 20 Kasım 2022).

CNVD (2022). https://www.cnvd.org.cn/. (Erişim Tarihi, 10 Ekim 2022).

CVE (2022). Search CVE list, https://cve.mitre.org/. (Erişim Tarihi, 02 Ağustos 2022).

DataTheorem (2022). https://www.datatheorem.com/. (Erişim Tarihi, 27 Temmuz 2022).

EDB (2022). Exploit database, https://www.exploit-db.com/. (Erişim Tarihi, 25 Ekim 2022).

Gartner (2022). 2022 Gartner magic quadrant for application security testing. https://www.synopsys.com/software-integrity/resources/analyst-reports/gartner-magic-quadrant-appsec.html (Erişim Tarihi, 24 Ekim 2022).

GitHub (2022). Features. https://github.com/features/security. (Erişim Tarihi, 25 Ekim 2022).

GitLab (2022). A comprehensive software innovation platform. https://about.gitlab.com/platform/?stage=plan. (Erişim Tarihi, 25 Ekim 2022).

HLC Software (2022). Enterprise security. https://www.hcltechsw.com/enterprise-security. (Erişim Tarihi, 17 Eylül 2022).

Invicti (2022). Produst overview, https://www.invicti.com/product/. (Erişim Tarihi, 20 Ekim 2022).

İTU (2022). Snippet kullanımı. https://bidb.itu.edu.tr/seyir-defteri/blog/2013/09/06/snippet-kullan%C4%B1m%C4%B1. (Erişim Tarihi, 28 Ekim 2022).

JFrog (2022). What is a software vulnerability? https://jfrog.com/knowledge-base/software-vulnerability/. (Erişim Tarihi, 25 Ekim 2022).

JVN (2022). Japan vulnerability notes. https://jvn.jp/en/. (Erişim Tarihi, 30 Eylül 2022).

Micro Focus (2022). Application security. https://www.microfocus.com/en-us/cyberres/application-security. (Erişim Tarihi, 25 Ekim 2022).

NTT (2022), Application security testing. https://services.global.ntt/cs-cz/services-and-products/security/application-security/application-security-testing. (Erişim Tarihi, 26 Ekim 2022).

NVD (2022). National vulnerability database, https://nvd.nist.gov/. (Erişim Tarihi, 25 Kasım 2022).

Onapsis (2022). Automated security testing designed for SAP applications. https://onapsis.com/solutions/application-security-testing. (Erişim Tarihi, 25 Ekim 2022).

Rapid (2022). Vulnerability & exploit database. https://www.rapid7.com/db/. (Erişim Tarihi, 17 Ekim 2022).

Rapid7 (2022). Dynamic application security testing. https://www.rapid7.com/products/insightappsec/. (Erişim Tarihi, 25 Ekim 2022).

Rosencrance, L. (2021). What is a vulnerability assessment?, www.techtarget.com/ searchsecurity/definition/vulnerability-assessment-vulnerability-analysis (Erişim Tarihi, 13 Ekim 2022).

Synopsis (2022). Binary code and binary analysis. https://www.synopsys.com/glossary/what-is-binary-code-binary-analysis.html. (Erişim Tarihi, 25 Ekim 2022).

Sykn (2022. Open source risk management made for developers. https://snyk.io/solutions/application-security/. (Erişim Tarihi, 25 Ekim 2022).

Veracode (2022). Veracode solution. https://www.veracode.com/. (Erişim Tarihi, 20 Ekim 2022).