T.C.
SİVAS CUMHURİYET ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

YAZILIM MALZEME LİSTELERİNE DAYALI GÜVENLİK AÇIĞI ANALİZİ VE CI/CD SÜREÇLERİNDE OTOMATİK GÜVENLİK AÇIĞI TARAMASI İÇİN YENİ BİR MODEL

YÜKSEK LİSANS TEZİ

Ömercan KAĞIZMANDERE

(20219257001)

Bilgisayar Mühendisliği Ana Bilim Dalı

Tez Danışmanı: Dr. Öğr. Üyesi Halil ARSLAN

SİVAS

TEMMUZ 2024

Ömercan KAĞIZMANDERE'nin hazırladığı ve **"YAZILIM MALZEME LİSTELERİNE DAYALI GÜVENLİK AÇIĞI ANALİZİ VE CI/CD SÜREÇLERİNDE OTOMATİK GÜVENLİK AÇIĞI TARAMASI İÇİN YENİ BİR MODEL"** adlı bu çalışma aşağıdaki jüri tarafından **BİLGİSAYAR MÜHENDİSLİĞİ ANA BİLİM DALI**'nda **YÜKSEK LİSANS TEZİ** olarak kabul edilmiştir.

Tez Danışmanı	**Dr. Öğr. Üyesi Halil ARSLAN** Sivas Cumhuriyet Üniversitesi	……
Jüri Üyesi	**Dr. Öğr. Üyesi Hakan KEKÜL** Sivas Cumhuriyet Üniversitesi	……..……
Jüri Üyesi	**Dr. Öğr. Üyesi Ali DURDU** Ankara Sosyal Bilimler Üniversitesi	……..……..

Bu tez, Sivas Cumhuriyet Üniversitesi Fen Bilimleri Enstitüsü tarafından **YÜKSEK LİSANS TEZİ** olarak onaylanmıştır.

Prof. Dr. Nevcihan GÜRSOY
FEN BİLİMLERİ ENSTİTÜSÜ MÜDÜRÜ

Bu tez, Sivas Cumhuriyet Üniversitesi Senatosu'nun 20.08. 2014 tarihli ve 7 sayılı kararı ile kabul edilen Fen Bilimleri Enstitüsü Lisansüstü Tez Yazım Kılavuzuna (Yönerge)'nda belirtilen kurallara uygun olarak hazırlanmıştır.

ETİK

Sivas Cumhuriyet Üniversitesi Fen Bilimleri Enstitüsü, Tez Yazım Kılavuzu (Yönerge)'nda belirtilen kurallara uygun olarak hazırladığım bu tez çalışmasında;

- ✓ Bütün bilgi ve belgeleri akademik kurallar çerçevesinde elde ettiğimi,
- ✓ Görsel, işitsel ve yazılı tüm bilgi ve sonuçları bilimsel ahlak kurallarına uygun olarak sunduğumu,
- ✓ Başkalarının eserlerinden yararlanılması durumunda ilgili eserlere, bilimsel normlara uygun olarak atıfta bulunduğumu ve atıfta bulunduğum eserlerin tümünü kaynak olarak gösterdiğimi,
- ✓ Bütün bilgilerin doğru ve tam olduğunu, kullanılan verilerde herhangi bir değişiklik yapmadığımı,
- ✓ Tezin herhangi bir bölümünü, Sivas Cumhuriyet Üniversitesi veya bir başka üniversitede, bir başka tez çalışması olarak sunmadığımı; beyan ederim.

08/07/2024

Ömercan KAĞIZMANDERE

KATKI BELİRTME VE TEŞEKKÜR

Bu tez çalışmasını hazırlama sürecimde desteğini esirgemeyen değerli aileme, yüksek lisans eğitimim boyunca tez danışmanlığımı üstlenerek yardımcı olan hocam Dr. Öğr. Üyesi Halil ARSLAN'a çok teşekkür ederim.

ÖZET

YAZILIM MALZEME LİSTELERİNE DAYALI GÜVENLİK AÇIĞI ANALİZİ VE CI/CD SÜREÇLERİNDE OTOMATİK GÜVENLİK AÇIĞI TARAMASI İÇİN YENİ BİR MODEL

Kağızmandere, Ömercan

Bilgisayar Mühendisliği Anabilim Dalı

Danışman: Dr. Öğr. Üyesi Halil ARSLAN

2024, xi+57 Sayfa

Yazılım malzeme listesi, yazılım güvenliği ve yazılım tedarik zinciri yönetiminde önemli bir bileşen olarak 2018 yılında ortaya çıkmıştır. Yazılım malzeme listesi, yazılımları oluşturan bileşenlerin bir listesi olarak sunulan bir envanterdir. Son yıllarda yazılım ürünlerinin zafiyet içerip içermediği o ürünün kullanıcıları açısından düzenli olarak kontrol edilmesi gereken bir olgudur. Bu çalışma, yazılım malzeme listesi kavramı temelinde yazılım bileşenlerinin sistematik bir şekilde belirlenmesi ve bu bileşenler üzerinden güvenlik açığı analizlerinin yapılmasını ele almaktadır. Bir yazılım ürününün kendisinin güvenlik açığı içermemesi o yazılım ürününün güvenli olduğu anlamına gelmez. Yazılım projeleri tek başına incelendiğinde herhangi bir güvenlik açığı içermese de bileşenlerinde güvenlik açıkları olabilir. Ürünün bağımlılıklarında ya da bileşenlerinde yer alan güvenlik açıkları siber saldırganlar için o ürünün istismar edilmesi için yeterli olabilmektedir. Yazılım bileşenlerinden kaynaklı güvenlik açıklarının yol açtığı tahribatı en aza indirmek, siber güvenlik çalışmalarının temelini oluşturur. Bu çalışmada, yazılım geliştirme/dağıtım ortamlarında (CI/CD) yazılım malzeme listesinin (SBOM) otomatik olarak üretilmesinin ve bu malzeme listesi üzerinden zafiyet analizinin yapılmasının gerekliliği gösterilmiş ve buna uygun bir model önerilmiştir.

Anahtar kelimeler: SBOM, Yazılım Malzeme Listesi, Güvenlik Açığı Analizi, Yazılım Güvenliği

ABSTRACT

A NEW MODEL FOR VULNERABILITY ANALYSIS BASED ON SOFTWARE BILL OF MATERIALS AND AUTOMATED VULNERABILITY SCANNING IN CI/CD PROCESSES

Kağızmandere, Ömercan

Department of Computer Engineering

Advisor: Asst. Prof. Dr. Halil ARSLAN

2024, xi+57 pages

The software bill of materials (SBOM) emerged in 2018 as an important component in software security and software supply chain management. SBOM is an inventory presented as a list of the components that make up software. In recent years, whether software products contain vulnerabilities is a phenomenon that should be checked regularly by the users of that product. This paper deals with the systematic identification and vulnerability analysis of software components based on the concept of software bill of materials. The fact that a software product itself does not contain vulnerabilities does not mean that the software product is secure. Even if software projects do not contain any vulnerabilities when examined alone, there may be vulnerabilities in their components. Vulnerabilities in the dependencies or components of the product may be sufficient for cyber attackers to exploit that product. Minimizing the damage caused by vulnerabilities in software components is the basis of cyber security efforts. In this study, the necessity of automatically generating software bill of materials in software development/deployment environments (CI/CD) and performing vulnerability analysis on this bill of materials is demonstrated and a suitable model is proposed.

Keywords: Software Bill of Materials, Vulnerability Analysis, Software Security

İÇİNDEKİLER

Sayfa

ÖZET ..... **Vİ**

ABSTRACT ..... **Vİİ**

İÇİNDEKİLER ..... **Viİİ**

ŞEKİLLER LİSTESİ ..... **X**

ÇİZELGELER DİZİNİ ..... **Xİ**

1. GİRİŞ **1**

1.1 Araştırma Soruları 2

1.2 Araştırmanın Önemi 2

1.3 Araştırmanın Amacı 3

2. LİTERATÜR TARAMASI **4**

3. ISO 25010 KALİTE MODELİ VE GÜVENLİK KRİTERİ **9**

3.1 Fonksiyonel Uygunluk 9

3.2 Güvenirlik 9

3.3 Kullanılabilirlik 10

3.4 Uyumluluk 10

3.5 Performans Etkinliği 10

3.6 Güvenlik 10

3.7 Taşınabilirlik 11

3.8 Bakım Yeteneği ve Korunabilirlik 11

4. YAZILIM MALZEME LİSTESİ **12**

4.1 SBOM Oluşturma 13

4.2 SBOM Formatları 14

4.2.1 SPDX 14

4.2.2 CycloneDX 15

4.2.3 Syft 16

4.3 Yazılım Tedarik Zinciri Güvenliğinde SBOM'un Rolü 16

4.4 SBOM'un Faydaları 17

5. METODOLOJİ **19**

5.1 GIT 21

5.1.1 GitHub 22
5.2 Jenkins 24
5.3 SBOM Tool 25
5.3.1 Bileşen Algılama / Component Detection 25
5.4 Bomber 27
5.4.1 Güvenlik Açığı 28
5.4.2 Zafiyet Değerlendirme Kavramı 29
6. DENEYSEL ÇALIŞMALAR 33
6.1 Deneysel Çalışmada Kullanılan Projelerde Bulunan Bulgular 34
6.1.1 OpenRA 34
6.1.2 Runtime 35
6.1.3 Efcore 37
6.1.4 Abp 38
6.1.5 Aspnetboilerplate 39
6.1.6 Captura 40
6.1.7 Spectre.console 42
6.1.8 RestSharp 43
6.1.9 Radarr 44
6.1.10 Ocelot 45
7. SONUÇLAR VE TARTIŞMA 48
KAYNAKÇA 51

ŞEKİLLER LİSTESİ

Sayfa

Şekil 3.1 ISO/IEC 25010 Kalite Modeli 9
Şekil 4.1 OpenRA projesi için SPDX örneği 15
Şekil 4.2 CycloneDX XML Format 16
Şekil 5.1 Önerilen Model 20
Şekil 5.2 Örnek Bileşen Tespiti 26
Şekil 5.3 CVSS V2.0 Ratings 30
Şekil 5.4 CVSS V3.0 Ratings 30
Şekil 5.5 CVE Yapısı 30
Şekil 6.1 Bomber üzerinden alınan örnek zafiyet analiz raporu 33
Şekil 6.2 OpenRA bağımlılık listesi 35
Şekil 6.3 OpenRA için bulunan zayifetler 35
Şekil 6.4 Runtime bağımlılık listesi 36
Şekil 6.5 Runtime için bulunan zafiyetler 36
Şekil 6.6 Efcore bağımlılık listesi 37
Şekil 6.7 Efcore için bulunan zafiyetler 38
Şekil 6.8 Abp bağımlılık listesi 38
Şekil 6.9 Abp için bulunan zafiyetler 39
Şekil 6.10 Aspnetboilerplate bağımlılık listesi 39
Şekil 6.11 Aspnetboilerplate için bulunan zafiyetler 40
Şekil 6.12 Captura bağımlılık listesi 41
Şekil 6.13 Captura için bulunan zafiyetler 41
Şekil 6.14 Spectre.console bağımlılık listesi 42
Şekil 6.15 Spectre.console için bulunan zafiyetler 42
Şekil 6.16 RestSharp bağımlılık listesi 43
Şekil 6.17 RestSharp için bulunan zafiyetler 44
Şekil 6.18 Radarr bağımlılık listesi 44
Şekil 6.19 Radarr için bulunan zafiyetler 45
Şekil 6.20 Ocelot bağımlılık listesi 46
Şekil 6.21 Ocelot için bulunan zafiyetler 46

ÇİZELGELER DİZİNİ

Sayfa

Çizelge 5.1 Bileşen algılamanın desteklediği kütüphaneler 27
Çizelge 6.1 Deneysel çalışma için seçilen projeler 33
Çizelge 7.1 İncelenen projelerin zafiyet listesi.....48

1. GİRİŞ

Günümüz yazılım projeleri, çok farklı sistemlerle entegre olmak durumundadır ve farklı yazılım kütüphanelerine bağımlıkları vardır. Bu nedenle bir sistemde bulunan güvenlik açığı sömürülerek o sistem üzerinden diğer sistemler de tehdit edilebilir. Diğer bir deyişle yazılım sistemlerinde projenin kendinde bir güvenlik açığı olmaması demek o projenin güvenli olduğunu göstermemektedir. Projenin bağımlılıklarında güvenlik açığı olması durumunda ana projede de güvenlik açığı var demektir. Saldırılar normal bir yazılım tedarik zincirinin herhangi bir noktasında gerçekleşebilir ve bu saldırılar günümüz dünyasında daha görünür, yıkıcı ve pahalı hale gelmektedir. Bunun önüne geçebilmek için bir projenin bağımlılıklarını öğrenmek kritik seviyede önemli hale gelmiştir.

Yazılım ürünlerinin kalitesini değerlendirmek için kılavuz ve öneriler sağlamak için geliştirilen ISO 25010 kalite modeli, 8 ana ve 31 alt kriterden oluşan hiyerarşik bir yapıdır (ISO/IEC 25010:2011). ISO 25010 içerisinde güvenlik (security) 8 ana kriterden biridir. Güvenlik, bir ürünün veya sistemin bilgileri ve verileri koruduğunu gösteren kalite karakteristiğidir (Peters vd., 2020), (Pratama vd., 2021). Bir yazılım ürününün ISO 25010 Kalite Modeli bağlamında Güvenlik kategorisinin ele alınması sadece mevcut ürünün incelenmesiyle ortaya konulamamaktadır. Yazılım ürününün bağımlılıklarında ortaya çıkabilecek bir zafiyet ilgili ürünün güvenlik parametrelerini de riske atmaktadır. Bu bağlamda bir yazılım ürününün bağımlılıklarının belirlenmesi, bu bağımlılıkların içerdiği zafiyetlerin incelenebilmesi ve hem yazılım geliştiricilere hem de ilgili tüketicilere bir yazılım ürününe ilişkin şeffaflık ve görünürlük sağlaması açısından SBOM (Software Bill of Materials) önerilmiştir (Arora vd., 2023). Dolayısıyla bir projenin SBOM'u, o projenin bağımlılıklarını görebilmenin en güncel yaklaşımıdır. SBOM, bir yazılımı oluşturan bileşenlerin bir listesidir.

Bu çalışma ile görünürde hiçbir güvenlik açığı olmayan projelerin bağımlılıklarında var olan güvenlik açıklıklarından kaynaklı zafiyetler barındırabileceklerinin ispatlanması amaçlanmaktadır. Bu amaca yönelik uçtan uca bir model oluşturulması hedeflenmektedir. Bu modelin, bir uygulamanın geliştirme aşamasından canlı kullanıma alınması (CI/CD – Continuous Integration / Continuous Delivery)

aşamasına kadar gerçekleşen iş hattının (pipeline) bir parçası olması gerektiği ifade edilmiştir. Sonraki çalışmalarda ortaya konulan modelin CI/CD ortamlarına bir adım olarak eklemlenmesi sağlanarak ilgili güvenlik analizlerinin yazılım geliştirme süreçlerinde dikkate alınması sağlanacaktır. Bu çalışmada öncelikle, bir yazılım ürününün kullandığı bağımlılıkların takibi ve dokümantasyonu için dinamik olarak SPDX (Software Package Data Exchange – Yazılım Paketi Veri Değişimi) standardında SBOM'unun üretilmesi gerçekleştirilmiştir. SPDX, yazılım paketlerini ve bağımlılıklarını tanımlamak için kullanılan endüstriyel bir standarttır. Tüm satıcılar, programlama dilleri ve çerçevelerle çalışmak üzere tasarlanmıştır (Butler vd., 2022) Çalışma kapsamında, yaygın geliştirici ve kullanıcı desteği olan açık kaynak 10 farklı proje incelenmiştir. Dinamik olarak SBOM'ları çıkarılan projelerin daha sonra ürün ve bağımlılık bazlı zafiyet analizleri gerçekleştirilmiştir. Sonuç olarak, ister açık kaynak, isterse de derlenmiş kod olarak dağıtımı yapılan projelerin, SBOM verilerinin hem ISO 25010 hem de siber güvenlik açısından incelenmesi gereken en önemli parametrelerden biri olabileceği gösterilmiştir.

1.1 Araştırma Soruları

Yukarıdaki tanımlara göre, çalışmamızın temel araştırma soruları aşağıdaki gibidir;

AS-1. SBOM'lar kaynak koddan otomatik olarak üretilebilir mi?
AS-2. Otomatik olarak çıkarılan SBOM listeleri aracılığıyla kaynak kod bağımlılıklarının güvenlik açıkları bulunabilir mi?
AS-3. Mevcut ürün ve bağımlılıkları arasında güvenlik açıkları üzerinden bir korelasyon/ilişki var mıdır?

1.2 Araştırmanın Önemi

Bir yazılım ürünü, hedeflenen görevleri gerçekleştirmek için pek çok alt bileşene/bağımlılığa ihtiyaç duymaktadır. Bu bağımlılıklar, kullanılan yazılım ürününün kendisi kadar ürün bütünlüğünü güvenlik açısından etkilemektedir. Bu nedenle bir sistemde bulunan güvenlik açığı sömürülerek o sistem üzerinden diğer sistemler de tehdit edilebilir. Diğer bir deyişle yazılım sistemlerinde projenin kendinde bir güvenlik açığı olmaması demek o projenin güvenli olduğunu göstermemektedir. Projenin bağımlılıklarında güvenlik açığı olması durumunda ana projede de güvenlik açığı var demektir. Saldırılar normal bir yazılım tedarik

zincirinin herhangi bir noktasında gerçekleşebilir ve bu saldırılar günümüz dünyasında daha görünür, yıkıcı ve pahalı hale gelmektedir. Bunun önüne geçebilmek için bir projenin bağımlılıklarını öğrenmek kritik seviyede önemli hale gelmiştir. Bağımlılıklarda bulunan bir güvenlik açığının ana projeye büyük zararlar vermesi söz konusudur. Bunu engellemek için yöntemler sunmak araştırmanın önemini artırmaktadır.

1.3 Araştırmanın Amacı

Bir SBOM'un birincil amacı, yazılım geliştirme ve yönetim süreçlerinde şeffaflığı, izlenebilirliği ve güvenliği artırmaktır (NTIA, 2021). SBOM'lar, belirli bir ağdaki her bir yazılım bileşeninin kökenine ilişkin bilgiler sağlar. Potansiyel güvenlik açıklarını tespit etmek, lisans uyumluluğunu değerlendirmek ve tedarik zinciri risklerini yönetmek için kullanılabilirler. SBOM'lar, yazılım geliştirme uygulamalarının güvenlik, uyumluluk ve risk yönetimi uygulamalarıyla yakın uyum gerektirdiği sektörlerde büyük önem taşımaktadır (Bauer vd., 2020).

Bu çalışma ile görünürde hiçbir güvenlik açığı olmayan projelerin bağımlılıklarında var olan güvenlik açıklıklarından kaynaklı zafiyetler barındırabileceklerinin ispatlanması amaçlanmaktadır. Bu amaca yönelik uçtan uca bir model oluşturulması hedeflenmektedir.

2. LİTERATÜR TARAMASI

Yazılım Malzeme Listesi (SBOM), yazılım geliştirmenin ayrılmaz bir parçası olan bileşenlerin ve bağımlılıkların ayrıntılı bir envanterini sağlayarak yazılım tedarik zinciri güvenliğinin sağlanmasında kritik bir görevi yerine getirir. Bununla birlikte, SBOM'ların paylaşımında, potansiyel veri tahrifatı ve yazılım satıcıları arasında kapsamlı bilgilerin ifşa edilmesi konusundaki tereddütler de dahil olmak üzere çok sayıda zorluk bulunmaktadır. Bu engeller, SBOM'ların yaygın biçimde benimsenmesini ve kullanılmasını engellese de SBOM paylaşımı için daha güvenli ve esnek bir mekanizmaya olan ihtiyacı ortaya çıkarmaktadır. Bu durumdan hareketle yapılan SBOM çalışmalarının ayrıntıları incelenmiştir.

Axelsson vd., açık kaynak yazılım (OSS) topluluğunun SBOM'ları benimsemede ne kadar ilerlediğini ve Yazılım Paketi Veri Değişimi (SPDX) formatına odaklanarak mevcut SBOM'ların nasıl geliştiğini araştırmıştır. Bu araştırmanın amacı doğrultusunda GitHub'daki OSS projelerinde SBOM'ların arandığı, içeriklerinin ve evrimlerinin analiz edildiği bir arşiv çalışması yapmışlardır (Axelsson vd., 2023).
Adewumi vd., mevcut OSS kalite değerlendirme modellerini kalite özelliklerine, kullandıkları metodolojiye ve uygulama alanlarına göre sınıflandırarak yeni modellerin formülasyonuna ve geliştirilmesine rehberlik etmek için sistematik bir literatür taraması yapmışlardır (Adewumi vd., 2015).

Stoddard vd., araştırmalarında SBOM üretimi konusunda önemli topluluk tartışmaları olsa da SBOM paylaşımına odaklanan tartışmaların daha az olduğunu belirtmişlerdir. Çalışmaları, halihazırda kullanılan SBOM paylaşım çözümlerini vurgulamak ve okuyucuların SBOM'ların keşfi, erişimi ve taşınmasıyla ilgili ihtiyaçlarına göre uygun paylaşım çözümlerini keşfetmelerine yardımcı olmaktadır (Stoddard vd., 2023).

Camp vd., SBOM standartlarını temel alarak sistematik güvenlik açığı sınıflarına odaklanmışlardır. Yaptıkları çalışma sonucunda, SBOM'un yalnızca tedarik zincirini güvence altına almakla kalmayıp aynı zamanda yazılım bileşenlerini tanımlayabilme konusundaki sorunlara karşı umut verici bir çözüm olduğunu dile getirmişlerdir (Camp vd., 2021).

Xia vd., SBOM paylaşımı için blok zinciri (blockchain) destekli bir mimari sunarak, SBOM'un kapsamını yapay zekâ sistemlerini kapsayacak şekilde genişletmişlerdir. Böylece Yapay Zekâ Malzeme Listesi (AIBOM) terimini türetmişlerdir. Önerilen SBOM paylaşım mekanizmasının fizibilitesini ve esnekliğini göstererek yazılım tedarik zincirlerini güvence altına almak için yeni bir çözüm ortaya koymuşlardır (Xia vd., 2023).

Ding vd., her ürün için eksiksiz bir elektronik dosya oluşturmak için kullanılan, kurumsal büyük verilere dayalı bir SBOM oluşturma yöntemi önermişlerdir. Yapılan çalışma ile manuel işlem büyük ölçüde azaltılmış ve verilerin doğruluğu artırılmıştır (Ding vd., 2019).

Kemppainen yaptığı çalışmada, bir SBOM aracını test ederken ve seçerken, yazılım geliştirme ortamı, kullanılan yazılım paket yöneticileri ve istenen SBOM formatı gibi bazı hususların dikkate alınmasının önemini belirtmiştir (Kemppainen, 2023).

Chaora vd., yazılımın toplumsal altyapı ile entegrasyonunu incelemiş ve güçlü bir yazılım tedarik zinciri güvenliğine duyulan ihtiyacı vurgulamıştır. Makale, yazılımın endüstriyel, imalat ve belediye teknolojileri de dahil olmak üzere çeşitli sektörlerde yaygın olarak kullanılmasının güvenilir ve güvenli bir yazılım tedarik zinciri gerektirdiğinin altını çizmektedir (Chaora, 2023). Yazarlar, SBOM'lar gibi yazılım tedarik zincirlerinde risk yönetimini geliştirmeye yönelik çeşitli girişimleri tartışmışlardır. Bununla birlikte, makale SBOM dağıtımında destek ve heves eksikliği de dahil olmak üzere bazı zorlukların da altını çizmektedir. Makalede ayrıca, bilinmeyen yazılımlarla ilişkili risklerin azaltılması ve kritik altyapı bileşenlerinin daha iyi anlaşılması da dahil olmak üzere SBOM'un benimsenmesinin faydaları da özetlenmektedir. Çalışma, SBOM'ların tedarik zinciri güvenliğindeki kritik rolünü vurgulamakta ve bu alanda gelecekte yapılacak araştırma ve geliştirmeler için değerli içgörüler sağlamaktadır. Yazarlar, siber güvenlik protokollerini güçlendirmek ve yazılım şeffaflığını ve yazılım bileşenlerinin güvenilirliğini artırmak için SBOM'ların yaygın bir şekilde uygulanmasını ve entegrasyonunu savunmaktadır.

Nocera vd., çalışmalarında açık kaynaklı yazılım projelerinde SBOM'ların benimsenmesine odaklanmışlardır (Nocera vd., 2023). Güncel olarak açık kaynaklı yazılım projelerinde SBOM kullanım durumunu incelemişlerdir. Araştırmalarına göre, SBOM'ların kullanımında hafif ama artan bir eğilim vardır; bu da geliştiricilerin yazılım tedarik zincirinde risk azaltma ve görünürlüğün değeri konusunda daha bilinçli hale geldiğini göstermektedir. Bu keşiflerden yola çıkan Stalnaker vd., SBOM'un oluşturulması ve kullanımında yer alan tarafların yaşadığı zorlukların kapsamlı bir analizini yapmıştır. Yazarlar bu zorlukları kabul etmekte ve uygulanabilir çözümlerin ve gelecekteki araştırma yönlerinin ana hatlarını çizerek yazılım tedarik zinciri güvenliğinin güçlendirilmesine ilişkin mevcut tartışmaya katkıda bulunmaktadırlar. Uygulayıcıların tutumlarını, benimseme eğilimlerini ve sorunlarını ele alan bu çalışmalar, toplu olarak SBOM'lara ilişkin ayrıntılı bir anlayış sunmakta ve yazılım tedarik zinciri genelinde güvenlik uygulamalarının ilerletilmesi için önemli bir temel oluşturmaktadır (Stalnaker vd., 2024).

Harer vd. araştırmalarında, makine öğrenimi kullanarak güvenlik açığı keşfi için otomatik bir yöntem sunarak, yazılım güvenliğinin artırılması konusunda yeni bir bakış açısı sunmaktadır (Harer vd., 2018). Bu çalışma, yazılım tedarik zinciri saldırıları, risk değerlendirme teknikleri ve SBOM'ların kullanımıyla ilgili sorunları inceleyen daha önceki araştırmaların başlattığı daha büyük tartışmaya katkıda bulunmaktadır. Önceki araştırmalar SBOM'lar gibi çerçeveler ve yazılım tedarik zincirinde artan geçişkenlik gibi risk azaltma stratejilerinin gerekliliğine odaklanırken, Harer ve arkadaşlarının çalışması güvenlik açıklarını proaktif olarak bulmanın teknik yönünü ele almaktadır. Makine öğreniminin güvenlik açığı tespitine entegrasyonu, yazılım güvenlik önlemlerini geliştirmek ve yazılım sistemi esnekliğini güçlendirmek için veri odaklı bir yaklaşım sağlamak için uyumludur. Bir bütün olarak ele alındığında, bu çalışmalar otomatik tespit, risk yönetimi ve şeffaflık gibi çeşitli hususları vurgulayarak yazılım güvenliği alanındaki zorlukların ve gelişmelerin kapsamlı bir şekilde kavranmasına katkıda bulunmaktadır.

Yan vd., tarafından gerçekleştirilen bir çalışmada, yazarlar yazılım tedarik zincirlerinin saldırı yüzeyini araştırmış ve üç temel hususu anlamak için 50 açık kaynaklı Java projesini analiz etmişlerdir. Özellikle, bağımlı bileşenlerin tedarik zincirlerindeki güvenlik açıklarından ne ölçüde etkilendiği, tedarikçi yazılımın

güvenlik açıklarından ne ölçüde etkilendiği ve yazılım tedarik zincirlerindeki güvenlik açıklarının belirli konumlarının neler olduğuna odaklanmışlardır. Önemli bir bulgu olarak, "açık kaynaklı yazılım tedarik zincirindeki güvenlik açıklarından kaynaklanan güvenlik sorununun, bağımlı bileşenleri üzerinde geniş bir etkiye sahip olabileceği ve bu durumun topluluk tarafından dikkatle ele alınması gerektiği" sonucuna varmışlardır (Yan vd., 2021).

Zahan vd. tarafından 1,63 milyon JavaScript npm paketinin analiz edildiği bir çalışmada, kötü niyetli bir aktörün yararlanabileceği ek paketler içeren birden fazla giriş noktası bulunmuştur. (Zahan vd., 2022).

Wang vd., tarafından gerçekleştirilen bir başka çalışmada, yazarlar Java açık kaynak kütüphaneleriyle ilişkili riskleri incelemiş ve kütüphaneyle ilişkili birçok risk bulmuşlardır. Bu çalışmalar, yazılım tedarik zinciri paketlerinin üçüncü taraf sağlayıcılarıyla ilişkili gizli tehlikeleri örneklendirmeye yardımcı olmaktadır (Wang vd., 2020).

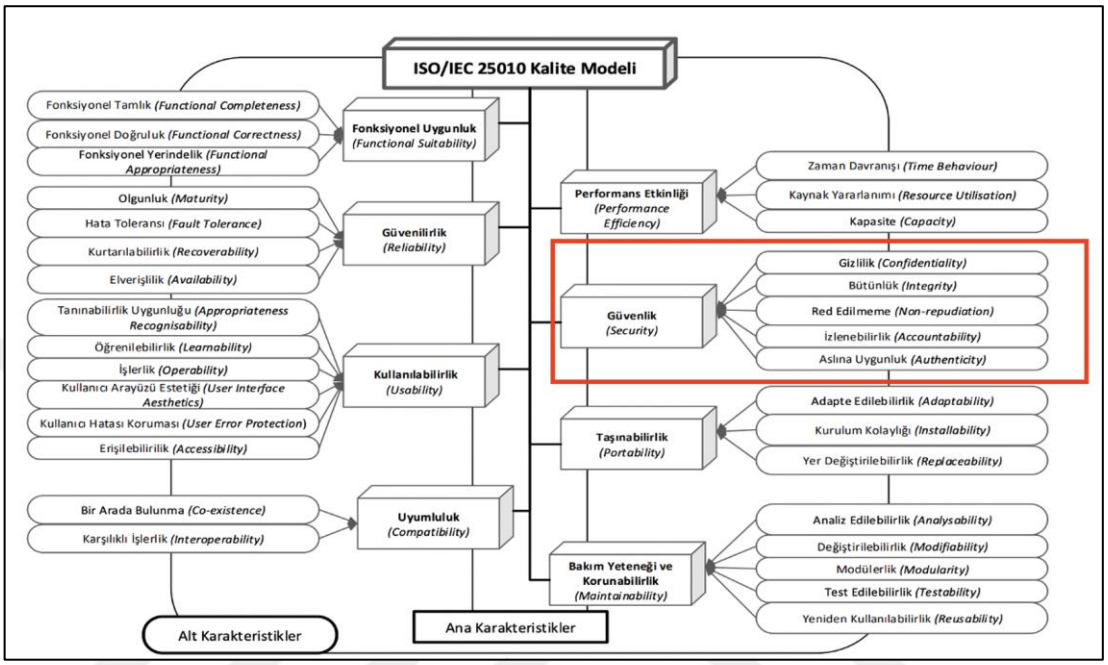
Kloeg vd., SBOM üretim ve tüketimine doğrudan dahil olan iş paydaş gruplarını incelemiştir. Bu çalışmanın temel amacı, hangi grupların SBOM'un benimsenmesini teşvik edebileceğini veya engelleyebileceğini ve bu davranışın arkasındaki mantığı belirlemektir. Grup temsilcileriyle görüşmeler yaparak, SBOM'un benimsenmesine ilişkin paydaşlara özgü riskleri, faydaları, endişeleri ve teşvikleri belirlemişlerdir. Analiz sonucunda, SBOM'un benimsenme potansiyelinin Sistem Entegratörleri ve Yazılım Satıcıları arasında daha yüksek olduğunu göstermektedir. Aynı zamanda, B2B müşterileri ve Bireysel Geliştiriciler, SBOM'un benimsenme sürecini engelleyen en az motivasyona sahiptir. Bunların SBOM tüketen ve tedarik eden ana paydaşlar olduğu göz önüne alındığında, bu teknolojinin genel benimsenme potansiyelinin şu anda sınırlı olduğu ve önemli bir dış itici güç gerektirdiği sonucuna varmışlardır. (Kloeg vd., 2024).

İncelenen çalışmalardan da görülebileceği üzere SBOM'un, yazılım zafiyetlerinden kaynaklı güvenlik ihlallerinin önüne geçilebilmesi için kilit bir rol oynamaya aday bir metodoloji olduğu ifade edilebilir. Satın alınan yazılımın

bağımlılıklarının/bileşenlerinin bilinmesi, alıcılar açısından zafiyetli yazılım sorununa çözüm üretebilecektir.

3. ISO 25010 KALİTE MODELİ VE GÜVENLİK KRİTERİ

ISO 25010 kalite modeli Şekil 3.1'de gösterildiği gibi 8 ana ve 31 alt kriterden oluşan hiyerarşik bir yapıdır (ISO/IEC 25010:2011) ISO 25010 içerisinde güvenlik 8 ana kriterden biridir (Estdale vd., 2018).



Şekil 3.1 ISO/IEC 25010 Kalite Modeli

3.1 Fonksiyonel Uygunluk

Belirlenen koşullar altında kullanıldığında, bir ürünün veya sistemin sağladığı fonksiyonların, belirtilen veya kastedilen gereksinimleri karşıladığını, uygun olduğunu gösteren karakteristiktir. Bu özellik Fonksiyonel Tamlık, Fonksiyonel Doğruluk ve Fonksiyonel Uygunluk alt özelliklerinden oluşur (França vd., 2015).

3.2 Güvenirlik

Belirlenen koşullar altında, bir sistemin, ürünün veya bileşenin belirli bir zaman aralığında, belirtilen fonksiyonları yeterli bir şekilde yerine getirdiğini gösteren kalite karakteristiğidir. Olgunluk, Hata Toleransı, Kurtarılabilirlik ve Elverişlilik alt özelliklerinden oluşur.

3.3 Kullanılabilirlik

Belirli bir kullanım bağlamında etkinlik, verimlilik ve memnuniyet ile belirtilen hedefleri gerçekleştirmek için bir ürünün veya sistemin belirli kullanıcılar tarafından kullanılabilir olduğunu gösteren kalite karakteristiğidir. Tanınabilirlik Uygunluğu, Öğrenilebilirlik, İşlerlik, Kullanıcı Arayüzü Estetiği, Kullanıcı Hatası Koruması ve Erişilebilirlik alt özelliklerinden oluşur.

3.4 Uyumluluk

Aynı donanım veya yazılım ortamını paylaşırken, bir ürünün, sistemin veya bileşenin diğer ürünler, sistemler veya bileşenlerle bilgi alışverişi yapabilmesi ve/veya gerekli fonksiyonları yerine getirebilmesini gösteren kalite karakteristiğidir. Bir Arada Bulunma ve Karşılıklı İşlerlik alt özelliklerinden oluşur.

3.5 Performans Etkinliği

Belirtilen koşullar altında kullanılan kaynakların miktarına göre performansının etkin olduğunu gösteren kalite karakteristiğidir. Zaman Davranışı, Kaynak Yararlanımı ve Kapasite alt özelliklerinden oluşur.

3.6 Güvenlik

Bir ürünün veya sistemin bilgileri ve verileri koruduğunu gösteren kalite karakteristiğidir. Güvenliğin 5 alt karakteristiği şunlardır:

1- Gizlilik: Bir ürünün veya sistemin, verilere yalnızca erişime yetkili olanlara erişebilmesini sağlamasıdır.
2- Bütünlük: Bir sistemin, ürünün veya bileşenin, bilgisayar programlarına veya verilere yetkisiz erişime veya modifikasyona engel olmasıdır.
3- Red Edilmeme: Olayların ve eylemlerin gerçekleştiğini kanıtlama yeteneğidir. Böylelikle olayların veya eylemlerin daha sonra reddedilmesi mümkün olmamaktadır.
4- İzlenebilirlik: Bir işletmenin faaliyetlerinin benzersiz şekilde izlenebilmesi yeteneğidir.
5- Aslına Uygunluk: Bir konunun veya kaynağın kimliğinin iddia edildiği gibi kanıtlanabilmesidir (Yıldız, 2014).

3.7 Taşınabilirlik

Bir sistemin, ürünün veya bileşenin bir donanımdan, yazılımdan veya diğer işletimsel veya kullanım ortamından diğerine geçirilebileceği etkinlik derecesini ve verimliliğini gösteren kalite karakteristiğidir. Adapte Edilebilirlik, Kurulum Kolaylığı ve Yer Değiştirebilirlik alt özelliklerinden oluşur.

3.8 Bakım Yeteneği ve Korunabilirlik

Bir ürünün veya sistemin bakım yapan kişiler tarafından etkin ve verimli bir şekilde değiştirilebilmesini, düzeltilebilmesini, geliştirilebilmesini vb. gösteren kalite karakteristiğidir. Analiz Edilebilirlik, Değiştirilebilirlik, Modülerlik, Test Edilebilirlik, Yeniden Kullanılabilirlik alt özelliklerinden oluşur. (Şimşek, 2018).

ISO 25010 kalite modelinin detaylarında da görülebileceği üzere Güvenlik ana kriter olmakla birlikte, yazılım kalitesini bir bütün olarak ele almaktadır. ISO 25010 Kalite Modeli, yazılım ürününün bağımlılıkları/bileşenleri üzerine somut bir vurguda bulunmamaktadır. Kalite modelinin vurguladığı Güvenlik ana kriterinin sağlanabilmesi için yazılım bileşenlerinin de belirtilen alt kriterler bağlamında ele alınması gerekliliği açıktır.

4. YAZILIM MALZEME LİSTESİ

Yazılım malzeme listesi (Software Bill of Materials – SBOM), bir yazılımı oluşturan bileşenlerin bir listesidir (Sehgal vd., 2023). Bir SBOM, yazılım bileşenlerini, bu bileşenler hakkındaki bilgileri ve aralarındaki tedarik zinciri ilişkilerini tanımlar ve listeler. Belirli bir SBOM'da yer alan bilgilerin miktarı ve türü, sektör ve SBOM tüketicilerinin ihtiyaçları gibi faktörlere bağlı olarak değişiklik gösterebilir.

Günümüzün yazılım paketleri genellikle çok sayıda üçüncü taraf bileşen içerir. Şirketler, güvenliği ve işlevselliği korumak için bu bileşenlerin her birini aktif olarak izlemeli ve yönetmelidir. SBOM'lar eski bir kavramın yeni bir versiyonudur. Satıcılar tarihsel olarak tedarik zinciri yönetiminde ürünlerini oluşturan birçok parçayı tanımlamak için malzeme listelerini kullanmışlardır. Örneğin, marketten satın aldığımız gıdanın içindekiler listesi etkili bir şekilde bir ürün reçetesidir. Malzeme listesi fikrinin yazılıma uygulanması ise daha yenidir. ABD Başkanı Biden yönetiminin ABD'de siber güvenliği artırmanın bir yolu olarak SBOM'ları vurgulayan bir idari emir yayınladığı Mayıs 2021'de daha çok gündeme gelmiştir (Sehgal vd., 2023). ABD federal hükümetine satış yapan yazılım tedarikçileri, yönetmelik uyarınca ürünleri için SBOM'ları sağlamak zorundadır. Bu amaçla, kuruluşların bu bileşenleri takip etmek için bir yazılım malzeme listesi kullanması gerekmektedir. Makine tarafından okunabilen bu liste, bir yazılım parçasının çeşitli bağımlılıklarını ve öğelerini içerir (Stoddard vd., 2023). Yazılım malzeme listesi, bir uygulamanın geliştirilmesinde ve sunulmasında yer alan tüm bileşen parçalarını ve yazılım bağımlılıklarını listeler. SBOM'lar, tedarik zincirlerinde ve üretimde kullanılan malzeme listelerine (BOM) benzer. BT sektöründeki tüm satıcılar için, bir uygulamanın oluşturulduğu temel kod bileşenlerini doğru bir şekilde tanımlayacak ortak bir çerçeve sunmaktadır (Sehgal vd., 2023).

Tipik bir SBOM, lisans bilgilerini, sürüm numaralarını, bileşen ayrıntılarını ve satıcıları içerir. Bu liste, başkalarının yazılımların neler içerdiğini anlamalarına ve buna göre hareket etmelerine olanak tanıyarak hem üretici hem de kullanıcı için riskleri azaltır.

Bir SBOM'un temel amacı, bileşenleri ve bunların birbirleriyle olan ilişkilerini benzersiz ve açık bir şekilde tanımlamaktır. Bunu yapabilmek için aşağıdaki temel bilgilerin bir kombinasyonu gereklidir (Muriri, 2019).

- Author Name/ Yazar Adı: SBOM girişinin yazarı (bu her zaman tedarikçi olmayabilir).
- Supplier Name/ Tedarikçi Adı: Birden fazla adı veya takma adı not etme yeteneği de dahil olmak üzere, SBOM girişindeki bileşenin tedarikçisinin adı veya kimliği. Yazar ve tedarikçi aynı olduğunda, tedarikçi birinci taraf yetkili bileşeni tanımlamaktadır. Yazar ve tedarikçi farklı olduğunda, yazar farklı bir tedarikçiden alınan bir bileşen hakkında bir iddiada bulunmaktadır.
- Component Name/ Bileşen Adı: Birden fazla adı veya takma adı not etme yeteneği de dahil olmak üzere bir veya daha fazla bileşen adı/adı. Bileşen adları tedarikçi adlarını taşıyabilir. Bileşen (ve tedarikçi) adları, genel bir ad alanı:ad yapısı kullanılarak aktarılabilir.
- Version String / Sürüm Dizesi: Sürüm bilgilerinin formatı serbest biçimlidir ancak yaygın endüstri kullanımına uygun olmalıdır.
- Component Hash / Bileşen Karması: Bir yazılım bileşenini tanımlamanın en iyi yolu, benzersiz bir tanımlayıcı görevi gören bir şifreleme karması kullanmaktır.
- Unique Identifier / Benzersiz Tanımlayıcı: Karmaya ek olarak, her bileşenin kendisini SBOM içinde tanımlayan bir kimlik numarasına sahip olması gerekir.
- Relationship / İlişki: Bileşen ile paket arasındaki ilişkiyi tanımlar. Çoğu durumda ilişki belirli bir bileşenin belirli bir pakete dahil olduğu anlamına gelir (Muriri, 2019).

4.1 SBOM Oluşturma

Bir SBOM oluşturmak için tedarikçi kendi oluşturduğu bileşenleri tanımlar, temel bileşen bilgilerini ve bu bileşenler için ek nitelikleri üretir ve doğrudan dahil edilen tüm bileşenlerin listesini numaralandırır. SBOM bilgileri ideal olarak tedarikçinin yazılım oluşturma ve paketleme süreçlerinin ayrılmaz bir parçası olarak oluşturulacaktır ve bu da mevcut geliştirme araçlarında yapılacak değişikliklerle

gerçekleştirilebilir. Yazılım veya yazılım sistemleri oluşturan, değiştiren, paketleyen ve teslim eden her kuruluş tedarikçi olarak kabul edilir ve bu nedenle bileşenlerin tanımlanmasından ve SBOM'ların oluşturulmasından sorumludur. Bu, esasen SBOM amaçları için tedarikçi olarak kabul edilen sistem entegratörlerini de içerir. Bir kuruluş dahili olarak geliştirilen bileşenler için de tedarikçi olarak hareket edebilir. Dahil edilen bileşenler için SBOM'lar yukarı akış tedarikçilerinden temin edilebiliyorsa, bu SBOM'lar birincil SBOM ile birlikte verilir veya ona eklenir. Bu tür bilgilerin mevcut olmadığı durumlarda, bir tedarikçi "en iyi çaba" SBOM'larını sağlayabilir; bu da dahil edilen bir bileşen SBOM'unun yazarının bileşenin tedarikçisiyle aynı olmayacağı gerçeğiyle belirtilecektir (Muriri, 2019).

4.2 SBOM Formatları

4.2.1 SPDX

Yazılım Paketi Veri Değişimi (Software Package Data Exchange - SPDX) belirtimi, yazılım bileşenleri hakkında bilgi iletmek için açık bir standart tanımlar. SPDX, yazılım malzeme listeleri (SBOM'ler) oluşturmak, lisans ve telif hakkı ayrıntılarını kapsüllemek, sürüm kimlikleri ve bilinen güvenlik açıkları gibi paket meta verileri sağlamak için kullanılır.

SPDX, on yıldan uzun bir süre önce geliştiricilerin açık kaynak lisanslarına uymasına yardımcı olmak için tasarlanmıştır. O zamandan beri, bağımlılık ağaçlarını tanımlamak ve SBOM'ları yaymak için yeni özelliklerle genişletilmiştir. SPDX, ISO'nun yazılım tedarik zinciri dokümantasyonu için uluslararası standart olarak kabul ettiği Eylül 2021'de dikkatleri üzerine çekmiştir.

SPDX, topluluk odaklı Linux Vakfı tarafından yürütülen bağımsız bir projedir (Gandhi vd., 2018). Mevcut standart, yazılım endüstrisinden ilgili taraflarca desteklenmiş ve yazılmıştır. Liste, Google ve Microsoft gibi büyük isimlerin yanı sıra Anchore ve Snyk gibi bitişik araç geliştiricilerini de içermektedir. SPDX, bu satıcıların büyük ölçekli yazılım tedarik zincirlerini yönetme, belgeleme ve sürdürme konusundaki deneyimlerinin doruk noktası olmuştur. Ancak proje dışarıdan katılıma açıktır. SPDX eserleriyle çalışan herkes veya şirket aylık genel toplantıya katılabilir veya posta listesine abone olabilir.

SPDX, yazılım paketlerini ve bağımlılıklarını açıklamak için endüstri standardı bir yöntemdir (Germonprez vd., 2014). Tüm satıcılar, programlama dilleri ve çerçevelerle çalışmak üzere tasarlanmıştır. Standart, paket formatları arasındaki farklılıkları ortadan kaldırarak ve ekosistem genelinde birlikte çalışabilirliği artırarak bir proje için bir SBOM üretme çabasını azaltır. Şekil 4.1'de SPDX json formatı görülebilir.


```
{
  "files": [
    {
      "fileName": "./OpenRA.Server.deps.json",
      "SPDXID": "SPDXRef-File--OpenRA.Server.deps.json-9AF11BD8EF6F2C5D9B49E2B0F7AAB7180F9D56D6",
      "checksums": [
        {
          "algorithm": "SHA256",
          "checksumValue": "48531a1ea210e84eebe7230dd127a1c77b4108231b6ae2c9b8a70c73241606fb"
        },
        {
          "algorithm": "SHA1",
          "checksumValue": "9af11bd8ef6f2c5d9b49e2b0f7aab7180f9d56d6"
        }
      ],
      "licenseConcluded": "NOASSERTION",
      "licenseInfoInFiles": [
        "NOASSERTION"
      ],
      "copyrightText": "NOASSERTION"
    },
    {
      "fileName": "./OpenRA.pdb",
      "SPDXID": "SPDXRef-File--OpenRA.pdb-9C6D0FCAD2448A927644BC35391F57250E96ED47",
      "checksums": [
        {
          "algorithm": "SHA256",
          "checksumValue": "ee522c24723722c0cc23b67bd6695843af64f97429bfdfb1f802163135b2ca78"
        },
        {
          "algorithm": "SHA1",
          "checksumValue": "9c6d0fcad2448a927644bc35391f57250e96ed47"
        }
      ],
      "licenseConcluded": "NOASSERTION",
      "licenseInfoInFiles": [
        "NOASSERTION"
      ],
      "copyrightText": "NOASSERTION"
    },
```


Şekil 4.1 OpenRA projesi için SPDX örneği

4.2.2 CycloneDX

OWASP (Open Web Application Security Project) tarafından geliştirilen bir SBOM formatıdır. Özellikle güvenlik odaklıdır ve yazılım bileşenlerinin güvenlik açıklıkları hakkında bilgi sağlar. Daha hafif bir yapıya sahiptir ve genellikle güvenlik açıklıklarının izlenmesi için tercih edilir (Url-1).

Cyclone DX ve SPDX'in her ikisi de veri yapısını çok farklı tanımlar. SPDX insan tarafından okunabilir format için iyi bir seçim olarak kabul edilebilirken, CycloneDx format spesifikasyonu onu mükemmel bir otomasyon hedefi haline getirir (Sehgal vd., 2023). Şekil 4.2'de CycloneDX'in XML formatı görülebilir.

XML format CycloneDx:


```
{
 "bomFormat": "SBOM",
 "specVersion": "2.1",
 "serialNumber": "urn:uuid:3e67188-792b-bc0a-a67575785bnbb79",
 "version": 1,
 "components": [
  {
   "type": "library",  "name": "acme-library",  "version": "2.0.0"    }  ] }
```


Şekil 4.2 CycloneDX XML Format

4.2.3 Syft

Syft, Anchore Inc. tarafından sürdürülen ve Apache-2.0 lisansı altında dağıtılan açık kaynaklı bir CLI uygulamasıdır (Url-2). Konteyner görüntülerini ve dosya sistemlerini tarama ve bulgularından bir SBOM üretme yeteneği sağlar. Tespit yetenekleri kataloglayıcılar aracılığıyla gerçekleştirilir ve v0.55.0 sürümünden itibaren 18 paket türü desteklenmektedir. Syft, tespitleri için yalnızca dosya sistemi verilerine bakar ve tespitlerinin içeriğini zenginleştirmek için herhangi bir dış kaynağa ulaşmaz. Bu yaklaşım, çok hızlı bir şekilde sonuç üretmesini sağlar. Bir konteyner görüntüsünün tipik bir syft taraması yalnızca birkaç saniye sürer. Ancak bu karar aynı zamanda eksik sonuçlar üretme maliyetini de beraberinde getirir. Bu gibi durumlarda, syft tarafından üretilen sonucu almak ve gerektiğinde ek bilgilerle geliştirmek SBOM tüketicisinin sorumluluğundadır (Gschrei, 2022).

4.3 Yazılım Tedarik Zinciri Güvenliğinde SBOM'un Rolü

SBOM'lar yazılım tedarik zincirinin güvenliğinin sağlanmasında çeşitli şekillerde kritik bir rol oynayabilir (Nguyen vd., 2023). Bu durum aşağıdaki şekilde özetlenebilir:

- Güvenlik açıklarının belirlenmesi: SBOM'lar yazılım tedarik zincirindeki güvenlik açıklarını tespit etmek için kullanılabilir. Bu bilgi, iyileştirme

çabalarını önceliklendirmek ve güvenlik açıklarının istismar edilmesini önlemek için kullanılabilir.

- Bağımlılıkların izlenmesi: SBOM'lar yazılım bileşenleri arasındaki bağımlılıkları izlemek için kullanılabilir. Bu bilgiler, bağımlılıklarda yapılacak güncellemelerle ortaya çıkabilecek potansiyel güvenlik açıklarını tespit etmek için kullanılabilir.
- İletişimin iyileştirilmesi: SBOM'lar geliştiriciler ve güvenlik ekipleri arasındaki iletişimi iyileştirebilir. Bu bilgiler, geliştirme sürecinin erken aşamalarında güvenlik endişelerini tespit etmek ve ele almak için kullanılabilir.

4.4 SBOM'un Faydaları

SBOM'ları kullanmanın bir dizi faydası vardır.

- Artan görünürlük: SBOM'lar yazılım tedarik zincirinde daha fazla görünürlük sağlar. Bu, potansiyel güvenlik risklerinin belirlenmesine ve ele alınmasına yardımcı olabilir.
- Güvenlik Açığı Yönetimi: SBOM'lar tedarik zinciri saldırıları riskini azaltmaya yardımcı olabilir. Bunun nedeni, güvenlik açıklarını belirlemek ve azaltmak için kullanılabilecek bilgiler sağlamalarıdır. Bir SBOM, bir alt bileşenin herhangi bir güvenlik açığı içerip içermediğini ve bir yama gerektirip gerektirmediğini belirlemek için faydalı olabilir. Bu aynı zamanda bu güvenlik açığı nedeniyle risk altında olabilecek herhangi bir alt bileşenin belirlenmesine de yardımcı olur. Ayrıca, bir SBOM önceki yama ve güncellemelerin ayrıntılarını sağlayabilir. Güvenlik açığı değerlendirmesi ve yönetiminin tanıtılması ve otomatikleştirilmesi, düzeltme süresini kısaltabilir ve farkındalığı artırabilir
- İyileştirilmiş uyumluluk: SBOM'lar güvenlik gerekliliklerine uyumun iyileştirilmesine yardımcı olabilir. Bunun nedeni, yazılımın güvenli olduğunu göstermek için kullanılabilecek bilgiler sağlamalarıdır.
- Lisans Yönetimi: SBOM'lar yazılımın ve ilgili alt bileşenlerinin lisanslama gerekliliklerine uygunluğunu garanti edebilir.
- Fikri Mülkiyet: SBOM bilgileri fikri mülkiyet uygulamalarına yardımcı olabilir, çünkü bu tür verilere erişim kullanılan alt bileşenlerin, lisansların ve

güncelleme geçmişinin ayrıntılarını sağlar. SBOM savunucuları, öngörülemeyen işleri ve kod fazlalığını en aza indirdiklerini savunmaktadır. Kapsamlı alt bileşen detayları fikri mülkiyetin korunmasını kolaylaştırır (Url-3).

- Yüksek Güvence: SBOM'lar kaynağı, tedarikçileri hakkında bilgileri, değişikliklerin geçmişini ve bileşenler tedarik zinciri boyunca hareket ederken gözetim zincirini sağlayarak yazılımdaki alt bileşenlerin bütünlüğünü sağlayabilir. Ayrıca düşük kaliteli veya terk edilmiş yazılım unsurlarının hariç tutulmasını da teşvik eder. Ayrıca SBOM, ilgili yazılım tarafından desteklenen teknolojiler ve yazılımın zamanında güncellenmesine yardımcı olabilecek son kullanım tarihleri (End of Life - EOL) hakkında da bilgi içerir (Nguyen vd., 2023), (Arora vd., 2022).

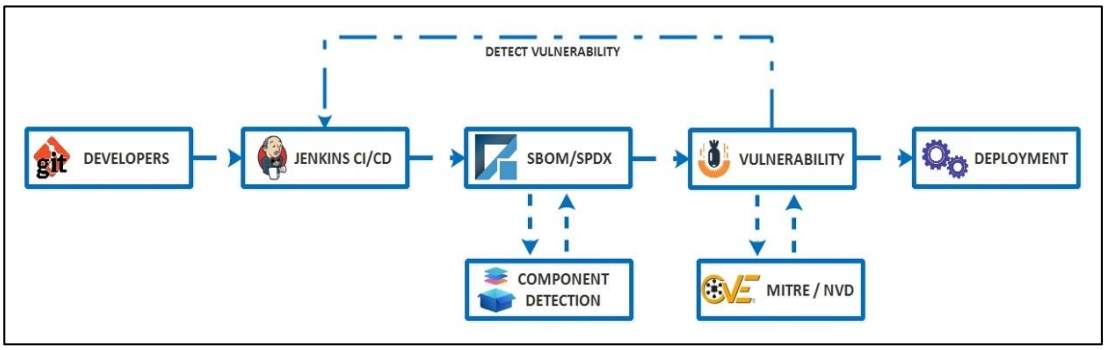
5. METODOLOJİ

Exploit, kelime olarak istismar etmek, sömürmek gibi anlamlara gelmektedir. Exploit, yazılımlarda veya bilgisayarlarda yer alan zafiyetlerden ve hatalardan faydalanmak için geliştirilmiş kodlar ve programlardır. Bu kodlar ve programlar, sistemi istismar etmek için kullanılarak gizliliğe, bütünlüğe ve erişilebilirliğe zarar verirler (Jacobs vd., 2021). Exploitler, sistemin en zayıf noktalarını bulur ve buradan sisteme sızarlar. Exploitler, dijital dünya için çok büyük risk oluştururlar. Siber saldırıların büyük bir kısmı exploit türünde gerçekleşmektedir (Kekül vd., 2021). Firmaların exploitlerden etkilenmemeleri için mevcut yazılım ve sistemlerini düzenli olarak aktif/pasif zafiyet analiz araçları ile taramaları ve ortaya çıkan açıklıklarını kapatmaları gerekmektedir. Bir yazılımın zafiyeti o ürünün içerdiği tüm bileşenlerin zafiyetlerinin bir birleşimidir. Bu nedenle etkili bir zafiyet analizi, ilgili yazılımı oluşturan tüm bileşenlerin analiz edilmesini gerektirir.

Bu çalışmada, bir yazılımın kaynak kodu üzerinden SBOM'unun üretildiği ve yazılım bileşenlerinin zafiyet analizinin yapılabildiği bir model önerilmiştir. Önerilen model Şekil 5.1'de gösterilmiştir. Bu model bir yazılım geliştirme sürecinden kullanıma alınmasına kadar olan iş hattına SBOM üretimi ve zafiyet taraması için iki yeni adım eklenmesini önermektedir. Önerilen modelin gerekliliği ve doğrulaması somut sonuçlarla gösterilmiştir. Bu model, geliştiricilerin kod birleştirme (pull request) istekleri aşamasından sürekli dağıtım (CI/CD) araçları üzerinden uygulama dağıtım adımı ile sonuçlanabilmesi için otomatik zafiyet analizi yapılmasını önermektedir. Önerilen model, ürünü oluşturan tüm yazılım bileşenlerinin zafiyet taramasının yapılabilmesi için otomatik SBOM üretimini önermektedir.

Çalışma kapsamında öncelikle, GitHub üzerinde yer alan popüler 10 proje (C# tabanlı) belirlenmiştir (Url-4). Bu projeler belirlenirken projelerin popülariteleri (star ve fork sayıları) dikkate alınmıştır. Ayrıca bu projelerin MITRE-CVE (Common Vulnerability and Exposures - Yaygın Güvenlik Açıkları ve Etkilenmeler) veritabanı üzerinden pasif zafiyet analizi temelinde mevcut zafiyetleri taranmış ve zafiyet içerip içermediklerine bakılmıştır. CVE, halka açık olarak sunulan bir zafiyet veritabanıdır. Bu veritabanının geliştirilme amacı zafiyetler hakkında yapılan bilgi paylaşımını

kolaylaştırmaktır. Bir CVE kaydı, zafiyet hakkında bir açıklama, bir zafiyet kimlik numarası ve en az bir halka açık referanstan oluşur (Hankin, 2022).



Şekil 5.1 Önerilen Model

Önerilen modelin doğrulanması için 10 farklı projenin belirlenmesinden sonra belirlenen projelerin dinamik olarak SBOM'ları oluşturulmuştur. Projelerin SBOM üretimi için SBOM aracı (sbom-tool) kullanılmıştır (Url-5). Çalışmada SBOM üretmek için kullanılan araç, Microsoft tarafından geliştirmiştir. Bu araç, her türlü yapı için SPDX 2.2 uyumlu SBOM'lar oluşturmaya yönelik yüksek düzeyde ölçeklenebilir ve kurumsal kullanıma hazır bir araçtır. SBOM tool, bileşenleri algılamak için bileşen algılama (Component Detection) kitaplıklarını ve bu bileşenlere ilişkin lisans bilgilerini belirlemek için ClearlyDefined API'yi kullanır (Url-5).

Seçilen projelerin SBOM'larının üretiminin ardından projelerin SBOM'ları üzerinden zafiyet taramaları gerçekleştirilmiştir. SBOM'ları üretilen projelerin bağımlılıklarının zafiyet içerip içermediklerini belirlemek için açık kaynak kodlu Bomber aracı kullanılmıştır (Url-6). Bu araç üzerinden yapılan taramalar sonucunda bir projenin bağımlılıklarının içerdiği zafiyetler elde edilmiş ve değerlendirilmiştir. Bomber, SBOM listeleri üzerinden zafiyet taraması yapan bir araçtır (Url-6). Çalışma kapsamında bir yazılımı oluşturan bileşenlerin içerebileceği zafiyetlerin belirlenmesi aşamasında kullanılmıştır. Bomber, JSON veya XML tabanlı CycloneDX, SPDX veya Syft biçimli SBOM'ları okuyabilir ve zafiyet taraması gerçekleştirebilir.

Yazılımlar açık kaynak veya derlenmiş kod olarak sunulabilir. GitHub veya herhangi bir genel kaynak deposundan erişilebilen üçüncü taraf bileşenlere açık kaynak kodlu yazılım olarak bakılabilir. Teknik olarak firmalar için şirket bünyesinde oluşturulan yazılımlar da açık kaynaktır; herkese açık değildir ancak dahili ekipler kaynak koda erişebilir. Derlenmiş/kapalı kaynak yazılım da dahili olabilir ancak bu genellikle harici satıcılardan satın alınan/temin edilen yazılımlardır (Url-6). Şirketler, her türlü açık kaynağı taramak ve güvenlik açığı verilerini sağlamak ve hatta bazı durumlarda SBOM'lar oluşturmak için GitHub, Sonatype, Snyk vb. gibi satıcılar tarafından sağlanan SCA (Software Composition Analysis) araçlarını kullanabilirler. Ancak derlenmiş/kapalı kaynaklı yazılımların bileşenlerini pasif zafiyet analizi araçları ile taranması mümkün olmamaktadır. Bu aşamada satıcılar tarafından sağlanan SBOM'lar devreye girer. SBOM'lar ile ilgili yazılımı oluşturan bileşenler sunulur ve Bomber gibi araçlarla SBOM ile sunulan bileşenlerin herhangi bir zafiyet içerip içermediği belirlenebilir.

Özet olarak GitHub üzerinde kaynak kodları yer alan en popüler 100 CSharp listesinde yer alan (Url-4) 10 projenin SBOM Tool üzerinden bağımlılıkları çıkarılmış, elde edilen SBOM'lar üzerinden zafiyet taraması gerçekleştirilmiştir. SBOM'lar üzerinden elde edilen zafiyetlerle doğrudan ilgili yazılım ürününe atanan zafiyetler karşılaştırılarak değerlendirilmiştir.

5.1 GIT

Git, projedeki tüm değişiklikleri görüntülemek amacıyla kullanılan bir version kontrol aracıdır. Bu da Git'i dünya çapındaki programcılar için üst düzey yardımcı araç haline getirir. Git, her boyuttaki projeyi yönetebilir. Projede çalışanların iş akışını denetleyebilen ve zaman içinde yapılan değişiklikleri yönetebilen bir sistemdir. Git ile projedeki gelişmelerin görüntülenebilmesi programcıların yanı sıra teknik olmayan kullanıcılar ve müşteriler için de faydalı olur. Kısaca bu sistem, birden fazla kullanıcının birbirinin işini aksatmadan birlikte çalışmasına olanak tanır (Blischak vd., 2016).

Kısaca Git, tüm değişikleri izleyen ve önceki sürümleri yedek olarak saklayan sistemdir. GitHub ise web tabanlı bir grafik arayüz sağlayan hakla açık yada kurumsal olarak hizmet veren Git deposu hizmetidir (Url-7). Bu hizmet,

geliştiricilerin yalnızca belirli bireysel değişiklikleri değil, grup olarak yaptıkları değişiklikleri de görmeyi sağlar. Bu sayede de karmaşık durumların ve hataların çözümünü kolaylaştırır.

Tüm yetenekleriyle GitHub, dünyanın en büyük kodlama topluluğudur. GitHub'a bir kod veya proje koymak, onun daha fazla görülmesini sağlar. Yazılımcılar bu platform üzerinde birçok farklı dilde kaynak kodları bulabilir ve herhangi bir değişiklik yapmak için Git'i kullanabilirler. GitHub, iş paylaşımını basit hale getirirken ekibe dahil edilen herkesin belli bir projede birlikte çalışmasını da sağlar.

5.1.1 GitHub

GitHub, yazılım geliştirme ve sürüm kontrolü için bulut tabanlı bir platformdur ve geliştiricilerin kodlarını depolamalarını ve yönetmelerini sağlar. Kullanıcı dostu ara yüzü sayesinde projeleri, bireyler ve ekipler için erişilebilir hale getirir ve Git kullanarak iş birliğini ve sürüm kontrolünü kolaylaştırır (Url-7).

Bu platform sayesinde yazılımcılar, birbirlerinin deneyimlerinden yeni şeyler öğrenebilir ve farklı projelere destek verebilir. Bu platformda geliştiriciler, kodlarını saklar ve yönetir (Url-8). Versiyon kontrol, adından da anlaşılabileceği gibi gerektiğinde çalışmanın geçmiş bir sürümünün çağrılabilmesi için dosya ya da veri kümesinde uygulanan bütün değişikliklerin depolanabileceği sistemdir. Sistem, proje üzerinde çalışan herkesin dosyanın en son sürümünde değişiklikler yapabilmesini ve aynı projede eş zamanlı görev alabilmesini sağlar (Zolkifli vd., 2018).

Versiyon kontrol yazılımcılar için oldukça önemlidir çünkü geliştiriciler proje sırasında özellikler eklemek, hataları düzeltmek için aynı anda kodu güncelleyebilir. Herhangi bir sorun kullanıcıları etkileyeceğinden bu değişiklikleri doğrudan kaynak kodunda yapmak mantıklı olmaz. Bunun yerine geliştiriciler, kodun kendi kopyalarıyla çalışır ve kod kapsamlı şekilde test edildikten sonra ana kod tabanına eklenir. GitHub'ın bu kadar popüler olmasının temel sebepleri olarak:

- Gelişmiş İş Birliği: GitHub'ın en büyük yeteneği; sürüm ve erişim kontrolü de dahil olmak üzere proje iş birliği özellikleridir. Mevcut ve geçmiş sürümler de dahil olmak üzere tüm dosyaları saklayan bir depo sunulur. Ekipler ortak çalışmak istediğinde erişim izni verilebilir. Erişim izni verilen

kişiler, projenin farklı özellikleri için geliştirmeler yapabilir. Böylece projede çalışan herkes, birbirinin işine müdahale etmeden değişikliklerini aynı anda uygulayabilir. Bunun için GitHub'da ayrı "branches" yani geliştirme alanları oluşturmak mümkündür. İşi biten ekip üyesi, kendi çalışmasını diğeriyle birleştirmek istediğinde "pull request" oluşturabilir. Bu isteğe onay verdiğinde iki farklı çalışma birbiriyle birleştirilmiş olur.

Hiçbir yazılım ilk seferde mükemmel olmaz. Bu nedenle GitHub deposunda projeyle ilgili yapılacakları listelemek ve sorunları raporlamak için de bir iş takip modülü bulunur. Bu alanda sorunlar ve yapılacaklar tartışılarak, biten görevler "çözüldü" olarak işaretlenebilir. Özetle GitHub kullanan geliştiriciler, herhangi bir işlevi geçersiz hale getirme ya da güncellemeleri kaybetme endişesi yaşamaz. Böylece proje, daha az sorunla minimum sürede tamamlanabilir.

- Kolay Dosya Yönetimi: GitHub'daki dosyalar tek bir cihaz veya ortamla sınırlı kalmaz. Bu platform, Git'in üstüne iyi bir grafik kullanıcı arayüzü (GUI) katmanı ekler. Git, kendi başına komut satırı, yani bilgisayarın metin tabanlı arayüzü üzerinde çalışır. Geliştiriciler, komut satırını nasıl kullanacağını bilse de dosyalarla etkileşim kurmanın en iyi yolu genellikle geliştirme ortamları üzerinde çalışmaktır. GitHub arayüzü, Git eylemlerini gerçekleştirmenin yanı sıra dosya geçmişini görüntülemenin kullanıcı dostu bir yolunu sunar. Bu, geliştiriciler için daha kullanışlı olmasının yanında Git'e alışmaya yeni başlayanlar için de daha erişilebilirdir. GitHub ile kolay dosya yönteminin diğer bir yararı ise bulut tabanlı altyapıdır. Bu platform üzerinden erişilmek istenilen depoya herhangi bir yer veya cihazdan rahatlıkla erişilebilir ve değişiklikler yönetilebilir. Bu dosyalar, HTML, CSS, JavaScript, veriler ve görüntüler de dahil olmak üzere pek çok belgeyi içerebilir.
- Sosyal Ağ: GitHub, kaynak kod versiyonlama sisteminin çok daha ötesinde bir sistemdir. Tüm kullanıcılar sistem üzerindeki genel (public) projelere katılabilir ve etkinliklerini görüntüleyebilecekleri bir profile sahiptirler. Bu profil herkese açık şekilde paylaşılabilir. Bu sosyal ağ, geliştiricileri her türlü açık kaynak projesini keşfetmeye ve katkıda bulunmaya teşvik eder. Ayrıca yazılımcılar, GitHub üzerinden bir portföy oluşturulabilir. Bu portföy üzerinden işe alım uzmanları tarafından keşfedilme imkanı kazanabilirler.

- Açık Kaynak Projeleri: Bir bütün olarak GitHub kullanımı, açık kaynak iş birliğine öncülük ederek birçok yazılım teknolojisinin geliştirilmesine katkıda bulunmaktadır. Günümüzde yaygın olarak kullanılan birçok araç ve yazılım, GitHub depoları üzerinde geliştirilmiştir.
- Özel Depolar: GitHub ücretsiz olarak hizmet vermektedir. Ancak ekipler, ücretli bir planda kodlarını gizli tutarak platform üzerinden iş birliği yapabilirler. Ayrıca bu ekosistem, kuruluşları dahili iş birliği araçlarıyla donatan teknolojik çözümler de sunar.

5.2 Jenkins

Jenkins, sürekli entegrasyon amacıyla oluşturulmuş eklentilerle Java'da yazılmış açık kaynaklı bir otomasyon aracıdır. Jenkins, yazılım projelerini, sürekli olarak derlemek ve test etmek için kullanılır. Geliştiricilerin proje değişikliklerini entegre etmesini ve kullanıcıların yeni bir yapı elde etmesini kolaylaştırır. Ayrıca çok sayıda test ve dağıtım teknolojisi ile entegre olarak yazılımın sürekli olarak teslim edilmesini sağlar. Jenkins ile kuruluşlar otomasyon yoluyla yazılım geliştirme süreçlerini hızlandırabilirler. Jenkins, yapı, belge, test, paket, aşama, dağıtım, statik analiz ve çok daha fazlası dahil olmak üzere her türlü geliştirme yaşam döngüsü sürecini entegre edebilmektedir.

Jenkins, eklentiler yardımıyla Sürekli Entegrasyonu (Continuous Integration - CI) gerçekleştirir. Eklentiler, çeşitli DevOps aşamalarının entegrasyonuna izin verir. Belirli bir aracın entegre edilmesi istendiğinde, o araç için eklentileri yüklemek yeterlidir. Örneğin: Git, Maven 2, Amazon EC2, HTML vb. (Smart, 2011).

Sürekli Entegrasyon (CI) erken hata keşfi ve genel bir kalite değerlendirmesi ve iyileştirmesi amacıyla özel sunuculardaki derleme sürecini otomatikleştirmeyi amaçlamaktadır (Zampetti vd., 2021). Bu gibi nedenlerden dolayı, CI endüstride ve açık kaynakta oldukça benimsenmiştir. Onun evrimi olan Sürekli Teslimat (Continuous Delivery - CD), kısa döngülü sürümlerin gerçekleştirilmesine yardımcı olmaktadır.

5.3 SBOM Tool

SBOM aracı, her türlü yazılım ürünü için SPDX 2.2 uyumlu SBOM'lar oluşturmak için son derece ölçeklenebilir ve kurumsal kullanıma hazır bir araçtır. Araç, bileşenleri algılamak için Bileşen Algılama (Component Detection) kütüphanelerini ve bu bileşenlerin lisans bilgilerini doldurmak için ClearlyDefined API'sini kullanır (Url-5). SBOM aracı CI/CD işlem hatlarına (GitHub Action ve Azure DevOps Pipeline gibi) entegre edilebilir.

Aşağıdaki bir örneği verilen komut ile SBOM Tools aracı çalıştırılabilir:
sbom-tool generate -b <drop path> -bc <build components path> -pn <package name> -pv <package version> -ps <package supplier> -nsb <namespace uri base> (Url-5).

"generate -b <drop path>" parametresi, sbom aracının derleme dizinidir. Projeye ait tüm dosyalar derlenerek SBOM'un dosyalar bölümüne eklenecektir. "-bc <build_components_path" parametresi, projeye ait kaynak kodların yer aldığı dizindir. SBOM aracı, paketi oluşturmak için hangi bileşenlerin kullanıldığını belirlemek üzere *.csproj veya package.json gibi proje dosyalarını aramak için bu dizini tarayacaktır. Diğer parametreler paket adı, sürümü ve isim uzayı gibi tanımlayıcı bilgileri belirlemek için kullanılır. Üretilen SBOM'u benzersiz bir şekilde tanımlayan benzersiz bir isim uzayı verilebilir. SBOM içindeki isim uzayı için benzersiz bir tanımlayıcı oluşturulur. Ancak tüm kuruluş için ortak olacak bir temel URI'ye ihtiyaç vardır. Örneğin, -nsb parametresi için örnek bir değer *https://companyName.com/teamName* şeklinde belirlenebilir. Bu durumda SBOM aracı *https://companyName.com/teamName/<packageName>/<packageVersion>/<new-guid>* gibi görünen bir isim uzayı oluşturacaktır.

5.3.1 Bileşen Algılama / Component Detection

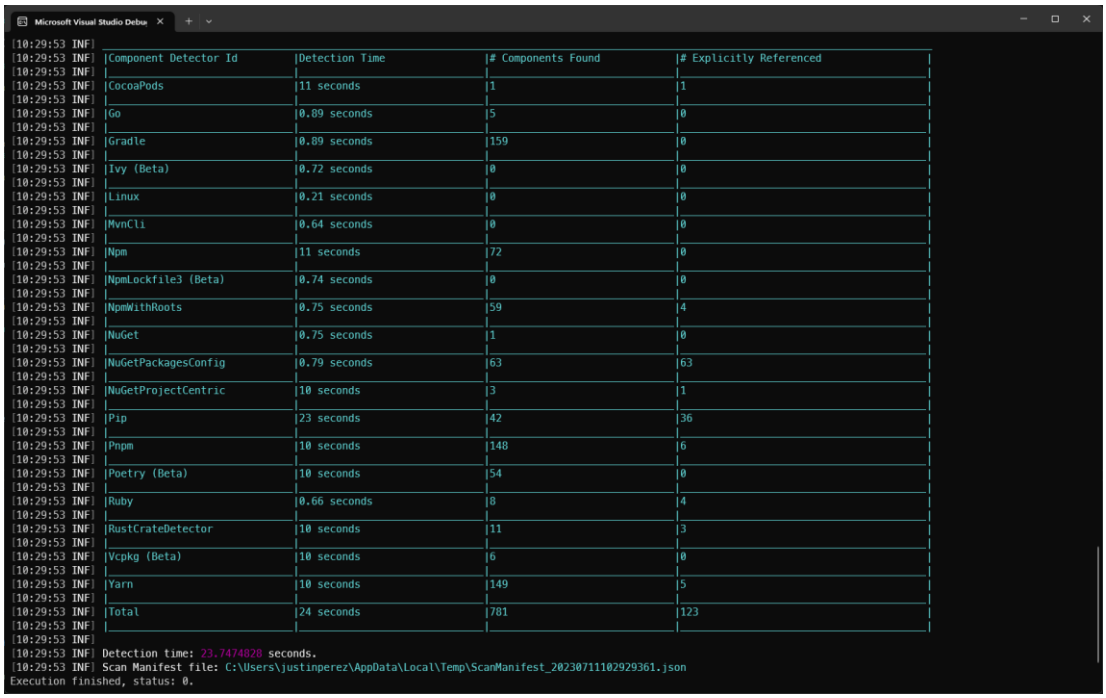
Bileşenler, BT altyapısının birincil yapı taşlarıdır. Sistemlerin genel işlevselliği ve işleyişinde farklı rollere hizmet ederler (Laan, 2017). İki genel bileşen kategorisi vardır. Bunlar yazılım bileşenleri ve ağ bileşenleridir. Yazılım bileşenleri, bir sistemin yazılım altyapısını oluşturan tek tek kod parçalarını, uygulamaları veya programları ifade eder (Paik vd., 2005). Yazılım bileşenleri yönetimiyle ilgili zorluklar bu tezde sorulan sorularla doğrudan ilgilidir. Yazılım bileşenleri

uygulamaları, kütüphaneleri, çerçeveleri ve modülleri/sınıfları içerir (Crnkovic vd., 2010).

Uygulamalar, belirli görevleri veya işlevleri yerine getiren son kullanıcı yazılımlarını ifade eder. Örnek olarak kelime işlemciler ve web tarayıcıları verilebilir. Kütüphaneler, geliştiricilerin önceden var olan işlevleri uygulamalara dahil etmesine olanak tanıyan yeniden kullanılabilir kod modüllerini ifade eder. Çerçeve, yazılım uygulamalarının üzerine inşa edilebileceği bir temel sağlayan kapsamlı bir araç kütüphaneleri ve kurallar setini tanımlar. Modüller ve sınıflar, kodun sürdürülebilirliğini artırmaya yönelik belirli işlevler sağlayan bir program içindeki daha küçük kod birimleridir (Crnovic vd.,2010).

Bileşen Algılama (Component Detection - CD), derleme zamanında kullanılmak üzere tasarlanmış bir paket tarama aracıdır. Çeşitli paket ekosistemlerinde tespit edilen tüm bileşenlerin grafik tabanlı bir çıktısını üretir (Url-9).

Bileşen Tespiti, uygulamalardaki bağımlılıkları tespit etmek için bir kütüphane olarak da kullanılabilir. Şekil 5.2'de örnek bileşen tespiti görülmektedir.



Şekil 5.2 Örnek Bileşen Tespiti

Bileşen Algılama aracı, Çizelge 5.1'de sunulan ekosistemdeki kütüphanelerin algılanmasını destekler.

Çizelge 5.1 Bileşen algılamanın desteklediği kütüphaneler

Ekosistem	Tarama	Grafik Oluşturma
CocoaPods	√	√
Go	√	X
Gradle (lockfiles only)	√	X
Linux (Debian, Alpine, Rhel, Centos, Fedora, Ubuntu)	√ Sfyt aracılığıyla	X
Maven	√	√
NPM (include Yarn, Pnpm)	√	√
NuGet (include Paket)	√	√
Pip (Python)	√	√
Poetry (Python, lockfiles only)	√	X
Ruby	√	√
Rust	√	√

5.4 Bomber

Bir satıcı, kapalı kaynak ürünlerinden biri için Yazılım Malzeme Listesi (SBOM) sunduğunda bu veri genellikle JSON formatında bir dosya şeklindedir. Bu dosya üzerinden temelde, SBOM içinde listelenen bileşenlerden herhangi birinin güvenlik açığı olup olmadığının ve bu bileşenlerin ne tür lisanslara sahip olduğunun öğrenilmesi istenmektedir. Bu yaklaşım, ürünü kullanarak ne tür bir risk alındığının belirlenmesine yardımcı olacaktır (Url-6).

Bomber ürününün temel amacı, bir SBOM'da tanımlanan bileşenler için güvenlik açıklarını ve lisans bilgilerini bulmaktır. Bomber, herhangi bir JSON veya XML tabanlı CycloneDX formatını veya JSON SPDX veya Syft formatlı bir SBOM'u okuyabilir ve herhangi bir güvenlik açığı olup olmadığını oldukça hızlı bir şekilde belirleyebilir.

Yazılım açık ya da kapalı kaynak kodlu olabilir. GitHub'da yer alan üçüncü taraf bileşenlere veya herhangi bir genel kod deposuna açık kaynak olarak bakılabilir. Teknik olarak, şirketlerin dahili olarak oluşturduğu yazılımlar da açık kaynaklıdır - halka açık değildir, ancak dahili ekipler bunu görebilir. Kapalı kaynaklı yazılımlar da dahili olabilir, ancak bunlar genellikle harici satıcılardan satın alınan yazılımlardır.

Şirketler, GitHub, Sonatype, Snyk gibi satıcılar tarafından sağlanan SCA araçlarını kullanarak her türlü açık kaynağı tarayabilir ve güvenlik açığı verileri sağlayabilir ve

hatta bazı durumlarda SBOM'lar oluşturabilir. Ancak henüz kod görünürlüğü olmayan kapalı kaynak yazılımlar için bunları yapabilmeleri mümkün değildir. Bu noktada SBOM'lar ve Bomber gibi araçlar devreye girer. SBOM'lar kaynak kod erişimi olmayan yazılımlar için bileşen analizi sağlar ve Bomber gibi araçlarla da SBOM'daki herhangi bir bileşenin güvenlik açığı olup olmadığı belirlenebilir.

Bomber güvenlik açığı bilgileri için birden fazla kaynağı destekler. Bunlara sağlayıcı (providers) adı verilir. Aktif olarak Bomber varsayılan sağlayıcı olarak OSV'yi (Open Source Vulnerabilities) kullanmaktadır, ancak Sonatype OSS Index veya Snyk'i de kullanılabilir.

OSV, bomber için varsayılan sağlayıcıdır. Açık kaynak için güvenlik açığı bilgisi üretmeye ve tüketmeye yönelik açık, hassas ve dağıtılmış bir yaklaşımdır. Herhangi bir hizmete kaydolmadan, bir parola veya token alınmadan kullanılabilir (Url-6).

5.4.1 Güvenlik Açığı

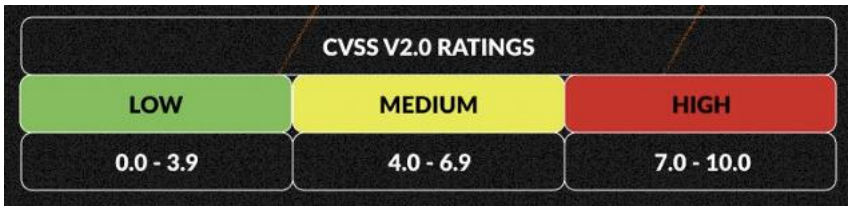
Güvenlik açığı, bir sistem, ağ veya yazılım uygulamasındaki zayıflık veya kusur olarak tanımlanır. Bir saldırgan, sistemi veya verileri tehlikeye atmak için bundan yararlanabilir. Güvenlik açıkları, yetkisiz erişim, saldırılar veya diğer güvenlik ihlalleri için potansiyel giriş noktalarını temsil eder ve bir BT altyapısı içinde çeşitli seviyelerde bulunabilir. Bunlar yazılım ve ağlardan, işletim sistemleri ve yapılandırma ayarlarına kadar uzanır. Güvenlik açıkları insan eylemlerinden de kaynaklanabilir (Safianu vd., 2016). Yazılım zafiyetleri programlama hatalarından, güvensiz konfigürasyonlardan ya da güncel olmayan / yamalanmamış yazılımlardan kaynaklanır (Sibal vd., 2017). Ağ güvenlik açıkları, ağ bileşenlerinin, yönlendiricilerin, anahtarların ve güvenlik duvarlarının yapılandırılmasındaki veya tasarımındaki sorunlardan kaynaklanmaktadır. İşletim sistemi güvenlik açıkları, saldırganların güvenlik açıklarından yararlanmasına ve sistem üzerinde yetkisiz kontrol elde etmesine izin verebilecek bir işletim sisteminin tasarımındaki veya uygulanmasındaki kusurları ifade eder. Konfigürasyon zafiyetleri sistemlerin, uygulamaların veya cihazların güvensiz veya optimal olmayan konfigürasyonlarını ifade eder. İnsan eylemleri, zayıf parola yönetimi, sosyal mühendislik veya en iyi güvenlik uygulamaları konusunda yetersiz çalışan eğitimi anlamına gelir. Hepsi de

yetkisiz erişim elde etmek veya sistem bütünlüğünü tehlikeye atmak için potansiyel olarak istismar edilebilir.

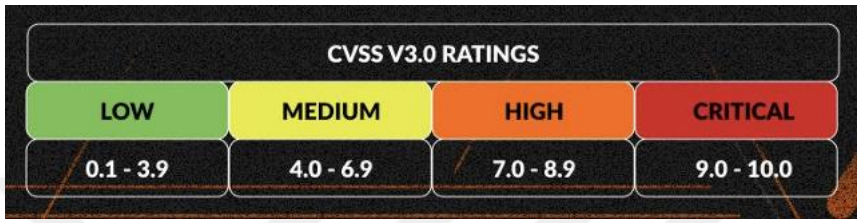
Güvenlik genellikle konuşlandırılmış sistemlerdeki açıklardan faydalanmaya çalışan saldırganlar ile bunu imkansız hale getirmek isteyen güvenlik çalışanları arasındaki bir silahlanma yarışı olarak görülür. Bu nedenle, ortaya çıkan eğilimler büyük ölçekli sorunlara dönüşmeden önce karşı önlemler hakkında düşünebilmek için güvenlik alanında ortaya çıkan eğilimleri bilmek arzu edilir. Güvenlik açığı raporlarının büyük ve kamuya açık kaynaklarından biri MITRE'de bulunan Ortak Güvenlik Açıkları ve Maruziyetler (Common Vulnerability Exposure - CVE) veritabanıdır. CVE'nin sıkça sorulan sorular bölümüne göre, "CVE, kamuoyunca bilinen sorunlar için ortak isimler sağlamayı amaçlayan bir bilgi güvenliği açıkları ve maruziyetler listesidir. CVE'nin amacı, bu 'ortak numaralandırma' ile ayrı güvenlik açığı yetenekleri (araçlar, depolar ve hizmetler) arasında veri paylaşımını kolaylaştırmaktır." Bu büyük ölçüde doğru gibi görünmektedir: diğer veri tabanları, CVE'de de yer alan bir güvenlik açığına atıfta bulunduklarında genellikle CVE tanımlayıcılarını içerirler. Halka açık veriler üzerinde yapılan tek büyük çalışma 2007 yılında CVE üzerinde yapılmıştır. Bu çalışma, Common Weakness Enumeration (CWE) adı verilen bir sınıflandırma sistemi yardımıyla girişlerin manuel olarak sınıflandırılmasına dayanmaktadır. CWE, yazılım zayıflıkları için eksiksiz bir sözlük olmayı amaçlamaktadır (Neuhaus vd., 2010).

5.4.2 Zafiyet Değerlendirme Kavramı

Zafiyet, bir sistemde, web uygulamasında, network kurulumunda olan ve kötü saldırılara sebep olacak sistemsel problemlere verilen addır. Şirketler ve tehdit avcıları herhangi bir zafiyete karşı sistemleri sürekli olarak tarayıp, buldukları zafiyetleri raporlarlar. Bu zafiyetlerin zararlılık seviyesini belirlemek için CVSS (Common Vulnerability Scoring System) adı verilen bir sistem kullanılır. Günümüzde 3 versiyonu kullanılmaktadır. Ancak 2.0 versiyonu da hala kabul görmektedir (Url-10). Ortak Güvenlik Açığı Puanlama Sistemi (CVSS), yazılım güvenlik açıklarının ciddiyetini değerlendirmek için standartlaştırılmış bir yöntemdir. CVSS güvenlik uzmanları, yazılım satıcıları ve devlet kurumları tarafından güvenlik açığı giderme faaliyetlerine öncelik vermek için kullanılır (Mell vd., 2006), (Url-12).

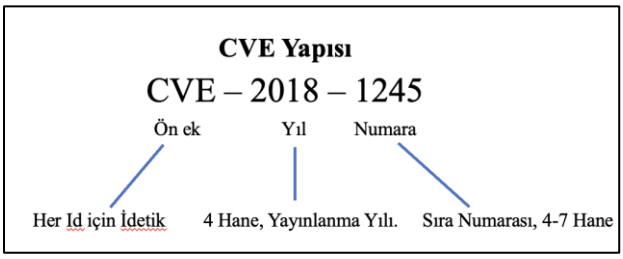


Şekil 5.3 CVSS V2.0 Ratings



Şekil 5.4 CVSS V3.0 Ratings

CVE (Common Vulnerabilities and Exposures) numarası, genel zafiyet ve etkilenmeler olarak çevrilebilecek, güvenlik açıklarını kimliklendirebilmek için kullanılan bir tanımlayıcıdır. CVE numarasının ilk 4 numarası, zafiyetin çıkış yılını, sonraki 4 haneli rakamlar ise rastgele atanmış belirteçleri ifade eder.



Şekil 5.5 CVE Yapısı

CWE (Common Weakness Enumeration) numarasını ise CVE'deki zafiyetin sebebi olarak adlandırılabilir. Örneğin, bir uygulamada keşfedilen SQL enjeksiyonu zafiyeti genel olarak CVE numarasını oluştururken, SQL enjeksiyonun kendisi başlı başına bir CWE değerine sahiptir.

Common Weakness Enumeration (CWE), saldırganlar tarafından sistemlere veya verilere yetkisiz erişim elde etmek için kullanılabilecek yazılım zayıflıklarının bir

listesidir. CWE, yazılım zayıflıklarını tanımlamak, sınıflandırmak ve kategorize etmek için yaygın olarak kullanılan bir standarttır (Martin vd., 2011).

CWE, yazılım sistemlerindeki güvenlik açıklarını tespit etmek için kullanılabilecek bir araçtır. Geliştiriciler bu aracı kullanarak düzeltme çabalarını daha etkili bir şekilde önceliklendirebilirler. Bunun nedeni, CWE'nin yazılım zayıflıkları hakkında iletişim kurmak için ortak bir dil sağlamasıdır. Bu da geliştiriciler, güvenlik uzmanları ve satıcılar arasındaki iletişimi geliştirmeye yardımcı olabilir. Geliştiriciler CWE'yi kullanarak en kritik güvenlik açıklarını belirleyebilir ve öncelikle bunların ele alınmasını sağlayabilir, böylece güvenlik ihlali riskini azaltabilirler (Url-11).

CWE hiyerarşik bir zayıflık taksonomisi sunar. Taksonominin en üst seviyesi aşağıdaki kategorilere ayrılmıştır:

- Girdi Doğrulama: Girdi doğrulamadaki zayıflıklar saldırganların sistemlere kötü amaçlı kod enjekte etmesine izin verebilir.
- Veri İşleme: Veri işlemedeki zayıflıklar saldırganların hassas verilere erişmesine veya bunları değiştirmesine olanak tanıyabilir.
- Kaynak Yönetimi: Kaynak yönetimindeki zayıflıklar saldırganların sistem kaynaklarını tüketmesine veya sistemlere yetkisiz erişim sağlamasına izin verebilir.
- Güvenlik Özellikleri: Güvenlik özelliklerindeki zayıflıklar saldırganların güvenlik kontrollerini atlamasına izin verebilir.
- Mimari Sorunlar: Mimari konulardaki zayıflıklar sistemleri saldırılara karşı daha savunmasız hale getirebilir.

CWE kullanımının aşağıda ifade edilen çeşitli faydaları vardır:

- Standartlaştırılmış dil: CWE, yazılım zayıflıkları hakkında iletişim kurmak için standartlaştırılmış bir dil sağlar. Bu, geliştiriciler, güvenlik uzmanları ve satıcılar arasındaki iletişimi geliştirmeye yardımcı olabilir.
- Geliştirilmiş güvenlik açığı yönetimi: CWE, güvenlik açığı yönetim süreçlerini iyileştirmek için kullanılabilir. Bu, güvenlik açıklarının tutarlı bir şekilde tanımlanmasını, sınıflandırılmasını ve önceliklendirilmesini sağlamaya yardımcı olabilir.

- Azaltılmış risk: CWE, yazılım güvenlik açıklarının istismar edilme riskini azaltmaya yardımcı olabilir. Bu, kuruluşların siber saldırılardan korunmasına yardımcı olabilir.

6. DENEYSEL ÇALIŞMALAR

Çalışma kapsamında önerilen modeli doğrulamak için öncelikle GitHub üzerinde kaynak kodları yer alan popüler C# projeleri arasından 10 proje belirlenmiştir. Deneysel çalışma kapsamında incelenen projeler ve bazı özellikleri Çizelge 6.1'de listelenmiştir.

Çizelge 6.1 Deneysel çalışma için seçilen projeler

No	Proje Adı	Star	Fork	Repo URI (github)
1	OpenRA	13768	2638	OpenRA
2	Runtime	13403	4391	dotnet
3	Efcore	13060	3064	dotnet
4	Abp	11916	3275	abpframework
5	Aspnetboilerplate	11420	3761	aspnetboilerplate
6	Captura	9426	1754	MathewSachin
7	Spectre.console	9336	421	spectre.console
8	RestSharp	9239	2335	restsharp
9	Radarr	8554	889	Radarr
10	Ocelot	7999	1613	ThreeMammals

İncelenen projelerin zafiyetlerini belirlemek için MITRE-CVE sistemi üzerinden sorgulamalar gerçekleştirilmiştir. Değerlendirilen projelere ait MITRE üzerinde herhangi bir zafiyet kaydına rastlanmamıştır. Bunun üzerine değerlendirmeye alınan projelerin SBOM'ları oluşturulmuş ve bu SBOM'lar üzerinden zafiyetleri araştırılmıştır. Bomber üzerinden elde edilen örnek bir zafiyet analiz raporu (Radarr projesine ait) Şekil 6.1'de sunulmuştur.

TYPE	NAME	VERSION	SEVERITY	VULNERABILITY	EPSS %
npm	traverse	7.22.11	CRITICAL	CVE-2023-45133	24%
	postcss	6.0.23	MODERATE	CVE-2021-23382,SNYK-JAVA-ORGWEBJARSNPM-1255641,SNYK-JS-POSTCSS-1255640	N/A
		6.0.23	MODERATE	CVE-2023-44270	18%
		8.4.29	MODERATE	CVE-2023-44270	18%
		8.4.23	MODERATE	CVE-2023-44270	18%
	css-tools	4.3.1	MODERATE	CVE-2023-48631	14%
	color-string	0.3.0	MODERATE	CVE-2021-29060	53%
nuget	System.Security.Cryptography.Pkcs	6.0.0	HIGH	BIT-dotnet-2023-29331,BIT-dotnet-sdk-2023-29331,CVE-2023-29331	N/A

Şekil 6.1 Bomber üzerinden alınan örnek zafiyet analiz raporu (Radarr projesi)

Proje bağımlılıklarının içerdiği zafiyetler Severity ve EPSS değerine göre incelenmeli ve ilgili güvenlik açığının doğurabileceği etkiler ürün kullanıcıları tarafından dikkate alınmalıdır. Bomber tarafından sunulan zafiyet analiz raporunda ifade edilen Exploit Tahmin Puanlama Sistemi (Exploit Prediction Scoring System - EPSS), bir yazılım güvenlik açığının gerçek dünya ortamında istismar edilme

olasılığını tahmin etmeye yönelik veri odaklı bir çalışmanın sonucudur. Amacı, ağ savunucularının güvenlik açığını giderme çabalarına daha iyi öncelik vermelerine yardımcı olmaktır. Diğer endüstri standartları, bir güvenlik açığının doğuştan gelen özelliklerini yakalamak ve ciddiyet ölçümleri sağlamak için faydalı olsa da tehdidi değerlendirme yetenekleri sınırlıdır. EPSS, CVE'den gelen mevcut tehdit bilgilerini ve gerçek dünyadaki istismar verilerinden yararlandığı için bu boşluğu doldurur. EPSS modeli 0 ile 1 (%0 ile 100) arasında bir olasılık puanı üretir. Puan ne kadar yüksek olursa, bir güvenlik açığından yararlanma olasılığı da o kadar yüksek olur. Sunulan bu değer, zafiyetin önceliğini belirlemede yardımcı olacaktır (Jacobs vd., 2023). Severity, zafiyetin teknik bir perspektiften değerlendirilmesiyle ilgilidir. Zafiyetin etkisi ve olası sonuçları üzerine odaklanır. Severity statüleri önem derecesine göre Critical, High, Moderate, Low değerlerini içerebilir.

Deneysel çalışma kapsamında seçilen tüm projeler için SBOM'lar üretilmiş ve üretilen bu SBOM'lar üzerinden zafiyet analizleri gerçekleştirilmiştir.

6.1 Deneysel Çalışmada Kullanılan Projelerde Bulunan Bulgular

Aralık 2023 – Ocak 2024 arasında yapılan analiz çalışmaları sonucunda deneysel çalışma için seçilen 10 farklı projeye ait SBOM tabloları ve zayifetleri devam eden alt başlıklarda sunulmuştur.

6.1.1 OpenRA

OpenRA projesi Command & Conquer gibi erken Westwood oyunları için açık kaynaklı gerçek zamanlı strateji oyun motorudur. Windows, Linux, BSD ve Mac OS X üzerinde çalışabilir.

OpenRA projesinin bağımlılıkları SBOM Tool ile çıkarılmıştır ve bu bağımlılıklar Şekil 6.2'de gösterilmiştir. Buna göre 28 tane bağımlılık bulunmuştur. Bulunan bağımlılıklarda toplamda 18 tane açıklık referansı çıkmıştır.

Component Detector Id	Detection Time	# Components Found	# Explicitly Referenced
CocoaPods	0,15 seconds	0	0
Go	0,15 seconds	0	0
Gradle	0,15 seconds	0	0
Ivy (Beta)	0,059 seconds	0	0
Linux	0,0037 seconds	0	0
MvnCli	0,058 seconds	0	0
Npm	0,15 seconds	0	0
NpmLockfile3 (Beta)	0,14 seconds	0	0
NpmWithRoots	0,14 seconds	0	0
NuGet	0,14 seconds	0	0
NuGetPackagesConfig	0,14 seconds	0	0
NuGetProjectCentric	0,15 seconds	28	18
Pip	0,14 seconds	0	0
Pnpm	0,14 seconds	0	0
Poetry (Beta)	0,14 seconds	0	0
Ruby	0,14 seconds	0	0
RustCrateDetector	0,14 seconds	0	0
SPDX22SBOM	0,14 seconds	0	0
Vcpkg (Beta)	0,14 seconds	0	0
Yarn	0,14 seconds	0	0
Total	0,16 seconds	28	18

Şekil 6.2 OpenRA bağımlılık listesi

OpenRA projesinde bulunan bağımlılıkların zafiyet taraması Bomber aracı ile gerçekleştirilmiştir. İçerdiği zafiyetlere ait bulgular Şekil 6.3'te gösterilmiştir.

TYPE	NAME	VERSION	SEVERITY	VULNERABILITY	EPSS %
nuget	NuGet.CommandLine	4.4.1	MODERATE	BIT-dotnet-2022-30184,BIT-dotnet-sdk-2022-30184,CVE-2022-30184	N/A

Total vulnerabilities found: 1

RATING	COUNT
CRITICAL	0
HIGH	0
MODERATE	1
LOW	0

Şekil 6.3 OpenRA için bulunan zayifetler

Bomber taraması sonrasında CVE-2022-30184 numaralı zafiyet bulunmuştur. Bu zafiyet .NET ve Visual Studio'da ortaya çıkan bilgi ifşası zafiyetini ifade etmektedir.

6.1.2 Runtime

Runtime projesi bulut, mobil, masaüstü ve IoT uygulamaları için platformlar arası bir çalışma zamanı kütüphanesidir. Runtime projesinin bağımlılıkları SBOM Tool ile çıkarılmıştır ve bu bağımlılıklar Şekil 6.4'de gösterilmiştir. 351 tane bağımlılık bulunmuştur. Bulunan bağımlılıklarda toplamda 80 tane açıklık referansı çıkmıştır.

Component Detector Id	Detection Time	# Components Found	# Explicitly Referenced
CocoaPods	2,1 seconds	0	0
Go	2,1 seconds	0	0
Gradle	2,1 seconds	0	0
Ivy (Beta)	0,056 seconds	0	0
Linux	0,0035 seconds	0	0
MvnCli	0,055 seconds	0	0
Npm	2,1 seconds	1	0
NpmLockfile3 (Beta)	2,1 seconds	0	0
NpmWithRoots	2,1 seconds	211	17
NuGet	2,1 seconds	0	0
NuGetPackagesConfig	2,1 seconds	1	1
NuGetProjectCentric	2,1 seconds	138	62
Pip	2 seconds	0	0
Pnpm	2 seconds	0	0
Poetry (Beta)	2 seconds	0	0
Ruby	2 seconds	0	0
RustCrateDetector	2 seconds	0	0
SPDX22SBOM	2 seconds	0	0
Vcpkg (Beta)	2 seconds	0	0
Yarn	2 seconds	0	0
Total	2,1 seconds	351	80

Şekil 6.4 Runtime bağımlılık listesi

Runtime projesinde bulunan bağımlılıkların zafiyet taraması Bomber aracı ile gerçekleştirilmiştir. İçerdiği zafiyetlere ait bulgular Şekil 6.5'te gösterilmiştir.

TYPE	NAME	VERSION	SEVERITY	VULNERABILITY	EPSS %
nuget	System.Security.Cryptography.Xml	6.0.0	MODERATE	BIT-dotnet-2022-34716,BIT-dotnet-sdk-2022-34716,CVE-2022-34716	N/A
	System.Security.Cryptography.Pkcs	6.0.1	HIGH	BIT-dotnet-2023-29331,BIT-dotnet-sdk-2023-29331,CVE-2023-29331	N/A

Total vulnerabilities found: 2

RATING	COUNT
CRITICAL	0
HIGH	1
MODERATE	1
LOW	0

Şekil 6.5 Runtime için bulunan zafiyetler

Bomber taraması sonrasında ciddiyet seviyesi yüksek ve moderate olan iki farklı zafiyet bulunmuştur. Ciddiyet seviyesi yüksek olan CVE-2022-34716 numaralı zafiyet .NET Spoofing Güvenlik Açığını ifade etmektedir. Ciddiyet seviyesi moderate olan CVE-2023-29331numaralı zafiyet .NET Framework ve Visual Studio Hizmet Reddi Güvenlik Açığını ifade etmektedir.

6.1.3 Efcore

EF Core projesi, .NET için modern bir nesne-veritabanı eşleyicisidir. LINQ sorgularını, değişiklik takibini, güncellemeleri ve şema geçişlerini destekler. Efcore projesinin bağımlılıkları SBOM Tool ile çıkarılmıştır ve bu bağımlılıklar Şekil 6.6'da gösterilmiştir. Projeye ait 190 adet bağımlılık bulunmuştur. Bulunan bağımlılıklarda toplamda 47 tane açıklık referansı çıkmıştır.

Component Detector Id	Detection Time	# Components Found	# Explicitly Referenced
CocoaPods	0,16 seconds	0	0
Go	0,16 seconds	0	0
Gradle	0,16 seconds	0	0
Ivy (Beta)	0,054 seconds	0	0
Linux	0,0035 seconds	0	0
MvnCli	0,053 seconds	0	0
Npm	0,15 seconds	0	0
NpmLockfile3 (Beta)	0,15 seconds	0	0
NpmWithRoots	0,15 seconds	0	0
NuGet	0,15 seconds	0	0
NuGetPackagesConfig	0,16 seconds	1	1
NuGetProjectCentric	0,38 seconds	189	46
Pip	0,15 seconds	0	0
Pnpm	0,15 seconds	0	0
Poetry (Beta)	0,15 seconds	0	0
Ruby	0,15 seconds	0	0
RustCrateDetector	0,15 seconds	0	0
SPDX22SBOM	0,15 seconds	0	0
Vcpkg (Beta)	0,15 seconds	0	0
Yarn	0,15 seconds	0	0
Total	0,39 seconds	190	47

Şekil 6.6 Efcore bağımlılık listesi

Efcore projesinde bulunan bağımlılıkların zafiyet taraması Bomber aracı ile gerçekleştirilmiştir. İçerdiği zafiyetlere ait bulgular Şekil 6.7'de gösterilmiştir.

Efcore, CVE Mitre'de kontrol edildiğinde herhangi bir zafiyet bulunmamıştır. Bomber taraması sonrasında ciddiyet seviyesi yüksek iki farklı açık bulunmuştur. Bu zayifetler incelendiğinde CVE-2023-36414 numaralı zafiyet Azure Identity SDK Uzaktan Kod Yürütme Güvenlik Açığını ifade etmektedir. İstismar edilebilirliği (EPSS) %56 oranındadır.

TYPE	NAME	VERSION	SEVERITY	VULNERABILITY	EPSS %
nuget	Newtonsoft.Json	10.0.2	HIGH	CWE-755	N/A
	Azure.Identity	1.7.0	HIGH	CVE-2023-36414	56%

Total vulnerabilities found: 2

RATING	COUNT
CRITICAL	0
HIGH	2
MODERATE	0
LOW	0

Şekil 6.7 Efcore için bulunan zafiyetler

6.1.4 Abp

ASP.NET Core için Açık Kaynak Web Uygulama Çerçevesi projesidir. .NET ve ASP.NET Core platformları üzerinde en iyi uygulamalarla kurumsal yazılım çözümleri oluşturmak için fikir sahibi bir mimari sunar. Temel altyapı, üretime hazır başlangıç şablonları, uygulama modülleri, kullanıcı arayüzü temaları, araçlar, kılavuzlar ve belgeler sağlar. abp projesinin bağımlılıkları SBOM Tool ile çıkarılmıştır ve bu bağımlılıklar Şekil 6.8'de gösterilmiştir. Abp projesinde 3600 tane bağımlılık ve 215 tane açıklık referansı bulunmuştur.

Component Detector Id	Detection Time	# Components Found	# Explicitly Referenced
CocoaPods	1,1 seconds	0	0
Go	1,1 seconds	36	0
Gradle	1,1 seconds	0	0
Ivy (Beta)	0,054 seconds	0	0
Linux	0,0032 seconds	0	0
MvnCli	0,053 seconds	0	0
Npm	1,1 seconds	90	0
NpmLockfile3 (Beta)	1,1 seconds	0	0
NpmWithRoots	1,1 seconds	397	9
NuGet	1,1 seconds	0	0
NuGetPackagesConfig	1,1 seconds	0	0
NuGetProjectCentric	1,9 seconds	315	118
Pip	1,1 seconds	0	0
Pnpm	1,1 seconds	0	0
Poetry (Beta)	1,1 seconds	0	0
Ruby	1,1 seconds	0	0
RustCrateDetector	1,1 seconds	0	0
SPDX22SBOM	1,1 seconds	0	0
Vcpkg (Beta)	1,1 seconds	0	0
Yarn	1,1 seconds	2762	88
Total	1,9 seconds	3600	215

Şekil 6.8 Abp bağımlılık listesi

Abp projesinde bulunan bağımlılıkların zafiyet taraması Bomber aracı ile gerçekleştirilmiştir. İçerdiği zafiyetlere ait bulgular Şekil 6.9'te gösterilmiştir.

Total vulnerabilities found: 190

RATING	COUNT
CRITICAL	18
HIGH	81
MODERATE	76
LOW	15

Şekil 6.9 Abp için bulunan zafiyetler

Bomber taraması sonucunda 190 tane zafiyet bulunmuştur. Bunların 18'i kritik, 81'i yüksek, 76'sı moderate ve 15 tanesi düşük ciddiyete sahiptir.

6.1.5 Aspnetboilerplate

ASP.NET Boilerplate projesi bir Web Uygulama Framework'üdür. ASP.NET Boilerplate projesinin bağımlılıkları SBOM Tool ile çıkarılmıştır ve bu bağımlılıklar Şekil 6.10'da gösterilmiştir. ASP.NET Boilerplate projesinde 434 tane bağımlılık, 90 tane açıklık referansı bulunmuştur.

Component Detector Id	Detection Time	# Components Found	# Explicitly Referenced
CocoaPods	0,17 seconds	0	0
Go	0,17 seconds	0	0
Gradle	0,17 seconds	0	0
Ivy (Beta)	0,055 seconds	0	0
Linux	0,0034 seconds	0	0
MvnCli	0,054 seconds	0	0
Npm	0,17 seconds	0	0
NpmLockfile3 (Beta)	0,17 seconds	0	0
NpmWithRoots	0,17 seconds	0	0
NuGet	0,17 seconds	0	0
NuGetPackagesConfig	0,17 seconds	0	0
NuGetProjectCentric	0,54 seconds	434	90
Pip	0,17 seconds	0	0
Pnpm	0,17 seconds	0	0
Poetry (Beta)	0,17 seconds	0	0
Ruby	0,17 seconds	0	0
RustCrateDetector	0,17 seconds	0	0
SPDX22SBOM	0,17 seconds	0	0
Vcpkg (Beta)	0,17 seconds	0	0
Yarn	0,17 seconds	0	0
Total	0,55 seconds	434	90

Şekil 6.10 Aspnetboilerplate bağımlılık listesi

ASP.NET Boilerplate projesinde bulunan bağımlılıkların zafiyet taraması Bomber aracı ile gerçekleştirilmiştir. İçerdiği zafiyetlere ait bulgular Şekil 6.11'de gösterilmiştir.

TYPE	NAME	VERSION	SEVERITY	VULNERABILITY	EPSS %
nuget	System.Text.Encodings.Web	4.4.0	CRITICAL	BIT-dotnet-2021-26701,BIT-dotnet-sdk-2021-26701,CVE-2021-26701	N/A
	System.Security.Cryptography.Xml	4.4.0	MODERATE	BIT-dotnet-2022-34716,BIT-dotnet-sdk-2022-34716,CVE-2022-34716	N/A
		4.4.0	HIGH	CVE-2018-0765	71%
		4.4.0	HIGH	CVE-2018-0764	71%
	System.Data.SqlClient	4.8.1	MODERATE	CVE-2022-41064	16%
	NuGet.Protocol	6.3.1	HIGH	CVE-2023-29337	62%
	NuGet.Common	6.3.1	HIGH	CVE-2023-29337	62%
	Microsoft.Owin.Security.Cookies	3.0.1	HIGH	BIT-dotnet-2022-29117,BIT-dotnet-sdk-2022-29117,CVE-2022-29117	N/A
	Microsoft.Owin	2.1.0	HIGH	BIT-aspnet-core-2020-1045,CVE-2020-1045	N/A
		2.1.0	HIGH	BIT-dotnet-2022-29117,BIT-dotnet-sdk-2022-29117,CVE-2022-29117	N/A
		3.0.1	HIGH	BIT-dotnet-2022-29117,BIT-dotnet-sdk-2022-29117,CVE-2022-29117	N/A
		3.0.1	HIGH	BIT-aspnet-core-2020-1045,CVE-2020-1045	N/A
	Azure.Identity	1.7.0	HIGH	CVE-2023-36414	56%

Total vulnerabilities found: 13

RATING	COUNT
CRITICAL	1
HIGH	10
MODERATE	2
LOW	0

Şekil 6.11 Aspnetboilerplate için bulunan zafiyetler

Bomber taraması sonucunda 13 tane zafiyet bulunmuştur. Bunlardan 1 tanesi kritik ve 10 tanesi yüksek ciddiyet seviyesine sahiptir. Bu zafiyetlerden CVE-2018-0765, CVE-2018-0764'ün EPSS'si %71, CVE-2023-29337'nin EPSS'si %62, CVE-2023-36414 EPSS'i %56 gibi ciddi değerler içeren istismar edilebilirlik puanına sahiptir.

6.1.6 Captura

Captura, ekran, ses, imleç, fare tıklamaları ve tuş vuruşlarını yakalama aracıdır. Captura projesinin bağımlılıkları SBOM Tool ile çıkarılmıştır ve bu bağımlılıklar Şekil 6.12'de gösterilmiştir. Projeye ait 42 adet bağımlılık ve 23 adet açıklık referansı bulunmuştur.

Component Detector Id	Detection Time	# Components Found	# Explicitly Referenced
CocoaPods	0,08 seconds	0	0
Go	0,079 seconds	0	0
Gradle	0,079 seconds	0	0
Ivy (Beta)	0,053 seconds	0	0
Linux	0,0031 seconds	0	0
MvnCli	0,052 seconds	0	0
Npm	0,078 seconds	0	0
NpmLockfile3 (Beta)	0,078 seconds	0	0
NpmWithRoots	0,078 seconds	0	0
NuGet	0,08 seconds	1	0
NuGetPackagesConfig	0,078 seconds	0	0
NuGetProjectCentric	0,12 seconds	41	23
Pip	0,078 seconds	0	0
Pnpm	0,078 seconds	0	0
Poetry (Beta)	0,078 seconds	0	0
Ruby	0,078 seconds	0	0
RustCrateDetector	0,078 seconds	0	0
SPDX22SBOM	0,078 seconds	0	0
Vcpkg (Beta)	0,078 seconds	0	0
Yarn	0,078 seconds	0	0
Total	0,13 seconds	42	23

Şekil 6.12 Captura bağımlılık listesi

Captura projesinde bulunan bağımlılıkların zafiyet taraması Bomber aracı ile gerçekleştirilmiştir. İçerdiği zafiyetlere ait bulgular Şekil 6.13'te gösterilmiştir.

TYPE	NAME	VERSION	SEVERITY	VULNERABILITY	EPSS %
nuget	System.Drawing.Common	4.5.0	CRITICAL	BIT-dotnet-2021-24112,BIT-dotnet-sdk-2021-24112,CVE-2021-24112	N/A
	Newtonsoft.Json	11.0.2	HIGH	CWE-755	N/A
		10.0.2	HIGH	CWE-755	N/A

Total vulnerabilities found: 3

RATING	COUNT
CRITICAL	1
HIGH	2
MODERATE	0
LOW	0

Şekil 6.13 Captura için bulunan zafiyetler

Bomber taraması sonucunda 1 kritik, 2 yüksek ciddiyete sahip olmak üzere 3 tane zafiyet bulunmuştur. Kritik olan CVE-2021-24112 zafiyeti, .NET Core Uzaktan Kod Yürütme Güvenlik Açığını ifade eder. CWE-755 zayıflığı, İstisnai Durumların Uygun Olmayan Şekilde Ele Alınması olarak ifade edilmektedir.

6.1.7 Spectre.console

Konsol uygulamaları oluşturmayı kolaylaştıran bir .NET kütüphanesidir. Spectre.console projesinin bağımlılıkları SBOM Tool ile çıkarılmıştır ve bu bağımlılıklar Şekil 6.14'te gösterilmiştir. 118 tane bağımlılık, 13 tane açıklık referansı bulunmuştur.

Component Detector Id	Detection Time	# Components Found	# Explicitly Referenced
CocoaPods	0,079 seconds	0	0
Go	0,078 seconds	0	0
Gradle	0,078 seconds	0	0
Ivy (Beta)	0,056 seconds	0	0
Linux	0,0032 seconds	0	0
MvnCli	0,055 seconds	0	0
Npm	0,079 seconds	1	0
NpmLockfile3 (Beta)	0,08 seconds	0	0
NpmWithRoots	0,08 seconds	105	5
NuGet	0,077 seconds	0	0
NuGetPackagesConfig	0,077 seconds	0	0
NuGetProjectCentric	0,1 seconds	12	8
Pip	0,077 seconds	0	0
Pnpm	0,077 seconds	0	0
Poetry (Beta)	0,077 seconds	0	0
Ruby	0,077 seconds	0	0
RustCrateDetector	0,077 seconds	0	0
SPDX22SBOM	0,077 seconds	0	0
Vcpkg (Beta)	0,077 seconds	0	0
Yarn	0,077 seconds	0	0
Total	0,11 seconds	118	13

Şekil 6.14 Spectre.console bağımlılık listesi

Spectre.console projesinde bulunan bağımlılıkların zafiyet taraması Bomber aracı ile gerçekleştirilmiştir. İçerdiği zafiyetlere ait bulgular Şekil 6.15'te gösterilmiştir.

TYPE	NAME	VERSION	SEVERITY	VULNERABILITY	EPSS %
npm	postcss	8.4.6	MODERATE	CVE-2023-44270	18%
	minimist	1.2.5	CRITICAL	CVE-2021-44906	84%

Total vulnerabilities found: 2

RATING	COUNT
CRITICAL	1
HIGH	0
MODERATE	1
LOW	0

Şekil 6.15 Spectre.console için bulunan zafiyetler

Bomber taraması sonucunda bulunan zafiyetlerden CVE-2023-44270 güvenlik açığı, harici güvenilmeyen CSS'i ayrıştırmak için PostCSS kullanan CSS yorumlayıcıları etkilemektedir. Bir saldırgan CSS'yi, PostCSS tarafından CSS yorumu olarak ayrıştırılan kısımları içerecek şekilde hazırlayabilir. PostCSS tarafından işlendikten sonra, bir yorumda yer almasına rağmen CSS düğümlerinde (kurallar, özellikler) PostCSS çıktısına dahil edilecektir.

CVE-2021-44906 güvenlik açığı, bir javascript kütüphanesi olan minimist'in 1.2.5 versiyonundan önceki sürümlerinde Prototip Kirliliğine karşı savunmasız olduğunu göstermektedir. Zafiyetin istismar edilebilirlik puanı (EPSS) %84 gibi çok yüksek bir değere sahiptir.

6.1.8 RestSharp

RestSharp, .NET için hafif bir HTTP API istemcisidir. RestSharp projesinin bağımlılıkları SBOM Tool ile çıkarılmıştır ve bu bağımlılıklar Şekil 6.16'da gösterilmiştir. Projede 311 tane bağımlılık, 26 tane açıklık referansı bulunmuştur.

Component Detector Id	Detection Time	# Components Found	# Explicitly Referenced
CocoaPods	0,068 seconds	0	0
Go	0,067 seconds	0	0
Gradle	0,067 seconds	0	0
Ivy (Beta)	0,052 seconds	0	0
Linux	0,0035 seconds	0	0
MvnCli	0,051 seconds	0	0
Npm	0,066 seconds	0	0
NpmLockfile3 (Beta)	0,066 seconds	0	0
NpmWithRoots	0,066 seconds	0	0
NuGet	0,066 seconds	0	0
NuGetPackagesConfig	0,066 seconds	0	0
NuGetProjectCentric	0,18 seconds	81	25
Pip	0,066 seconds	0	0
Pnpm	0,066 seconds	0	0
Poetry (Beta)	0,066 seconds	0	0
Ruby	0,066 seconds	0	0
RustCrateDetector	0,066 seconds	0	0
SPDX22SBOM	0,065 seconds	0	0
Vcpkg (Beta)	0,066 seconds	0	0
Yarn	0,077 seconds	230	1
Total	0,19 seconds	311	26

Şekil 6.16 RestSharp bağımlılık listesi

RestSharp projesinde bulunan bağımlılıkların zafiyet taraması Bomber aracı ile gerçekleştirilmiştir. İçerdiği zafiyetlere ait bulgular Şekil 6.17'de gösterilmiştir.

TYPE	NAME	VERSION	SEVERITY	VULNERABILITY	EPSS %
nuget	System.Drawing.Common	4.5.0	CRITICAL	BIT-dotnet-2021-24112,BIT-dotnet-sdk-2021-24112,CVE-2021-24112	N/A
	Newtonsoft.Json	11.0.2	HIGH	CWE-755	N/A
		10.0.2	HIGH	CWE-755	N/A

Total vulnerabilities found: 3

RATING	COUNT
CRITICAL	1
HIGH	2
MODERATE	0
LOW	0

Şekil 6.17 RestSharp için bulunan zafiyetler

Bomber taraması sonucunda bulunan zafiyetlerden CVE-2021-24112, .NET Core Uzaktan Kod Yürütme Güvenlik Açığını ifade etmektedir. Kritik önem derecesine sahiptir.

6.1.9 Radarr

Radarr, Usenet ve BitTorrent kullanıcıları için film koleksiyonu yöneticisidir. Radarr projesinin bağımlılıkları SBOM Tool ile çıkarılmıştır ve bu bağımlılıklar Şekil 6.18'de gösterilmiştir. Projeye ait 1117 tane bağımlılık, 174 tane açıklık referansı bulunmuştur.

Component Detector Id	Detection Time	# Components Found	# Explicitly Referenced
CocoaPods	0,77 seconds	0	0
Go	0,77 seconds	0	0
Gradle	0,77 seconds	0	0
Ivy (Beta)	0,057 seconds	0	0
Linux	0,0032 seconds	0	0
MvnCli	0,056 seconds	0	0
Npm	0,77 seconds	1	0
NpmLockfile3 (Beta)	0,77 seconds	0	0
NpmWithRoots	0,77 seconds	0	0
NuGet	0,77 seconds	0	0
NuGetPackagesConfig	0,77 seconds	0	0
NuGetProjectCentric	1,3 seconds	142	50
Pip	0,77 seconds	0	0
Pnpm	0,77 seconds	0	0
Poetry (Beta)	0,77 seconds	0	0
Ruby	0,77 seconds	0	0
RustCrateDetector	0,77 seconds	0	0
SPDX22SBOM	0,77 seconds	0	0
Vcpkg (Beta)	0,77 seconds	0	0
Yarn	0,77 seconds	974	124
Total	1,3 seconds	1117	174

Şekil 6.18 Radarr bağımlılık listesi

Radarr projesinde bulunan bağımlılıkların zafiyet taraması Bomber aracı ile gerçekleştirilmiştir. İçerdiği zafiyetlere ait bulgular Şekil 6.19'da gösterilmiştir.

TYPE	NAME	VERSION	SEVERITY	VULNERABILITY	EPSS %
npm	traverse	7.22.11	CRITICAL	CVE-2023-45133	24%
	postcss	6.0.23	MODERATE	CVE-2021-23382,SNYK-JAVA-ORGWEBJARSNPM-1255641,SNYK-JS-POSTCSS-1255640	N/A
		6.0.23	MODERATE	CVE-2023-44270	18%
		8.4.29	MODERATE	CVE-2023-44270	18%
		8.4.23	MODERATE	CVE-2023-44270	18%
	css-tools	4.3.1	MODERATE	CVE-2023-48631	14%
	color-string	0.3.0	MODERATE	CVE-2021-29060	53%
nuget	System.Security.Cryptography.Pkcs	6.0.0	HIGH	BIT-dotnet-2023-29331,BIT-dotnet-sdk-2023-29331,CVE-2023-29331	N/A

Total vulnerabilities found: 8

RATING	COUNT
CRITICAL	1
HIGH	1
MODERATE	6
LOW	0

Şekil 6.19 Radarr için bulunan zafiyetler

Proje bağımlılıklarına ait 8 farklı zayifet bulunmuştur. Bunlardan 1'i kritik, 1'i yüksek, 6 tanesi moderate seviyesindedir. CVE-2021-29060 zafiyeti moderate olmasına rağmen %53 EPSS'ye sahiptir. Color-String sürüm 1.5.5 ve altında, uygulama sağlandığında ve hazırlanmış geçersiz bir HWB dizesini kontrol ettiğinde ortaya çıkan bir Düzenli İfade Hizmet Reddi (ReDOS) güvenlik açığını ifade etmektedir. CVE-2023-45133 zafiyeti, kritik seviyede bir zafiyettir ve %24 EPSS değerine sahiptir. Zafiyet, JavaScript yazmak için kullanılan bir derleyici olan Babel'in 7.23.2 ve 8.0.0-alpha.4 sürümlerinden önceki '@babel/traverse' ve 'babel-traverse' sürümlerinin tümünde, 'path.evaluate()' veya 'path.evaluateTruthy()' dahili Babel yöntemlerine dayanan eklentiler kullanıldığında, bir saldırgan tarafından özel olarak hazırlanmış kodu derleme sürecinde rastgele kod yürütülmesine yol açabilir.

6.1.10 Ocelot

Ocelot, .NET için geliştirilmiş API Ağ geçididir. Ocelot projesinin bağımlılıkları SBOM Tool ile çıkarılmıştır ve bu bağımlılıklar Şekil 6.20'de gösterilmiştir. Projede, 418 adet bağımlılık ve 98 adet açıklık referansı bulunmuştur.

Component Detector Id	Detection Time	# Components Found	# Explicitly Referenced
CocoaPods	0,1 seconds	0	0
Go	0,099 seconds	0	0
Gradle	0,099 seconds	0	0
Ivy (Beta)	0,057 seconds	0	0
Linux	0,0033 seconds	0	0
MvnCli	0,056 seconds	0	0
Npm	0,098 seconds	0	0
NpmLockfile3 (Beta)	0,098 seconds	0	0
NpmWithRoots	0,098 seconds	0	0
NuGet	0,1 seconds	9	0
NuGetPackagesConfig	0,098 seconds	0	0
NuGetProjectCentric	0,6 seconds	384	96
Pip	7,9 seconds	25	2
Pnpm	0,098 seconds	0	0
Poetry (Beta)	0,098 seconds	0	0
Ruby	0,098 seconds	0	0
RustCrateDetector	0,098 seconds	0	0
SPDX22SBOM	0,098 seconds	0	0
Vcpkg (Beta)	0,098 seconds	0	0
Yarn	0,098 seconds	0	0
Total	7,9 seconds	418	98

Şekil 6.20 Ocelot bağımlılık listesi

Ocelot projesinde bulunan bağımlılıkların zafiyet taraması Bomber aracı ile gerçekleştirilmiştir. İçerdiği zafiyetlere ait bulgular Şekil 6.21'de gösterilmiştir.

TYPE	NAME	VERSION	SEVERITY	VULNERABILITY	EPSS %
nuget	System.Text.Encodings.Web	4.3.0	MODERATE	CVE-2017-0248	59%
		4.3.0	MODERATE	CVE-2017-0256	43%
		4.3.0	HIGH	CVE-2017-0247	69%
		4.3.0	HIGH	CVE-2017-0249	50%
		4.3.0	CRITICAL	BIT-dotnet-2021-26701,BIT-dotnet-sdk-2021-26701,CVE-2021-26701	N/A
		4.4.0	CRITICAL	BIT-dotnet-2021-26701,BIT-dotnet-sdk-2021-26701,CVE-2021-26701	N/A
		4.5.0	CRITICAL	BIT-dotnet-2021-26701,BIT-dotnet-sdk-2021-26701,CVE-2021-26701	N/A
	Newtonsoft.Json	9.0.1	HIGH	CWE-755	N/A
	Microsoft.AspNetCore.Authentication.JwtBearer	3.0.0	MODERATE	BIT-aspnet-core-2021-34532,CVE-2021-34532	N/A
	MessagePack	1.7.3.4	MODERATE	CVE-2020-5234	82%

Total vulnerabilities found: 10

RATING	COUNT
CRITICAL	3
HIGH	3
MODERATE	4
LOW	0

Şekil 6.21 Ocelot için bulunan zafiyetler

Bomber taraması sonucunda toplamda 10 farklı zafiyet bulunmuştur. CVE-2021-26701 zafiyeti, kritik öneme sahiptir. .NET Core uzaktan kod yürütme güvenlik açığını ifade etmektedir. CVE-2020-5234 zafiyetinin EPSS değeri %82'dir. C# ve Unity için MessagePack'in 1.9.11 ve 2.1.90 sürümlerinden önceki sürümlerinde,

güvenilmeyen verilerin hash çarpışmaları ve yığın taşması nedeniyle DoS saldırısına yol açabileceği bir güvenlik açığını tanımlamaktadır.

7. SONUÇLAR VE TARTIŞMA

İncelenen projelerin bağımlılıklarında bulunan zafiyetlerin listesi Çizelge 7.1'de sunulmuştur.

Çizelge 7.1 İncelenen projelerin zafiyet listesi

No	Proje Adı	SBOM Zafiyet Sayısı
1	OpenRA	1
2	Runtime	2
3	Efcore	2
4	Abp	190
5	Aspnetboilerplate	13
6	Captura	3
7	Spectre.console	2
8	RestSharp	2
9	Radarr	8
10	Ocelot	10

Deneysel çalışma sonucunda ortaya çıkan bulgular da göstermektedir ki bir yazılım ürününün kendisinin zafiyet içermemesi o yazılım ürününün güvenli olduğunu göstermemektedir. Ürünün bağımlılıklarında yer alan zafiyetler saldırganlar için o ürünü istismar etmek için yeterli olacaktır. Yazılım bileşenlerinden kaynaklı zafiyetlerin yol açtığı tahribatı göstermesi açısından 2021 yılının son günlerinde ortaya çıkan log4j zafiyeti incelenebilir (Ferreira vd., 2023).

SBOM Tool tarafından bulunan güvenlik açığı sayılarına bakıldığında bazı paketlerin diğerlerinden çok daha savunmasız olduğu görülmektedir. En savunmasız paketlerin yazılım geliştirmede çok yaygın olduğu düşünüldüğünde, bu bulgular üçüncü taraf yazılımları değerlendirirken yazılım sağlayıcılara yardımcı olacaktır. Bu araştırma SBOM'ların yazılım tedarik zinciri güvenliğinin değerlendirilmesindeki potansiyelini ortaya koymaktadır. Bununla birlikte, yazılım sağlayıcıların SBOM'lardan yararlanarak elde edilen sonuçlara tamamen güvenebilmeleri için SBOM alanında hala önemli çalışmalar yapılması gerekmektedir.

Çalışma kapsamında incelenen açık kaynak kodlu 10 C# projesinin öncelikle kaynak kodları üzerinden SBOM'ları üretilmiştir. Üretilen SBOM'lar pasif zafiyet analiz aracı Bomber ile analiz edilerek yazılım bileşenlerinin içerdiği zafiyetler taranmıştır. Bir yazılım paketi olarak tek başına incelendiğinde herhangi bir zafiyet içermiyor gibi görünen yazılımların bileşenlerinde zafiyetlerin olabileceği gösterilmiştir. Bu

bulgu bize, yazılım geliştirme/dağıtım ortamlarında (CI/CD) yazılım malzeme listesinin (SBOM) otomatik olarak üretilmesinin ve bu malzeme listesi üzerinden zafiyet analizinin yapılmasının gerekliliğini göstermiştir. Bu gereklilik CI/CD süreçleri için yeni bir model önerilmesi gerçeğini ortaya çıkarmıştır. Önerilen bu model Şekil 5.1'de gösterilmiştir. Bu model, bir geliştiricinin pull request isteği ortaya çıktığında CI/CD sürecinin bir parçası olarak iki yeni adımın sürece dahil edilmesini ele almaktadır. Bu adımlar, sbom-tool ile kaynak kod üzerinden otomatik SBOM üretilmesini ve bu SBOM listesi üzerinden Bomber ile zafiyet taramasını içermektedir. Bu adımların gerekliliği örnek 10 proje üzerinden doğrulanmıştır. Bu gerekçeler bağlamında, kaynak koddan otomatik SBOM çıkarımının ve SBOM üzerinden zafiyet taraması yapılmasının Jenkins, GitHub Action vb. CI/CD pipeline'larına eklemlenmesi gerektiğini düşünmekteyiz.

Sonuç olarak, bu çalışma kapsamında ortaya atılan üç araştırma sorusuna yanıt aranmıştır. SBOM'ların kaynak koddan otomatik olarak çıkarılabildiği ve otomatik olarak çıkarılan SBOM listeleri aracılığıyla güvenlik açıklarının bulunabildiği görülmüştür. Son soruya cevap olarak, güvenlik açıkları açısından ürün ve bağımlılık arasında bir korelasyon olmadığı gözlemlenmiştir.

Gelecek çalışmalarda yazılım kalite modelleme tekniklerini ve bunların yazılım tedarik zinciri güvenlik risklerini karakterize etmek için nasıl kullanılabileceğini araştıran araştırmalar son derece ilgi çekici olacaktır. SBOM statik analiz araçlarından elde edilen bulguları kullanan kalite modellerinin geliştirilmesi, CI/CD ortamlarına entegrasyona olanak tanıyacak potansiyel bir araştırma alanı olacaktır. Yazılım kalite modelleme yaklaşımları, yazılım sağlayıcılarının SBOM yapısının (yani standartlara uygunluk) yanı sıra SBOM içeriğinin değerlendirilmesini kullanarak yazılım tedarik zincirlerinin kalitesini takip etmelerine ve izlemelerine olanak tanıyacaktır. Yazılım kalite (QA) uygulayıcıları ve kullanıcılar tedarik zinciri güvenlik kalite hedeflerini belirleyerek ve sürdürerek fayda sağlayabilirler.

Gelecek çalışmalarda önerilen modelin uçtan-uca uygulandığı Jenkins entegrasyonunun gerçekleştirilmesi planlanmaktadır. Aktif zafiyet analiz araçlarına göre performans, verimlilik ve canlı ortamlarda sürekli tarama imkânı sunan pasif zafiyet analiz araçlarının giderek popüler olması nedeniyle pasif zafiyet analiz

araçları üzerinden SBOM üretimi ve zafiyet taramasına yönelik çalışmalar önemli bir araştırma alanı olarak göze çarpmaktadır. Günümüzde yazılımın artan karmaşıklığı ve 3. parti entegrasyonların artması nedeniyle güvenliğin maliyeti de artacaktır. Kaynak kodda bağımlılıklar arttıkça güvenlik açıkları da aynı oranda artacaktır. Bu nedenle bağımlılıklardaki güvenlik açıklarını tespit etmek için yeni yapılar oluşturulacaktır. Bu da CI/CD süreçlerinde kendine yer bulacak otomatize bir yapı olacaktır.

KAYNAKÇA

Adewumi, A., Misra, S., & Omoregbe, N. (2015, October). Evaluating open source software quality models against ISO 25010. In 2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing (pp. 872-877). IEEE.

Arora, A., & Garman, C. (2023). Analysis of software bill of materials tools. Cyber Security: A Peer-Reviewed Journal, 6(4), 334-355.

Arora, A., Wrght, V., & Garman, C. (2022). Strengthening the security of operational technology: Understanding contemporary bill of materials. *Journal of Critical Infrastructure Policy*, *3*(1), 111-135.

Axelsson, V., & Larsson, F. (2023). Understanding the Software Bill Of Material for supply-chain management in Open Source projects.

Bauer, A., Harutyunyan, N., Riehle, D., Schwarz, GD. (2020). Challenges of tracking and documenting open-source dependencies in products: A case study. In: Ivanov, V., Kruglov, A., Masyagin, S., Sillitti, A., Succi, G. (eds) Open-source systems. IFIP advances in information and communication technology, vol 582. Springer, Cham.

Blischak, J. D., Davenport, E. R., & Wilson, G. (2016). A quick introduction to version control with Git and GitHub. PLoS computational biology, 12(1), e1004668.

Butler, S., Gamalielsson, J., Lundell, B., Brax, C., Mattsson, A., Gustavsson, T., ... & Lönroth, E. (2022). Considerations and challenges for the adoption of open source components in software-intensive businesses. Journal of Systems and Software, 186, 111152.

Chaora, A., Ensmenger, N., & Camp, L. J. (2023, September). Discourse, Challenges, and Prospects Around the Adoption and Dissemination of Software Bills of Materials (SBOMs). In *2023 IEEE International Symposium on Technology and Society (ISTAS)* (pp. 1-4). IEEE.

Camp, L. J., & Andalibi, V. (2021). SBOM vulnerability assessment & corresponding requirements. NTIA Response to Notice and Request for Comments on Software Bill of Materials Elements and Considerations.

Crnkovic, I., Sentilles, S., Vulgarakis, A., & Chaudron, M. R. (2010). A

classification framework for software component models. IEEE Transactions on Software Engineering, 37(5), 593-615.

Ding, X., Zhao, F., Yan, L., & Shao, X. (2019, October). The Method of Building SBOM Based on Enterprise Big Data. In 2019 3rd International Conference on Electronic Information Technology and Computer Engineering (EITCE) (pp. 1224-1228). IEEE.

Estdale, J., & Georgiadou, E. (2018). Applying the ISO/IEC 25010 quality models to software product. In Systems, Software and Services Process Improvement: 25th European Conference, EuroSPI 2018, Bilbao, Spain, September 5-7, 2018, Proceedings 25 (pp. 492-503). Springer International Publishing.

Ferreira, P., Caldeira, F., Martins, P., & Abbasi, M. (2023, February). Log4j Vulnerability. In International Conference on Information Technology & Systems (pp. 375-385). Cham: Springer International Publishing.

França, J. M., & Soares, M. S. (2015, April). SOAQM: Quality Model for SOA Applications based on ISO 25010. In *ICEIS (2)* (pp. 60-70).

Gandhi, R., Germonprez, M., & Link, G. J. (2018, January). Open data standards for open source software risk management routines: An examination of SPDX. In *Proceedings of the 2018 ACM International Conference on Supporting Group Work* (pp. 219-229).

Germonprez, M., Kendall, J. E., Kendall, K. E., & Young, B. (2014). Collectivism, creativity, competition, and control in open source software development: reflections on the emergent governance of the SPDX® working group. *International Journal of Information Systems and Management*, *1*(1-2), 125-145.

Gschrei, A. (2022) Defining a framework for automated Software-BOM generation from package metadata in a CI/CD environment.

Hankin, C., & Malacaria, P. (2022). Attack dynamics: an automatic attack graph generation framework based on system topology, CAPEC, CWE, and CVE databases. Computers & Security, 123, 102938.

Harer, J. A., Kim, L. Y., Russell, R. L., Ozdemir, O., Kosta, L. R., Rangamani, A., ... & Lazovich, T. (2018). Automated software vulnerability detection with machine learning. *arXiv preprint arXiv:1803.04497*.

Jacobs, J., Romanosky, S., Edwards, B., Adjerid, I., & Roytman, M. (2021).

Exploit prediction scoring system (epss). Digital Threats: Research and Practice, 2(3), 1-17.

Kekül, H., Ergen, B., & Arslan, H. (2021). A multiclass hybrid approach to estimating software vulnerability vectors and severity score. Journal of Information Security and Applications, 63, 103028.

Kemppainen, P. (2023). Managing 3rd Party Software Components with Software Bill of Materials.

Kloeg, B., Ding, A. Y., Pellegrom, S., & Zhauniarovich, Y. (2024). Charting the Path to SBOM Adoption: A Business Stakeholder-Centric Approach.

Laan, S. (2017). IT infrastructure architecture: Infrastructure building blocks and concepts third edition.

Martin, B., Brown, M., Paller, A., Kirby, D., & Christey, S. (2011). 2011 CWE/SANS top 25 most dangerous software errors. *Common Weakness Enumeration, 7515*.

Mell, P., Scarfone, K., & Romanosky, S. (2006). Common vulnerability scoring system. *IEEE Security & Privacy*, *4*(6), 85-89.

Muirí, É. Ó. (2019). Framing software component transparency: Establishing a common software bill of material (SBOM). NTIA, Nov, 12.

Neuhaus, S., & Zimmermann, T. (2010, November). Security trend analysis with cve topic models. In *2010 IEEE 21st International Symposium on Software Reliability Engineering* (pp. 111-120). IEEE.

Nguyen, P., Durlauf, S., & Tikalsky, M. (2023). Software Bill of Materials: A Catalyst to a More Secure Software Supply Chain (Doctoral dissertation, Acquisition Research Program).

Nocera, S., Romano, S., Di Penta, M., Francese, R., & Scanniello, G. (2023, October). Software Bill of Materials Adoption: A Mining Study from GitHub. In *2023 IEEE International Conference on Software Maintenance and Evolution (ICSME)* (pp. 39-49). IEEE.

NTIA Multistakeholder Process on Software Component Transparency Framing Working Group. (2019). Framing Software Component Transparency: Establishing a Common Software Bill of Material (SBOM). United States Department of Commerce.

Paik, I., & Park, W. (2005). Software component architecture for an information

infrastructure to support innovative product design in a supply chain. Journal of organizational computing and electronic commerce, 15(2), 105-136.

Peters, E., & Aggrey, G. K. (2020). An ISO 25010 based quality model for ERP systems. Adv. Sci. Technol. Eng. Syst. J, 5(2), 578-583.

Pratama, A. A., & Mutiara, A. B. (2021). Software quality analysis for halodoc application using iso 25010: 2011. Int. J. Adv. Comput. Sci. Appl, 12(8), 383-392.

Safianu, O., Twum, F., & B, J. (2016). Information System Security Threats and Vulnerabilities: Evaluating the human factor in data protection. International Journal of Computer Applications, 143(5), 8–14.

Sehgal, V. V., & Ambili, P. S. (2023, April). A Taxonomy and Survey of Software Bill of Materials (SBOM) Generation Approaches. In Analytics Global Conference (pp. 40-51). Cham: Springer Nature Switzerland.

Sibal, R., Sharma, R., & Sabharwal, S. (2017). Prioritizing software vulnerability types using multi-criteria decision-making techniques. *Life Cycle Reliability and Safety Engineering*, *6*, 57-67.

Smart, J. F. (2011). Jenkins: The Definitive Guide: Continuous Integration for the Masses. *O'Reilly Media, Inc.*

Stalnaker, T., Wintersgill, N., Chaparro, O., Di Penta, M., German, D. M., & Poshyvanyk, D. (2024, February). BOMs Away! Inside the Minds of Stakeholders: A Comprehensive Study of Bills of Materials for Software Systems. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering* (pp. 1-13).

Stoddard, J. T., Cutshaw, M. A., Williams, T., Friedman, A., & Murphy, J. (2023). *Software Bill of Materials (SBOM) Sharing Lifecycle Report* (No. INL/RPT-23-71296-Rev000). Idaho National Lab.(INL), Idaho Falls, ID (United States).

Şimşek Yağlı, B. (2018). ISO 25010 kalite modeli çerçevesinde teknoloji mağazalarının internet sitelerinin çok kriterli analizi: Türkiye örneği (Master's thesis, A MULTI-CRITERIA ANALYSIS OF TECHNOLOGY STORES'WEBSITES WITHIN THE FRAME OF ISO 25010 QUALITY MODEL: THE CASE OF TURKEY).

Wang, Y., Chen, B., Huang, K., Shi, B., Xu, C., Peng, X., ... & Liu, Y. (2020,

September). An empirical study of usages, updates and risks of third-party libraries in java projects. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)* (pp. 35-45). IEEE.

Xia, B., Zhang, D., Liu, Y., Lu, Q., Xing, Z., & Zhu, L. (2023). Trust in Software Supply Chains: Blockchain-Enabled SBOM and the AIBOM Future. arXiv preprint arXiv:2307.02088.

Yan, D., Niu, Y., Liu, K., Liu, Z., Liu, Z., & Bissyandé, T. F. (2021, December). Estimating the attack surface from residual vulnerabilities in open source software supply chain. In *2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS)* (pp. 493-502). IEEE.

Yıldız, E., Bilgen, S., Tokdemir, G., Cagiltay, N. E., & Erturan, Y. N. (2014). Analysis of B2C mobile application characteristics and quality factors based on ISO 25010 quality model. In *Mobile Web Information Systems: 11th International Conference, MobiWIS 2014, Barcelona, Spain, August 27-29, 2014. Proceedings 11* (pp. 261-274). Springer International Publishing.

Zahan, N., Zimmermann, T., Godefroid, P., Murphy, B., Maddila, C., & Williams, L. (2022, May). What are weak links in the npm supply chain?. In *Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice* (pp. 331-340).

Zampetti, F., Geremia, S., Bavota, G., & Di Penta, M. (2021, September). Ci/cd pipelines evolution and restructuring: A qualitative and quantitative study. In *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)* (pp. 471-482). IEEE.

Zolkifli, N. N., Ngah, A., & Deraman, A. (2018). Version control system: A review. Procedia Computer Science, 135, 408-415.

İnternet Kaynakları

Url-1 "Cyclonedx" [Online]. Available: https://cyclonedx.org/ [Accessed: 20-Nov-23]

Url-2 "Anchore Syft" [Online]. Available: https://github.com/anchore/syft [Accessed: 10-May-24]

Url-3 "Software Identification Ecosystem Option Analysis. (2023). In

Cybersecurity and Infrastructure Security Agency. CISA." [Online]. Available: https://www.cisa.gov/resources-tools/resources/software-identification-ecosystem-option-analysis [Accessed: 10-May-24]

Url-4 "Github Ranking, 'Top 100 Stars in C#'" [Online]. Available: https://github.com/EvanLi/Github-Ranking/blob/master/Top100/CSharp.md [Accessed: 20-Nov-23]

Url-5 "SBOM Tool, 'microsoft/sbom-tool'" [Online]. Available: https://github.com/microsoft/sbom-tool [Accessed: 01-Nov-23]

Url-6 "Bomber, 'devops-kung-fu/bomber'" [Online]. Available: https://github.com/devops-kung-fu/bomber/blob/main/doc/providers/osv.md [Accessed: 01-Nov-23]

Url-7 "Github" [Online]. Available: https://github.com/ [Accessed: 10-Dec-23]

Url 8 "GitHub Hakkında Bilmeniz Gereken Her Şey" [Online]. Available: https://l24.im/Jzcq [Accessed: 10-Dec-23]

Url 9 "Component Detection" [Online]. Available: https://github.com/microsoft/component-detection [Accessed: 10-Dec-23]

Url 10 "Zafiyet Analizi" [Online]. Available: *https://l24.im/Iqaj* [Accessed: 10-Dec-23]

Url-11 "CVE Mitre" [Online]. Available: *https://cwe.mitre.org/* [Accessed: 20-Dec-23]

Url-12 "Common Vulnerability Scoring System v3.0: Specification Document" [Online]. Available: *https://www.first.org/cvss/v3.0/specification-document* [Accessed: 20-Dec-23]