T.C.
HALİÇ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI
BİLGİSAYAR MÜHENDİSLİĞİ PROGRAMI

KAYNAK KOD GÜVENLİĞİNDE STATİK ANALİZ ÖLÇÜTLERİ İLE PERFORMANS DEĞERLENDİRMESİ

YÜKSEK LİSANS TEZİ

Hazırlayan
Tolun ARDAHANLI

Danışman
Dr. Öğr. Üyesi Faruk BULUT

İstanbul – 2019

T.C.
HALİÇ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI
BİLGİSAYAR MÜHENDİSLİĞİ PROGRAMI

KAYNAK KOD GÜVENLİĞİNDE STATİK ANALİZ ÖLÇÜTLERİ İLE PERFORMANS DEĞERLENDİRMESİ

YÜKSEK LİSANS TEZİ

Hazırlayan
Tolun ARDAHANLI

Danışman ve Tez Jürisi
Dr. Öğr. Üyesi Faruk BULUT (Danışman)
Dr. Öğr. Üyesi Zeynep TURGUT (Üye)
Dr. Öğr. Üyesi Selçuk SEVGEN (Üye)

İstanbul – 2019

FEN BİLİMLERİ ENSTİTÜSÜ MÜDÜRLÜĞÜNE

Bilgisayar Mühendisliği Anabilim Dalı Yüksek Lisans Programı Öğrencisi Tolun ARDAHANLI tarafından hazırlanan **"Kaynak Kod Güvenliğinde Statik Kod Analiz Ölçütleri İle Performans Değerlendirilmesi"** adlı tez çalışma jürimizce Yüksek Lisans tezi olarak kabul edilmiştir.

Tez Savunma Tarihi: 18.01.2019

Jüri Üyesinin Unvanı, Adı, Soyadı ve Kurumu	İmzası
Jüri Üyesi : Dr. Öğr. Üy. Faruk BULUT : Danışman / İstanbul Rumeli Üniversitesi	
Jüri Üyesi : Dr. Öğr. Üy. Zeynep TURGUT : Asıl Üye / Haliç Üniversitesi	
Jüri Üyesi : Dr. Öğr. Üy. Selçuk SEVGEN : Asıl Üye / İstanbul Üniversitesi	

Bu tez Enstitü Yönetim Kurulu tarafından belirlenen jüri üyeleri tarafından uygun görülmüş ve Enstitü Yönetim Kurulunun kararıyla kabul edilmiştir.

Prof. Dr Temel SAVAŞKAN
Fen Bilimleri Enstitüsü
Vekil Müdür

ÖNSÖZ

Bu çalışma 2016 – 2019 yılları arasında T.C. Haliç Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalı Bilgisayar Mühendisliği Programı'nın bilimsel araştırma ve uygulama çalışmalarına verdiği destek ile hazırlanmıştır.

Bu çalışmamın tamamlanması süresince gayret ve özveriyle çalışmamı takip eden, yönlendiren, gösterdiği sabır ve hoşgörüyle destek olan danışmanım Sayın Dr. Öğr. Üyesi Faruk BULUT'a çok teşekkür ederim.

Akademik çalışmamda desteklerini esirgemeyen, "Vakıf Emeklilik ve Hayat A.Ş.", "Endpoint Bilgi Teknolojileri Güvenliği Ar-Ge A.Ş." ve "Halk Hayat ve Emeklilik A.Ş." personellerine teşekkürlerimi bir borç bilirim.

İstanbul, 2019 Tolun ARDAHANLI

İÇİNDEKİLER

KISALTMALAR

AAIT : Actionable Alert Identification Techniques (Eylemleştirilebilir Uyarı Tanımlama Teknikleri)

Acc : Accuracy (Sistemde doğru olarak yapılan tahminlerin tüm tahminlere oranı)

API : Application Programming Interface (Uygulama Programlama Arabirimi)

CAS : Center for Assured Software (Güvenli Yazılım Merkezi)

CLOC : Count Lines Of Code (Kod Satırlarını Say)

CWE : Common Weakness Enumeration (Ortak Zayıflık Numaralandırması)

Err : Error (Hata) (Sistemde yanlış olarak yapılan tahminlerdir)

F1-Score : Precision (Hassaslık) ve Recall (Anımsama) nın harmonik ortalamasıdır

FN : False Negative (Yanlış Negatif)

FP : False Positive (Yanlış Pozitif)

JULIET : SAMATE ekibinin test verisetine verdiği kod adı

NIST : National Institute of Standards and Technology (Ulusal Standartlar ve Teknoloji Enstitüsü)

NSA : National Security Agency (Ulusal Güvenlik Ajansı)

P : Precision (Hassaslık) (Pozitif olan tahminlerin ne kadarının doğru olduğunu gösterir)

POL : Polyspace aracının kısa adı

R : Recall (Anımsama) (Pozitif olan durumların ne kadarının doğru olduğunu gösterir)

SAMATE : Software Assurance Metrics and Tool Evaluation (Yazılım Güvence Ölçütleri ve Araç Değerlendirme)

SCA : Static Code Analysis (Statik Kod Analizi) veya Fortify aracının kısa adı

SCATPA : Static Code Analysis Tools Performance Analysis (Statik Kod Analiz Araçları Performans Analizi)

SDLC : Software Development Life Cycle (Yazılım Geliştirme Yaşam Döngüsü)

SSDLC : Secure Software Development Life Cycle (Güvenli Yazılım Geliştirme Yaşam Döngüsü)

TN : True Negative (Doğru Negatif)

TP : True Positive (Doğru Pozitif)

XSS : Cross Site Scripting (Siteler Arası Betik)

ŞEKİLLER

Sayfa No.

TABLOLAR

Sayfa No.

GENEL BİLGİLER

Adı ve Soyadı : Tolun ARDAHANLI
Anabilim Dalı : Bilgisayar Mühendisliği
Programı : Bilgisayar Mühendisliği
Tez Danışmanı : Dr. Öğr. Üyesi Faruk BULUT

Tez Türü ve Tarihi : Yüksek Lisans – Ocak 2019

ÖZET

KAYNAK KOD GÜVENLİĞİNDE STATİK ANALİZ ÖLÇÜTLERİ İLE PERFORMANS DEĞERLENDİRMESİ

Bilişim sektöründe aktif olarak çalışan yazılımcıların geliştirdiği yazılım ürünleri üzerinde hatalar bulunmasının yanı sıra güvenlik zafiyetleri de sıklıkla görülmektedir. Geliştirilen yazılımın ne seviyede güvenli olduğu herhangi bir standart çerçevesinde değerlendirilmemektedir. Yazılım kaynak kodlarında olası güvenlik açıklarını keşfetmek için Statik Kod Analiz araçları kullanılmaktadır. Fortify, Polyspace, SonarCloud (open), Checkmarx, AppScan ve AttackFlow gibi birçok ücretli ve ücretsiz test aracı mevcuttur ve bu araçlar yazılımların değerlendirmesini yapmaktadır. Araçlar kaynak kodları analiz ederken yanlış pozitif (False Positive), yanlış negatif (False Negative), doğru pozitif (True Positive) ve doğru negatif (True Negative) gibi değerlendirme sonuçları üretirler. Yanlış pozitif (False Positive), kaynak kodda gerçekte güvenlik sorunu olmayan ve statik analiz araçları tarafından yanlışlıkla var olduğu tespit edilen güvenlik açıklarını tanımlamaktadır. Yanlış negatif (False Negative) ise statik kod analiz araçları ile bulunamayan ve gerçekte mevcut olan güvenlik açığı şeklinde ifade edilmektedir. Güvenlik taraması sırasında, yanlış negatif (False Negative) ve yanlış pozitif (False Positive) lerden dolayı, taramayı yapan, taramayı yaptıran, ilgili değerlendirme aracını satın almak üzere olan ve aynı zamanda güvenlik taramasından geçirilmemiş programı canlı sistemlerinde barındırarak hizmet vermek zorunda olan aktörler mağduriyet yaşamaktadır.

Bu çalışmanın temel amacı, farklı statik kod analiz araçlarının, güvenlik açıklarını tespit etme performanslarını analiz etmektir. Aynı zamanda doğruluğundan emin olunulmuş ve tekrarlanabilir bir yöntemi ortaya koyarak objektif değerlendirme

sonuçları ile doğabilecek sorunların önüne geçmektir. Bu tez çalışması içerisinde yazılım sektöründe kullanılan Fortify, Polyspace, SonarCloud (open), Checkmarx, AppScan ve AttackFlow gibi test araçlarından sadece Polyspace ve Fortify incelenmiştir. Çalışma içerisinde kâr amacı barındıran ticari ürün Fortify ile akademik çalışmalarda kullanılan Polyspace ürünleri incelenerek performansları kıyaslanmıştır.

Bu çalışmada ilk olarak seçilen JULIET test veri kümesi üzerinde yazılım test araçlarının başarımları incelenmiştir. Daha sonra ilgili test araçları farklı istatistiksel yöntemler kullanılarak analiz edilmiş ve kıyaslanmıştır. Aşamalara ait sürelerin kısaltılabilmesi için filtreleme ve otomatize etmek amacıyla çeşitli betikler oluşturulmuştur. Ayrıca başarımları hesaplayabilmek adına SCATPA adı verilen bir proje de geliştirilmiştir. Gerçekleştirilen çalışmada aynı zayıflık grubunu (Common Weakness Enumeration, CWE) test etmeyi amaçlayan yazılım araçları birbirleri ile karşılaştırılmış ve ilgili araçların farklılıkları (avantaj – dezavantajları) ortaya konmuştur.

Bu tez çalışması alanında bir ilk olup, ilgili alanda çalışmak isteyen şahıs ve kuruluşlar için yol gösterici olması hedeflenmiştir.

Anahtar Kelimeler: CWE, JULIET, MITRE, NIST, SAMATE

GENERAL KNOWLEDGE

Name and Surname : Tolun ARDAHANLI
Field : Computer Engineering
Department : Computer Engineering
Supervisor : Asist. Prof. Dr. Faruk BULUT
Degree and Date : MSc – December 2018

ABSTRACT

PERFORMANCE EVALUATION WITH STATIC ANALYSIS METHODS IN SOURCE CODE SECURITY

Software vulnerabilities developed by software developers working actively in the IT sector, as well as security vulnerabilities are frequently seen. The level of security of the developed software is not evaluated in any standard framework. Static Code Analysis tools are used to discover possible vulnerabilities in software source codes. There are many paid and free testing tools such as Fortify, Polyspace, SonarCloud (open), Checkmarx, AppScan and AttackFlow, and these tools evaluate the software. While analyzing source codes, these tools produce evaluation results such as false positive, false negative, true positive and true negative. A false positive identifies vulnerabilities in the source code that actually do not have security problems and are found to be inadvertently present by static analysis tools. A false negative is expressed as a security vulnerability that cannot be found with static code analysis tools. During the security scan, who is scanning, who is requesting to make this scan, companies that are going to buy a scanning tool and companies that have to serve by hosting a security-free program in their live systems can be a victim, because of false negative and false positives.

The main objective of this study is to analyze the performance of different static code analysis tools and their detection vulnerabilities. At the same time to ensure that the accuracy of a reproducible method to reveal the results of the objective assessment can be avoided. Only Polyspace and Fortify were included in the test tools available on the market, such as Fortify, Polyspace, SonarCloud (open), Checkmarx, AppScan and AttackFlow. In the study, the product of the profit, Fortify, and

Polyspace products which is used in academic studies were examined and their performances were compared.

In this study, the performance of software testing tools on the JULIET test data set was investigated. Then, the related test tools were analyzed and compared using different statistical methods. In order to shorten the time of the stages, various scripts have been created for filtering and automating. In addition, a project called SCATPA was developed in order to calculate the performance. In the study, the software tools aimed at testing the same weakness group (Common Weakness Enumeration, CWE) were compared with each other and the differences of the related tools; advantages - disadvantages have been demonstrated.

As the thesis study is a first in its field, it is aimed to be a guide to the individuals and organizations who want to work in this field.

Keywords: CWE, JULIET, MITRE, NIST, SAMATE

1. GİRİŞ

Günümüzde, bilgisayar yazılımları kullanımı esnasında son kullanıcılar casus yazılımlarda kimlik hırsızlığından, güvenli bilginin kimliğine bürünen ara yazılımlara ve elektronik ticarette yapılan usulsüzlüklere kadar birçok güvenlik sorunu ile karşılaşmaktadır. Pek çok uygulama açık kaynak kodlu yazılımlar ve üçüncü partilerden temin edilen farklı seviyelerdeki yazılımlara fazlaca bağımlıdır. İşletim sistemlerine ve uygulamalara ait yama vb. eklentilerin internet üzerinden edinilmesi güvenlik zafiyetlerinin doğma olasılığını arttırmaktadır.

Bilgi güvenliği çok yönlü ortak bir sorundur ve kullanıcılar kullandıkları yazılımların güvenli olduğunu özellikle bilmek istemektedir.

Yazılım geliştiriciler, geliştirdikleri yazılımın yeterince güvenilir olduğundan emin olmalıdırlar. Yazılım yaşam döngüsü içerisinde kullanılan statik analiz araçları, yazılımcılara güvenlik seviyeleri üzerine bilgi vermekte kullanılan aracılardır. Bu bilgiler geliştirilen yazılımın "kullanım amacına" uygun olarak çalışır, yeterince güvenli olup olmadığını sınamaktadır. Statik kaynak kod güvenlik tarayıcıları (araçları) yazılım geliştirme yaşam döngülerine (Software Development Life Cycle - SDLC) dâhil edilerek otomatik şekilde kullanılabilmektedir. Yazılım geliştirme yaşam döngüleri içerisine statik güvenlik analiz araçlarının da dâhil edilmesi ile güvenli yazılım geliştirme yaşam döngüsü (Secure Software Development Life Cycle-SSDLC) adını almaktadır. İlgili yöntemler kaynak kodu çalıştırılmadan ve/veya canlı sistemlere yüklenmeden yazılım üzerinde yer alan güvenlik açıklarını tespit etmeye imkân vermektedir.

Günümüzde otomatize etmek amacıyla, statik kaynak kod güvenlik araçları, yazılım geliştirme yaşam döngülerine (SDLC) Şekil 1.1 de görüldüğü üzere kolaylıkla dâhil edilebilmektedir. Güvenlik seviyesi incelenmeden bir yazılım ürünü son kullanıcının kullanımına açılırsa, ilgili kullanıcının güvenlik açıkları kaynaklı itibar, ekonomik vb. kayıplar yaşaması olasıdır [1].

2013 tarihli G. Diaz ve J.R. Bermejo ya ait "Static analysis of source code security assesment of tools against SAMATE" başlıklı çalışma ile 2015 tarihli K. Goseva-Popstojanova ve A. Perhinschi ye ait "On the capability of static code analysis to detect security vulnerabilities" çalışmalarının her ikisinde de kaynak kodları üzerinde performans değerlendirmesi yer almaktadır. G.Diaz ve J.R.Bermejo ya ait çalışma çok eski olduğu için geçerliliğini kaybetmiş ve SAMATE tarafından kullanımdan kaldırılmış (deprecated) 45 ve 46 veri kümelerini kullanmaktadır [1-3].

G. Diaz ve J.R. Bermejo ya ait çalışmanın amacı, statik analiz araçlarının performansını tespit ederek, güvenlik açıklarını analiz eden iyi tanımlanmış ve tekrarlanabilir bir metodoloji ile objektif bir değerlendirme sağlamaktır. Çalışma içerisinde farklı tasarıma sahip olan dokuz aracın (CBMC, K8-Insight, PC-lint, Prevent, Satabs, SCA, Goanna, Cx-enterprise, Codesonar) performansı karşılaştırılmaktadır. Statik analiz araçları kullanılarak SAMATE'nin C dili için tasarlanmış olan referans veri setleri 45 ve 46 üzerinde test edilmiştir. Elde edilen sonuçlar performans metrikleri ile analiz edilmiştir [1].

K. Goseva-Popstojanova ve A. Perhinschi ye ait çalışmada statik kod analiz araçlarının güvenlik açıklarını tespit etme yeteneği ampirik (deneysel) olarak sınanmıştır. Statik kod analizi için yaygın olarak kullanılan üç ticari aracı değerlendirilmiş ve kıyaslama test paketi JULIET kullanılarak bir deney gerçekleştirilmiştir. Analiz ve değerlendirme yapmak için deney yapılması ve sonuçların istatistiksel teste dâhil edilmesi bu çalışmanın literatürde yer alan diğer çalışmalardan farklı yönleridir. Kontrollü denemeye ek olarak ilgili çalışma deneysel değerlendirme ve üç açık kaynak kodlu programa dayalı örnek olay incelemelerini içermektedir [2].

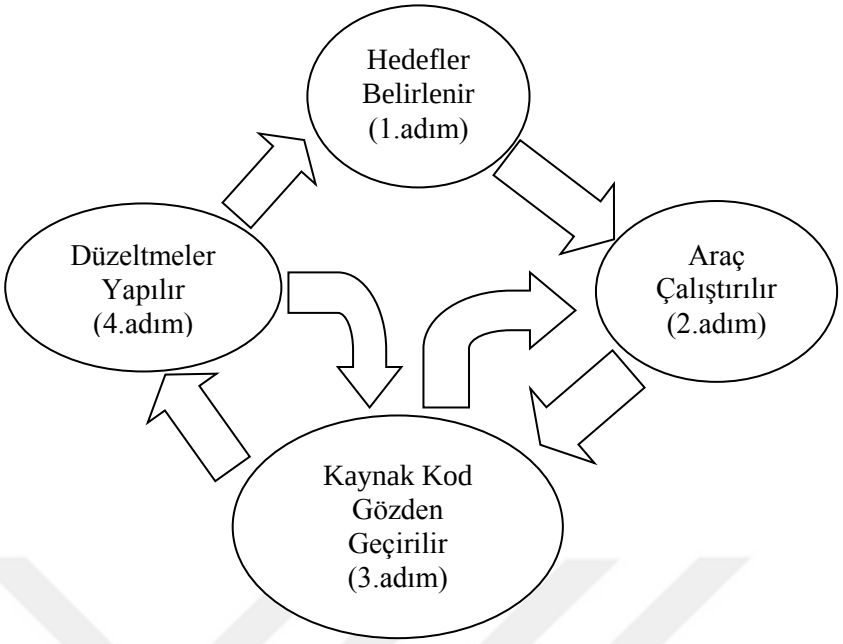
G. Diaz ve J.R. Bermejo ya ait çalışmada kullanılan "performans değerlendirmesi" ve K. Goseva-Popstojanova ve A. Perhinschi ye ait çalışmada kıyaslama amacı ile kullanılan "JULIET veri setinin" güncel hali bu çalışmanın temelini oluşturmaktadır.

S. Heckman ve L. Williams'a ait çalışmanın amacı, eylemleştirilebilir uyarı tanımlama tekniklerinin (AAIT) kanıt temelli seçimini bilgilendirmek için mevcut araştırma sonuçlarını sentezlemektir. Teknikler alarm tipi seçimi kullanabilmek için, bağlamsal bilgi, veri füzyonu, grafik teorisi, makine öğrenme, matematiksel ve istatistiksel modelleri kullanmaktadır. Eylemleştirilebilir uyarıları sınıflandırmak ve öncelik sırasına koymak için dinamik algılama, çeşitli değerlendirme ölçütleri ile bir örnekle değerlendirilmektedir [4].

L.K. Shar ve H.B. Kuan Tan a ait çalışmanın amacı, SQLI veya XSS'ye açık olması muhtemel olan yazılım bileşenleri yerine, belirli program bildirimlerini tahmin etmek için kullanılabilecek statik kod öznitelikleri önererek, var olan analizörlere alternatif veya tamamlayıcı bir çözüm sunulmasıdır. SQLI ve XSS açıklarını önlemek için yaygın olarak web uygulamalarında uygulanan giriş temizleme kodunun gözlemlerinden, kod modellerini karakterize eden bir dizi statik kod özelliği önerilmektedir. SQLI ve XSS açıklarını tahmin etmek için önerilen statik öznitelikleri ve bilinen güvenlik açığı verilerini yansıtan geçmiş bilgilerden güvenlik açığı tahmin modelleri oluşturulmaktadır [5].

M. Mouzarani ve B. Sadeghiyan'a ait çalışmanın amacı, belirli güvenlik açıklarının özelliklerinden bağımsız olan genişletilebilir bir güvenlik açığı algılama yöntemi tasarlamaktır. Bu nedenle ilk olarak yazılım açıklarını belirlemek için genel bir model önerilmektedir. İlgili modele dayanarak, çalıştırılabilir kodlardaki belirli açıkları tespit etmek için genel bir şartname yöntemi ve genişletilebilir bir algoritma sunulmaktadır. Önerilen yöntem Valgrind ikili enstrümantasyon çerçevesi için bir eklenti olarak uygulanmaktadır. Güvenlik açıkları Vex adı verilen Valgrind orta düzey dili kullanılarak belirlenmektedir [6].

Bu çalışmaların hepsinin ortak noktası "statik kod analiz" yöntemi üzerine inşaa edilmiş ve farklı tekniklerin kullanılmış olmasıdır. Her birinde kendi içerisinde özgün yöntemler kullanmıştır.



Şekil 1.1 Yazılım yaşam döngüsünde kod gözden geçirme süreci

Ayrıca statik kaynak kod güvenlik araçlarının, standart araç olarak kullanılmaları henüz ülkemizde tam olarak yaygınlaş(a)mamıştır. Statik analiz araçlarının kendi iç tasarımları, hangi güvenlik açıklarının (CWE) algılanabileceğini belirleyebilmektedir. Ticari olan araçların kaynak kodlarına ve tasarım bilgilerine erişilemediğinden, kamusal bilgiye de ulaşılamamaktadır. Analiz araçları tarafından bulunan güvenlik açıklarında kullanılan "sınıflandırma yöntemi" birbirlerinden farklıdır ve sonuçların karşılaştırılmasını zorlaştırmaktadır. Performanslarını, tespit edebildikleri zaafiyet sayılarını, kullanılabilirliklerini ve tarafsız şekilde hangisinin hangi ortam için en iyi olduğunu belirleyebilmek için gerçek bir ölçüt gerekmektedir.

Bu araçlar, pek çok yazılım geliştirme ekibi için, araç setinin bir parçası olmaya başlasa da, güçlü ve zayıf noktalarının net bir şekilde anlaşılması gereksinimi devam etmektedir. Bu çalışma, bu konunun bazı yönlerini aydınlatmaya katkıda bulunmayı hedeflemiştir.

Bu tez çalışmasında farklı statik analiz tarayıcılarının (araçlarının), C/C++ dillerindeki güvenlik açıklarını tespit etme performanslarını analiz eden, doğruluğundan emin olunulmuş ve tekrarlanabilir bir yöntemi takip ederek, objektif değerlendirme sonuçlarını elde edilmiştir. İyi bilinen güvenlik açıklarını göz önünde bulunduran bir kıyaslama yöntemi kullanmak gerekmektedir. En iyi performans ölçütü, "araç ne kadar fazla doğru pozitif (TP) yakalıyor, ne kadar az yanlış pozitif

yakalıyor (FP) ve ne kadar az yanlış negatifi (FN) tespit ediyorsa" şeklindedir. Karşılaştırma ölçütü ise tekrarlanabilir, taşınabilir, ölçeklenebilir, hedef araçta en az değişiklik gerektiren ve kullanım kolaylığına sahip olmalıdır. Birkaç karşılaştırma ölçütü değerlendirilmiş ve bu kıstaslara uyan NIST organizasyonunun SAMATE ekibine ait JULIET referans veriseti güncel sürümünün kullanılmasına karar verilmiştir. SAMATE güvenlik açığı kategorilerinin çoğunda C/C++, java, J2EE ve PHP gibi programlama dilleri için imkân sağlayan test ölçütü JULIET tir.

Çalışma, tespit edilen güvenlik açıklarının sayısal açıdan etkinliklerini araştırmaktadır ve yaygın olarak kabul görmüş metrik kümesi kullanmaktadır. İlgili değerlendirme, analiz araçların kullanımı ve etkinliklerinin nasıl artıracaklarına içermekte, önerilerde bulunmaktadır.

Şekil 1.2 de Cloc uygulamasına girdiğimiz parametreler ile JULIET test senaryolarında bulunan C/C++ kodlarına ait detaylara ulaşılmaktadır. İlgili detaylar, programlama dillerine göre, dosya adedini, boş satırlar ile açıklama satırlarını ve kod satırlarını içermektedir.

Bu bilgiler genel olarak test senaryolarının ne kadar büyüklükte olduğunu ve araçlar tarafından güvenlik taramalarının yapılması esnasında kod derleme sürelerininin kaba şekilde tahmin edilmesi olanağını sağlamaktadır.

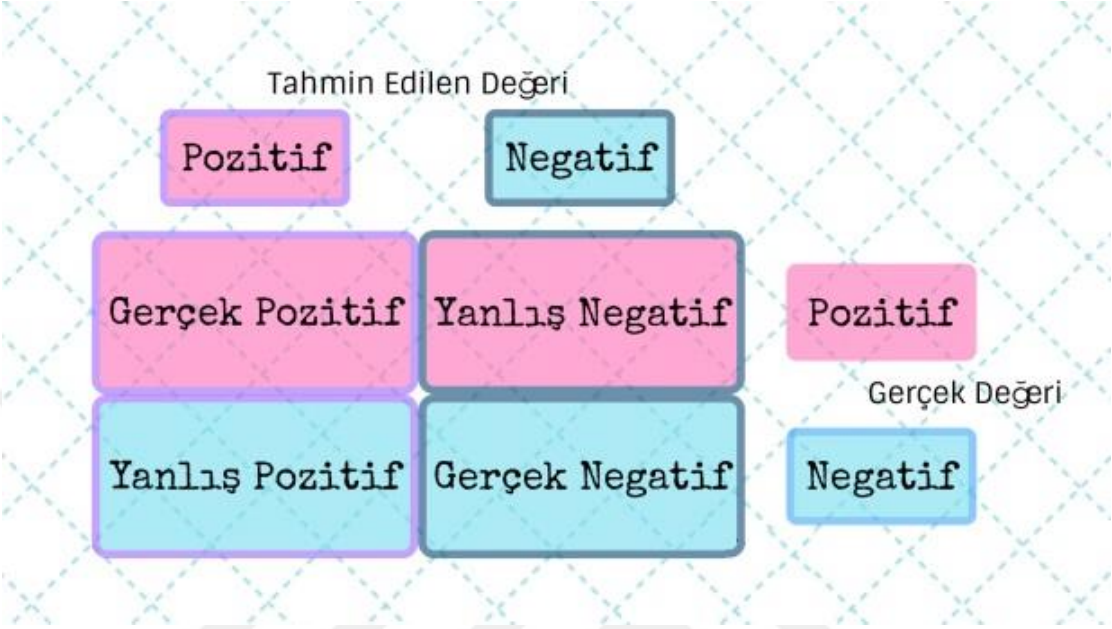
```
D:\Tolun\Downloads>cloc-1.80.exe --include-lang=C,C++,"C/C++ Header" D:\C\testca
ses
  106075 text files.
  105996 unique files.
     647 files ignored.

github.com/AlDanial/cloc v 1.80  T=265.00 s (398.7 files/s, 49596.8 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
C                            54484         773552        1635030        4861502
C++                          46668         818613        1202086        3338481
C/C++ Header                  4496         185622          70516         257748
-------------------------------------------------------------------------------
SUM:                        105648        1777787        2907632        8457731
-------------------------------------------------------------------------------
```

Şekil 1.2 Cloc uygulamasının Juliet test senaryolarını değerlendirmesi

Çalışma içerisinde oluşturulmuş SCATPA adlı araç, statik analiz araçlarından çıkan tarama raporlarını inceleyerek, oluşturulan ölçek ile örtüştürerek ilgili araçların değerlendirmesini yapmaktadır. Dosyalama işlemleri ve toplu tarama işlemlerin otomatize yapılabilmesi ile alakalı betikleri, taramalar sonunda ortaya çıkan işlenebilir tarama raporları ve SCATPA aracı ilgili bağlantıda yer almaktadır [7-8].

Bir karışıklık (confusion) matrisi, gerçek değerlerin bilinmekte olduğu bir dizi test verisi üzerinde, bir sınıflandırma modelinin performansını tanımlamak için kulanılmaktadır. Örnek bir karışıklık matrisi Şekil 1.3'de yer almaktadır [9].



Şekil 1.3 Karışıklık matrisi örneği

JULIET ölçeği ile kullanılan manifest.xml dosyasının örnek bir kesiti Şekil 1.4 de görülmektedir. Manifest.xml dosyası, hangi test senaryosunun, hangi kısmında ne tür bir güvenlik açığı bulunduğuna dair gerçekleştirilen testlerde referans olarak kullanılan bir dosyadır.

```
1: <?xml version="1.0" encoding="utf-8"?>
2: <container>
3:   <testcase>
4:     <file
path="CWE78_OS_Command_Injection__wchar_t_listen_socket_w32_spawnvp_84.h"/>
5:     <file
path="CWE78_OS_Command_Injection__wchar_t_listen_socket_w32_spawnvp_84_bad.cpp"
>
6:       <flaw line="137" name="CWE-078: Improper Neutralization of Special
Elements used in an OS Command ('OS Command Injection')"/>
7:     </file>
8:     <file
path="CWE78_OS_Command_Injection__wchar_t_listen_socket_w32_spawnvp_84_goodG2B.
cpp"/>
9:     <file
path="CWE78_OS_Command_Injection__wchar_t_listen_socket_w32_spawnvp_84a.cpp"/>
10:   </testcase>
11:   ….
12:   <testcase>
13:     <file
path="CWE773_Missing_Reference_to_Active_File_Descriptor_or_Handle__w32CreateFi
le_66a.c"/>
14:     <file
path="CWE773_Missing_Reference_to_Active_File_Descriptor_or_Handle__w32CreateFi
le_66b.c">
15:       <flaw line="28" name="CWE-773: Missing Reference to Active File
Descriptor or Handle"/>
16:     </file>
17:   </testcase>
18:   ….
19: </container>
```

Şekil 1.4 JULIET manifest.xml dosyasının örnek bir kesiti

Bu çalışmanın temelini oluşturan çalışmalardan farklı olarak, güncel ve geçerliliği kabul edilen JULIET sürüm 1.3 ile performans değerlendirmeleri yapılacaktır.

JULIET gibi kaynak kodları gözden geçirilerek, ne tür güvenlik açıklarına sahip ve ne tür güvenlik açıklarına sahip olmadığı kesinleştirilmiş bir veri setini kullanarak, tekrarlanabilir testler gerçekleştirilebilmektedir. İçeriği bilinmeyen ve paylaşılmayan statik analiz araçlarını teste tabii tutmak suretiyle oluşturulan raporların, ne kadar doğru (TP ve TN), ne kadar eksik ve/veya yanlış (FP ve FN) tespitlerde bulunulduğunu kanıtlanmıştır.

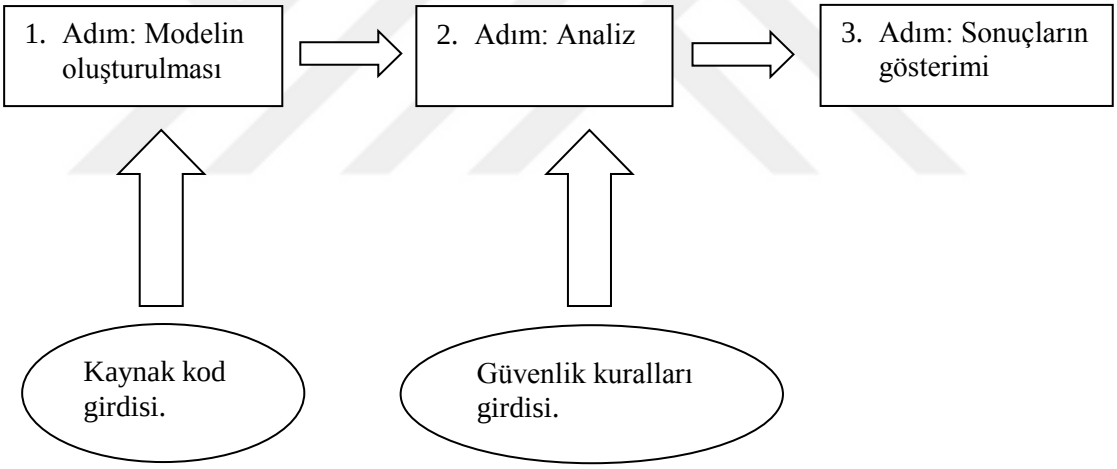
Bu çalışmada, eksik ve/veya yanlış (FP ve FN) tespitlerden dolayı:

- Güvenlik taraması sırasında taramayı yapan,
- Taramayı yaptıran,
- İlgili güvenlik aracını satın almak üzere olan,
- Güvenlik taramasından geçirilmemiş programı canlı sistemleri barındırarak hizmet vermek zorunda olan kullanıcıların mağduriyetlerinin engellenmesi hedeflenmiştir.

2. STATİK KAYNAK KOD GÜVENLİK ANALİZ ARAÇLARI

Bu bölümde mevcut statik kaynak kodu güvenlik araçlarının en önemli özellikleri incelenmiştir. Farklı araç kategorileri tarafından kullanılan çeşitli yaklaşımlar ve bu araçların dezavantajları da bu bölümde yer almaktadır. İlgili araçlar, kaynak kod parçaları üzerinde benzer bir çalışma prensibi sergilemektedir. Çalışma prensipleri;

- Analiz edilmek istenilen kodun bir model haline dönüştürülme gerekliliğini,
- Model farklı kural ve özellikler ile analiz edilme gerekliliğini,
- Sonuçların bir analist tarafından incelenme gerekliliğini içermektedir.



Şekil 2.1 Yazılım analiz süreci

Şekil 2.1 de gösterildiği üzere oluşturulan modelde yer alan kaynak kod; söz dizimsel ve anlam bilimsel analiz, soyut söz dizimi, ayrıştırma vb. çeşitli teknikler kullanılarak anlam ifade eden bir sonuca dönüştürülür. İlgili model fonksiyonlar arasındaki ilişkinin incelenmesi için prosedürel (yerel) analiz, izleme akışı, veri akışı, işaretçi adlandırma vb. yöntemler kullanır.

Seçilen araca bağlı olarak, ortaya çıkabilecek olası sorunlar değişkenlik göstermektedir. Bu nedenle açık kaynak kodlu araçlar üzerinde yer alan sorunlar derinlemesine analiz edilebilirken, açık kaynak kodlu olmayan araçlar için bu seviyede bir analizin gerçekleştirilmesi mümkün değildir.

2.1. SAMATE Projesi

SAMATE açılımı "Software Assurance Metrics and Tool Evaluation" (Yazılım Güvence Ölçütleri ve Araç Değerlendirme), yazılım analiz araçlarının değerlendirmesini mümkün kılacak yöntemler geliştirilmesini amaçlayan bir projedir. Araç ve tekniklerin etkinliğini ölçmek, ilgili araç ve yöntemlerdeki boşlukları belirleyerek, yazılım güvencesini geliştirmeye adanmış bir NIST projesi olarak 2006 yılında yayın yapmaya başlamıştır [10].

SAMATE projesinin kapsamı geniştir: işletim sistemlerinden güvenlik duvarlarına, SCADA'dan web uygulamalarına, kaynak kodu güvenlik analizleri ile "yapım aşaması düzeltme" yöntemlerine kadar pek çok içeriğe sahiptir [11].

Test senaryosu (Test Case), statik analiz aracının belirlenen ortak zayıflık numaralandırmasında (CWE) kullanılabilecek en küçük birim olarak kabul edilmektedir. Buradaki amaç, zayıflığın oluşturulması ve ilgili statik analiz aracının, oluşturulan zayıflığı yakalama performansının sınanmasıdır.

Ulusal Güvenlik Ajansı (NSA)'nın Güvenli Yazılım Merkezi (CAS) tarafından, statik analiz araçlarını test edebilmek isteyenler için JULIET Test Suit oluşturulmuştur [12]. JULIET içinde, **118** adet ortak zayıflık numaralandırması (CWE) olarak **64.099** adet test durumu (test case) yer almaktadır [13].

2.2. CWE (Common Weakness Enumeration)

Ortak Güvenlik Açıkları ve Etiketlendirmelerin Karşılığı olan CWE (diğer adı ile CVE), MITRE Corporation tarafından tutulan, güvenlik açıkları ve saldırıların yer aldığı bir kütüphanedir. 1999 yılında başlatılan veritabanı halka açıktır ve ücretsizdir.

CWE içerisinde herhangi bir güvenlik açığı, saldırgan tarafından ağa sızmak için kullanılabilecek hata olarak tanımlanmaktadır [14].

2.3. CWE Çalışma Prensibi

CWE'nin amacı, kamuya açık olan her güvenlik açığı veya maruz kalma için, kuruluşların kendi uygulamalarında güvenlik konularını yakından takip etmelerini sağlayan standart bir tanımlayıcı sunmaktır. Yazılımları, bilinen güvenlik açıklarından korumak amacıyla veritabanında yer alan bilgilerin kullanılmasına ek olarak, çeşitli kuruluşlar da CWE'ye karşı güvenlik araçlarını test etmek için CWE veritabanını temel olarak kullanabilirler [15].

CWE projesi başlatıldığında, her güvenlik aracı kendi güvenlik açığı ve maruz kalma tanımlayıcılarını kullanmıştır. Bu, güvenlik araçlarını birbirlerine karşı test etmeyi imkânsız hale getirerek kurum ve kuruluşların kendilerine ait bir güvenlik yaklaşımı belirlemesini zorlaştırmıştır. Çünkü kullandıkları farklı araçların kendilerine farklı sonuçlar vermesi olasıdır. Bir test ortamında CWE tanımlayıcılarının kullanılması, geliştiriciler ve güvenlik ürünü grupları arasında sorunsuz veri alışverişi yapılabilmesini sağlar. Aynı zamanda uygulama güvenliği ile ilgili test araçlarını ve hizmetlerini değerlendirmek için kullanılabilecek bir temel de sunar.

CWE, dağıtım ortamlarının genel güvenliğini artırmak için erişilebilecek bir güvenlik açığı yönetim ürünü tasarlamak veya herhangi bir güvenlik açığı hizmetine veritabanı sağlamak için kullanılır. Ayrıca, geliştiricilere, güvenlik ve BT yöneticilerine sunulan geliştirme ve dağıtım platformlarını güncel tutmalarına yardımcı olarak bu platformlardaki güvenlik açıklarından kaynaklanan tehditlere karşı daha iyi korunmalarını sağlar. Bu amaçla yama yönetimi ürünleri ve hizmetlerinin kapsamlı bir listesini sunar.

CWE ayrıca, tespit edilen yeni güvenlik açıkları konusunda sizi uyaran kapsamlı bir veri / olay korelasyon hizmetleri listesi sunar. Ancak bu liste ve önceki listelerle ilgili bazı örtüşmeler vardır. Son olarak, CWE proje web sitesinde izinsiz giriş tespit ürünleri ve hizmetlerini belirlemek için iyi bir kaynak mevcuttur [15].

Otomatik güvenlik testi için birlikte çalışabilir şartnameler sunmak üzere tasarlanan Ulusal Standartlar ve Teknoloji Enstitüsü (NIST) güvenlik içeriği otomasyon protokolü için destek de bulunmaktadır.

CWE tanımlayıcıları, belirli bir güvenlik açığı veya maruz kalma olup olmadığını belirlemek için verilen (iletilen) güvenlik açıklarının kolayca

sınıflandırılmasını sağlamaktadır [14].

MITRE Corporation'ın Ortak Zayıflık Numaralandırma (CWE) temelli kaynak kodu açıkları Tablo 2.1'de yer almaktadır.

Tablo 2.1 Kaynak kod güvenlik açıklarının tam listesi (mitre)

Sıra	CWE Kodu	CWE Açıklaması	C/C++ Test Senaryo Adedi
1	15	External Control of System or Configuration Setting	48
2	23	Relative Path Traversal	2400
3	36	Absolute Path Traversal	2400
4	78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	4800
5	90	Improper Neutralization of Special Elements used in a LDAP Query ('LDAP Injection')	480
6	114	Process Control	576
7	121	Stack-based Buffer Overflow	4968
8	122	Heap-based Buffer Overflow	5922
9	123	Write-what-where Condition	144
10	124	Buffer Underwrite ('Buffer Underflow')	2048
11	126	Buffer Over-read	1452
12	127	Buffer Under-read	2048
13	134	Uncontrolled Format String	2880
14	176	Improper Handling of Unicode Encoding	48
15	188	Reliance on Data/Memory Layout	36
16	190	Integer Overflow or Wraparound	2592
17	191	Integer Underflow (Wrap or Wraparound)	1584
18	194	Unexpected Sign Extension	1152
19	195	Signed to Unsigned Conversion Error	1152
20	196	Unsigned to Signed Conversion Error	18
21	197	Numeric Truncation Error	864
22	222	Truncation of Security-relevant Information	18
23	223	Omission of Security-relevant Information	18
24	226	Sensitive Information Uncleared Before Release	72
25	242	Use of Inherently Dangerous Function	18
26	244	Improper Clearing of Heap Memory Before Release ('Heap Inspection')	72
27	247	Reliance on DNS Lookups in a Security Decision	18
28	252	Unchecked Return Value	630
29	253	Incorrect Check of Function Return Value	684
30	256	Plaintext Storage of a Password	96
31	259	Use of Hard-coded Password	96
32	272	Least Privilege Violation	252
33	273	Improper Check for Dropped Privileges	36
34	284	Improper Access Control	216
35	319	Cleartext Transmission of Sensitive Information	192
36	321	Use of Hard-coded Cryptographic Key	96
37	325	Missing Required Cryptographic Step	72
38	327	Use of a Broken or Risky Cryptographic Algorithm	54

Tablo 2.1 (devam) Kaynak kod güvenlik açıklarının tam listesi (mitre)

39	328	Reversible One-Way Hash	54
40	338	Use of Cryptographically Weak PRNG	18
41	364	Signal Handler Race Condition	18
42	366	Race Condition within a Thread	36
43	367	Time-of-check Time-of-use (TOCTOU) Race Condition	36
44	369	Divide By Zero	864
45	377	Insecure Temporary File	144
46	390	Detection of Error Condition Without Action	90
47	391	Unchecked Error Condition	54
48	396	Declaration of Catch for Generic Exception	54
49	397	Declaration of Throws for Generic Exception	20
50	398	Indicator of Poor Code Quality	181
51	400	Uncontrolled Resource Consumption ('Resource Exhaustion')	720
52	401	Improper Release of Memory Before Removing Last Reference ('Memory Leak')	1658
53	404	Improper Resource Shutdown or Release	384
54	415	Double Free	962
55	416	Use After Free	459
56	426	Untrusted Search Path	192
57	427	Uncontrolled Search Path Element	480
58	440	Expected Behavior Violation	1
59	457	Use of Uninitialized Variable	948
60	459	Incomplete Cleanup	36
61	464	Addition of Data Structure Sentinel	48
62	467	Use of sizeof() on a Pointer Type	54
63	468	Incorrect Pointer Scaling	37
64	469	Use of Pointer Subtraction to Determine Size	36
65	475	Undefined Behavior For Input to API	36
66	476	NULL Pointer Dereference	348
67	478	Missing Default Case in Switch Statement	18
68	479	Signal Handler Use of a Non-reentrant Function	18
69	480	Use of Incorrect Operator	18
70	481	Assigning instead of Comparing	18
71	482	Comparing instead of Assigning	18
72	483	Incorrect Block Delimitation	20
73	484	Omitted Break Statement in Switch	18
74	500	Public Static Field Not Marked Final	1
75	506	Embedded Malicious Code	158
76	510	Trapdoor	70
77	511	Logic/Time Bomb	72

Tablo 2.1 (devam) Kaynak kod güvenlik açıklarının tam listesi (mitre)

78	526	Information Exposure Through Environmental Variables	18
79	534	Information Exposure Through Debug Log Files	36
80	535	Information Exposure Through Shell Error Message	36
81	546	Suspicious Comment	90
82	561	Dead Code	2
83	562	Return of Stack Variable Address	3
84	563	Unused Variable	512
85	570	Expression is Always False	16
86	571	Expression is Always True	16
87	587	Assignment of a Fixed Address to a Pointer	18
88	588	Attempt to Access Child of a Non-structure Pointer	80
89	590	Free of Memory not on the Heap	2680
90	591	Sensitive Data Storage in Improperly Locked Memory	96
91	605	Multiple Binds to Same Port	18
92	606	Unchecked Input for Loop Condition	480
93	615	Information Exposure Through Comments	18
94	617	Reachable Assertion	306
95	620	Unverified Password Change	18
96	665	Improper Initialization	193
97	666	Operation on Resource in Wrong Phase of Lifetime	90
98	667	Improper Locking	18
99	672	Operation on a Resource after Expiration or Release	47
100	674	Uncontrolled Recursion	2
101	675	Duplicate Operations on Resource	192
102	676	Use of Potentially Dangerous Function	18
103	680	Integer Overflow to Buffer Overflow	576
104	681	Incorrect Conversion between Numeric Types	54
105	685	Function Call With Incorrect Number of Arguments	18
106	688	Function Call With Incorrect Variable or Reference as Argument	18
107	690	Unchecked Return Value to NULL Pointer Dereference	960
108	758	Reliance on Undefined, Unspecified, or Implementation-Defined Behavior	581
109	761	Free of Pointer not at Start of Buffer	576
110	762	Mismatched Memory Management Routines	3564
111	773	Missing Reference to Active File Descriptor or Handle	144
112	775	Missing Release of File Descriptor or Handle after Effective Lifetime	144
113	780	Use of RSA Algorithm without OAEP	18
114	785	Use of Path Manipulation Function without Maximum-sized Buffer	18

Tablo 2.1 (devam) Kaynak kod güvenlik açıklarının tam listesi (mitre)

115	789	Uncontrolled Memory Allocation	960
116	832	Unlock of a Resource that is not Locked	18
117	835	Loop with Unreachable Exit Condition ('Infinite Loop')	6
118	843	Access of Resource Using Incompatible Type ('Type Confusion')	80
		TOPLAM TEST ADEDİ:	**61387**

Tablo 2.1 içerisinde her bir açığa atanmış kod numarası (CWE kodu sütunu), ilgili açığa ait açıklama (CWE açıklaması sütunu) ve CWE koduna ait test senaryo adedi (C/C++ test senaryo adedi sütunu) bilgileri yer almaktadır. Tablo 2.1'de listelenen kaynak kodu güvenlik açıklarını bulmak ve belirlemek için, bir kaynak kodu statik analiz aracı, ilgili güvenlik açığıyla örtüşen hatayı bulmalıdır. Hangi güvenlik açığının, hangi dosya ve kaçıncı satırda olduğu genel bir listesi mevcuttur. Bazı kod türleri (CWE), ek analiz yaklaşımları gerektirmektedir. Tablo 2.1 JULIET sürüm 3.0 içeriğinde bulunan 118 ana başlığı ve her bir ana başlığın altında bulunan test adet sayısını göstermektedir.

2.4. Statik Kaynak Kod Güvenlik Analiz Araçlarının Kategorileri

Statik güvenlik analiz araçlarını kategorize etmek için farklı şemalar mevcuttur. Sınıflandırma, anladıkları dile veya aranan zayıflık türlerine göre yapılabilir. Başka bir kıstas, analiz edilecek kodun uzunluğudur. Yani küçük uygulamalar veya büyük projelere göre kullanılabilecek araç değişkenlik gösterebilir. Bu çalışmanın amacı ile paralellik gösteren taksonominin Chess B. ve West J. Tarafından gerçekleştirilmiştir [16]. İlgili taksonomi, araçların genel amacını baz alır ve aşağıdaki başlıklarda farklılıklar göstermektedir [16].

Stil kontrolü: Bu araçlar, tip denetleyicisinden çok daha seçici ve daha yüzeysel kurallar dizisi uygular. İşlev çağrılarındaki tutarsızlıklar, bazı yerlerde dönüş sayıları, değişken sayıları ile çağrılan işlevler ve diğer argümanları geçiren işlev çağrıları gibi problemleri bulmak için sözlü ve söz dizimsel analiz temelli kontroller gerçekleştirirler. Bu analizin, daha sonra bahsedilen diğer türlerle karşılaştırıldığında açıkça sınırlamaları mevcuttur (çalışma zamanında neler olduğunu simüle etmeye dayalı bir analiz yapmamak gibi).

Program anlayışı: Bu araçlar kullanıcıların bir proje kodunu anlamalarına yardımcı olmak için tasarlanmıştır. Birçok *"Tümleşik Geliştirme Ortamı"na* dâhil edilirler ve programcıların bir programın çalışma şekli hakkında fikir sahibi olmalarına yardımcı olurlar. Kodları anlamak ve güvenlik açıklarını keşfetmek için güvenlik analizi gerçekleştirilmesine yardımcı olurlar, ancak her durumda, tüm kodun gözden geçirilmesi gerekmektedir ve zaman isterine sahiptir.

Program doğrulama ve özellik denetimi: Bu araçlar bir şartnameyi ve bir kod grubunu kabul eder ve daha sonra kodun belirtiminin doğru bir şekilde uygulandığını kanıtlamaya çalışır.

Son olarak, bu sınıflandırma en modern ve sofistike araçları ifade eder. Bu statik analiz araçlarına Chess B. ve West J. Tarafından verilen isimleri kullanılmıştır: "hata bulma" araçları ve "güvenlik incelemesi" araçları. Projelerin yüz binlerce veya milyonlarca kod satırının güvenlik analizini yaptıkları için her iki araç türü de tercih edilmiştir [16].

2.5. Hata Tespit Statik Analiz Araçları

"Hata bulma" araçları, programın programcı tarafından arzulanan şekilden farklı biçimde hareket edeceği yerlerde uyarı vererek yönlendirme yapmaktadır. Bu araçların iki önemli özelliği vardır:

- Kolay kullanım: Araçlar, güvenlik açıklarını gösterebilen, koddaki kalıpları tanımlayan, önceden tanımlanmış bir kurallar kümesi içerir. Bu küme çok iyi araçlarda genişletilebilmektedir.
- Genellikle, çok sayıda; yüz binlerce veya milyonlarca kod satırına sahip uygulamaları analiz etmek için tasarlanmıştır.

Çoğu hata (bug) bulma aracı düşük sayıda yanlış pozitif (FP) üretebilmektedir. Şüpheli bir güvenlik açığı tespit edildiğinde, kodların olası olay dizisini gösteren araçların uygulanmasından sonra örneklendirilmektedir.

2.6. Güvenlik İncelemesi Statik Analiz Araçları

Bu araçlar, önceki araçların pek çok tekniğini birleştirir, ancak belirli güvenlik açıklarını belirleme konusunda daha katı hedefleri vardır. İlgli araçların üreticileri tarafından iddia edildiği üzere tasarım aşamasında özellik denetleyicileri ve "hata arama" sınıfı araç teknikleri uygulanır. Çünkü pek çok güvenlik açığı bilgisi, kısaca program özellikleri olarak ifade edilebilir. Bu araçların tasarımcıları, yanlış pozitifler (FP) ile negatifler (FN) arasındaki dengenin uygun olanını tercih ederler. Bu araçlar, kod içerisinde, "hata bulma" araçları ile karşılaştırıldığında daha fazla yanlış pozitif üreten, aracı çalıştırdıktan sonra manuel olarak gözden geçirilmesi gereken, birçok noktayı göstermeyi tercih etmektedir.

2.7. Araçların Değerlendirilebilmesi için Karşılaştırma Ölçütü

Statik analiz araçlarını karşılaştırmanın en umut verici yollarından biri, aynı kodu analiz etmek ve bunları karşılaştırmak için kullanmaktır. Ancak analiz edilecek doğru kodu, herkes tarafından doğru kabul edilebilecek şekilde seçme zorunluluğu mevcuttur. Birçok başka disiplinde olduğu gibi, araçları karşılaştırmak için bir kıyaslama gereklidir. Bir karşılaştırma ölçütü, yanlış pozitifler ve yanlış negatifler arasındaki dengeye ilişkin fikir birliğine varmaya çalışarak, sonuçların karşılaştırılmasına yardımcı olmalıdır. Belki de bu kıyaslama, örneğin OWASP (Açık Web Uygulama Güvenliği Projesi), SANS (SysAdmin, Denetim, Ağ ve Güvenlik) Enstitüsü, CERIAS (Eğitim ve Araştırma Merkezi) gibi, topluluk oyuncuları tarafından referans standart olarak kullanılmasını da motive edebilir (Bilgi Güvenirliği ve Güvenliği ve diğerleri) [17].

Giriş bölümünde de belirtildiği gibi, iyi bir karşılaştırma ölçütü çok sayıda özelliğe sahip olmalıdır:

- Maliyeti sonuçların değeri ile karşılaştırılabilir olmalıdır.
- Güvenilir olmak için, güvenlik açığı algılama araçları için ölçüt, aynı araç üzerinde birden çok kez çalıştırıldığında benzer sonuçlar bildirmelidir.
- Belirli bir alandaki farklı araçların karşılaştırılmasına izin vermesi gerektiği için kolayca taşınabilir olmalıdır. Uygulamada, iş yükü, taşınabilirlik üzerinde

daha fazla etkiye sahip olan bileşen olup, etki alanındaki bir takım araçların güvenlik açığı algılama yeteneklerini kullanabilmelidir.

- Alakalı sonuçları bildirmek için bir karşılaştırma ölçütü gerçek dünyayı temsil etmelidir. Temsili gerçekçi bir koda dayanmalı (örnek) ve gerçekçi bir zayıflık kümesi içermeli (CWE) ve temsiliyet oranını artırmak için ölçeklenebilir olmalıdır.
- Bir karşılaştırma ölçütü, değerlendirilen hedef araçlarda minimum değişiklik gerektirmelidir.
- Karşılaştırma ölçütünü uygulamak ve mümkün olduğunca çalıştırma kolaylığına sahip olmalıdır. İdeal olarak, kullanılmaya hazır bir bilgisayar programı mevcut değilse, kıyaslamanın nasıl gerçekleştirileceğini ve yürütüleceğini ayrıntılı olarak belirten bir belge olarak sağlanmalıdır. Buna ek olarak, karşılaştırma ölçüt (benchmark) uygulaması en kısa sürede yapılabilmelidir.

NIST SAMATE ("Yazılım Güvencesi Ölçütleri ve Aracı Değerlendirme") projesi tüm bu özellikleri sergilemesi nedeniyle bu çalışma içerisinde kıstas ölçüt olarak seçilmiştir.

Minimum bir işlevi gerçekleştirmek için kaynak kod üzerinde çalışan güvenlik statik analiz aracı detaylı birtakım görevleri yerine getirmelidir. Herhangi bir aracın bulması gereken minimum açıklık setinin seçimi için SAMATE tarafından önerilen ölçütler:

- Kullanılan güvenlik açığı taksonomisi, MITRE tarafından farklı kod karmaşıklıklarını da dikkate alan ve yaygın olarak kabul edilen “Ortak Zayıflık Numaralandırması (CWE)” olmalıdır.
- Güvenlik açıkları mevcut kodda bulunabilmeli ve kötü niyetli bir kullanıcının bunları tanıması ve kullanması için oldukça kolay anlaşılır olmalıdır.

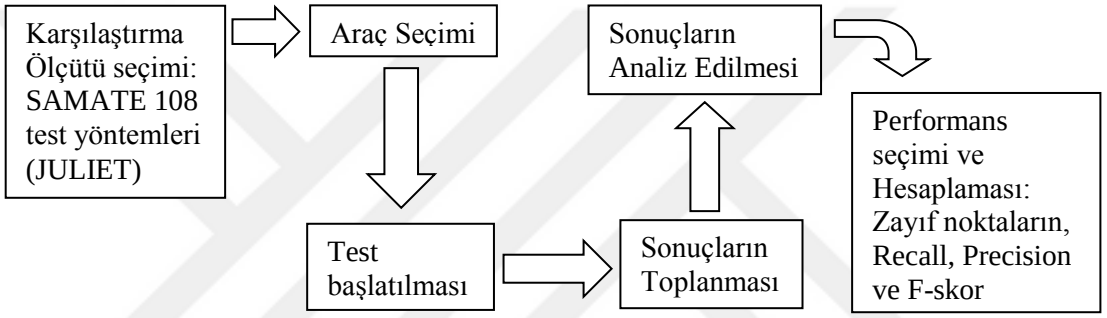
SAMATE, bir aracın güvenlik açıklarının bazılarınım (veya tümünün) algılanıp test edilip edilmediğini kontrol etmek için tasarlanmış, farklı diller için çok sayıda sınama da sunmaktadır. Örneğin, amaç, statik aracın belirli bir güvenlik açığını tespit edip etmediğini bilmek ise, ilgili testi seçilmelidir. Ardından, statik araç testteki

kod üzerinde yürütülür ve aracın gerçekten açığı algılayıp algılamadığı görülür. SAMATE, seçilen güvenlik açıklarının çoğunu (veya tümünü) kapsayan bazı "test yöntemleri" sunmaktadır.

SAMATE, bir referans haline gelmek için mümkün olan tüm çabayı göstermektedir ve statik analiz araçlarına uygulandığında elde edilen bilgilerin paylaşımı için kullanımı önerilmektedir. Bu çalışma içerisinde ayrıntılı olmasa da, ilgili testler daha önce referans olarak kullanılmıştır. SAMATE testlerinin alt kümeleri seçilebilmekte ve testler istenildiği gibi daraltılıp genişletilebilmektedir. Bazı SAMATE testleri, Motorola kodlama standartlarındaki güvenlik kurallarının kapsamını anlamak için çalışma yapmakta kullanılmıştır. NIST SAMATE projesi, kaynak kodundaki güvenlik açıklarını tespit eden statik analiz araçlarındaki araştırmaları ilerletmek için 2018 yılında JULIET in 1.3 sürümünü yayınlamıştır [18]. SAMATE'nin ana hedefleri, test setlerine dayanan deneysel (ampirik) araştırmaları mümkün kılmak, araçlardaki iyileştirmeleri için teşvik etmektir. Aynı zamanda objektif olarak üretim yazılımındaki kullanımlarını göstermek yoluyla araçların daha geniş ve daha hızlı bir şekilde benimsenmesini teşvik etmektir.

3. DEĞERLENDİRME METODOLOJİSİ

Bu bölüm, SAMATE ile karşılaştırmayı yapmak için kullanılan metodolojiyi, değerlendirme metriklerini, SAMATE içinde çalışma amacıyla seçilen testleri ve statik analiz araçlarını içermektedir. Son olarak, bu çalışmanın ana araştırma soruları yer almaktadır. Şekil 3.1, içerisinde kullanılan yöntem adım adım gösterilerek bu çalışmada izlenen yaklaşım özetlenmiştir.



Şekil 3.1 Yazılım değerlendirme metodolojisi

İşlem basamaklarında uygulanan yöntemlerde:

- Karşılaştırma ölçütü seçimi ile teste girecek aracın algılaması gereken güvenlik açıklarının listesi netleştirilmektedir.
- Araç seçimi ile JULIET test veri kümesine karşı sınanacak araç seçilmektedir.
- Test sürecinin başlatılması ile seçilen araçların JULIET test veri kümesi üzerinde yürütülmesi gerçekleştirilmektedir.
- Sonuçlar, geliştirilen SCATPA aracı ile değerlendirilebilmeleri için işlenebilir olmaları gerekmektedir.
- Sonuçların analizi ile oluşan raporları inceleyen SCATPA kullanılarak performans metrikleri elde edilmektedir.
- Performans metrikleri ile karışıklık matrisi oluşturulmakta; ilgili matrise dayanan performans değerlendirmesi gerçekleştirilmektedir.

Böylesi bir süreç sonucunda, her araç ile yapılan güvenlik taraması karışıklık

matrisi verileri: yanlış negatif (FN), yanlış pozitif (FP) ve doğru pozitif (TP) değerleri oluşturulmaktadır. İlgili metrikler araçların performanslarını ölçebilmeyi ve/veya değerlendirebilmeyi mümkün kılmaktadır.

3.1. NIST SAMATE Test Grubu

SAMATE, statik güvenlik test aracı satıcılarının, iddialarını doğrulayan araştırmacılara ve satın alacaklara yardımcı olmak adına yazılım güvencesi araçları için spesifikasyonlar ve otomatik test süitleri (veri kümeleri) geliştirmiştir.

SAMATE için bir güvenlik açığı "sistem güvenlik gereksinimlerinde, tasarım, uygulama veya çalıştırmada yanlışlıkla tetiklenebilecek veya kasıtlı olarak istismar edilebilecek, sistemin güvenlik ilkesini ihlal eden yazılımdır" şeklinde tanımlanmaktadır.

SAMATE referans veri kümesi, C, C++, Java, PHP gibi farklı diller için seçilmiş güvenlik açıklarına sahip, binlerce yazılım örneğinin (testleri) çevrimiçi, halka açık bir deposudur. Her bir test yöntemi, test dosyası, güvenlik açığı tarifi, CWE sınıflandırması, kod karmaşıklığı türü, güvenlik açığı yeri ve diğer test durumu metadata bilgilerini sağlamaktadır. Yanlış pozitif (FP) oranı ölçmek için SAMATE, güvenlik açıkları olmayan vakalar veya güvenlik açıklarının giderildiği vakalara da sahiptir.

Her kaynak kodu statik analiz aracının sahip olması gereken işlevsel gereklilikler şunlardır: Kaynak kodda en az bir yazılım zayıflığı kümesi (CWE ID) bulunan güvenlik açıklarının raporlanması, türü ve yeri, bir bulgu raporu üretildiğinin belirlenmesi.

SAMATE'nin tanımladığı gibi, "bir test paketi, özel bir amaç için açıkça seçilen test vakaları topluluğudur". Her bir test yöntemi, test edilen araç tarafından gerekli belirli bir işlevselliği gerçekleştirilebilmesini sağlayan bir atomik program içermektedir. Test paketleri ve test durumları SAMATE veri setinde bulunmaktadır.

Olası tüm karmaşıklık değişkenleri de dâhil olmak üzere, Tablo 2.1'de listelenen C/C++'da yazılmış kod için eksiksiz kaynak kodu güvenlik açıklarını kapsaması nedeniyle SAMATE test paketi 108 (diğer adıyla JULIET), bu çalışma için seçilmiştir. Seçilen tüm analiz araçları en azından C kodu ile çalışabilmektedir. Test paketi 108, C kodu için test edilen araç tarafından üretilen yanlış pozitif oranı

incelemeye de izin vermektedir. Bunun için ilgili test paketinde, test paketi 108'in düzeltilmiş sürümleri de mevcuttur.

Araçları test paketi 108 üzerinde tarama gerçekleştirilmesinin amacı, güvenlik açığının bulunmasıdır. Bulunan güvenlik açıklarının sayısını kontrol etmek, hangilerinin hangi araçkar tarafından tespit edilmediğini görmek ve her araç için yanlış pozitif oranını elde etmektir.

Araçlara ait sonuçları ve raporları analiz ederken yapılması gereken bazı hususlar bir sonraki bölümde yer almaktadır [19-20].

3.2. JULIET Test Ölçeği

Yanlış pozitiflik (FP), bir programda güvenlik açığı olarak görülen ve bildirilen bir güvenlik açığıdır. Fazla sayıda yanlış pozitif raporlanması, pek çok sorunu tetikleyebilmektedir. Herhangi bir geliştirici, yanlış pozitif içeren uzun bir liste ile karşı karşıya kalırsa, listede gizli olan önemli verileri kaçırabilmektedir. Yanlış pozitifler aynı zamanda geliştiricilerin tarama araçlarının kullanışlı olmadığını düşünmelerine de sebep olmaktadır.

Bununla birlikte, birçok farklı disiplinde belirsizlik ile ilgili durumlarda olduğu gibi yanlış negatifler (FN) daha kötü olarak değerlendirilmektedir. Sahte bir negatif, araç ile bulunmayan bir güvenlik açığı olarak tanımlanır. Yanlış pozitif (FP) ile ilişkili ceza, aracın sonuçlarını kontrol etmek için kullanılan süre miktarıdır. Yanlış negatif (FN) ile ilişkili ceza ise daha büyüktür. Çünkü bu durumda güvenlik açığı kod içerisinde farkedilmeden kalmaya devam edecektir. Bir araç somut bir açığı (yanlış negatif) bulamazsa, başka bir yöntemle bulmaya çalışmak, yanlış pozitifleri düzeltmekten daha zordur.

Doğru pozitifler (TP), programda tespit edilen ve doğru kategoride (CWE) iletilen güvenlik açıklarıdır. Bu güvenlik açıkları, geliştiriciler tarafından hızlı şekilde dikkate alınarak düzeltilmesine öncelik verilmesi gereken açıklardır.

Doğru negatifler (TN), kaynak kodun gerçekte bir güvenlik açığı olmamasını beklemektedir. Herhangi bir şekilde güvenlik bildirimi yapılmamış ise (mevcut olmadığı teyit edilmiş ise), bu doğru negatif (TN) olarak değerlendirilmelidir. JULIET test kümesinin yapısı gereği bunu ölçmek için test kümesi içerisinde bulunan C/C++ dosyalarına ait ikincil fonksiyon adında “good” kelimesi, yanlış pozitif (FP) olarak

listelenmemiş ise doğru negatif (TN) olarak değerlendirilmelidir. Bu işlem, üzerinde çalışılan bilgisayarda günler sürebilecek vakit alacağı ve hesaplamalarda kullanılmayacağı öngörülerek hesap edilmemiştir [19].

3.3. Değerlendirme Metrikleri

SAMATE test yöntemleriyle ilişkili olarak hesaplanacak ilk metrik, her araç için güvenlik açığı yüzdesidir. Bir araç en azından Tablo 2.1 içerisinde yer alan güvenlik açıklarını tespit edebilmelidir.

Önceki bölümde, yanlış pozitif (FP) ve negatifler (FN) ile ilişkili olarak statik analiz araçlarının maruz kaldığı sorunları ortaya konmuştur. Ölçümler esnasında elde edilen değerler:

- TP (doğru pozitifler): algılanan gerçek güvenlik açıklarının sayısını (kod içerisinde bulunan zayıflıkları),
- FP (yanlış pozitifler): aslında bulunmamasına karşın algılanan ve/veya yanlış kategorilendirilen güvenlik açıklarının sayısını,
- FN (yanlış negatifler): kod içerisinde bulunamayan toplam güvenlik açıklarının sayısını ifade etmektedir.

Yanlış pozitif (FP) ve negatiflerin (FN) ölçümü, her aracın kıyaslama işlemi sonrasında elde edilen bilgiler ile hesaplanmalıdır. Güvenlik açıklarının ölçülme biçiminden bağımsız olarak, F-Skor metriği güvenlik açığı algılama araçlarını karakterize etmektir. Aslında, hassasiyet tespiti bağlamında tanımlanabilen iki yöntemin (hassasiyet ve anımsama) harmonik ortalamasını temsil etmektedir.

3.3.1. Hassaslık (Precision): Doğru tespit edilen zayıflıkların, tespit edilen tüm zayıflıkların sayısına oranıdır. Hassasiyet, Pozitif öngörülen değer (PPV) olarak da adlandırılır Denklem 3.2'de yer almaktadır.

$$P = \frac{TP}{TP + FP} \tag{3.2}$$

3.3.2. Anımsama (Recall): Doğru tespit edilen güvenlik açıklarının bilinen güvenlik açıklarının sayısına oranıdır. Bu bağlamda Anımsama (Recall), Doğru Pozitif Oranı -True Positive Rate (TPR) veya Duyarlılık olarak da anılır, Denklem 3.3'de yer almaktadır.

$$R = \frac{TP}{TP + FN} \quad \textbf{(3.3)}$$

3.3.3. Harmonik Ortalama:

n: değişken sayısını (adet), X_n: *n* değişken değerini ifade etmek üzere Harmonik Ortalama formülü Denklem 3.4'de yer almaktadır.

$$H = \frac{n}{\frac{1}{X1} + \frac{1}{X2} + \ldots + \frac{1}{Xn}} = \frac{n}{\sum_{i=1}^{n} \frac{1}{x_i}} \quad \textbf{(3.4)}$$

3.3.4. F1-skor

Hassaslık (precision) ve Anımsama (recall) nın harmonik ortalamasıdır, Denklem 3.5 içerisinde yer almaktadır.

$$F_1 = 2 \cdot (\frac{precision\ X\ recall}{precision + recall}) \quad \textbf{(3.5)}$$

3.3.5. F2-skor

Ağırlıkların hassasiyetten daha yüksek öneme sahip olduğu belirtilmektedir F0,5-Skor ise hassaslık (precision), anımsamadan (recall) daha fazla vurgulamaktadır.

Fβ-Skor formülü Denklem 3.6'da yer almaktadır.

$$F_\beta = (1 + \beta^2) \cdot \frac{(precision\ X\ recall)}{((\beta^2\ X\ precision) + recall)} \quad \textbf{(3.6)}$$

Bir araç için F-Skor 0 ile 1 arasında değer almaktadır. %65 hassasiyet oranı elde edilen (Precision) bir araç, verilen uyarının %65 olasılıkla doğru olma ihtimaline sahip olduğu anlamına gelmektedir. 0.8 Anımsamanın (Recall) elde edildiği bir durum bilinen tüm güvenlik açıklarının %80'inin tespit edildiğini ve %20'sinin kaçırıldığını ifade etmektedir. Bu durumda F-Skor yaklaşık 0.72 olacaktır. Karşılaştırma ölçütü (benchmark) kullanıcısının amacına bağlı olarak, F-Skor bu üç ölçüm araçların performans sıralamasını oluşturmak için kullanılmaktadır.

3.4. Seçilen Araçlar

Araçları seçme ölçütleri, araçların performanslarını, kullanılabilirliklerini, kapsadıkları güvenlik açıklarını ve sayılarının karşılaştırılmasına izin vermektedir. Bu çalışma içerisinde analiz edilen tüm araçlardan iki ticari çözüm seçilerek test aşamasında kullanılmıştır.

Bazı durumlarda ticari araçlara ait deneme sürümü dahi elde etmek zordur. Ticari araçların, daha fazla dil desteğine sahip oldukları, Tablo 2.1'de yer alanlara kıyasla daha geniş güvenlik açıklarını kapsadıkları, daha iyi kullanılabilirlik ve yanlış pozitifleri (FP) azaltmak için yardımcı ek araçlar bulundurdukları bilinmektedir.

Bu çalışmanın başlangıcında 6 adet uygulamanın kullanılması hedeflenmiştir. Bunlar: Polyspace, Fortify, Checkmarx, SonarCloud (open), AppScan ve AttackFlow'dur.

IBM firmasına ait AppScan ürününden sorumlu Türkiye şubesi tez çalışması isteğine yanıt vermemiştir.

AttackFlow, "Polyspace aracının çalışmada olma gerekliliği" sabit olduğundan ve AttackFlow'un Polyspace aracının desteklediği C/C++ dillerini desteklememesi nedeniyle testlerde kullanılamamıştır.

Checkmarx firmasının ürününe direk erişimin bulunmaması, limitli lisanslamaya dayanması (ör: bir dosya tarama limitli lisans) ve taramaları yapan personelin "her CWE kodunu aracın farklı tespit ettiğini" iletmesi sonrasında ilgili firma ile iletişim kurulamamıştır.

SonarCloud (open) a ait "sonarscan" ve "sonarqube" uygulamalarına ait destek ekibi ile görüşülmüştür. Ek araştırma ve gerçekleştirilen testler sonucunda tarama yapılarak bulut (cloud) ortamına yükleme yapılabilmiştir. Ancak SonarCloud (open), üzerinde çalışılması gereken fiziksel raporların indirilmesine olanak sağlamamaktadır. Bunun yapılabilmesi için tamamen ayrı bir uygulama daha geliştirerek, uygulama programlama arabirimlerinin (API) kullanılması gerekmektedir. Bu nedenle bu çalışma içerisinde yer almamaktadır. Polyspace ve Fortify araçları kıyaslanmıştır.

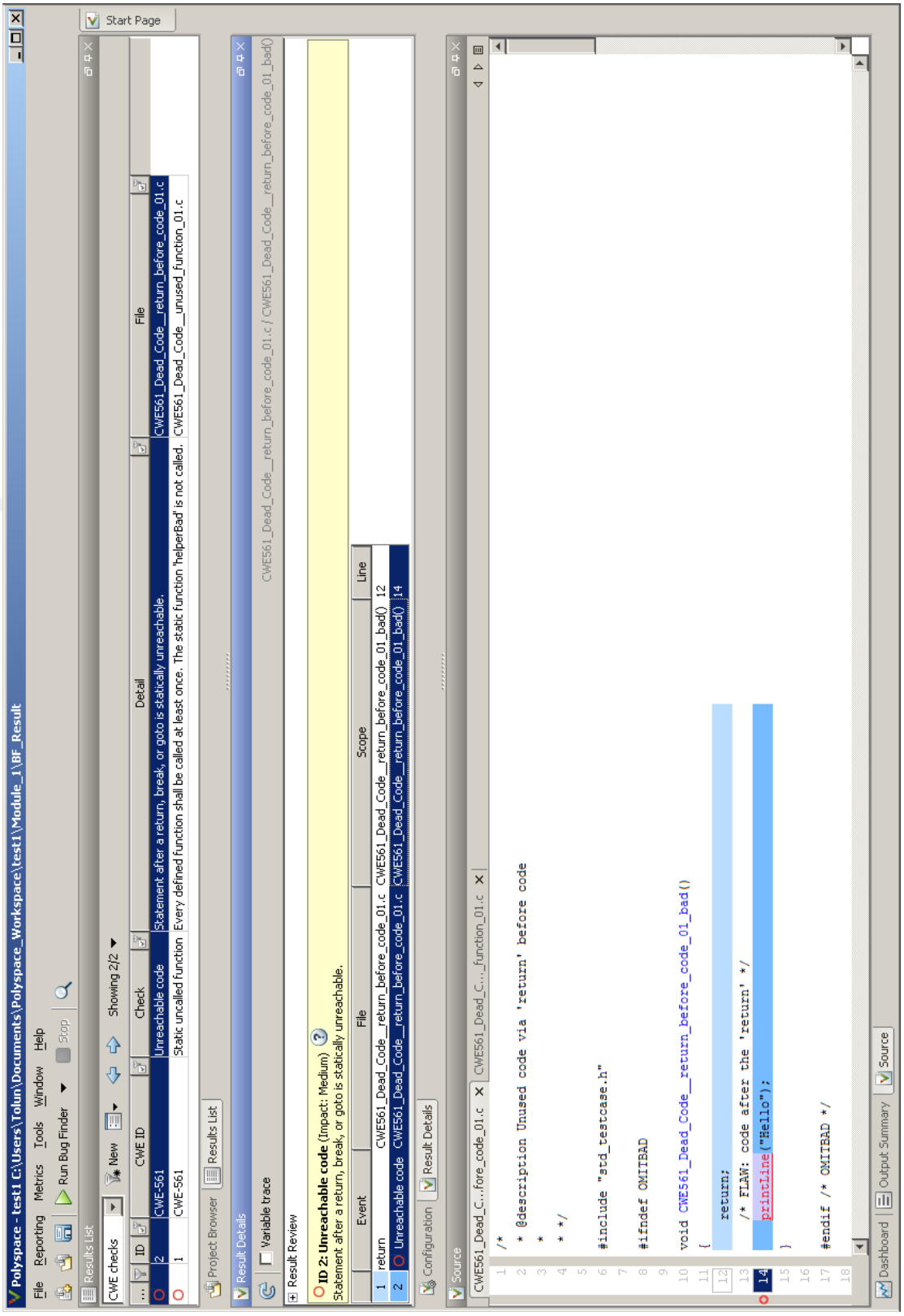
Tablo 3.1 İncelenerek kararlaştırılmış statik analiz araçları

ARAÇLAR	KARAKTERİSTİK ve DEĞERLENDİRİLMESİ
POLYSPACE	MatLab araç kutularından Polyspace, içerisinde bulundurduğu BugFinder ile 155 adet kategoride desteği bulunur iken, Tablo 2.1 güvenlik açıkları kategorilerinin %64.41'ini kapsamaktadır.
FORTIFY	Lider güvenlik inceleme aracı. Toplamda 278 adet kategoriyi destekler iken, Tablo 2.1 güvenlik açıkları kategorilerinin %48.30'unu kapsamaktadır.

3.4.1. Polyspace (POL)

Polyspace araç kutusunda bulunan BugFinder, bütün platformlar için (gömülü sistemler de dâhil olmak üzere) C/C++ dilleri için bir "hata bulma" aracıdır. Bir C programını, orijinal programın bir soyutlaması olan Boolean programına dönüştürür. Daha sonra Boolean programı bir Model Denetçisine geçirilir. Dayanak (predicate) soyutlama, dizi sınırları (ara bellek taşmaları), işaretçi güvenliği, istisnalar ve kullanıcı tarafından belirlenen denetim akışı gibi özellikler için en uygun araçtır [21].

BugFinder aracına ait ekran görüntüsü Şekil 3.7'de yer almaktadır.

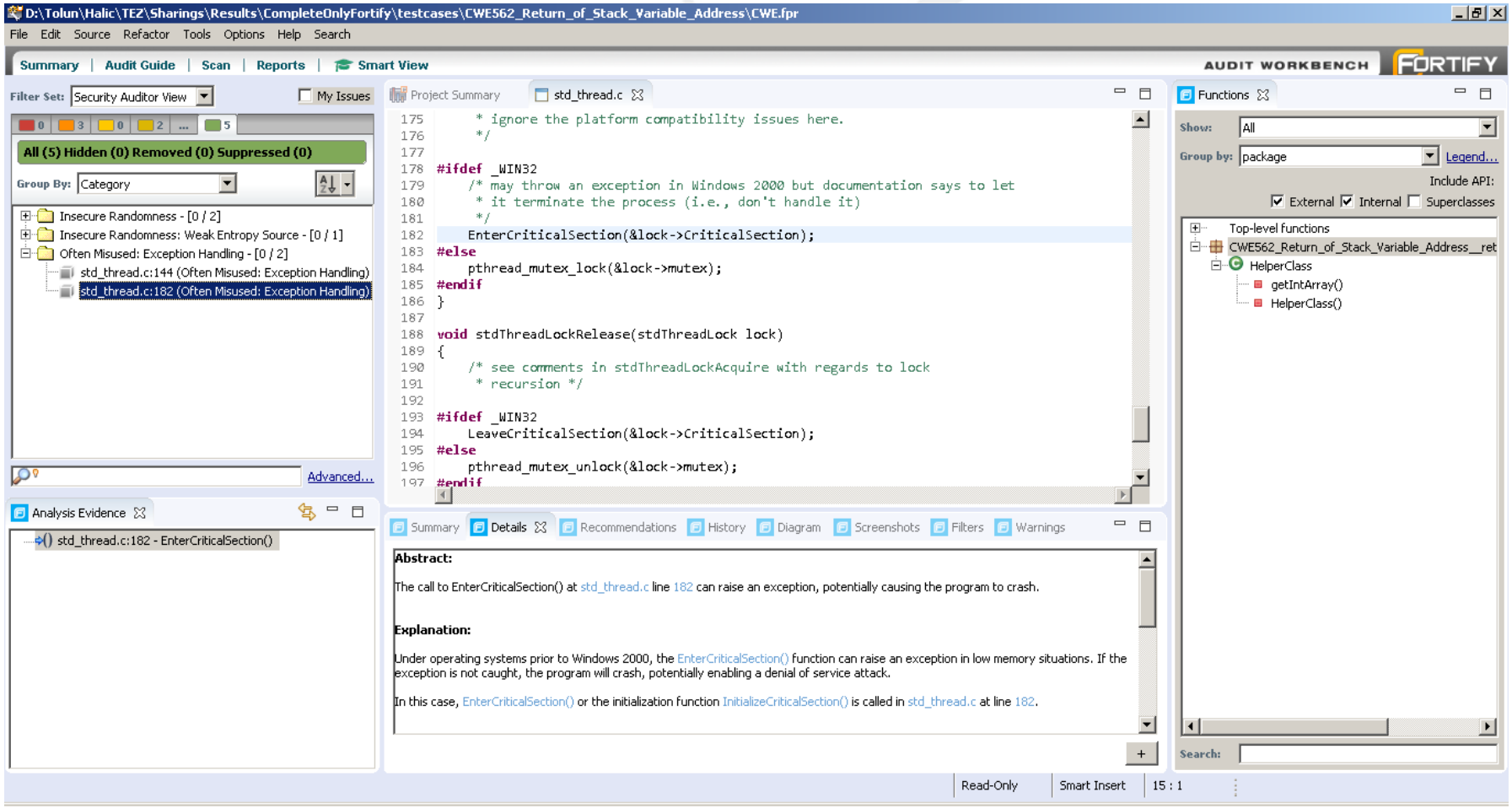


Şekil 3.7 Polyspace BugFinder aracı ekran görüntüsü

3.4.2. Fortify (SCA)

Fortify yazılım ailesinin bir ürünü olup, günümüzde Hewlett-Packard firmasından MicroFocus firmasına geçmiştir. SCA "güvenlik incelemesi" tipinde bir araçtır. Sözlü, sözdizimsel ve semantik analiz, kontrol akışı ve veri akışı analizi için kullanır. Pek çok farklı güvenlik kuralını kullanarak, özel analizcilerin çalıştırdığı kodun ara modelini oluşturur. Kullandığı algoritma veya tekniklerin türüne ilişkin ayrıntıları gösteren hiçbir kamuya açık belge sunmamaktadır. SCA, C/C++, C#, ASP.NET, VB.NET, COBOL, CFML, HTML, Java, JavaScript, AJAX, JSP, PHP, PL/SQL, Python, Visual Basic, VBScript ve XML gibi yirmibeş (25) dili kapsamaktadır. Java dilinde geliştirildiğinden birçok platformu desteklemektedir. SCA'nın 1.000'den fazla güvenlik açığı kategorisi için ayrıntılar sunduğu belirtilmektedir [22].

Fortify Audit Workbench aracına ait ekran görüntüleri Şekil 3.8'de görülmektedir.



Şekil 3.8 Fortify Audit Workbench aracı ekran görüntüsü

Her bir aracın Tablo 2.1 kullanılarak gerçekleştirilen testlerinde iddia edilen kapsama sayısından farklı sayıda algılama oranına ulaşılmıştır.

Araçları çalıştırma prosedürü kolay, ancak açıkça farklı özelliklerine ve farklı kullanılabilirlik düzeylerine sahip olması nedeniyle her araç için farklıdır.

Araçların kendi içlerinde değerlendirilmelerinin haricinde, ortak destekledikleri güvenlik kategorilerinin de belirlenerek, birbirleri arasında karşılaştırılması, toplam 38 adet kategoride yani ölçeğin %32.20 lik kısmında gerçekleştirilmiştir.

Çalışmaya dâhil edilemeyen araçlar ve nedenleri ilerleyen bölümlerde yer almaktadır

3.4.3. SonarCloud (open)

SonarSource firmasına ait ve piyasada hızla yaygınlaşmaya başlayan "hata bulma" ve "yazılım kalite denetim" ürünleri mevcuttur [23-24].

SonarCloud (open) açık kaynak ürünler için tamamen ücretsiz hizmet vermektedir ve sınırsızca kullanılabilmektedir. Kapalı kodlu projelere ise yazılım satır sayısına bağlı olarak lisans karşılığında hizmet vermektedir. Fortune 100 şirketlerinin kırk üç (43) ünde kurumsal lisans ile kullanılmaktadır. Çok kısa bir sürede güvenlik başlık sayılarını artıracaklarını belirtmişlerdir. ABAP, Apex, C/C++, Objective-C, COBOL, C#, CSS, Flex, Go, HTML, Java, JS, Kotlin, PHP, PL/I, PLSQL, Python, RPG, Ruby, Scala, Swift, TS, TSQL, VB6, VB ve XML gibi yirmibeş (25) üzerinde dili desteklemektedir.

3.4.4. Checkmarx

Checkmarx firmasına ait olan ürün, SonarCloud ve Fortify kadar tercih edilmemekle birlikte kodları derlemeden tarayan bir yaklaşıma sahiptir [25].

Araştırmacılara yüksek bilgi paylaşım ortamı sağlamasına karşın ürünlerine ulaşmak katı lisanslama modelinden dolayı zorludur. Bu çalışma içerisinde JULIET test sürecine dâhil edilmek istenmesine rağmen temsilci yerel kuruluş tarafından destek alınamamıştır. Java, C#, PHP, Python, Groovy, Ruby, Android, Objective-C, HTML5, C++, JS, ASP, VB, VB6, PLSQL, Perl, Apex, Scala, Swift, Go, JSP, TS ve VBScript gibi toplamda yirmi (20) dili desteklemektedir [26].

3.4.5. AppScan Source

IBM firmasına ait, piyasada nadir tercih edilen dinamik ve statik taramalar yapabilen bir ürün ailesine mensuptur [27].

AppScan Source ürünü IBM in statik analiz yapabilen ürünüdür. Ürün farklı işletim sistemlerinde farklı dilleri desteklemektedir. Dolayısıyla her programlama dili her işletim sisteminde desteklenmemektedir. C/C++, Objective-C, COBOL, ColdFusion, Java, JSP, JS, Perl, PHP, PLSQL, Python, TSQL, C#, ASP, VB, ASP ve VB6 gibi toplamda onyedi (17) programlama dilini desteklemektedir [28].

3.4.6. AttackFlow

Statik analiz araçlarına daha büyük önem vermeyi planlayan sürekli gelişim içerisinde bulunan bir üründür.

Günümüzde Java, C#, Android ve JSP olmak üzere toplam dört (4) programlama dilini desteklemektedir [29].

3.5. Araştırma Soruları

Bu çalışma daha önce açıklanan metodolojiyi izleyerek, seçilen iki statik kaynak kodu güvenlik analiz aracı için, şu sorulara cevap aramaktadır:

- SAMATE test 108 paketinde yer alan farklı kategorilere sahip güvenlik açıklarını tespit etme açısından iki araç nasıl karşılaştırılır?
- Yanlış pozitifleri (FP) raporlama açısından iki araç nasıl karşılaştırılır?
- İki araç, güvenlik açıkları açısından Tablo 2.1'de yer alan bilgiler ile nasıl kıyaslanır?
- Analiz edilen araçlar için doğru pozitif (TP) / yanlış pozitif (FP) oranı nedir?
- Önerilen metodoloji içerisinde SAMATE karşılaştırma ölçütlerinin yeterlilik derecesi nedir?

JULIET test senaryoları bünyesinde bulunan 118 adet CWE ana başlıkta bulunan toplam 64.099 adet test senaryosunun her bir araç ile sınanması ciddi vakit kaybına sebebiyet vereceğinden Şekil 3.9 de görünen pratik Matlab betiği geliştirilmiştir.

Bu betik kullanılarak ilgili statik analiz aracının desteklemediği CWE ana başlıklarını tarama listesinden çıkartılmaktadır. Bu amaçla sadece ölçek ile statik analiz aracının örtüştüğü CWE ana başlıklarını belirleyerek "CWEcomparison.xlsx" adındaki tabloya kaydetmektedir. İlgili betikten de anlaşılacağı üzere sadece "ölçek – kullanılan statik analiz aracına" ait ortak CWE anabaşlıklarını değil, "ölçek - statik analiz araçlarının" toplam ortak kesişim CWE başlıklarını da belirleyerek, birbirlerine karşı yapılacak analizlerde kullanılmak üzere tabloya kaydedilmektedir.

Tablo 3.2 de tarama amacıyla kullanılacak statik analiz araçlarının kritik özellikleri karşılaştırmalı olarak değerlendirilmektedir.

Tablo 3.2 Statik analiz araçlarının kritik özellik karşılaştırması

KRİTER	**Polyspace** (BugFinder)	**Fortify** (SCA)	**SonarCloud** (Open)	**Checkmarx**	**AttackFlow**	**AppScan** (Source)
C/C++ desteği	+	+	+	+	-	+ (sadece Lin. Win.)
Lisans Türü	Ücretli	Ücretli	Ücretsiz (sadece açık kaynak projeler)	Ücretli	Ücretli	Ücretli
Windows desteği	+	+	+	+	+	+
Linux desteği	+	+	+	-	-	+
MacOS desteği	+	+	+	-	-	+
Test ortamlarına erişim / lisans temin süreci	Kolay	Zor	Kolay	Zor	Kolay	Zor
Tarama raporlarının fiziksel temin edilme süreci	Hızlı (uygulama içinden)	Hızlı (uygulama içinden)	Zaman alıcı (API kullanılarak)	Hızlı (uygulama içinden)	Hızlı (uygulama içinden)	Hızlı (uygulama içinden)

```
1: global xlApp, book;
2: try
3:    xlApp = actxGetRunningServer('Excel.Application');
4: catch ME %#ok
5:    xlApp = actxserver('Excel.Application');
6: end
7: xlApp.Visible = 1;
8: book =
xlApp.Workbooks.Open('D:\Tolun\Halic\TEZ\CWEcomparison.xlsx');
9: %Read CWEs
10: CWEs_Juliet =
cell2mat(book.Sheets.Item('Sheet1').Range('A3:A121').Value);
11: CWEs_Polyspace =
cell2mat(book.Sheets.Item('Sheet1').Range('C3:C157').Value);
12: CWEs_Fortify =
cell2mat(book.Sheets.Item('Sheet1').Range('D3:D280').Value);
13: %PREPARE INTERSECTIONS
14: intsec_p = intersect(CWEs_Polyspace, CWEs_Juliet);
15: intsec_f = intersect(CWEs_Fortify, CWEs_Juliet);
16: intsec_pf = intersect(CWEs_Polyspace, CWEs_Fortify);
17: ProductsIntersection = intersect(CWEs_Juliet, intsec_pf);
18: %Prepare Intersection result
19: sendToDocument('F', 'Common CWEs with Juliet\n(only
Polyspace)', intsec_p)
20: sendToDocument('G', 'Common CWEs with Juliet\n(only Fortify)',
intsec_f)
21: sendToDocument('I', 'Common CWEs with Juliet\n(intersection of
All)', ProductsIntersection)
22: book.Save;
23: book.Close;
24: xlApp.Quit;
25: delete(xlApp);
26: clear intsec* xlApp book ans;
27:
28: function sendToDocument(column,columnTitle,data)
29: global book
30:   book.Sheets.Item('Sheet1').Columns.Item(column).Delete;
31:   book.Sheets.Item('Sheet1').Columns.Item(column).Insert;
32:   book.Sheets.Item('Sheet1').Range(strcat(column,'1')
).EntireColumn.Font.Bold=1;
33:   book.Sheets.Item('Sheet1').Range(strcat(column,'1')
).EntireColumn.HorizontalAlignment = 3;
34:   book.Sheets.Item('Sheet1').Range(strcat(column,'1')
).EntireColumn.VerticalAlignment = 2;
35:   book.Sheets.Item('Sheet1').Range(strcat(column,'1')
).EntireColumn.ColumnWidth='32';
36:   book.Sheets.Item('Sheet1').Range(strcat(column,'1')).Value2
= sprintf(columnTitle);
37:   LastPoint = length(data)+2;
38:   updatedRange = book.ActiveSheet.Range([strcat(strcat(column,
'3:'),column),num2str(LastPoint)]);
39:   updatedRange.Value2 = data;
40: end
```

Şekil 3.9 Ölçek ve statik analiz araçları için ortak CWE kod tespit betiği

4. ARAÇLARIN SAMATE TEST YÖNTEMLERİ İLE SINANMASI

4.1. Çalışmada Kullanılacak Teknolojilerin Seçimi

Ölçek olarak SAMATE projesine ait JULIET test senaryoları topluluğu (test suite), seçilen statik analiz araçları tarafından sınanmış ve araçların üreteceği "işlenebilir raporlar" biriktirilmiştir. Biriktirilen raporlar, VisualStudio 2015 C# ile geliştirilen SCATPA (Static Code Analysis Tools Performance evAluation) adlı projje ile değerlendirmeye alınmıştır. Son aşama olarak araçların (Polyspace ve Fortify) hem kendi içlerindeki performansı, hem de birbirlerine karşı performanslarının değerlendirilebilmesi için sonuç raporları oluşturulmuştur. Ayrıca Matlab programında ortak CWE kesişimlerinin bulunması için, toplu dosyalama işlemleri ve toplu tarama işlemleri için çeşitli betikler oluşturulmuştur. Betiklere CD ekinden ya da internet üzerinden paylaşılan adres aracılığı ile erişilebilmektedir [7].

4.2. Çalışmada Kullanılacak Platform

Üzerinde çalışılacak sistem, çalışmalara başlamadan modifikasyonlarda bulunulmuş ve sistem bakımı, donanım yükseltme işlemleri gerçekleştirilmiştir.

Tablo 4.1 Çalışmada kullanılan bilgisayar teknik özellikleri

İşlemci Tipi	2. Nesil Intel Core i7
İşlemci Kodu	i7-2670QM
İşlemci Hızı	2.2 GHz - 3.10 GHz (turbo ile)
Çekirdek sayısı	4 – 8 (HT teknolojisi ile)
Ön Bellek	6 MB
İşletim Sistemi	Windows 7
Ana bellek	8 GB
Bellek Tipi	DDR3-1333MHz
Sabit Disk	120Gb (SSD) + 320Gb (SATA)

Şekil 4.1'deki kaynak kod, 108 numaralı test topluluğu JULIET içinde mevcut olan CWE562 güvenlik açığını içeren (helperBad) ve açığın giderilmiş (helperGood1) fonksiyonlarını içeren bir C programına aittir. Böylelikle statik analiz araçlarının performanslarının değerlendirilebilmesi için ortam oluşturulmuştur.

```
1: #include "std_testcase.h"
2: #ifndef OMITBAD
3: static char *helperBad()
4: {
5:     char charString[] = "helperBad string";
6:     char *ptrCharString;
7:     ptrCharString = &charString[1];
8:
9:     return ptrCharString;
10: }
11: void
CWE562_Return_of_Stack_Variable_Address__return_pointer_buf_01_bad()
12: {
13:     printLine(helperBad());
14: }
15: #endif
16: #ifndef OMITGOOD
17: static char *helperGood1()
18: {
19:     static char charString[] = "helperGood1 string";
20:     char *ptrCharString;
21:     ptrCharString = &charString[1];
22:
23:     return ptrCharString;
24: }
25: static void good1()
26: {
27:     printLine(helperGood1());
28: }
29: void
CWE562_Return_of_Stack_Variable_Address__return_pointer_buf_01_good()
30: {
31:     good1();
32: }
33: #endif
34: #ifdef INCLUDEMAIN
35: int main(int argc, char * argv[])
36: {
37:     srand( (unsigned)time(NULL) );
38: #ifndef OMITGOOD
39:     printLine("Calling good()...");
40:
CWE562_Return_of_Stack_Variable_Address__return_pointer_buf_01_good();
41:     printLine("Finished good()");
42: #endif
43: #ifndef OMITBAD
44:     printLine("Calling bad()...");
45:
CWE562_Return_of_Stack_Variable_Address__return_pointer_buf_01_bad();
46:     printLine("Finished bad()");
47: #endif
48:     return 0;
49: }
50: #endif
```

Şekil 4.1 SAMATE test paketi 108 test yöntemi 562

Seçilen statik analiz araçlarına ait işlenebilir tarama raporlarına ait çıktılar çok farklı düzenlere sahiptir. Ancak her birinin temel olarak iletmeleri gereken başlıklar vardır. Bunlar, CWE kodu, ilgili dosya adı, CWE açıklamasıdır.

Raporlarda, temel başlıklara odaklanıldığında işlemlerin çok daha hızlı ilerlediği görülmüştür. Statik analiz araçlarına ait örnek rapor çıktıları Şekil 4.2 ve Şekil 4.3'da görülmektedir.

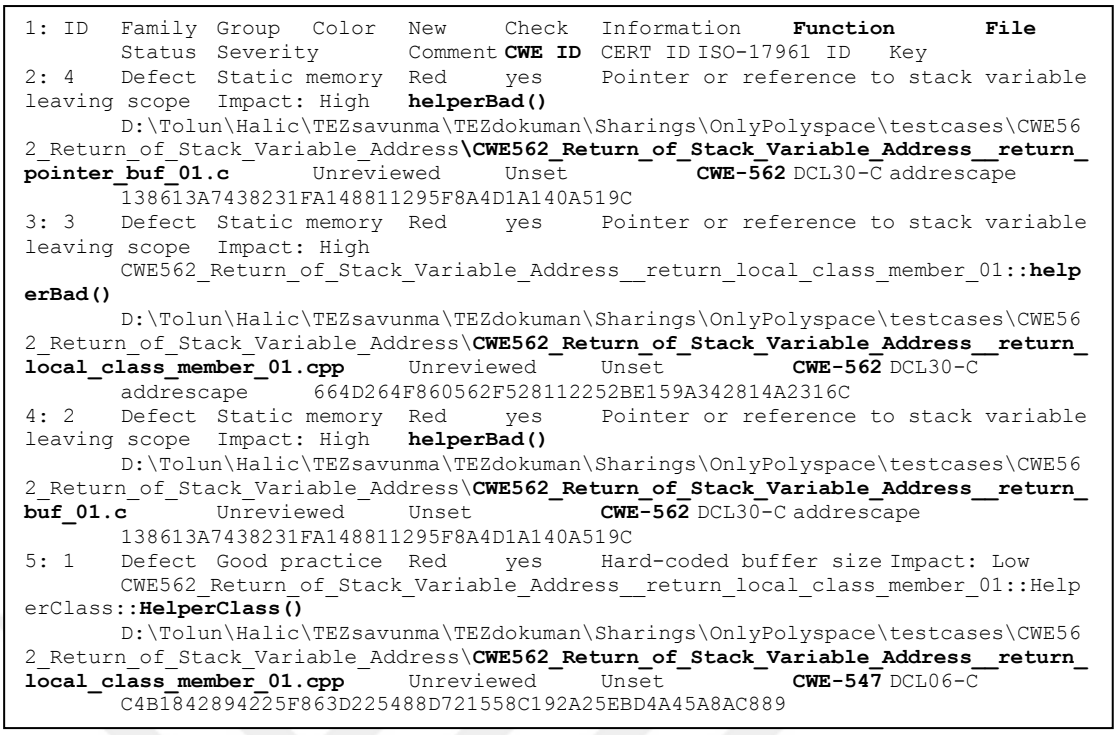
Şekil 4.2 Fortify statik analiz aracına ait örnek bir rapor kesitini içermektedir. Bu rapor içerisinde geliştirilen SCATPA uygulamasına ait "ReportSection", "SubSection", "GroupSection", "groupTitle" ve "Issue" etiketleri (tags) değerlendirilmeye alınarak TP, TN, FP ve FN hesapları yapılmaktadır. Örneğin, "ReportSection" ve "SubSection" etiketleri aktif olanlar değerlendirmeye alınarak, diğer alt kırılım etiketlerine ("GroupSection", "groupTitle" ve "Issue") ilerlenir.

```
1: <?xml version="1.0" encoding="UTF-8" standalone="yes"?>
2: <ReportDefinition type="standard">
3:     <TemplateName>Fortify Developer Workbook</TemplateName>
3:     <TemplateName>Fortify Developer Workbook</TemplateName>
4:     <TemplatePath></TemplatePath>
4:     <TemplatePath></TemplatePath>
5:     <LogoPath>/MF_logo.jpg</LogoPath>
5:     <LogoPath>/MF_logo.jpg</LogoPath>
6:     <Footnote>Copyright 2018 Micro Focus or one of its affiliates.</Footnote>
6:     <Footnote>Copyright 2018 Micro Focus or one of its affiliates.</Footnote>
7:     <UserName></UserName>
7:     <UserName></UserName>
8:     <ReportSection enabled="false" optionalSubsections="false" />
9:     <ReportSection enabled="false" optionalSubsections="false" />
10:     <ReportSection enabled="true" optionalSubsections="true">
11:         <Title>Results Outline</Title>
11:         <Title>Results Outline</Title>
12:         <SubSection enabled="true">
13:             <Title>Vulnerability Examples by Category</Title>
13:             <Title>Vulnerability Examples by Category</Title>
14:             <Description>Results summary of all issue categories.
Vulnerability examples are provided by category.</Description>
14:             <Description>Results summary of all issue categories.
Vulnerability examples are provided by category.</Description>
15:             <IssueListing listing="true" limit="-1">
16:                 <Refinement></Refinement>
16:                 <Refinement></Refinement>
17:                 <Chart chartType="list">
18:                     <Axis>CWE</Axis>
18:                     <Axis>CWE</Axis>
19:                     <MajorAttribute>Analysis</MajorAttribute>
19:                     <MajorAttribute>Analysis</MajorAttribute>
20:                     <GroupingSection count="2">
21:                         <groupTitle>CWE ID 338</groupTitle>
21:                         <groupTitle>CWE ID 338</groupTitle>
22:                                         <MajorAttributeSummary />
23:                                         <Issue
iid="EA32D9C324DBDF1E4AB00857012C9FE1" ruleID="CB0D4A41-677E-49CF-926F-
F8EFBB7AF919" />
24:                                         <Issue
iid="16AC7C56A41EB3C62176171C8336AFDF" ruleID="CB0D4A41-677E-49CF-926F-
F8EFBB7AF919" />
25:                     </GroupingSection>
26:                     <GroupingSection count="1">
27:                         <groupTitle>CWE ID 331</groupTitle>
27:                         <groupTitle>CWE ID 331</groupTitle>
28:                         <MajorAttributeSummary />
29:                                         <Issue
iid="8AE4C4212A3D70A706E0031000678095" ruleID="3ACF7C34-9BA7-48B9-B679-
4575D5257CBD" />
30:                     </GroupingSection>
31:                     <GroupingSection count="2">
32:                         <groupTitle>CWE ID 248</groupTitle>
32:                         <groupTitle>CWE ID 248</groupTitle>
33:                         <MajorAttributeSummary />
34:                                         <Issue
iid="8A7E5697F2DDFCE5CD590698C61E8187" ruleID="9505C356-A385-4B5F-8946-
7210D8DEF10E" />
35:                                         <Issue
iid="851687C83939CB8305B1CF516C51EDAE" ruleID="9505C356-A385-4B5F-8946-
7210D8DEF10E" />
36:                     </GroupingSection>
37:                 </Chart>
38:             </IssueListing>
39:         </SubSection>
40:     </ReportSection>
41: </ReportDefinition>
```

Şekil 4.2 Fortify aracına ait örnek rapor kesiti

4.3. Polyspace Karışıklık Analizi Çalışması

Şekil 4.3 Polyspace statik analiz aracına ait örnek bir rapor kesitini içermektedir. Bu rapor geliştirilen SCATPA uygulamasında “Function”, “File” ve “CWE ID” başlıklarını değerlendirmeye alarak TP, TN, FP ve FN hesaplarını yapılmaktadır. Örneğin, “File” ve “CWE ID” bilgileri JULIET ölçeğini manifest.xml dosyası ile kıyaslanmaktadır.



Şekil 4.3 Polyspace aracına ait örnek rapor kesiti

Tablo 4.2 SCATPA aracılığı ile oluşturulmuş, Polyspace ürününe ait karışıklık matrisi çıktısını içermektedir. İlgili tablo içerisinde yer alan veriler incelendiğinde: CWE562, Polyspace e 3 defa sorulmuş ve 3 defa da doğru cevap verdiği görülmektedir Dolayısıyla TP sütununda 3 (üç), diğer sütunlarda 0 (sıfır) olarak yer almaktadır. CWE562 için Polyspace gayet başarılı bir sonuç elde edilmektedir. Diğer bir CWE kodu olan CWE134 de ise, 2880 defa sorgu gerçekleştirilmiş, 3078 defa yanlış pozitif (FP), 708 defa yanlış negatif (FN) ve 0 (sıfır) TP sonuç alınmıştır. Polyspace, CWE134 güvenlik açıklarını desteklediğini belirtmesine rağmen yüksek doğruluk seviyesinde çıktı üretmemiştir.

Tablo 4.2 (devam) Polyspace karışıklık matrisi

38	CWE415	962	946	121	-	1308
39	CWE416	459	940	94	-	573
40	CWE426	192	576	17	-	0
41	CWE427	480	3934	0	-	0
42	CWE457	948	3402	4	-	802
43	CWE467	54	24	0	-	54
44	CWE468	37	17	0	-	19
45	CWE469	36	67	12	-	18
46	CWE475	36	16	0	-	36
47	CWE476	348	787	42	-	288
48	CWE478	18	59	0	-	18
49	CWE479	18	110	0	-	54
50	CWE480	18	27	0	-	0
51	CWE481	18	59	0	-	18
52	CWE482	18	8	0	-	18
53	CWE484	18	110	0	-	18
54	CWE534	36	220	0	-	0
55	CWE535	36	16	0	-	0
56	CWE561	2	0	0	-	2
57	CWE562	3	0	0	-	3
58	CWE587	18	8	12	-	0
59	CWE590	2680	3773	487	-	600
60	CWE606	480	503	176	-	0
61	CWE665	193	253	12	-	0
62	CWE666	90	40	0	-	90
63	CWE667	18	8	12	-	0
64	CWE672	47	16	43	-	0
65	CWE675	192	1366	43	-	0
66	CWE676	18	6	14	-	0
67	CWE681	54	177	0	-	0
68	CWE685	18	8	0	-	18
69	CWE690	960	1396	335	-	638
70	CWE758	581	1468	173	-	0
71	CWE762	3564	11230	844	-	636
72	CWE780	18	8	0	-	0
73	CWE785	18	44	0	-	18
74	CWE789	960	2470	315	-	235
75	CWE832	18	8	12	-	0
76	CWE843	80	274	12	-	0

4.4. Fortify Karışıklık Analizi

Tablo 4.3 Fortify karışıklık matrisi

Sıra	CWE Kodu	Test Adedi	FORTIFY			
			FP	FN	TN	TP
1	CWE15	48	74	3	-	44
2	CWE23	2400	600	1800	-	0
3	CWE78	5796	1218	4578	-	0
4	CWE90	480	1190	0	-	1070
5	CWE114	576	708	0	-	1284
6	CWE124	2048	2447	1020	-	0
7	CWE126	1452	1375	1147	-	0
8	CWE127	2048	3153	1020	-	0
9	CWE134	2880	4332	0	-	6450
10	CWE176	48	116	18	-	0
11	CWE190	3960	17641	1482	-	0
12	CWE195	1152	2583	129	-	1085
13	CWE226	72	786	0	-	0
14	CWE242	18	0	18	-	0
15	CWE244	72	1230	0	-	102
16	CWE247	18	51	0	-	51
17	CWE252	630	102	450	-	144
18	CWE253	684	102	648	-	0
19	CWE256	96	966	0	-	0
20	CWE259	96	668	0	-	163
21	CWE272	252	0	252	-	0
22	CWE284	216	0	216	-	0
23	CWE319	192	2136	0	-	0
24	CWE321	96	274	0	-	0
25	CWE325	72	186	0	-	0
26	CWE327	54	765	0	-	18
27	CWE328	54	0	0	-	54
28	CWE338	18	0	0	-	18

Tablo 4.3 SCATPA aracılığı ile oluşturulmuş, Fortify ürününe ait karışıklık matrisini içermektedir. İlgili tabloda yer alan veriler incelendiğinde: CWE562, Fortify a 3 defa sorulmuş ve 3 defa da yanlış cevap elde edilmiştir. Dolayısıyla TN kolonunda 3 (üç), diğer kolonlarda 0 (sıfır) değer elde edilmektedir. CWE562 için Fortify tamamen başarısız görünmektedir. Diğer bir CWE kodu olan CWE134 de ise, 2880 defa sorgu yürütülmüş, 4332 defa yanlış pozitif (FP) ve 6450 doğru pozitif (TP) sonuç görülmüştür. Fortify, CWE134 güvenlik açıklarını desteklediğini belirtmesine rağmen

zaman maliyetine sebep olabilecek rapor üretmiştir.

Tablo 4.3 (devam) Fortify karışıklık matrisi

29	CWE364	18	0	18	-	0
30	CWE367	36	0	36	-	0
31	CWE377	144	0	54	-	159
32	CWE391	54	0	54	-	0
33	CWE396	54	306	0	-	0
34	CWE397	20	4	19	-	0
35	CWE398	181	153	127	-	0
36	CWE400	720	2055	306	-	0
37	CWE401	1658	1994	632	-	1628
38	CWE404	384	256	174	-	210
39	CWE415	962	322	440	-	500
40	CWE416	459	551	148	-	18
41	CWE457	948	1131	558	-	171
42	CWE475	36	102	0	-	0
43	CWE476	348	434	96	-	205
44	CWE480	18	36	0	-	0
45	CWE526	18	0	18	-	0
46	CWE561	2	0	1	-	1
47	CWE562	3	0	3	-	0
48	CWE563	512	113	172	-	272
49	CWE570	16	13	4	-	0
50	CWE571	16	2	15	-	0
51	CWE591	96	404	0	-	126
52	CWE615	18	0	0	-	138
53	CWE665	193	1	192	-	0
54	CWE674	2	0	2	-	0
55	CWE676	18	33	0	-	0
56	CWE690	960	2395	334	-	0
57	CWE780	18	0	0	-	18

Tablo 4.4 SCATPA aracılığı ile oluşturulmuş, Fortify ve Polyspace ürünlerine ait karşılaştırmalı karışıklık matrisi çıktısıdır. Başarı kıstası doğru pozitif (TP) sayısını olabildiğince yüksek, yanlış pozitif (FP) ve yanlış negatif (FN) sayılarının olabildiğince düşük biçimde elde etmektir. Tablodaki veriler incelendiğide: CWE475, analiz araçlarına 36 defa sorulmuş ve Polyspace 36 defa da doğru cevap vermiştir (TP). TP sütununda 36 (otuzaltı) değer elde edilirken 16 (onaltı) adet yanlış pozitif (FP) görülmektedir. Fortify ise 102 (yüziki) defa yanlış pozitif (FP) ve 0 (sıfır) doğru pozitif (TP) sonuç üretmiştir. Fortify, CWE475 güvenlik açıklarını desteklediğini

belirtmesine rağmen zaman maliyetine sebep olurken, Polyspace ürünü için bu oran 16 (onaltı) yanlış pozitif (FP) ile çok daha düşük seviyededir.

4.5. Karşılaştırmalı Karışılık Analizi

Fortify karşılaştırmalı karışıklık matrisinde sadece 10 (on) adet doğru pozitif (TP) elde etmiştir. Polyspace ise 17 (onyedi) adet doğru pozitif (TP) elde etmiştir. CWE244 ve CWE338 test kodu üzerinde her iki araç benzer sonuçlar elde etmiştir Ancak her iki araçta ortak şekilde 9 (dokuz) test kodunda doğru pozitiflik (TP) kıstasından başarısız olmuşlardır. Sadece doğru pozitif (TP) sayılarında Polyspace, Fortify ‘dan üstün çıkmıştır.

Tablo 4.4 üzerinde normalizasyon uygulanarak TP verilerinin test adetlerini aşmaması sağlanmıştır. Normalizasyon yapılan CWE kodları:

- CWE114
- CWE134
- CWE195
- CWE244
- CWE364
- CWE377
- CWE415
- CWE416

Yanlış pozitif (FP) ler, mevcut olmayan ve/veya yanlış yönlendirmeler içerdiğinden gereksiz iş yükü anlamına gelmekte ve zaman kayıplarını arttırmaktadır.

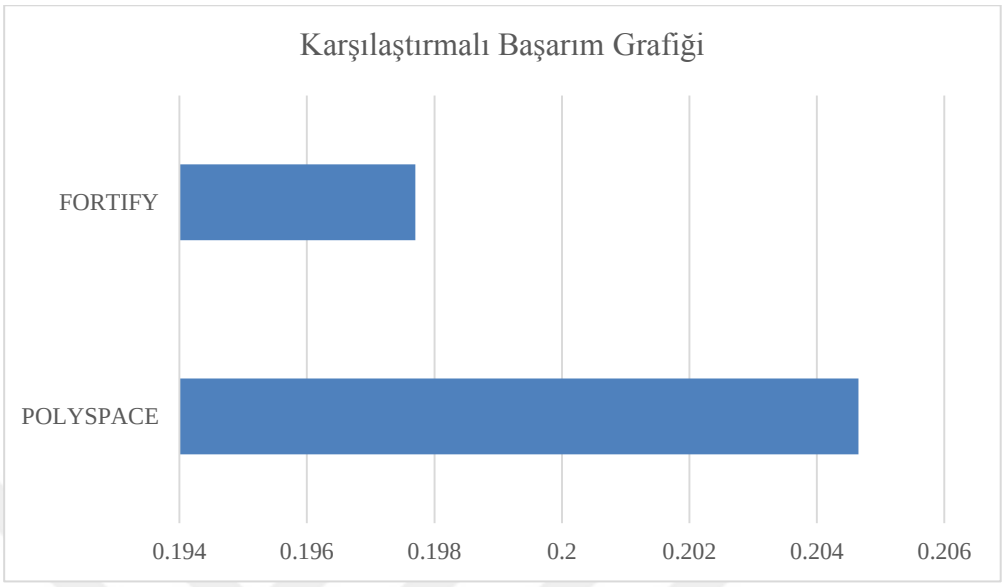
Yanlış negatif (FN) ler, gerçekte mevcut olan güvenlik açıklarını, tespit edememesi ve/veya mevcut olmadıkları şeklinde belirtilmesi anlamına gelmektedir. Bu durum prestij, maddi kayıp ve yerine göre can kayıplarına sebep olabilmektedir. İlgili yanlışların görmezden gelinmemesi ve çok ciddiye alınması gerekmektedir.

Kritik olan yanlık negatif (FN), doğru pozitif (TP) ve doğruluk (Accuracy) başlıklarında Polyspace üstün gelmiştir. Fortify ise, bsadece zaman kayıpları ile ilgili olan yanlış pozitif (FP) başlığında üstün gelmiştir.

Tablo 4.4 Karşılaştırmalı karışıklık matrisi

Sıra	CWE Kodu	Test Adeti	POLYSPACE					FORTIFY				
			FP	FN	TN	TP	Acc	FP	FN	TN	TP	Acc
1	CWE15	48	9	17	-	0	**0**	74	3	-	44	**0.364**
2	CWE23	2400	3524	1321	-	444	**0.084**	600	1800	-	0	**0**
3	CWE78	5796	24867	366	-	1312	**0.049**	1218	4578	-	0	**0**
4	CWE114	576	522	128	-	327	**0.335**	708	0	-	576	**0.449**
5	CWE124	2048	3551	0	-	385	**0.098**	2447	1020	-	0	**0**
6	CWE126	1452	2827	0	-	0	**0**	1375	1147	-	0	**0**
7	CWE127	2048	6524	0	-	0	**0**	3153	1020	-	0	**0**
8	CWE134	2880	3078	704	-	0	**0**	4332	0	-	2880	**0.399**
9	CWE190	3960	21716	1347	-	1074	**0.044**	17641	1482	-	0	**0**
10	CWE195	1152	3392	0	-	1152	**0.254**	2583	129	-	1085	**0.286**
11	CWE226	72	32	24	-	0	**0**	786	0	-	0	**0**
12	CWE242	18	9	0	-	18	**0.667**	0	18	-	0	**0**
13	CWE244	72	32	0	-	72	**0.692**	1230	0	-	72	**0.055**
14	CWE252	630	432	216	-	108	**0.143**	102	450	-	144	**0.207**
15	CWE321	96	18	0	-	0	**0**	274	0	-	0	**0**
16	CWE325	72	68	0	-	0	**0**	186	0	-	0	**0**
17	CWE327	54	942	0	-	0	**0**	765	0	-	18	**0.023**
18	CWE328	54	1095	0	-	0	**0**	0	0	-	54	**1**
19	CWE338	18	8	0	-	18	**0.692**	0	0	-	18	**1**
20	CWE364	18	344	0	-	18	**0.05**	0	18	-	0	**0**
21	CWE367	36	220	0	-	18	**0.076**	0	36	-	0	**0**
22	CWE377	144	1550	0	-	36	**0.023**	0	54	-	144	**0.727**
23	CWE391	54	108	0	-	18	**0.143**	0	54	-	0	**0**
24	CWE398	181	302	49	-	0	**0**	153	127	-	0	**0**
25	CWE401	1658	1165	1017	-	674	**0.236**	1994	632	-	1628	**0.383**
26	CWE404	384	1428	30	-	0	**0**	256	174	-	210	**0.328**
27	CWE415	962	946	121	-	962	**0.474**	322	440	-	500	**0.396**
28	CWE416	459	940	94	-	459	**0.307**	551	148	-	18	**0.025**
29	CWE457	948	3402	4	-	802	**0.191**	1131	558	-	171	**0.092**
30	CWE475	36	16	0	-	36	**0.692**	102	0	-	0	**0**
31	CWE476	348	787	42	-	288	**0.258**	434	96	-	205	**0.279**
32	CWE480	18	27	0	-	0	**0**	36	0	-	0	**0**
33	CWE561	2	0	0	-	2	**1**	0	1	-	1	**0.5**
34	CWE562	3	0	0	-	3	**1**	0	3	-	0	**0**
35	CWE665	193	253	12	-	0	**0**	1	192	-	0	**0**
36	CWE676	18	6	14	-	0	**0**	33	0	-	0	**0**
37	CWE690	960	1396	335	-	638	**0.269**	2395	334	-	0	**0**
38	CWE780	18	8	0	-	0	**0**	0	0	-	18	**1**
	TOPLAM	**29886**	**85544**	**5841**	**-**	**8864**	**0.205**	**44882**	**14514**	**-**	**7786**	**0.198**

Şekil 4.4'te statik analiz araçlarının başarımları değerlendirilmiştir. Grafik, Polyspace'in Fortify'dan daha yüksek başarıma sahip olduğunu göstermektedir.



Şekil 4.4 Karşılaştırmalı başarım grafiği

Tablo 4.5 de Sadece doğru pozitifliğe (TP) sahip güvenlik açıklarının yakalanma yüzdeleri karşılaştırılmıştır. Buna göre, Polyspace var olan pozitiflerin %40.63'ünü doğru tespit edebilmiştir. Fortify ise, var olan pozitiflerin %33.63'ünü doğru tespit edebilmiştir.

Tablo 4.5 Doğru pozitif (TP) oranı

Sıra	CWE Kodu	Test Adeti	POLYSPACE		FORTIFY	
			TP	%	TP	%
1	CWE15	48	0	**0**	44	**91.67**
2	CWE23	2400	444	**18.5**	0	**0**
3	CWE78	5796	1312	**22.64**	0	**0**
4	CWE114	576	327	**56.77**	576	**100**
5	CWE124	2048	385	**18.8**	0	**0**
6	CWE126	1452	0	**0**	0	**0**
7	CWE127	2048	0	**0**	0	**0**
8	CWE134	2880	0	**0**	2880	**100**
9	CWE190	3960	1074	**27.12**	0	**0**
10	CWE195	1152	1152	**100**	1085	**94.18**
11	CWE226	72	0	**0**	0	**0**
12	CWE242	18	18	**100**	0	**0**
13	CWE244	72	72	**100**	72	**100**
14	CWE252	630	108	**17.14**	144	**22.86**
15	CWE321	96	0	**0**	0	**0**
16	CWE325	72	0	**0**	0	**0**
17	CWE327	54	0	**0**	18	**33.33**
18	CWE328	54	0	**0**	54	**100**
19	CWE338	18	18	**100**	18	**100**
20	CWE364	18	18	**100**	0	**0**
21	CWE367	36	18	**50**	0	**0**
22	CWE377	144	36	**25**	144	**100**
23	CWE391	54	18	**33.33**	0	**0**
24	CWE398	181	0	**0**	0	**0**
25	CWE401	1658	674	**40.65**	1628	**98.19**
26	CWE404	384	0	**0**	210	**54.69**
27	CWE415	962	962	**100**	500	**51.98**
28	CWE416	459	459	**100**	18	**3.922**
29	CWE457	948	802	**84.6**	171	**18.04**
30	CWE475	36	36	**100**	0	**0**
31	CWE476	348	288	**82.76**	205	**58.91**
32	CWE480	18	0	**0**	0	**0**
33	CWE561	2	2	**100**	1	**50**
34	CWE562	3	3	**100**	0	**0**
35	CWE665	193	0	**0**	0	**0**
36	CWE676	18	0	**0**	0	**0**
37	CWE690	960	638	**66.46**	0	**0**
38	CWE780	18	0	**0**	18	**100**
	TOPLAM	**29886**	**8864**	**40.63**	**7786**	**33.63**

Statik analiz araçlarının hata miktarı yanlış negatif (FN) ve yanlış pozitif (FP) değerlerinin toplamı şeklindedir. JULIET ölçeğine göre statik analiz araçları arasında toplam 29.886 adet ortak test yapılmıştır.

Tablo 4.6 de statik analiz araçlarına ait hata yapma adetleri listelenmiştir. Buna göre Polyspace 91.385 adet ile en yüksek hata yapma oranına sahip iken Fortify 59.396 adet hata yapmıştır. Fortify bu açıdan Polyspace'e göre üstünlük göstermiştir.

Tablo 4.6 Statik analiz araçlarının hata miktarı

	Test Adedi	**POLYSPACE**		**FORTIFY**	
		FP	**FN**	**FP**	**FN**
		85544	5841	44882	14514
TOPLAM	**29886**	**91385**		**59396**	

Tablo 4.4 içerisnde yer alan bazı CWE kodlarını karşılaştırıldığında aslında her bir aracın diğerine farklı üstünlükleri olduğu görülmektedir.

CWE114 koduna ait toplam 576 adet test mevcuttur. Fortify 576 adedini de doğru tespit ederken, 708 adet yanlış tespitte bulunmuştur. Polyspace ise 327 adedini doğru tespit ederken, 522 adet yanlış tespitte bulunmuştur.

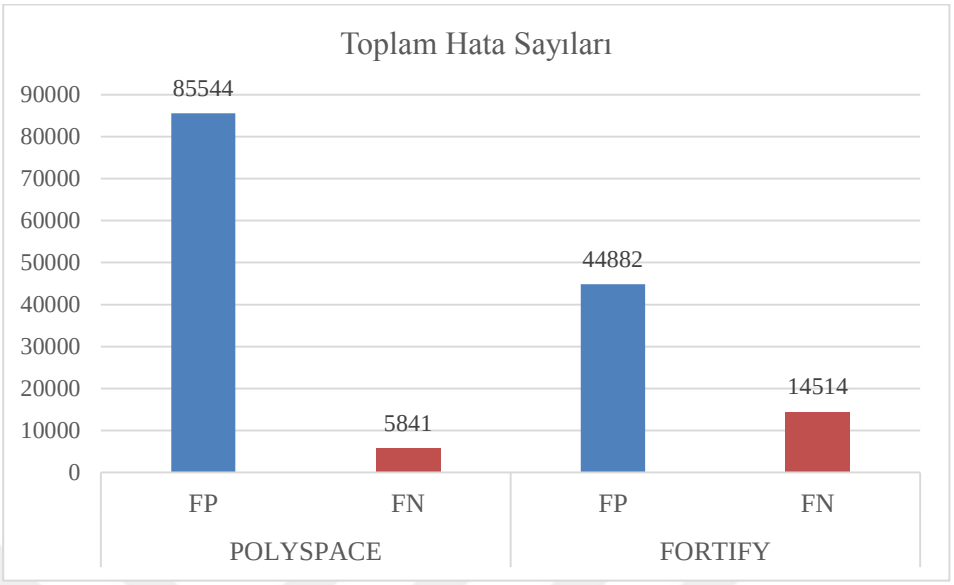
CWE195 koduna ait toplam 1.152 adet test mevcuttur. Polyspace 1.152 adedini de doğru tespit ederken, 3.392 adet yanlış tespitte bulunmuştur. Fortify ise 1085 adedini doğru tespit ederken, 2.583 adet yanlış tespitte bulunmuştur.

CWE244 koduna ait toplam 72 adet test mevcuttur. Her iki statik analiz aracı da 72 adet doğru tespitte bulunmuştur. Polyspace 32 adet ek yanlış tespitte bulunduğunu, Fortify ise 1.230 adet ek yanlış tespitte bulunduğunu iletmişlerdir.

CWE398 koduna ait toplam 181 adet test mevcuttur. Her iki statik analiz aracı da doğru tespitte bulunamamıştır. Polyspace 302 adet yanlış tespitte bulunurken, 49 adette hiç tespit edememiştir. Fortify ise 153 adet yanlış tespitte bulunduğunu iletirken, 127 adet hiç tespit edememiştir.

Şekil 4.5'de statik analiz araçlarının hata yapma sayıları; yanlış pozitif (FP) ve yanlış negatif (FN) değrerleri kıyaslanmıştır. Daha önce açıklandığı üzere, yanlış pozitif (FP) olmayan güvenlik açığının varmış gibi ve/veya yanlış sınıflandırılması anlamına gelirken, yanlış negatif (FN) güvenlik açığı olduğu halde yokmuş ve/veya tespit edilememesi anlamına gelmektedir. İlgili grafik Polyspace'in yüksek yanlış pozitif (FP) ve düşük yanlış negatif (FN) ürettiğini, Fortify'ın ise düşük yanlış pozitif

(FP) üretirken yüksek yanlış negatif (FN) ürettiği göstermektedir.



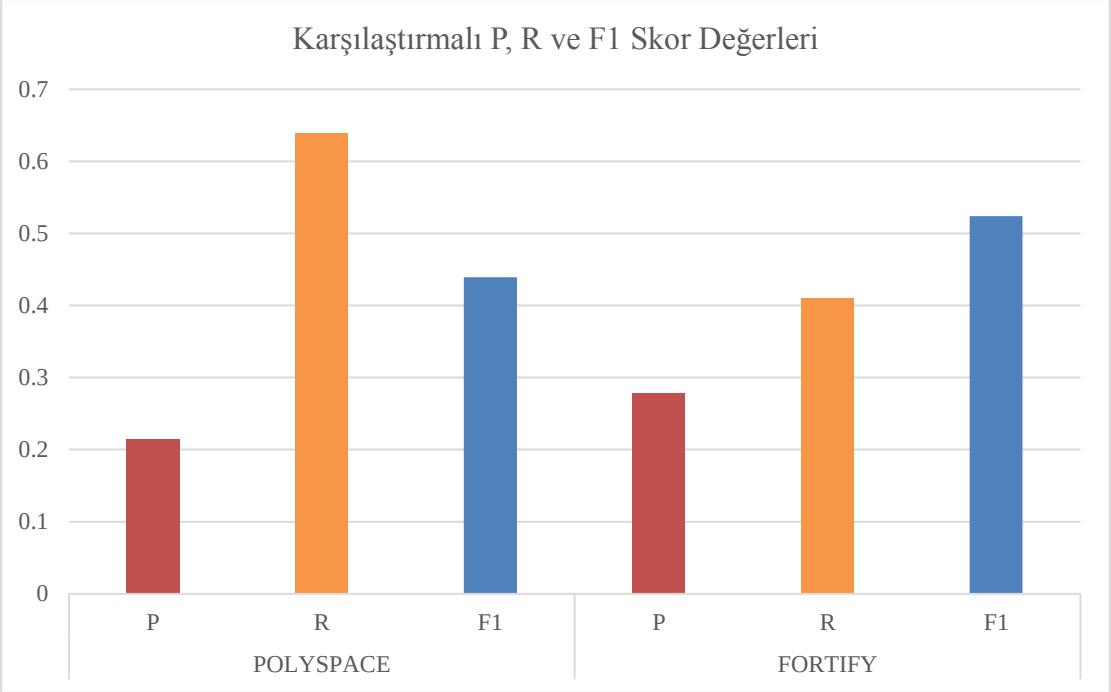
Şekil 4.5 Toplam hata sayıları

Tablo 4.7'de istatistiksel hesaplamalar hassaslık (Precision), anımsama (Recall) ve F1-skor kullanılarak, statik analiz araçlarının, var olan güvenlik açıklarını ne kadar doğru tespit edebildikleri, güvenlik açığı olarak tespit edilenlerin ne kadarının doğru olduğu ve gerçekleştirilen testlerin doğruluk oranı hesaplanmıştır.

Tablo 4.7 Statik test araçlarının performans tablosu

Sıra	CWE Kodu	Test Adedi	POLYSPACE			FORTIFY		
			P	R	F1	P	R	F1
1	CWE15	48	0	0	#N/A	0.373	0.936	0.533
2	CWE23	2400	0.112	0.252	0.155	0	0	#N/A
3	CWE78	5796	0.05	0.782	0.094	0	0	#N/A
4	CWE114	576	0.385	0.719	0.502	0.449	1	0.619
5	CWE124	2048	0.098	1	0.178	0	0	#N/A
6	CWE126	1452	0	#N/A	#N/A	0	0	#N/A
7	CWE127	2048	0	#N/A	#N/A	0	0	#N/A
8	CWE134	2880	0	0	#N/A	0.399	1	0.571
9	CWE190	3960	0.047	0.444	0.085	0	0	#N/A
10	CWE195	1152	0.254	1	0.404	0.296	0.894	0.444
11	CWE226	72	0	0	#N/A	0	#N/A	#N/A
12	CWE242	18	0.667	1	0.8	#N/A	0	#N/A
13	CWE244	72	0.692	1	0.818	0.055	1	0.105
14	CWE252	630	0.2	0.333	0.25	0.585	0.242	0.343
15	CWE321	96	0	#N/A	#N/A	0	#N/A	#N/A
16	CWE325	72	0	#N/A	#N/A	0	#N/A	#N/A
17	CWE327	54	0	#N/A	#N/A	0.023	1	0.045
18	CWE328	54	0	#N/A	#N/A	1	1	1
19	CWE338	18	0.692	1	0.818	1	1	1
20	CWE364	18	0.05	1	0.095	#N/A	0	#N/A
21	CWE367	36	0.076	1	0.141	#N/A	0	#N/A
22	CWE377	144	0.023	1	0.044	1	0.727	0.842
23	CWE391	54	0.143	1	0.25	#N/A	0	#N/A
24	CWE398	181	0	0	#N/A	0	0	#N/A
25	CWE401	1658	0.367	0.399	0.382	0.449	0.72	0.554
26	CWE404	384	0	0	#N/A	0.451	0.547	0.494
27	CWE415	962	0.504	0.888	0.643	0.608	0.532	0.568
28	CWE416	459	0.328	0.83	0.47	0.032	0.108	0.049
29	CWE457	948	0.191	0.995	0.32	0.131	0.235	0.168
30	CWE475	36	0.692	1	0.818	0	#N/A	#N/A
31	CWE476	348	0.268	0.873	0.41	0.321	0.681	0.436
32	CWE480	18	0	#N/A	#N/A	0	#N/A	#N/A
33	CWE561	2	1	1	1	1	0.5	0.667
34	CWE562	3	1	1	1	#N/A	0	#N/A
35	CWE665	193	0	0	#N/A	0	0	#N/A
36	CWE676	18	0	0	#N/A	0	#N/A	#N/A
37	CWE690	960	0.314	0.656	0.424	0	0	#N/A
38	CWE780	18	0	#N/A	#N/A	1	1	1
	ORTALAMA		**0.215**	**0.639**	**0.439**	**0.278**	**0.41**	**0.524**

Şekil 4.6'da statik analiz araçlarının, hassaslık (precision), anımsama (recall) ve harmonik ortalama (F1-score) değerleri ile performans analizi yapılmıştır. Polyspace hassaslık (precision) ve harmonik ortalama (F1-score) olarak Fortify'ın gerisinde kalmıştır. Ancak anımsama (recall) konusunda ise Fortify'dan üstündür. Harmonik ortalama (F1-score) değerlerine bakıldığında ise Fortify'ın daha yüksek başarı sergilediği görülmektedir.



Şekil 4.6 Karşılaştırmalı P, R ve F1 Skor değerleri

5. BULGULAR, TARTIŞMA ve İLERİ ÇALIŞMALAR

Gerçekleştirilen tez çalışması içerisinde başkaları tarafından geliştirilen yazılımlar üzerinde bir güvenilirlik seviyesi hesaplama yöntemi önerilmiştir. Önerilen yöntem ile "güvenlik sorunu var" veya "yok" olarak etiketler elde etmek mümkündür. Bir statik analiz aracının başarım seviyesinin ölçüm kriterleri ele alınmıştır.

Bu bağlamda uygulama güvenliği alanında hizmet verilen kurumlarda yapılan kanıta dayalı konsept çalışmalarında yanlış bir yöntem izlendiği anlaşılmıştır. Çünkü statik analiz araçlarının test süreçlerinde ilgili kurumlar tarafından geliştirilmiş yazılımlar ölçek olarak kullanılmaktadır. Bu da doğruluğu kanıtlanmamış bir ölçek ile ilerlenildiğini göstermektedir. Statik analiz araçları arasında aynı programın ölçek olarak kullanılması dahi gerçekçi sonuç üretilmemesine neden olmaktadır. Ya da hangi aracın yanlış negatif (FN) ya da yanlış pozitif (FP) ürettiği anlaşılamaz.

Çalışmada statik analiz araçlarının karışıklık matrisini çıkartma amacıyla geliştirilen SCATPA uygulamasının yüksek zaman isterine sahip olduğu görülmüştür. Seçilen statik analiz araçlarına ait işlenebilir rapor formatları farklılık gösterdiğinden zaman gereksinimi yükselmektedir. Polyspace aracına ait ortalama bir tarama 40dk içerisinde gerçekleştirilmektedir. Polyspace ise kendi içindeki raporu yaklaşık 10-12 saatte üretmektedir. Zaman isterinin düşürülmesi adına metinsel formatın kullanımının html/xml formatlarının ReportGenerator adındaki rapor üretme aracının kullanılması avantaj sağlayacaktır.

İlerleyen çalışmalarda, birden fazla programlama dilinin ve farklı test araçları incelemelere dâhil edilmesi faydalı olacaktır. Lisanslı test araçları da satın alınarak geniş kapsamlı bir araştırma gerçekleştirilebilir. Ancak bilinen ve tekrar edilebilen bir ölçek oluşturulması zorunluluğu mevcuttur. Farklı standartlara sahip güncel ölçekler çalışmaya dâhil edilerek detay artırılabilir.

Tez çalışması içerisinde ifade edilen mağduriyetlerin yaşanmaması adına önerilen yöntem, süreçler üzerinde denenerek kanıtlanmıştır. Yazılım sektörleri içerisinde çok bilinen bazı statik analiz ürünlerinin aslında eksik/yetersiz noktalarının olduğu, çokça yanlış pozitif (FP) üreterek zaman maliyetlerini yükselttiği ve C/C++

dillerinin derleyicilerinin tamamını desteklemediği gözlemlenmiştir.

Literatürde yer alan G.Diaz ve J.R. Bermejo' na ait çalışma tez çalışmasına benzerlik gösterse de farklı ölçekler kullanmıştır ve çok eski bir çalışmadır. İlgili çalışmaya ait ölçekler günümüz şartlarını kapsamadığından kullanımdan kaldırılmış (deprecated) durumdadır [30]. Aynı zamanda günümüzde artık kullanılmayan statik analiz araçlarının çıktılarına sahiptir [1]. Bu nedenle tez çalışması içerisinde önerilen yöntem ilgili alanda sunulan en güncel yaklaşım ve sonuçları içermektedir.

6. SONUÇ

Bu çalışma içerisinde statik analiz araçlarının performans ölçümüne dair mevcut yöntemlerin kıyaslanması ve performans analizlerini içermektedir. İlgili kıyaslama ve performans analizlerinin ortaya konmasında güncel bir veri seti kullanılmıştır.

Kullanılan yöntem gerçek (TP) ve yanlış pozitif (FP) oranlarının kullanımını içermektedir. Güvenlik açıklarının kapsamı derecelendirilmiş, statik analiz araçlarının performansının hesaplanması için yaygın olarak bilinen ölçütleri esas alınmıştır. Böylelikle bir şirketin SAMATE test aracı üzerinde elde edilen hassasiyet (Precision), anımsama (Recall), F-skor ve güvenlik açığı ölçümlerini analiz ederek kendisine en uygun aracı bulması mümkün hale getirilmiştir.

Bu çalışma, bilinen ticari aracın (Fortify) diğer analiz edilen araçtan (Polyspace) daha iyi bir performans, kullanılabilirlik ve güvenlik seviyesine sahip olduğunu nesnel olarak göstermektedir. Bununla birlikte, analiz edilen tüm araçlar farklı güvenlik açığı türleri için farklı sonuçlar üretmektedir.

Bu araçların çoğunun bir yazılım üzerinde uygulanması güvenilir sonuçlar elde etmek için yeterli değildir ve ham sonuçlar (ilk uygulamanın sonuçları) mutlaka gözden geçirilmelidir. Elbette araçların otomatik olarak çalıştırılması, özellikle uzun kodların analizleri için zorlu bir ilk adım içermesine karşın yeterli değildir. Sonuçları deneyimli bir kullanıcı veya ekip (güvenlik becerileri ve hedef kodda kullanılan dilde deneyim) ile dikkatli bir şekilde analiz etmek her zaman gereklidir.

Güvenlik açıklarını aramak için statik kaynak kod analizi için araçların herhangi bir geliştirme organizasyonunun güvenlik politikasını benimsemesi mümkündür. Ancak araçların iç tasarımları ve raporlama çıktı formatları farklıdır, bu nedenle önemli ölçüde farklı sonuçlar üretmektedir.

Kapsamlı bir güvenlik analizi gerçekleştirmek için güvenlik açıkları ve dil kapsamı artırılmalı, Doğru (TP) ve yanlış (FP) pozitiflik oranları özellikle ele alınmalıdır.

Farklı tasarımlar ve farklı algılama algoritmaları / keşif yöntemleri ile çeşitli araçların kullanılması, bir projenin gerçek analizini yaparken kapsamlı sonuçlar elde edilmesini sağlayabilir. Bununla birlikte, bazı ticari araçların yüksek fiyatları nedeniyle bu çözüm yalnızca profesyonel geliştirme ekipleri için kullanılabilirdir.

Statik kaynak kodu güvenlik analizörlerinin performansını objektif olarak değerlendirmek ve karşılaştırmak için bir ölçüt olarak SAMATE testlerinin kullanılması teşvik edilmelidir. Belli bir proje için hangi aracın kullanılacağına karar verirken, SAMATE araçları güvenilir bir başlangıç noktası olarak kullanılabilir. SAMATE'nin yeni paketler eklenerek içerdiği güvenlik açıklarına ait tanımları genişletmek mümkündür.

Kapsamlı güvenlik açıklarına dayanan araçların yürütülmesinin sonuçları için ortak bir çıktı biçimi tanımının oluşturulması, statik analiz araçlarının daha iyi ve daha güvenilir şekilde karşılaştırmalarına olanak tanımaktadır. Aynı zamanda her temel sorun için rapor sayısını normalize eden sonuç modları eklemek de faydalı olacaktır. Böylelikle bir işlevde arabellek taşması sorunu ortaya çıkarsa, tüm araçlar, farklı sonuçları ele almak yerine tek bir sorunla ilgilenebilirler.

Güvenilir bir endüstrinin sağlamlaştırılmasının temellerinden biri olarak statik analiz araçlarınının karşılaştırılması gerekmektedir.

Yazılımın güvenilirliğinin yükseltilmesi için çalışmalar yoğun bir biçimde sürdürülmektedir. Bu çalışmaların bazıları, örneğin, yazılımın güvenilirliğini değerlendirmektir. Benzer bir biçimde SAMATE'ye dayanan çalışmalar yapılması mümkündür.

Geliştirme aşamasında statik kod aracının daha iyi anlaşılması, nihai sürümden önce daha iyi bir güvenlik denetiminin gerçekleştirilmesine yardımcı olacaktır.

7. KAYNAKLAR

[1] G. Diaz and J.R. Bermejo (2013). Static analysis of source code security: assessment of tools against SAMATE tests. *Information and Software Technology*. 55 (2013) 1462-1476

[2] Katerina Goseva-Popstojanova and Andrei Perhinschi (2015). On the capability of static code analysis to detect security vulnerabilities. *Information and Software Technology*. 68 (2015) 18–33

[3] SAMATE C/C++. Erişim Tarihi: 9 Şubat 2018, **https://samate.nist.gov/SARD/view.php?tsID=45**

[4] Sarah Heckman and Laurie Williams (2010). A systematic literature review of actionable alert identification techniquesfor automated static code analysis. *Information and Software Technology*. 53 (2011) 363–387

[5] Lwin Khin Shar and Hee Beng Kuan Tan (2013). Predicting SQL injection and cross site scripting vulnerabilities through mining input sanitization patterns. *Information and Software Technology*. 55 (2013) 1767–1780

[6] Maryam Mouzarani and Babak Sadeghiyan (2016). Towards designing an extendable vulnerability detection method for executable codes. *Information and Software Technology*. 80 (2016) 231–244

[7] TEZ Çalışma Çıktıları, Tolun_SourcesScriptsSCATPAandResults.zip. Erişim Tarihi: 14 Ocak 2019, **https://drive.google.com/file/d/1DDg3wTXvTWhYqI_9sv2hdm3eLogeeWTm**

[8] Count Lines of Code (cloc) v1.80. Erişim Tarihi: 13 Temmuz 2018, **https://github.com/AlDanial/cloc/releases/tag/v1.80**

[9] Karışıklık Matrisi. Erişim Tarihi: 2 Ocak 2018, **https://veribilimcisi.com/2017/07/18/karisiklik-matrisi-nedir**

[10] SAMATE Software Assurance Metrics And Tool Evaluation. Erişim Tarihi: 04 Kasım 2018, **https://samate.nist.gov/SRD**

[11] Introduction to SAMATE. Erişim Tarihi: 9 Şubat 2018, **https://samate.nist.gov/index.php/Introduction_to_SAMATE.html**

[12] JULIET Documents. Erişim Tarihi: 9 Şubat 2018, **https://samate.nist.gov/SRD/around.php#juliet_documents**

[13] JULIET 1.3 C/C++. Erişim Tarihi: 9 Şubat 2018, **https://samate.nist.gov/SARD/view.php?tsID=108**

[14] CWE nedir? CWE ne yapar?. Erişim Tarihi: 20 Mayıs 2018, **https://www.checkmarx.com/glossary/cve-2/**

[15] A Community Developed List of Software Common Weakness Enumerations (CWE). Erişim Tarihi: 04 Ocak 2018, **https://cwe.mitre.org/data/index.html**

[16] Chess B. and West J. (2007). Secure Programming with static analysis. *AddisonWesley Software Security Series*

[17] Martin R. A. and Barnum S. (2008). Creating the Secure Software Testing Target List. *CSIIRW, Vol. 288, article no. 33*

[18] JULIET 1.3 Test Suite: Changes. Erişim Tarihi: 15 Haziran 2018, **https://www.nist.gov/publications/juliet-13-test-suite-changes-12**

[19] Center for Assured Software National Security Agency (2012). JULIET Test Suite v1.2 for C/C++ User Guide. Erişim Tarihi: 13 Temmuz 2018, **https://github.com/AlDanial/cloc/releases/https://samate.nist.gov/SRD/resources/Juliet_Test_Suite_v1.2_for_C_Cpp_-_User_Guide.pdf**

[20] Center for Assured Software National Security Agency (2012). Juliet 1.3 Test Suite: Changes From 1.2. Erişim Tarihi: 13 Temmuz 2018, **https://samate.nist.gov/SRD/resources/Juliet_1.3_Changes_From_1.2.pdf**

[21] MathWorks Polyspace BugFinder v2.5 Aracı. Erişim Tarihi: 06 Kasım 2017, **https://www.mathworks.com/products/polyspace-bug-finder.html**

[22] HPe Fortify Static Kaynak Kod Analiz Aracı. Erişim Tarihi: 18 Eylül 2017, **https://www.hpe.com/mysoftware/index**

[23] SonarCloud (open) desteklenen programlama dilleri. Erişim Tarihi: 7 Ocak 2019, **https://www.sonarsource.com/products/codeanalyzers/**

[24] SonarSource ürün listesi. Erişim Tarihi: 8 Ocak 2019, **https://www.sonarsource.com/**

[25] Checkmarx Statik Analiz Güvenlik aracı. Erişim Tarihi: 10 Ocak 2019, **https://www.checkmarx.com/products/static-application-security-testing/**

[26] Checkmarx desteklenen programlama dilleri. Erişim Tarihi: 10 Ocak 2019, **https://www.checkmarx.com/technology/supported-coding-languages/**

[27] IBM AppScan ürün listesi. Erişim Tarihi: 12 Ocak 2019, **https://www.ibm.com/security/application-security/appscan**

[28] IBM AppScan Source desteklenen programlama dilleri. Erişim Tarihi: 12 Ocak 2019, **https://www.ibm.com/support/knowledgecenter/en/SSS9LM_9.0.3/com.ibm.rational.appscansrc.install.doc/topics/system_requirements_language_support.html**

[29] AttackFlow ürün listesi ve desteklediği programlama dilleri. Erişim Tarihi: 14 Ocak 2019, **https://attackflow.com/**

[30] SARD Manual: What the Test Case Status Means. Erişim Tarihi: 9 Şubat 2018, **https://samate.nist.gov/SARD_Manual_TC_Status.html**

[31] SAMATE Software Assurance Metrics And Tool Evaluation. Erişim Tarihi: 04 Kasım 2017, **https://samate.nist.gov/**

[32] EBSCOHOST online kütüphane veritabanı. Erişim Tarihi: 04 Kasım 2017, **https://search.ebscohost.com/**

[33] SCIENCEDIRECT online kütüphane veritabanı. Erişim Tarihi: 04 Kasım 2017, **https://www.sciencedirect.com/**

[34] Supported CWE Standarts in MathWorks Polyspace Bug Finder v2.5. Erişim Tarihi: 06 Kasım 2017, **https://www.mathworks.com/help/releases/R2018a/bugfinder/ug/cwe-and-polyspace-results.html**

[35] Supported CWE Standarts in MicroFocus Fortify SCA v18.10. Erişim Tarihi: 01 Mart 2018, **C:\Program Files\Fortify\Fortify_SCA_and_Apps_18.10\Core\config\ExternalMetadata\externalmetadata.xml**

[36] Supported CWE Standarts in SonarSource SonarCloud (Open). Erişim Tarihi: 08 Mart 2018, **https://rules.sonarsource.com/c/tag/cwe**

[37] Microsoft Windows 7 64bit. Erişim Tarihi: 20 Eylül 2018, **https://www.microsoft.com/en-us/software-download/windows7**

8. ÖZGEÇMİŞ

1980 yılında Almanya'da doğdu. Babasının öğretmen olması nedeniyle, İlköğrenimini Kayseri (1991) ve Mardin (1995) de, orta öğrenimini Antalya Teknik Lisesi Bilgisayar Donanım (1999) bölümünde, yükseköğrenimini Kırıkkale Üniversitesi Bilgisayar Programlama (2003) programında ve Girne Amerikan Üniversitesi Bilgisayar Mühendisliği (2007) bölümünde tamamlamıştır.

Orta öğrenim hayatından başlamak üzere, yarı ve tam zamanlı yaklaşık yirmibeş (25) yıllık bilişim sektörü deneyimleri olmuştur. Önde gelen kurum ve kuruluşların "Yazılım Güvenliği" testleri gerçekleştirerek raporlar hazırlamıştır.

Tolun ARDAHANLI – Almanya (1980)
Gsm : (533) 544 44 46
E-mail : tolun.ardahanli@gmail.com

Eğitim:

Yüksek Lisans: Haliç Üniversitesi, Bilgisayar Mühendisliği
Lisans : Girne Amerikan Üniversitesi, Bilgisayar Mühendisliği
Ön Lisans : Kırıkkale Üniversitesi, Bilgisayar Programlama
Lise : Antalya Teknik Lisesi, Bilgisayar Donanım

Projeler:

Yıl	Projeler	Kullanılan teknolojiler
2003 - 2018	• Yüksek Lisans Öğrenci işleri otomasyonu • Yazılım kurulum ve kaldırım programı • Araç kontrol sistemi • Tank otomasyonu • Görme engellilere klavye okuyan virüs • Yazıcı-Kamera entegrasyon donanım ve yazılımı • E-posta kriptolojik fonksiyon geliştirme • Online dökümantasyon sistemi • Soğuk hava depoları uzaktan kontrol sistemi • Teknolojiler arası entegrasyon • Web servisi ve Web API geliştirilmesi • Bilgisayar-Telefon Entegrasyonu (CTI)	Delphi, Interbase, C#.Net, Java Script, jQuery,SQL Server, Asp.Net MVC, Pascal, Web Servis, ADO.NET, Entity Framework, LINQ, MS SQL, My SQL, ORACLE Linux, Gömülü Linux, Win CE, MS SQL CE, ARM

Yabancı Dil:

İngilizce: Okuma: Çok İyi, Yazma: Çok İyi, Anlama: Çok İyi, Konuşma: İyi
Almanca: Okuma: İyi, Yazma: Temel, Anlama: İyi, Konuşma: Temel

İlgi Alanlarım:

Yazılım geliştirme süreç yönetim teknikleri, Yazılım ve Sistem Güvenliği.

Turnitin Orijinallik Raporu

KAYNAK KOD GÜVENLİĞİND STATİK ANALİZ ÖLÇÜTLER İLE PERFORMANS DEĞERLENDİRMESİ Yazar Tolun Ardahanlı

Gönderim Tarihi: 05-Şub-2019 09:15AM (UTC+0300)
Gönderim Numarası: 1073269604
Kelime sayısı: 13100

Benzerlik Endeksi:
%10
Kaynağa Göre Benzerlik:
N/A
Internet Kaynakları:
%9
Yayınlar:
%4
Öğrenci Ödevleri:
%2

Kaynaklar:

1. %3 benzerlik
   citeseerx.ist.psu.edu
   İnternet Kaynağı

2. %1 benzerlik
   samate.nist.gov
   İnternet Kaynağı

3. %1 benzerlik
   www.grammatech.com
   İnternet Kaynağı

4. < %1 benzerlik
   cwe.mitre.org
   İnternet Kaynağı

5. < %1 benzerlik
   www.navigator44.ru
   İnternet Kaynağı

6. < %1 benzerlik
   Submitted to Haliç Üniversitesi
   Öğrenci Ödevi

7. < %1 benzerlik
   Katerina Goseva-Popstojanova, Andrei Perhinschi. "On the capability of static code analysis to detect security vulnerabilities", Information and Software Technology, 2015
   Yayın