ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL

PROFILING DEVELOPERS TO PREDICT VULNERABLE CODE CHANGES

M.Sc. THESIS

Tuğçe COŞKUN

Department of Computer Engineering

Computer Engineering Programme

JULY 2023

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL

PROFILING DEVELOPERS TO PREDICT VULNERABLE CODE CHANGES

M.Sc. THESIS

Tuğçe COŞKUN
(504201540)

Department of Computer Engineering

Computer Engineering Programme

Thesis Advisor: Assoc. Prof. Dr. Ayşe TOSUN KÜHN

JULY 2023

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

GÜVENLİK AÇIĞI KOD DEĞİŞİKLİKLERİNİ ÖNGÖRMEK İÇİN GELİŞTİRİCİLERİN PROFİLİNİ OLUŞTURMA

YÜKSEK LİSANS TEZİ

Tuğçe COŞKUN
(504201540)

İbilgisayar Mühendisliği Anabilim Dalı

Bilgisayar Mühendisliği Programı

Tez Danışmanı: Doç. Dr. Ayşe TOSUN KÜHN

TEMMUZ 2023

Tuğçe COŞKUN, a M.Sc. student of ITU Graduate School student ID 504201540 successfully defended the thesis entitled "PROFILING DEVELOPERS TO PREDICT VULNERABLE CODE CHANGES", which he/she prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor :	**Assoc. Prof. Dr. Ayşe TOSUN KÜHN** Istanbul Technical University	.....
Jury Members :	**Assoc. Prof. Dr. Yusuf YASLAN** Istanbul Technical University	.....
	Prof. Dr. Mehmet Bülent ÖRENCİK Istanbul Technical University	.....
		.....

Date of Submission : **17 June 2023**
Date of Defense : **20 July 2023**

To my mother,

FOREWORD

First of all, I want to take this opportunity to express my gratitude and respect to my advisor, Ayşe Tosun, for her support in my studies. Her career as a woman engineer and her computer science knowledge have been a source of motivation for me to complete my graduate education and write this thesis.

Secondly, I would like to thank my family and my friends for their endless support and effort in every important step in my life. I wouldn't have been able to deal with the challenges in my life without them.

July 2023 Tuğçe COŞKUN

TABLE OF CONTENTS

ABBREVIATIONS

HIN	**:** Heterogeneous Information Network
RWR	**:** Random Walk with Restart
NB	**:** Naive Bayes
SVM	**:** Support Vector Machine
RF	**:** Random Forest
XGB	**:** eXtreme Gradient Boosting
ML	**:** Machine Learning
ASF	**:** Apache Software Foundation
vcs	**:** version control systems
NVD	**:** National Vulnerability Database
CVE ID	**:** Common Vulnerabilities and Exposure Identifier
CWE	**:** Common Weakness Count
PCA	**:** Principal Component Analysis
ICA	**:** Independent Component Analysis
ASD	**:** Agile Software Development
R	**:** Recall
P	**:** Precision
F1	**:** F-measure
IR	**:** Inspection Ratio

SYMBOLS

S_{VD}	: Sub-matrix representing the relationship between Vulnerability and Developer
S_{DV}	: Sub-matrix representing the relationship between Developer and Vulnerability
S_{DD}	: Sub-matrix representing the relationship between Developer and Developer
S_{DP}	: Sub-matrix representing the relationship between Developer and Project
S_{PD}	: Sub-matrix representing the relationship between Project and Developer
T	: Transition Probability Matrix
T_{ij}	: Represents the random walker's likelihood of passing from node i to node j
λ	: Restart Probability
p_0	: Restart Vector
p_a	: Static Probabilities For Each Node

LIST OF TABLES

Page

LIST OF FIGURES

PROFILING DEVELOPERS TO PREDICT VULNERABLE CODE CHANGES

SUMMARY

Recently, researchers and practitioners have been interested in software vulnerability prediction and management. It is also possible to forecast software vulnerabilities using a variety of methods, most of which are based on features of code artifacts. Despite other research yielding encouraging outcomes, the significance of developers in introducing vulnerabilities has yet to be investigated. We characterize developers' vulnerability creation and vulnerability fixing habits in software development projects utilizing Heterogeneous Information Network (HIN) analysis.

The SmartSHARK public dataset was used to estimate vulnerable commits. Four projects with the most actual vulnerability data were chosen, and fix commits associated with security vulnerabilities were identified. The dataset's commitments to projects were used to generate the HIN matrices. Different relationships linking the two object types have different transition likelihoods, resulting in distinct random walk processes and ranking results. Our analysis of the Random Walk and Restart (RWR) algorithm on the HIN revealed that some developers are more often linked to vulnerabilities than others.

This thesis examines how they affect of developer profiles on predicting vulnerable commits and compares them to the code metrics-based approach. To extract all input features, we use the metrics we obtained by applying the RWR algorithm to the HIN and the aggregation of the code metrics. To create the prediction models, we use conventional machine learning methods, including Random Forest (RF), Naive Bayes (NB), and eXtreme Gradient Boosting (XGB), Support Vector Machine (SVM). We employed recall, accuracy, F-measure, and inspection rate to assess the efficacy of several classifying algorithms. In this thesis, Friedman and Nemenyi tests were used to establish that the differences across the models we created in terms of performance indicators were statistically significant.

We report our empirical study that we conducted to anticipate vulnerability-inducing changes across four Apache projects. In terms of recall, the technique focused on code metrics is 90% effective, as opposed to the technique focused on developer behavior profiling is 71% effective. We achieve a success rate of 89% by combining the feature sets generated by the two strategies. The findings demonstrated that the coding practices of developers may be beneficial for predicting vulnerabilities.

GÜVENLİK AÇIĞI KOD DEĞİŞİKLİKLERİNİ ÖNGÖRMEK İÇİN GELİŞTİRİCİLERİN PROFİLİNİ OLUŞTURMA

ÖZET

Yazılım güvenlik açığı tahmini ve yönetimi, son zamanlarda araştırmacıların ve uygulayıcıların ilgisini çekmiştir. Yazılım güvenlik açıklarını tahmin etmek için genellikle kod eserlerinin özelliklerine dayanan çeşitli teknikler de sunulur. Bu çalışmalar umut verici sonuçlar elde ederken, insan faktörlerinin güvenlik açıklarını tetiklemede oynadığı ana rol henüz çalışılmamıştır. Bu tezde güvenlik açıkları merceğinden geliştiricilerin profili çıkardık. Ayrıca, güvenlik açığı oluşturan taahhütleri tahmin etmede geliştirici profillerinin etkisini araştırdık ve bulguları kod ölçümlerine dayalı yaklaşımla karşılaştırdık. Bildiğimiz kadarıyla ilk kez biz güvenlik açıklarına göre geliştirici profili oluşturduk ve oluşturduğumuz bu geliştirici profilinin güvenlik açığı tahmin modellerindeki etkisini inceledik.

Yazılım güvenlik açıkları, kuruluşların itibarı üzerindeki negatif etkileri, yama oluşturma maliyetleri ve kullanıcı gizliliğinin ihlaline sebep olmasına rağmen yazılım araştırmacılarının ve uygulayıcılarının ilgisini çeker. Araştırmalar, yazılım geliştirme yaşam döngüsünde birçok teknik borç türü olduğunu gösteriyor. Teknik borcun birikmiş maliyeti, yazılım kalitesi üzerinde olumsuz bir etkiye sahiptir ve kusur olasılığını artırır. Yazılım sistemlerinde, ölçülemeyen ve geri ödenemez borçlar kusurlarla bir araya geldiğinde, teknik borç güvenlik açıklarına neden olabilir. Önceki araştırmalar, güvenlik açıkları ile tasarım kusurları ve kod karmaşası gibi teknik borç göstergeleri arasındaki korelasyona dair kanıtlar göstermektedir.

Geliştirici değişikliklerinin hata tahmini üzerinde iyi performans gösterdiği kanıtlanmıştır. Bazı araştırmalar, geliştirici profilini, elde edilen görev sayısına veya tamamlanan/test edilen yazılım modüllerinin sayısına, aktif programlama/test etme uzmanlığına, üretkenlik veya yıl cinsinden uzmanlık düzeyine göre tanımlamıştır. Öte yandan, bazı araştırmalarda belirli bir davranışa, alışkanlığa veya kodlama stiline göre tanımlanır. Ayrıca geliştirici profili, görev atama çalışmalarında ve geliştiricinin ekip çalışması üzerindeki etkisini ölçmek için kullanılmıştır. Çalışmamızda Heterojen Bilgi Ağı (Heterogeneous Information Network) yöntemini kullanarak güvenlik açıkları merceğinden geliştiricilerin profilini çıkardık ve güvenlik açığına neden olan taahhütleri tahmin etmede profil bilgilerinin etkisini inceleyerek sonuçları paylaştık.

Güvenlik açığı oluşturan taahhütleri tahmin etmek için, Java'da geliştirilen Apache Software Foundation deposuna ait 77 projeyi içeren SmartSHARK genel veri kümesini kullandık. Veri setinden çalışmamızla ilgili koleksiyonları belirledik. Veri setindeki MongoDB koleksiyonlarından bazıları çalışmamızla ilgili olmadığı için kaynak veri setinden özel MongoDB betikleri oluşturarak ilgi verileri elde ettik. SmartSHARK, kapsamlı bir veri kaynağı sağlar. Veri kümesi, çekme isteklerini, sorunları, e-postaları

ve taahhütleri içerir. Ancak, tahmin modelini oluşturmak için ihtiyaç duyduğumuz güvenlik açıklarını içermez. Bu nedenle, tüm projeler için manuel olarak analiz ederek Ulusal Güvenlik Açığı Veritabanından (NVD) güvenlik açığı kayıtlarını çıkardık. Ardından, en fazla sayıda gerçek güvenlik açığı verisine sahip dört proje seçtik, bu projeler ActiveMQ, Struts, Nifi ve Tika dır. Ardından, yine NVD den faydalanarak yazılım projelerindeki güvenlik açıklarıyla ilişkili düzeltme taahhütlerini bulduk. Ayrıca, bir güvenlik açığı tahmin modeli oluşturmak için güvenlik açığı sağlayan taahhütleri de bulmalıydık. Bunu ise SZZ algoritmasından faydalanarak yaptık. SZZ algoritması, bir sorun izleme sisteminde rapor edilen bir hatanın ve/veya güvenlik açığının giderildiği taahhütleri analiz ederek, onların hata veya güvenlik açığına neden olan taahhütlerini takip eder. Böylece projelerin güvenlik açıklarına neden olan taahhütlerini de elde ettik. 62 güvenlik açığı düzeltme işlemiyle eşleşen 187 güvenlik açığı tetikleyen taahhüt tespit ettik. Sonuç olarak seçtiğimiz dört proje için SmartSHARK veri setini güvenlik açığı veri seti oluşturarak genişlettik.

Homojen ağların aksine, Heterojen Bilgi Ağında (HBA) iki düğüm birden çok kenarla birbirine bağlanabilir ve bu kenarlar farklı ilişkileri temsil edebilir. HBA, benzerlik ölçümü, sınıflandırma, kümeleme ve sıralama gibi birçok veri madenciliği görevinde farklı alanlarda kullanılmaktadır. HBA analizi yoluyla geliştiricilerin profilini çıkarmak için Proje, Güvenlik Açıkları ve Geliştirici düğümlerinden oluşan bir ağ kurduk. Proje düğümü, analiz için seçilen projeleri temsil eder. Güvenlik Açığı düğümü, bu projelerde tespit edilen güvenlik açığı kayıtlarının CWE kodlu güvenlik açığı türlerini temsil eder. Geliştirici düğümü ise bu projelerdeki geliştiricileri temsil eder. Geliştiriciler tarafından yapılan kod taahhütlerine göre Proje ve Geliştirici düğümleri arasında iki özdeş ilişki tanımladık. Bir geliştiricinin bir güvenlik açığı oluşturabileceği ve/veya düzeltebileceği için Geliştirici ve Güvenlik Açığı düğümleri arasında iki ilişki tanımladık. Son olarak, aynı taahhüt üzerinde birlikte çalıştıkları için Geliştiriciler arasında bir ilişki tanımladık.

Veri setimiz, projelerdeki taahhütler aracılığıyla geliştirici bilgilerine ulaşmamızı sağladığından, HBA'nın matrislerini oluşturmak için tüm bu taahhütleri kullandık. Örneğin, Güvenlik Açığı ve Geliştirici arasındaki ilişki yalnızca güvenlik açığı oluşturan taahhütleri kullanırken, Geliştirici ile Güvenlik Açığı arasındaki ilişki yalnızca güvenlik açığı düzeltme taahhütlerini kullanır. Güvenlik açığı ile Geliştirici arasındaki ilişkiyi temsil eden alt matrisimiz için ağırlığı, bir geliştiricinin bir güvenlik açığı türünü tetiklediği taahhüt sayısına göre tanımladık. Geliştirici ile Güvenlik açığı arasındaki ilişkiyi temsil eden alt matrisimiz için ağırlık, bir geliştiricinin bir güvenlik açığı türünü düzelttiği taahhütlerin sayısıdır. Proje ve Geliştirici arasındaki ilişkiyi temsil eden alt matrisleri oluşturduk ve ilgili projeler için geliştiricilerin tüm taahhüt sayılarını bu alt matrislerde ağırlık olarak kullandık. Son olarak, Geliştirici ve Geliştirici ilişkisini temsil eden alt matrisin ağırlığını 1 olarak belirledik.

İki nesne türünü birbirine bağlayan farklı ilişkilerin farklı geçiş olasılıkları vardır, bu da farklı rastgele yürüyüş süreçleri ve sıralama sonuçlarıyla sonuçlanır. Rastgele Yeniden Başlatmalı Yürüyüş Algoritmasını (Random Walk with Restart), tüm ağın küresel yapısını yakalama yeteneği ile bilindiği için, geliştiriciler ve güvenlik açığı türleri arasındaki yakınlığı tahmin etmek için oluşturduğumuz HBA'nın düğümleri üzerinde kenarlarda gezinmek için uyguladık. Ağdaki bir düğümden rastgele bir

yürüyüşe çıkan rastgele bir sörfçü, her adımda iki seçeneğe sahiptir: (i) $1 - \lambda$ olasılıkla rastgele seçilmiş bir komşu düğüme giden yolu takip etmek; (ii) λ olasılıkla başlangıç düğümüne giden yolu takip etmek. Algoritmada λ yeniden başlatma olasılığı olarak ifade edilir. Bu sayı, algoritmayı kullanan araştırmalarda yaygın olarak 0,7 olarak tanımlanmıştır. Bu nedenle, bu tezde λ değerini 0,7 olarak belirledik. Her alt matrisi bir ayarlama faktörü ile çarparak sütun normalleştirilmiş geçiş olasılığı matrisini elde ettik. Ayarlama faktörü değerini ise 0,5 olarak ayarladık. Algoritmamızda, bir geliştiricinin profilini güvenlik açığıyla ilgili davranışlara göre ölçmeyi amaçladığımız için, *tohum* düğümleri olarak da adlandırılan yeniden başlatma düğümleri olarak güvenlik açığı düğümlerini belirledik. p_0, çekirdek düğümlerin yeniden başlatma vektörünü temsil eden sütun vektörü olsun. Bir geliştiriciyi, farklı güvenlik açıkları türleri için tetikleme davranışına göre değerlendirirken, yeniden başlatma vektörlerinin farklı olacağını belirtmeliyiz. Geliştirici tipi bir düğümün puanı, geliştiricilerin Güvenlik Açığı tipindeki güvenlik açığı türleriyle ilişkili puanıdır.

Elde ettiğimiz puanlar ve sıralamalar, bazı geliştiricilerin güvenlik açıklarıyla diğerlerinden daha fazla ilişkili olduğunu gösterdi. Bazı geliştiricilerin güvenlik açığı oluşturmaya diğerlerinden daha fazla dahil olduğunu gördük. CWE-ID için en yüksek puana sahip geliştirici, diğer geliştiricilere göre daha fazla güvenlik açığı oluşturan taahhütlere sahiptir diyebiliriz. Yaklaşımımızın geçerliliğini değerlendirmek, özellikle geliştiricilere güvenlik açığı yaratan davranışları için atadığımız sıraların doğru olduğunu kanıtlamak zordur. Ayrıca, güvenlik açığı oluşturan taahhütler nadiren gözlemlenir ve geliştirici faaliyetleri genellikle düzensizdir. Alternatif olarak, geliştirici profillerinin güvenlik açığına neden olan taahhütleri tahmin etme üzerindeki etkisini değerlendirmek için yazılım ölçümlerine dayalı başka bir yöntem kullandık. Güvenlik açıklarına neden olan taahhütleri tahmin ederken, geliştirici profilini kullanan yöntemi, kaynak koduna dayalı yazılım ölçümlerini kullanan yöntemle karşılaştırırız. Bunun için projelerdeki karmaşıklık, kapsam ve bağımlılık metriklerini kullandık ve metrikleri veri setinde dosya seviyesinden taahhüt seviyesine kadar topladık. Ayrıca, güvenlik açığı tahmini için aynı model tasarımını kullanarak geliştiricinin profilini temsil eden yeni bir dizi özellik ekledik. Bu çalışmada, yalnızca yazılım kodu metrikleri kullanılarak eğitilen modelleri Yazılım Metrikleri Modeli (YMM), geliştirici profili kullanılarak eğitilen modelleri Geliştirici Profili Modeli (GPM) olarak, her iki girdi özelliği kullanılarak eğitilen modelleri ise Yazılım Metrikleri ve Geliştirici Profili Modeli (YMGPM) olarak tanımladık.

Random Forest (RF), eXtreme Gradient Boost (XGB) ve Support Vector Machine (SVM), Naive Bayes (NB) gibi Makine Öğrenmesi (Machine Learning (ML)) sınıflandırıcılarını kullanarak güvenlik açığı tahmin modellerini eğittik. Farklı sınıflandırıcılar, farklı boyut küçültme yöntemleri ve öznitelik seçimi kullanarak oluşturduğumuz modellerin sonuçlarını karşılaştırdık. Farklı sınıflandırma yöntemlerinin performansını değerlendirmek için geri çağırma, kesinlik, F-ölçümü ve inceleme oranı kullandık. Bu metrikler, karışıklık matrisinden çıkarılır. Karışıklık matrisi, doğru ve yanlış tahmin edilen örneklerin sayısını temsil eder. Bu çalışmada, True Positive (TP), modelin geliştiricinin sistemde bir güvenlik açığı oluşturduğunu ve gerçek etiketin savunmasız bir taahhüt olduğunu öngördüğü durumları temsil eder. Öte yandan, True Negative (TN), hem tahmin edilen hem de gerçek etiketlerin savunmasız olduğu durumları gösterir. Performans ölçüleri açısından modeller arasındaki farkların

istatistiksel olarak anlamlı olduğu sonucuna varmak için Friedman ve Nemenyi testleri de uyguladık ($p < 0.05$).

Eğitim ve test için alt kümelere ayrılan veri setimizde, birden fazla 70%-30% bölme oluşturmak için çeşitli rastgele durum değerleriyle deneyleri 10 kez gerçekleştirdik ve ardından 10 deneme sonucunun ortalamasını aldık. XGB, SVM, RF ve NB sınıflandırma teknikleri, kod taahhütlerini savunmasız ve savunmasız olmayan taahhütler olarak sınıflandırmak için eğittik. Ayrıca öznitelik seçimi ve boyut küçültme yöntemlerinin algoritmaların performansına etkisi de değerlendirdik. Akademik çalışmalarda ML algoritmaları çok sayıda öznitelik veya öznitelikler arasındaki korelasyonlardan olumsuz etkilenebileceğinden, genellikle boyutsallık azaltma yöntemlerine başvurulur. Bu tezde, öznitelik sayısını 10'a indirerek Temel Bileşen Analizi (PCA) ve Bağımsız Bileşen Analizi (ICA) boyutluluk azaltma yöntemlerini kullandık. PCA doğrusal olarak yüksek boyutlu bir öznitelik vektörünü ilişkisiz bileşenlerle düşük boyutlu bir vektöre dönüştürürken, ICA, dönüştürülmüş vektörlerde istatistiksel olarak bağımsız bileşenler elde etmeye çalışır.

YMM'de, NB sınıflandırıcısı, geri çağırma metriği (0.89) açısından diğer sınıflandırıcılardan daha iyi performans gösterdi. Ayrıca, NB, ICA boyutsallık azaltma yöntemi ve en iyi 10 öznitelik seçim yönteminin bir kombinasyonu ile en iyi geri çağırma oranına (0,909) sahiptir. Diğer taraftan IR oranları sırasıyla 0,651 ve 0,698'dir. Bu, bu geri çağırma oranına ulaşmak için tüm taahhütlerin 65%'ini gözden geçirmek için çaba gösterilmesi gerektiği anlamına gelir. GPM sonuçlarına bakıldığında, RF sınıflandırıcının geri çağırma metriği açısından tüm sınıflandırıcılar arasında en iyi performansı gösterdiği görülmektedir. Ancak XGB sınıflandırıcı ve ICA boyutsallık azaltma yönteminin kombinasyonu ile en iyi geri çağırma oranını (0,710) elde ettik. NB sınıflandırıcısı (0.890), YMGPM ile tüm güvenlik açığı tahmin modellerinin sonuçları arasında en iyi geri çağırma oranına sahiptir. Boyut azaltma ve/veya en iyi 10 öznitelik yöntemlerini kullanan sınıflandırıcı kombinasyonlarında, NB sınıflandırıcı ve en iyi 10 öznitelik seçme yönteminin birleşimi en iyi geri çağırma oranına (0,888) sahiptir. Tüm modellerimiz için kesinlik değerleri düşüktür. Belki farklı özelliklerle çok daha başarılı modeller oluşturulabilir. Ancak bu çalışmada amacımız en iyi modeli elde etmek değil, geliştirici profilinin güvenlik açığı tahmini üzerindeki etkisini gözlemlemektir. Sonuçları incelediğimizde, YMGPM modelinin kesinlik değerleri açısından en yüksek sonuçları elde ettiğini söyleyebiliriz. Geliştirici profili tarafından verilen bilgileri kullandığımız GPM modelinin performansının algoritma ve farklı tekniklerle gelişmediğini gözlemledik. GPM'nin YMM'ye göre yeterli olmadığını söyleyebiliriz. Ancak YMGPM modelinde, her iki özellik setinin kombinasyonu ile performansta bir iyileşme gördük. Bu, profil bilgilerinin güvenlik açığına neden olan taahhütlerin tahmin edilmesine katkıda bulunduğunu gösterir.

YMGPM modelini, geliştirici profilinin güvenlik açığı tahmini üzerindeki etkisini görmek için tasarladık. Sınıflandırıcı kombinasyonları, boyut azaltma ve/veya en iyi 10 öznitelik seçme yöntemi ile toplam 24 farklı değerlendirme metriği ile sonuçlar elde ettik. Bu 24 farklı kombinasyonun 11'inde YMGPM, geri çağırma metriği açısından YMM'den daha iyi performans gösterdi. Bu da geliştirici profilinin güvenlik açığı tahmini üzerinde olumlu bir etkisinin olduğunun bir göstergesidir. Bu göstergeler ile geliştiricilerin güvenlik açığı yaratan kodlama davranışlarının güvenlik açığı

tahmininde etkili olabileceğine dair kanıtlar gözlemlediğimizi söyleyerek araştırma sorumuza cevap verebiliriz.

Bu tezde, oluşturduğumuz HBA üzerinde RWR algoritmasını kullanarak güvenlik açıkları açısından geliştirici profili elde ediyoruz. Geliştirici profili oluşturma, geliştirici profilinin güvenlik açığı tahmini üzerindeki etkisi incelenirken farklı şekillerde eklenebilir. Yöntemimiz, güvenlik açığı oluşturma ile ilişkili diğer düğümleri HBA'mıza ekleyerek de genişletilebilir. Ayrıca HBA düğümleri arasında kurduğumuz ilişkiler farklı şekillerde tanımlanabilir. CWE kodları kullanılarak güvenlik açıklarının tespitinde hatalı pozitifliklerin olması mümkündür. Ek olarak, HBA'daki ilişkilerin matris gösterimi, düğümler arasındaki ilgili tüm bağımlılıkları ve etkileşimleri yakalamayabilir. Bunların yanı sıra geliştiricilerin zaman içinde projelerde çok fazla değiştiğini de belirtmekte fayda var. Başka bir deyişle, güvenlik açığı yaratan davranışlar, her geliştiricinin ilişkili projesinde zaman içinde farklılık gösterir. Çalışmamızda belirli tarihler arasında yazılım yaşam döngüsündeki etkinliklerden yararlanarak veri setimizi oluşturuyoruz. Geliştiricinin deneyimine bağlı olarak, bu kodlama davranışı ve alışkanlıkları zamanla değişebilir. Bu, geliştiricilerin güvenlik açığının puanları tetiklediği ve sıralamaların zaman içinde değişebileceği anlamına gelir. Bu nedenle model, geliştiricilerin yeni kazanılan becerileri ve faaliyetleri ile sürekli olarak güncellenmelidir.

Bu tezde, geliştiricilerin güvenlik açığı oluşturma üzerinde bir etkiye sahip olduğuna inanarak bir geliştirici profili çıkarmak için tarihsel kodlama faaliyetlerine dayalı bir HBA tasarladık. Geliştiricilerin profilini, bir güvenlik açığı tahmin modeline girdi olarak kolayca kullanılabilecek bir dizi metriğe dönüştürmek için rastgele yeniden başlatmalı yürüyüş algoritmasını kullandık. Deneysel sonuçlarımız, geliştirici profilinin güvenlik açığı tahmini için faydalı bir metrik olduğunu ve diğer metriklerle birleştirildiğinde başarılı modeller oluşturulabileceğini göstermektedir.

1. INTRODUCTION

Software defects that allow an attacker to attack a system and steal data and hijack the system's functioning are called software vulnerabilities [1,2]. These can be caused by architectural flaws of the software, problems in the source code, poor access to the system or security controls. Due to their negative effects on firms' reputations, high cost of patch creation, and invasion of user privacy, software vulnerabilities attract the attention of software researchers and practitioners [3,4]. Due to many distinct types of vulnerabilities and their many causes, creating a vulnerability prediction model, on the other hand, is rather difficult. Using machine learning (ML) techniques to develop a vulnerability prediction model requires the same stages as other ML applications, such as dataset preparation, feature extraction, model development, and performance evaluation [5].

In addition, vulnerabilities are infrequently discovered and noticed since software teams are frequently unwilling to do. This lack of reporting and transparency can lead to significant security risks and potential harm to users. Some may argue that software teams have limited resources and must prioritize other aspects of their product, such as functionality and user experience, over security. On the other hand, it is critical for software teams to prioritize security and raise awareness in order to ensure the security and privacy of their users.

In order to successfully identify vulnerable and non-vulnerable code parts, one of the most important tasks in the ML workflow for predicting vulnerabilities is the identification of the best characteristics that explain vulnerable code parts. In the literature, there are academic studies that report vulnerability prediction models utilizing various metrics, including size, coupling, complexity, code churn, and fault history [6]–[9]. Walden et al. [6] utilized three web-based projects, one significantly larger than the others in size. Because vulnerable files are rare in large projects, predicting which files might contain vulnerabilities is found to be more difficult.

They point out that better predictive performance is more valuable in this case. The authors report a detection rate of 70.4% from the model using software metrics in this large-scale project. Shin et al. [7] built vulnerability prediction models using complexity, code churn, and developer activity metrics. The model with the code churn metrics has the highest detection rate of 80.5%. Using complexity metrics, authors report 68.9% detection rates, whereas they report 68.6% with the developer activity metrics. Shin and Williams [9] also constructed vulnerability prediction models using only the code complexity metric. They report accuracy values of up to 99% in their experiments. The authors state that using complexity metrics with low false positives is a viable approach, but they emphasize that it still misses many vulnerabilities.

Software errors that are injected into systems can also expose security vulnerabilities [10]. A previous study confirms that there exists a relationship between security vulnerabilities and software errors such as design flaws and code churn [10]. In a study published by Sultana et al. [11], authors discovered that classes having code smells are more vulnerable. Mumtaz et al. [12] also highlight that Feature Envy and Data Class code smells are the most common code smells in vulnerable classes.

As a matter of fact, these errors and code smells are often injected into software projects through developers. As software development is a human-intensive activity, developers' coding and collaboration activities, their experience as well as personal characteristics could have a significant impact on introducing software errors or defects. In the defect prediction field, researchers demonstrate that different characteristics of developers, i.e., developer profiles, are indicators of software bugs [13,14]. For instance, in their analysis on the effect of developer modifications on bug prediction, Nucci et al. [13] demonstrate that the findings of the proposed model outperform others. Nucci et al. [13] obtained accuracy rates from 68% to 94% with the model based on developer changes on defect prediction. The authors' results are more successful than the model proposed by Hassan [15] which uses code change entropy information, and the models in [16,17] which uses the number of developers as their predictors. Meneely et al. [18] proposed developers' social network-based metrics to identify the developer collaboration structure and used it as a feature for the defect prediction model. Lee et al. [19] proposed new criteria based on a set

of information from developer interactions, and used these criteria in their defect prediction models. Authors report that their proposed metrics, when used together with source code metrics and change history metrics, which are generally used for defect prediction, contribute significantly to improve defect prediction performance.

There are studies that conduct detailed reviews on developer profile modeling through systematic literature reviews [20,21] and systematic mapping study [22]. These reviews show that there exist many different ways, metrics or characteristics in the literature for defining developer profiles. For instance, Zhao et al. [23] define developer profile in terms of the quantity of tasks completed, while Meyer et al. [24] and Oliveira et al. [25] propose the number of software modules finished/tested to define a developer profile. There are other studies which utilize the level of expertise measured in terms of the number of years of active programming/testing [26]–[28]. Yan et al. [29] conducted an empirical study to profile developer expertise. For this purpose, they collected data from Github and Stack Overflow and extracted metrics to evaluate the expertise of developers in different skills. There are also other studies that describe the profile in accordance with a certain pattern of behavior, habit, or coding style, such as [30]–[32].

Finally, these developer profiles have been used in different software engineering tasks: Assigning a new issue to the most suitable developer [33], assigning a role such as a team leader to the most appropriate person in the project [34], identifying the experienced developers [35,36], analysis of developer behavior [37] and the most effective teamwork for building the collaborative team [38]. There is no study that identifies developers' profiles with respect to security vulnerabilities or development activities that cause introduction/injection of these vulnerabilities.

By our exploratory investigation of open source projects, we note that some developers are more frequently linked to vulnerabilities in software projects than the others. In fact, if we examine records such as CWE, we see that different types of security vulnerabilities, such as poor coding practices and intentional flaws, are among the root causes [1]. For example, SQL injection attacks are often caused by poor coding practices such that the user input is not properly sanitized before being executed as part

of a SQL query [39]. So, defining developers in terms of their vulnerability-inducing coding behavior motivates us to predict vulnerable code changes in software systems.

Two objectives of this thesis are (1) to create a developer profile through the perspective of software vulnerabilities and (2) to determine the extent to which the profile information predicts code changes that induce software vulnerabilities into the software systems. We utilize the Heterogeneous Information Network (HIN) approach to profile the developers through the perspective of vulnerabilities, taking inspiration from Yan et al. [29]. The HIN technique can help to identify the relationships between developers and vulnerabilities, providing a more comprehensive view of the security risks. This approach can also enable the development of targeted training programs to improve security practices among developers. We have identified vulnerabilities for the four projects we selected: ActiveMQ, Struts, Nifi, and Tika. After obtaining developer scores for various types of vulnerabilities in the HIN that we built on the selected projects, we have included these scores into our model for predicting vulnerability-inducing code changes. In perspective of this, we identify our research question as follows:

RQ : To what extent do developers' vulnerability-inducing coding behaviour explain software vulnerabilities?

2. LITERATURE REVIEW

This section is organized as two sub-sections that summarize existing studies on software vulnerability prediction models and on developer profiling.

2.1 Vulnerability Prediction

A software vulnerability is a flaw in a system's internal controls, security protocol, implementation, or information handling that might be used by a malevolent user to attack the system [1,2]. Software users who activate these affected parts/modules may suffer financial loss, and significantly damages their reputation as a result of the vulnerability and the consequent attack [40]. For example, in 2017, financial services company, Equifax, suffered from data breaches due to a proven software vulnerability. As a result of these breaches, personal information of 143 million people were stolen, and the company suffered a great loss of reputation. In fact, developers inject software vulnerabilities into the source code while working on the code. The expertise, development capabilities, and habits of software teams might also have significant impacts on introducing vulnerabilities to the code, but current research often reports examining the source code in development to detect vulnerability-prone code modules. However, some software teams may have high expertise and development abilities but utilize poor coding practices, resulting in vulnerable source code even at an early stage. Additionally, other factors such as time pressure and inadequate training can also contribute to introducing vulnerabilities to the source code.

Meneely and Williams [41] conducted an empirical case study examining the effects of developer activity on vulnerabilities. The authors present their results by examining the relationships between known vulnerabilities in open-source Red Hat Enterprise Linux four kernel project and developer activity metrics. According to their findings, a file modified by nine or more developers contains 16 times more vulnerabilities than a file modified by fewer than nine developers. This suggests that code changes made by a

large number of developers have negative consequences on the system's security. For example, when a file on an e-commerce site is updated by a few developers, potential vulnerabilities may be less. Conversely when a large organization's software project is modified by many developers, their system might become more vulnerable to potential attacks.

Codabux et al. [42] examined the code written by software developers during the software development lifecycle. They explain their behavior towards technical debt by investigating the code smells that the developers introduced, and the refactorings that they implemented. The SonarQube tool was used to collect data on code smells, bugs, coding violations, refactoring, and vulnerabilities, which were then used to calculate technical debt. These metrics illustrate how the coding practices of developers affect technical debt.

Nord et al. [10] suggest leveraging technical debt knowledge to detect vulnerabilities that are extremely hard to find with static code analysis alone. According to their discoveries the more types of design faults a file has, the more probable it is to have vulnerabilities. Files containing security vulnerabilities are also prone to increase code churn. As a result, Nord et al. [10] present a proof of a connection between security vulnerabilities and indicators of technical debt such as design errors and code churn.

The task of predicting whether a file contains vulnerabilities is more difficult than examining statistical relationships between vulnerabilities and other code features, as the vulnerable files are very rarely found in software systems. Many research studies have been conducted to predict software vulnerabilities using structural metrics. Shin and Williams [43], [44] conducted a study to estimate the location of security issues. The authors utilized complexity metrics on Mozilla JavaScript Engine and showed a correlation between these metrics and security issues. Although they only found a weak correlation, they report that there could be other factors that cause vulnerabilities.

Chowdhury and Zulkernine [45] examined the effects of using structural measures such as complexity, coupling, and cohesion in software vulnerability prediction models. They present a methodology based on these metrics and conducted empirical research on 52 versions Mozilla Firefox project. Researchers emphasize that the average

detection rate is 74.22%. In the literature, some studies presenting vulnerability prediction models use software metrics only [46], while others compare models based solely on software metrics with models based on alternative code structures [6,7].

Walden et al. [6] generate their own dataset. The dataset they created includes three web applications and 223 vulnerabilities detected in these applications. They use text mining technique and software metrics for vulnerability prediction and compare results. While the model using software metrics achieves recall rates of 66.3%, 70.4% and 76.9% for web applications, text mining-based models have higher recall rates. Zhang et al. [47] created vulnerability prediction models for the same three web applications in their work. They recommend a two-layer composite approach called VULPREDICTOR. Their recommendations use a combination of software metrics and text features to improve prediction accuracy. Their outcomes show that VULPREDICTOR outperforms other models.

Shin and Williams [46], while predicting vulnerabilities, also used software metrics, but they got poor precision and high recall numbers. Their vulnerability prediction models attain a degree of accuracy ranging from 12% and 52%, and recall between 15% and 83% at various prediction thresholds.

Shin et al. [7] finally investigated Mozilla Firefox web browser and Red Hat Enterprise Linux kernel to build vulnerability prediction models based on complexity, code complexity, and developer activity data. They acquired a recall of 70.8% and a precision of 4.6% in their studies using all measurements for Mozilla Firefox. Using discriminant analysis and all metrics, they achieved 68.8% recall and 7.5% accuracy for Red Hat Enterprise Linux kernel. They do not use a vulnerability-related metric, although they recommend a developer-focused metric. Instead, they define the metrics they use for developers based on code changes.

2.2 Profiling Developers

Developers play a crucial role in the software development life cycle [48]. They are responsible for writing, testing, and debugging code to produce well-functioning software. In addition, they work closely with other members of the team, such as

designers and project managers, to ensure that the software meets the needs of the end user. Without skilled developers, it would be impossible to create software products that empower our modern world.

Developers also need to have a good understanding of the project requirements and the programming languages and tools being used. They should be able to write efficient and maintainable code that can be easily modified or extended in the future. Additionally, developers need to be able to work well under pressure and be comfortable with the constant need to learn and adapt to new technologies and approaches. In summary, developers are an essential part of the software development process and their skills and expertise are critical to the success of any software project. Therefore, their contribution to the source code of a project is undeniable. That's why the developer profile is an interesting and yet, challenging topic.

Developer profiles have been defined in different ways in the literature [28,42,49]. The data created with these definitions have then been used to extract features or feature sets for different studies.

There are studies that conduct a detailed literature review to draw attention to profiling and to model the developer profile. Matturro et al. [22], for instance, conducted a systematic mapping study to identify the existing literature on soft skills in software engineering and to examine the soft skills in software engineering and their relationship to software engineering practices. The authors defined soft skills as a collection of abilities, habits, attitudes, and personality traits that enable people to perform better in the workplace, complement the technical skills needed to do their tasks, and influence how people conduct and interact with one another. Their mapping studies performed search on known databases such as ACM Digital Library, SpringerLink and IEEExplore. As a result, they read and analyzed 44 articles. They grouped these articles into categories such as analytical skills, creativity, listening skills, and critical thinking, and counted the frequencies of each category among the selected articles. Finally, they identified some soft skills drawn from selected papers. Matturro et al. [22] claim that their work has been a starting point, and suggest new work as an investigation of the impact of these skills on key factors of software project

success, such as problem solving, decision making, interpersonal communication, and teamwork.

Vishnubhotla et al. [20] conducted a systematic literature review for profile modeling. In [20], qualifications used to measure and predict the ability of individual software engineers and teams are categorized in the context of agile software development (ASD) for both individual engineers and teams. They considered articles published in Scopus, Science direct, ACM, and Wiley databases from 2001 to 2016. At the end of the search process, only 16 primary studies were selected on the keywords: capability, competence, team building, and ASD, along with alternative terms. They categorized qualifications for software engineers and their teams as professional, social, and innovative, identified subcategories, correlated qualifications for each category, and suggested attributes used to measure capabilities for an individual engineer. The results provide detailed talent measurement attribute catalogs that can assist software organizations in human resource management, such as identifying employee with sensitive skills, for both software engineers and agile teams.

Other works, such as the one reported by Meyer et al. [24], describe developer profile based on productivity. On their perceptions of productivity, developers might be approximately categorized into six groups: goal-oriented developers, focused developers, lone developers, balanced developers, leaders, and social developers, according to research done by the authors using data from 413 professional software developers.

One of the popularly discussed topics among papers on developer profiling is team formation, which entails giving a software development project to the best possible team. This understanding of developer profiles can help project managers determine which developers are best suited for specific tasks. In addition, understanding the developer profile of a project team can help managers identify potential skill gaps and design training programs to address them. By taking into account the skill sets and knowledge of each team member, project managers can optimize the development process and ensure the success of the project. Vishnubhotla et al. [38] examine the use of individual and team-level developer metrics in characterizing the capabilities of an

agile team, and the individuals that make up that team. As a result of employing the established developer profiles, they conduce to the election of relevant employees for agile teams. Kortum et al [50] investigated team cohesion for higher project success and productivity. They introduced a JIRA plugin called ProDynamics and report the characterization they have generated with this plugin. Six students examined the applicability of the tool with a software project. They rcontributed to the understanding of the effects of sprint planning and team cohesion. Costa et al. [51], utilized real-world data, and observed 16 developers and four project managers from eight Scrum projects. The tag-based profile method they presented is reported to facilitate multi-team formation, which provides high satisfaction. With this method, developers and project managers can evaluate team performance in real time and determine strategies to achieve targeted productivity in the team. For example, by determining which developers are more active with information from the data.

Several studies have also employed developer profiles to design developer recommendation systems [23,52,53]. Zanjani [52] created a developer expertise profile using code reviewer data. The author's profile was used for both the developer recommendation and the code reviewer recommendation.

In a study published by Sun et al. [54], authors introduced a method for suggesting developers based on the experience and practices of developers. They analyzed commit history to determine each developer's profile. When a problem request is made, the proposed method suggests a developer to fix the software issue if the files that need to be worked on and the files that the developer previously worked on are similar. When multiple developers are suggested, the list is sorted by relevance.

There exist studies that identify the developer profile based on specialization. In [55,56] the authors defined the developer profile as the specializations of the developers. But developers have differentiated their expertise with measurement metrics. In [55] Bhat et al. they used an issue based metric. In [56], Khatun and Sakib used the types of bugs contributed by the developers as their metrics.

In [27], authors refers to the amount of technical proficiency that developers in a certain sector possess. They evaluated developers according to the influence that they have on

the team dynamic. Mani et al. [28] describe the developer's specialization profile as a collection of Java classes with which the developer is likely conversant. With the proposed profiling, they were able to detect the API usage of the developers.

Dakhel et al. [49] also groups developers into "Expert" and "Novice" programmers. They see syntactic pattern mastery as a proof of programming knowledge and have proposed the distribution of syntax patterns in source code as a theoretical description of programming knowledge. Expert developers were distinguished from novices based on their commands of syntax pattern usage.

Ouni et al. [26] constructed developer profiles by utilizing the knowledge and experience of the developers to locate leading reviewers. Soares et al. [57] also profiled code reviewers to be assigned to the appropriate software project as part of a recommendation system. The authors used the rate of long-term pull requests from developers, the number of daily closed pull requests, and response time as metrics.

A recent study published by Codabux et al. [42] delved into the correlation between technical debt and a developer's profile. Through a thorough analysis of developers' coding practices, encompassing factors such as code smells and refactoring activities, the authors were able to create unique developer profiles and link these profiles with coding maturity and competence. The results of the study indicate that, while more experienced developers tend to produce fewer applications of average quality, they may also accumulate higher levels of technical debt. These findings suggest that a developer's experience and skill set may not necessarily translate into less technical debt, highlighting the importance of managing technical debt within software development projects.

Yan et al. [29] suggest using HINs in software, which are used in data mining tasks such as similarity measurement, classification, clustering and ranking, as well as in bioinformatics to identify gene-disease associations [58]. The authors utilize the HIN as a novel method to profile developer expertise in software organizations. They pooled data from Stack Overflow and Github to profile the developers. A HIN is basically a directed graph containing different types of objects and relationships. In the authors' study, nodes in the network were labeled as developers, projects, questions, or talent

terms. The team utilized the questions asked or answered by developers on Stack Overflow to establish connections between developers and questions. Additionally, they analyzed commit contributions, views, and stars from Github data to connect projects to developers. They also linked projects to talent terms using tag information. The team used an RWR algorithm to predict proximity within the generated network. This algorithm works by simulating a random walker that starts at a given node and follows the edges of the network, restarting at the starting node after certain number of steps. This process is repeated until convergence is achieved by effectively estimating the proximity between nodes in the network. Each developer is profiled by evaluating his or her expertise across all talent terms. To evaluate the performance of their proposed method, they compared their findings with a prior study, CPDScorer [59], and demonstrate that their technique is superior. Overall, the use of HINs and the RWR algorithm in analyzing expertise in software communities has proven to be a promising approach, providing new insights into the relationships between different types of objects and their connections.

3. METHODOLOGY

In this section, we first explain how we created the dataset that we utilized in the thesis. Second, we describe how we generate a developer profile based on their behavior leading to vulnerability-inducing changes, and third, we describe how the profile data affects predicting the code changes that lead to vulnerabilities.

3.1 Dataset

We utilize the SmartSHARK [60] public dataset, which contains 77 Java projects from the Apache Software Foundation (ASF) repository to predict vulnerability-inducing commits. The dataset that was obtained from GitHub project repositories comprises between 1,000 and 20,000 commits for the projects. In the dataset, there are 47,303 pull requests, 163,057 issues, 2,987,591 emails, and 366,322 commits.

3.1.1 Dataset preprocessing

We identified the collections relevant to this thesis and extracted them using custom MongoDB scripts in the source dataset, as several of the MongoDB collections in the dataset were irrelevant to our research. The following are the collections that were chosen:

- project: This collection contains the project names that are kept in SmartSHARK.
- vcs_system: All version control systems (vcs) from which data was gathered are included in this collection.
- commits: This collection contains information about various commits made to a project.
- people: This collection includes all individuals who worked on a project, i.e., contributed a commit.

- code_group_state: This collection contains, for each project commit, all code group states, e.g., all packages of the project at each commit.
- issue_system: This gives the issue management system of each project (e.g. JIRA, Bugzilla).

SmartSHARK is an extensive data repository. It does not, however, provide the vulnerabilities required to develop the vulnerability prediction model. Therefore, we retrieved vulnerability reports from the National Vulnerability Database (NVD) for all projects. Then, we chose four projects with the largest number of real vulnerability data. Struts, ActiveMQ, Tika, and Nifi are the four projects that are chosen for empirically investigating the research question of this thesis.

3.1.2 Dataset extension

We added a vulnerability dataset to the SmartSHARK dataset for the selected four projects. Vulnerabilities published in the NVD database are indexed by Common Vulnerabilities and Exposure Identifier (CVE ID). We extracted all CVE IDs and Common Weakness Number (CWE) IDs for the projects.

Fixing vulnerabilities in software projects can be a challenging task due to the multiple vulnerability-fixing commits that companies often submit. To tackle this issue, we manually gathered fixing commits from various web resources and project issue tracking systems and then matched them with software vulnerabilities. We utilized MSR2019 [61] dataset, which contains vulnerability data and fixing commits from open-source projects, as a benchmark to ensure our vulnerability-commit linkage was accurate. This dataset was collected using data from the NVD and 50 different project-specific web resources. After rigorously testing and validating our methods for extracting vulnerabilities, we successfully matched vulnerabilities and fixing commits for four projects from the MSR2019 [61] dataset.

Aside from just gathering the commits that resulted in vulnerabilities, we also acquired commits that induced vulnerabilities in order to construct a model for predicting vulnerabilities. To accomplish this, we employed the SZZ algorithm [62], a widely used approach for identifying changes that lead to bugs. The algorithm scrutinizes the

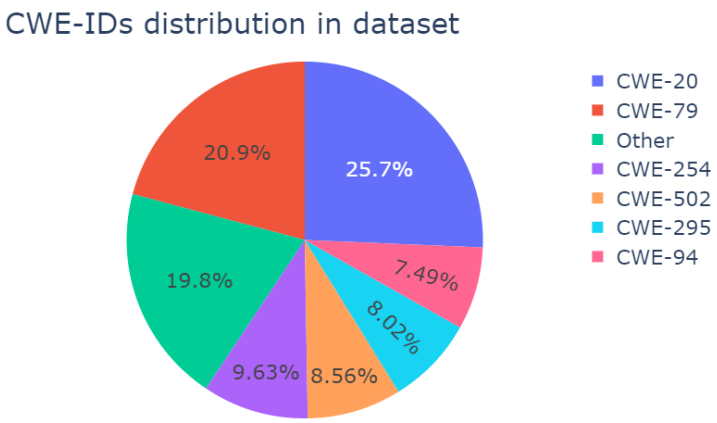


Figure 3.1 : Distribution of CWE-ID in the dataset.

commits on which a reported bug and/or vulnerability is fixed in order to backtrack to the commits that introduced/induced that bug or vulnerability. This thorough process ensures that we obtain the commits in which a vulnerability is likely to be injected into the system, rather than the commits in which that vulnerability is fixed, and hence, a vulnerability prediction model can be built on the inducing commits.

We linked 75 vulnerability fixing commits with 402 vulnerability inducing commits. Nonetheless, a large proportion of vulnerability inducing commits is linked to a few vulnerability fixing commits. We examined and filtered such commits since they might not be pushed to the system after development. We also removed certain vulnerabilities since their CWE-IDs were not described. Eventually, we discovered 187 vulnerability inducing commits that linked to 62 vulnerability fixing commits. According to our findings, the pie chart in Figure 3.1 depicts the CWE-ID vulnerability types and rates. The additional dataset of vulnerabilities that we retrieved is also freely accessible in CSV format [63].

3.2 HIN Formation

HIN is a directed graph composed of numerous objects and relationships. HIN, in contrast to homogeneous networks, permits two nodes to be connected by multiple edges, each of which can represent a distinct relation [64]. HIN is utilized in a variety of data mining applications, including classification, similarity measurement, ranking, and clustering [29,65,66]. Shi et al. [65] built HIN using bibliographic data to identify the most prominent authors and the respectable conferences where

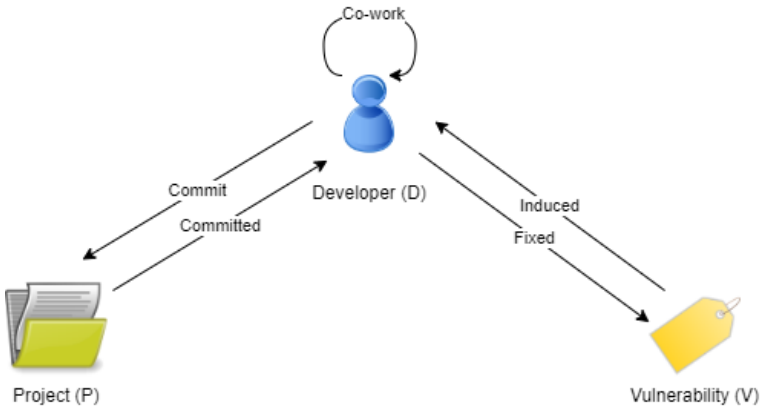


Figure 3.2 : Our Network Schema.

these authors presented their papers. HIN is commonly employed in bioinformatics, particularly in investigations of gene-disease relationship estimates [67,68].

We initially had to design a relationship table and network diagram to profile developers using HIN analysis, as shown in Figure 3.2. The network has many nodes, each of which is labeled with one of three categories: The developers who participated in the code changes in the chosen projects are indicated by the label Developer (D). The Project (P) label indicates the projects chosen for analysis. The vulnerabilities identified in these projects were identified by CWE codes. Detected vulnerabilities are specified with the Vulnerability (V) label. The edges between nodes D, P, and P represent a set of relationships. These relationships are specified in matrix form in the HIN and are derived from various data sources that occur throughout the development life cycle of the project in the dataset.

Our system has a specific approach for establishing relationships between Project and Developer nodes whenever a code commit is made. This approach involves generating two identical connections, which are intended to reflect the collaborative efforts of the developer involved. We create two links between the Developer and Vulnerability nodes. Because developers contribute to the project with commits, as is the nature of software development, these commits may introduce or fix vulnerabilities. In other words, while the developers inject a security vulnerability into the project through some of their commits, they might also remove them with some commits. This way, we are able to establish a more detailed and accurate understanding of a developer's relationship to a project based on their contributions. If two developers co-work and

work together on the same commit, it is known as co-working. Based on all these, the relationships we define for our HIN;

- Between Project and Developer nodes;
  - A project is contributed by a developer through multiple code changes (P→ D). For example, during the development of a web application, a developer first adds user registration and login functions, then edits the homepage and finally integrates a payment system. In this way, it contributes to the project by making multiple code changes.
  - Through a series of commits, a developer contributes to a project (D→ P). For example, a developer can add a new feature to a software application and contribute by submitting the code to the project's repository. At the same time, the developer can contribute to different projects.
- Between Developer and Vulnerability nodes;
  - A security vulnerability is created in the source code by a developer over a routine coding event, eg. feature development, bug fixing, refactoring (V→ D).
  - A developer fixes a reported vulnerability in an issue repository by modifying parts of the source code (D→ V). Developers frequently check and fix their code to avoid security vulnerabilities.
- Between Developer nodes;
  - Two developers collaborate on a single commit (D→ D). For example, one developer writes the code for a new feature, while another reviews and tests it before merging it into the master branch. Working together on the same commitment, they ensure that the code is of high quality and meets the team's standards, or one developer writes the code for a new feature while the other reviews and tests it before merging it into the master branch. Working together on the same commitment, they ensure that the code is of high quality and meets the team's standards.

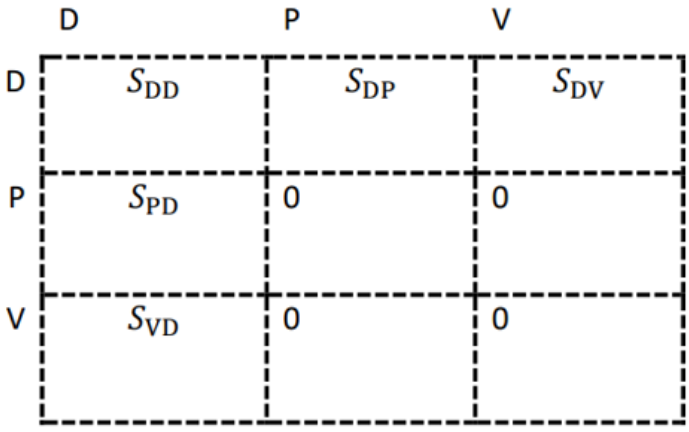


Figure 3.3 : Network in matrix form.

3.3 Extracting Relationship Data

The proposed associations between the new node types as well as their matrix representations are collected from various data sources during the course of the project's development life cycle. Our HIN nodes and the sub-matrices connecting them are shown in Figure 3.3.

Since we have an extensive dataset, we have all the developers' information contributing through commits of the selected projects. For example, we have the commit information that developers have made to projects, and the set of developers that they are working with on those commits. We used all this information while generating our matrices to create the HIN. The Vulnerability and Developer (S_{VD}) relationship matrix is created only with vulnerable commits. The Developer and Vulnerability (S_{DV}) relationship sub-matrix uses only vulnerability-fixing commits. Because various types of relationships have varying relevance, they require different weight calculation methods. We weight our S_{VD} sub-matrix based on the number of times a developer has added a type of vulnerability to the project through commits. In other words, it is about how many commits a d_1 developer has included a v_1 type vulnerability in the project. The weight of our S_{DV} sub-matrix is the number of commits that a developer has removed from the project by editing the development that caused a type of vulnerability. That is, it is related to number of commits the d_1 developer made to fix the v_1-type vulnerability from the project. All commits in the

dataset are used in matrices representing the relationship between the Developer and the Project. We construct the sub-matrices S_PD and S_DP, and weight them based on the aggregated number of commits made by each developer for the respective projects. That is, it is weighted according to the number of commits a d_1 developer has made to a p_1 project. The S_{PD} sub-matrix is the transpose of the S_{DP} sub-matrix. Lastly, the Developer and Developer (S_{DD}) relationship matrix represents two developers collaborate on a single commit. This sub-matrix is generated using author and implementer information from commits. In the matrix, if one developer is the committer and the other is the author, we set the appropriate value to 1 and weight it that way.

The dimensions of the (S_{VD}) sub-matrix is 18x650, while (S_{DV}) is 650x18. The dimensions of the sub-matrix S_{PD} is 4x650, the dimensions of the sub-matrix S_{DP} is 650x4. Finally, the dimensions of the (S_{DD}) sub-matrix is 650x650.

3.4 RWR Algorithm

RWR algorithm is appropriate for evaluating the significance of single or multiple object types developed with HIN. This algorithm is especially used in complex datasets such as social networks and biomedical data. It can also be used in various industrial applications. Various relationships between the two object types create different transition likelihoods, leading to various random walk loops and sorting outcomes [58,67,69].

We utilize RWR in accordance with the strategy suggested by Yan et al. [29]. We implement the RWR algorithm to traverse the nodes of the created HIN along with the edges to predict the closeness between the developers and the vulnerability types [70,71]. A random surfer embarking on a random walk from a node in the network has two options at each step: (i) follow the path to a randomly selected neighboring node with probability $1 - \lambda$; (ii) following the path back to the starting node with probability λ. In the algorithm λ is expressed as the restart probability. This number was commonly described as 0.7 in research utilizing the algorithm [58,68,72].

Moreover, Wang et al. [68] tried many different values to observe how the restart probability value affects the results of the RWR algorithm. The authors' experiments have performed the best when the restart probability was set to 0.7. In this thesis, we set λ to 0.7 for these reasons. In addition, the RWR algorithm has two parameters: The transition probability matrix T and the restart vector p_0. T_{ij} represents the random walker's likelihood of passing from node i to node j. According to the algorithm, we multiply each sub-matrix by an adjust factor. Thus, we have obtained the column-normalized transition probability matrix. We chose 0.5 as the value for the adjust factor.

As we aim to measure a developer's profile established on vulnerability-related behaviors, we define vulnerability nodes as restart nodes, also known as *seed* nodes, in our algorithm. Let p_0 be the column vector reflecting the seed nodes' restart vector. Please keep in mind that the restart vectors will vary when we evaluate a developer's vulnerability inducing behavior for different types of vulnerabilities.

The following formula (3.1) defines the RWR model . We got static probabilities for each node (p_a) by iteratively applying this method to the starting nodes of all nodes in the network until the closeness score, which represents convergence, was reached. The score of a type D node is the developers' score associated with vulnerability types in V.

$$p_a^{t+1} = (1-\lambda)T p_a^t + \lambda p_0 \tag{3.1}$$

3.5 Vulnerability Prediction

We employ the RWR algorithm on the created HIN. We compute the closeness score between developer nodes and vulnerability nodes. Thus, we acquire the scores associated with each vulnerability type for each developer. Our scores and rankings show that some developers induce vulnerabilities more often than others. In other words, some developers commit code improvements that more likely cause security vulnerabilities. This information can be used to identify high-risk developers and allocate resources for training and improvement. Additionally, it can also help

in identifying patterns or trends in vulnerability creation, which can inform the development of better coding practices and guidelines.

It is not easy to evaluate the correctness of our technique, mainly to demonstrate that the ranks we allocated to the developers for their vulnerability-inducing activities are correct. To identify and prioritize activities that create vulnerabilities, a process of continuous monitoring and evaluation is required. In addition, training and information programs are essential for increasing developers' security vulnerabilities. Rarely are vulnerable commits made, and developer activity is frequently erratic. Developers do not routinely commit to initiatives; rather, they contribute when new development is required or maintenance is necessary. This inconsistency makes our evaluation procedure difficult. Consequently, it is impossible to determine a suitable time to divide the project data by separating it into two, namely training and testing, and utilizing the ranks we received during training to validate a test set. As an alternative, we use a different approach based on software metrics to evaluate how developer profiles effect our model of predicting vulnerability-inducing contributions. To our knowledge, no research has been done to predict vulnerabilities using the profiles of developers.

We compare the strategy of utilizing software metrics based on the source code with the way developer profiles are used to predict commits that lead to vulnerabilities. In order to do that, we aggregate the metrics in the dataset from the file level to the commit level using the metrics for project complexity, coverage, and dependence. We also extend the model for vulnerability prediction by including new sets of characteristics that characterize the developer's profile. We finally train and evaluate models using attributes that just represent the developer profile. The models that are trained solely utilizing software code metrics are referred to as "Software Metrics Models" (SMM), the models that are trained to utilize developer profiles are referred to as "Developer Profile Models" (DPM), and the models that are trained to utilize both the input features (developer profiles and code metrics) are referred to as "Software Metrics and Developer Profile Models" (SMDPM).

Table 3.1 : A section from the results of the RWR algorithm.

ID	Developer	CWE-16	CWE-20	CWE-22	CWE-77	CWE-79	CWE-94	CWE-200	CWE-254
0	58bfd02202ca40f8bf1481a7	0,052757	0,001672	0,002142	0,000776	8,2E-05	5,34E-06	5,63E-05	2,14E-05
1	5e71ceaa1c4f2a5a2b64005c	0,026463	0,000876	0,001311	0,000693	7,56E-05	4,97E-06	5,24E-05	1,83E-05
2	5e71ceaa1c4f2a5a2b640084	0,026463	0,000876	0,001311	0,000693	7,56E-05	4,97E-06	0,175052	0,019463
3	5e71ceaa1c4f2a5a2b6400a7	0,028609	0,040361	0,001417	0,000749	8,18E-05	5,37E-06	5,66E-05	1,98E-05
4	59bfd9a5f2a4565fe934b025	0,052757	0,001672	0,002142	0,000776	8,2E-05	5,34E-06	5,63E-05	2,14E-05
5	5b6859bb26b42f0b33391b59	0,052757	0,001672	0,002142	0,000776	8,2E-05	5,34E-06	5,63E-05	2,14E-05
6	58c118c202ca40f8bfb1f2ae	0,026463	0,000876	0,001311	0,000693	7,56E-05	4,97E-06	5,24E-05	1,83E-05
7	5e71ceae1c4f2a5a2b64036a	0,000196	9,22E-05	0,018145	0,000752	8,03E-05	5,9E-06	5,63E-05	2,1E-05
8	5e71ceae1c4f2a5a2b64038c	0,00017	0,080288	0,000479	0,00061	0,009044	4,59E-06	4,84E-05	1,53E-05
9	58c11b86e4f89451f51d7c67	0,052757	0,001672	0,002142	0,000776	8,2E-05	5,34E-06	5,63E-05	2,14E-05
10	5b0fc423065f39020b8e07a3	0,000206	0,000207	0,000611	0,001147	0,000348	0,000143	4,93E-05	1,55E-05
11	5e71cebb1c4f2a5a2b640def	0,00017	7,94E-05	0,000479	0,00061	6,92E-05	4,59E-06	4,84E-05	1,53E-05
12	5e71cec01c4f2a5a2b6413dd	0,00017	7,94E-05	0,000479	0,00061	6,92E-05	4,59E-06	4,84E-05	1,53E-05
13	5e71cec01c4f2a5a2b6413e5	0,000197	9,29E-05	0,000558	0,000712	8,19E-05	6,11E-06	5,62E-05	1,78E-05
14	5e71cec01c4f2a5a2b6413eb	0,000198	0,000101	0,000561	0,000742	0,0001	1,57E-05	5,57E-05	1,76E-05
15	5e71cec01c4f2a5a2b6413f1	0,000197	9,32E-05	0,01375	0,000745	8,18E-05	6,17E-06	5,72E-05	2,02E-05

In this thesis, we train vulnerability prediction models utilizing four ML classifiers: Support Vector Machine (SVM), Random Forest (RF), eXtreme Gradient Boost (XGB), and Naive Bayes (NB). The purpose of this thesis is to contrast the outcomes of the models created utilizing various classifiers, dimensionality reduction techniques, and feature selection strategies.

To minimize overfitting and bias, XGB [73] is an expanded version of gradient-boosted decision trees that integrates system improvements like parallel processing, tree pruning, hardware optimization, and algorithmic enhancements like managing missing values and regularization. The technique seeks to increase model performance and computation speed by applying boosting to machines. To achieve the best performance, we perform hyperparameter tuning via grid search. Because XGB has numerous hyperparameters, tuning it is challenging.

SVM is a frequent classifier in prediction problems and a supervised learning model. If the dataset can be linearly split into classes, binary classification establishes the difference using infinite hyperplanes. The best SVM parameter set is determined by k-fold cross-validation [74], and we configure the SVM with a poly kernel to predict commits that would introduce vulnerabilities.

RF [75] is a classifier that employs the mean to increase the dataset's prediction precision and includes many decision trees in different subsets of the dataset. Compared to previous methods, the classifier requires less time to train. Even for a vast dataset with the majority of the data missing, it accurately predicts output when carried out effectively. Instead of depending on a single decision tree, RF considers the estimates from each tree and determines the outcome based on the plurality of these estimates. Greater precision and avoiding overfitting are two benefits of having more trees.

NB [76] is a Bayes theorem-based classifier. It is one of the most straightforward and efficient classifiers that aids in creating reliable models. We employ the Gaussian model, which presumes that the characteristics are distributed normally. This indicates that the model assumes that the continuous values used by the estimators are samples from the Gaussian distribution rather than discrete values.

3.6 Performance Evaluation

Recall, precision, F-measure, and inspection ratio are used to assess the effectiveness of our four classifiers, as illustrated in Table 3.2. The confusion matrix is used to retrieve these data. The confusion matrix indicates the number of incorrectly and adequately predicted examples. In this thesis, True Positive (TP) refers to examples where the model predicts a developer-induced system vulnerability while the actual label is also a vulnerable commit. True Negative (TN), on the other hand, illustrates commits in which both the actual and predicted labels are non-vulnerable. Recall metric is the percentage of commits observed as vulnerable and correctly classified. The fewer False Negative (FN) rates are, the closer the recall (TP/(TP+FN)) is to 100%. As software vulnerabilities become more and more critical, they need to be predicted efficiently and accurately. Therefore, the recall criterion is important for us in our evaluation. Also, we used Friedman and Nemenyi tests to determine that the variations in performance metrics across the models are statistically significant ($p < 0.05$).

We split the dataset into two subsets: training and testing. We use random state values to create different splits in the data that we divide into 70% training and 30% testing. With this method, we run the experiments 10 times. We use the evaluation metrics given in Table 3.2 when examining the performance of the experiments. We took the average of the results of 10 experiments. In this section, we explain the empirical results we obtained using our prediction models, which we named SMM, DPM, and SMDPM. Our findings reveal that SMDPM outperforms both SMM and DPM in terms of accuracy and precision. Moreover, our experiments indicate that the performance of all models improves when more data is used for training. These results suggest that SMDPM is a promising approach for predicting the target in this domain.

We used XGB,RF, SVM, and NB classifiers to train our vulnerability prediction models. In addition, we evaluated different techniques to improve the predictions of the models. The effects of feature selection and dimension reduction techniques on the performance of the models were examined. Using these techniques for each classifier, we evaluated their results. Table 4.1 presents all the results.

Table 3.2 : The definitions of evaluation metrics.

Evaluation Metrics	Formula	Descriptions
Recall(R)	$\frac{TP}{TP+FN}$	The degree to which the classifier recognizes commits that introduce vulnerabilities.
Precision(P)	$\frac{TP}{TP+FP}$	The proportion of accurately predicted vulnerability-inducing commits.
F1-Score(F1)	$\frac{2*Recall*Precision}{Recall+Precision}$	The harmonic mean of recall and precision
Inspection Rate(IR)	$\frac{TP+FP}{TP+TN+FP+FN}$	The proportion of estimated vulnerable commits in the dataset (TP+FP).

Too many features or correlations between features can have a negative impact on ML algorithms. To avoid this, the technique of dimensionality reduction is frequently used. Principal Component Analysis (PCA) and Independent Component Analysis (ICA) are utilized in this thesis. While PCA attempts to explain the majority of variables in the dataset with a single component, ICA attempts to identify the variables' independent components. These techniques enhance the performance of ML algorithms, enabling for more precise vulnerability predictions. Using these techniques, the number of features is reduced to 10.

Overall, our findings suggest that incorporating feature selection and dimensionality reduction techniques can significantly enhance the precision of vulnerability prediction models. However, the effectiveness of these techniques may vary depending on the specific classifier used. Further research is needed to explore the optimal combination of feature selection, dimensionality reduction, and classification techniques for vulnerability prediction.

4. RESULTS AND DISCUSSION

We employed the RWR algorithm on our HIN to establish the developer scores for every CWE-ID. We have presented a subset of the findings in Table 3.1. Our analysis reveal that some developers are more prone to inducing vulnerabilities than their peers. We have identified the developer with the highest CWE-ID score to have made a greater number of vulnerability-inducing commits as compared to other developers.

Based on our comprehensive research on HIN, we have found that the most frequently encountered vulnerability types are as follows: CWE-20, which accounts for 25% of the cases, followed by CWE-79 with 20%. In addition, we have identified three other significant vulnerability types that make up a combined 25% of the cases, namely CWE-254 with 9%, CWE-502 with 8%, and CWE-295 with 8%. Finally, CWE-94 accounts for 7% of the cases.

Vulnerability scores are standardized from 0 to 1, with 1 being the highest score, indicating a high probability of a developer introducing a vulnerability into the system. To provide a clear illustration, Figure 4.1 presents vulnerability scores for only a select group of developers and the six most common CWE types. Given the large number of developers and CWE types, we chose to focus on the top 15 developers with the highest scores in CWE-20, which is considered the most prevalent vulnerability type in this thesis. Additionally, the vulnerability scores for the other CWE types are also included in the graph. The results demonstrate that developers with high scores in the CWE-20 vulnerability also tend to have high scores in other vulnerability types. Therefore, if a developer is likely to induce a CWE-20 vulnerability, he or she is likely to induce other vulnerabilities as well. This highlights the importance of prioritizing security in the software development process.

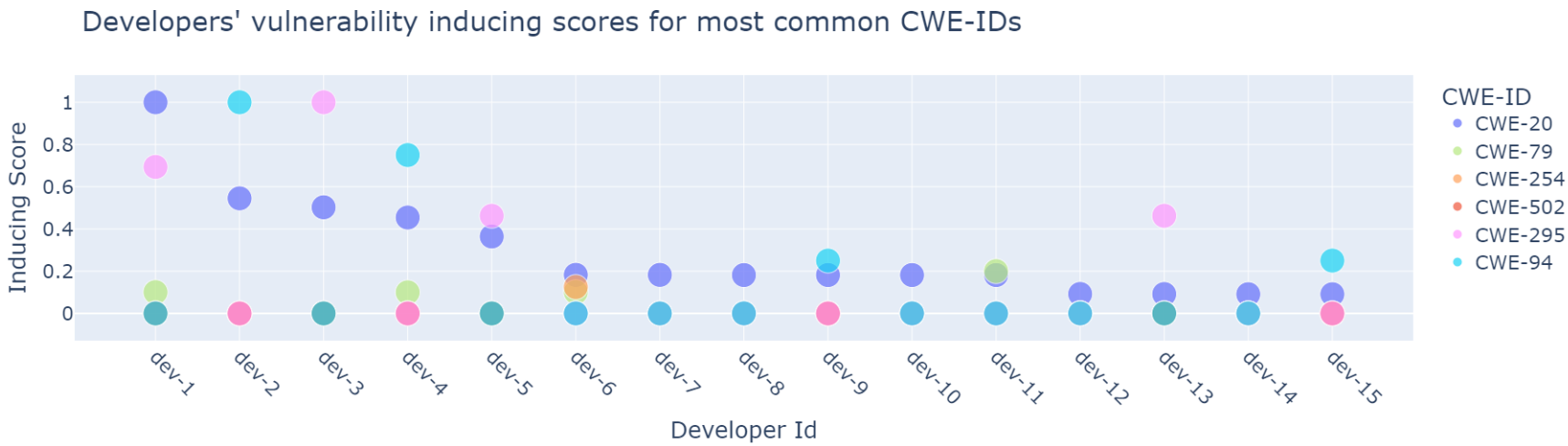


Figure 4.1 : Vulnerability inducing scores for 15 Developers we obtained. (sorted according to CWE-20 vulnerability type).

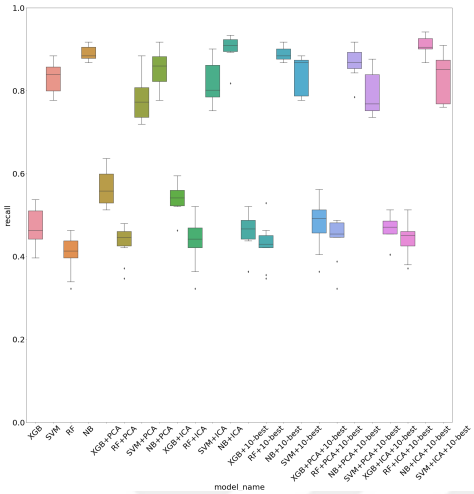
Table 4.1 : Vulnerability Prediction Model Results

Model Name	SMM				DPM				SMDPM			
	R	**P**	**F1**	**IR**	**R**	**P**	**F1**	**IR**	**R**	**P**	**F1**	**IR**
XGB	0.471	0.027	0.052	0.247	0.652	0.026	0.050	0.365	0.487	0.039	**0.073**	0.178
SVM	0.833	0.021	0.042	0.560	0.578	0.026	0.051	0.318	0.813	0.022	0.044	0.517
RF	0.408	**0.030**	**0.056**	**0.195**	0.664	0.026	0.051	0.364	0.454	**0.040**	**0.073**	**0.164**
NB	0.890	0.019	0.038	0.651	0.473	0.024	0.046	0.286	**0.890**	0.019	0.038	0.651
XGB+PCA	0.566	0.026	0.050	0.314	0.696	0.026	0.051	0.384	0.561	0.032	0.061	0.249
SVM+PCA	0.783	0.021	0.041	0.538	0.575	0.026	0.051	0.317	0.791	0.021	0.042	0.531
RF+PCA	0.433	0.029	0.054	0.217	0.662	0.027	0.051	0.361	0.494	0.037	0.069	0.192
NB+PCA	0.852	0.018	0.036	0.671	0.653	0.024	0.046	0.392	0.833	0.020	0.040	0.584
XGB+ICA	0.541	0.025	0.049	0.304	**0.710**	0.026	0.051	0.390	0.544	0.033	0.063	0.234
SVM+ICA	0.819	0.020	0.040	0.582	0.644	0.026	0.050	0.361	0.745	0.026	0.051	0.408
RF+ICA	0.434	0.029	**0.056**	0.210	0.669	0.026	0.051	0.365	0.500	0.039	0.072	0.186
NB+ICA	0.902	0.018	0.036	0.709	0.595	0.025	0.049	0.342	0.829	0.020	0.040	0.591
XGB+10-best	0.455	0.029	0.055	0.226	0.660	0.025	0.049	0.380	0.447	0.028	0.054	0.226
SVM+10-best	0.838	0.020	0.039	0.601	0.621	0.024	0.047	0.372	0.849	0.020	0.039	0.607
RF+10-best	0.428	0.030	**0.056**	0.205	0.645	0.025	0.049	0.365	0.425	0.030	0.056	0.204
NB+10-best	0.888	0.019	0.038	0.652	0.376	**0.029**	**0.053**	**0.197**	0.888	0.019	0.038	0.652
XGB+PCA+10-best	0.476	0.028	0.053	0.246	0.676	0.026	0.051	0.371	0.517	0.036	0.068	0.206
SVM+PCA+10-best	0.790	0.021	0.041	0.536	0.636	0.026	0.050	0.358	0.729	0.024	0.047	0.430
RF+PCA+10-best	0.443	0.028	0.054	0.223	0.668	0.027	0.051	0.363	0.512	0.037	0.070	0.196
NB+PCA+10-best	0.867	0.019	0.038	0.638	0.573	0.025	0.048	0.333	0.833	0.020	0.040	0.580
XGB+ICA+10-best	0.463	0.028	0.054	0.235	0.675	0.027	0.051	0.369	0.510	0.037	0.070	0.197
SVM+ICA+10-best	0.832	0.020	0.040	0.592	0.642	0.026	0.051	0.353	0.747	0.025	0.048	0.431
RF+ICA+10-best	0.441	0.029	**0.056**	0.214	0.669	0.027	0.052	0.364	0.504	0.038	0.072	0.188
NB+ICA+10-best	**0.909**	0.018	0.037	0.698	0.445	0.025	0.048	0.255	0.838	0.020	0.040	0.592

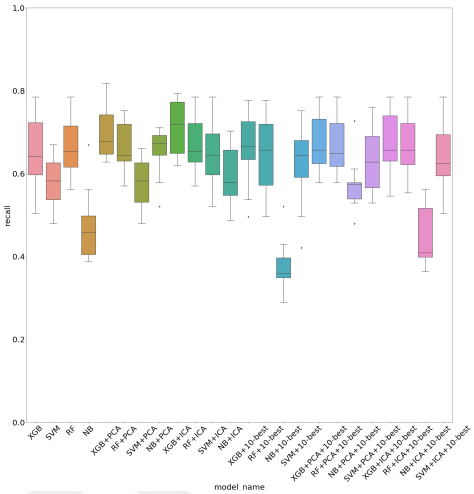
Table 5 summarizes our empirical findings for all trained vulnerability models. Regarding recall, the NB classifier outperforms the different classifiers in SMM (0.89). NB also gets the highest recall rate (0.909) when using the ICA dimensionality reduction technique and the top ten feature selection techniques. However, the inspection ratios are 0.651 and 0.698. That indicates that 65% of all commits must be reviewed to attain that recall rate. Finally, looking at the DPM findings, it is clear that the RF classifier outperforms all other classifiers regarding the recall. Nevertheless, the best recall rate is obtained by combining the XGB classifier and the ICA dimensionality reduction technique (0.710). Out of all the vulnerability prediction models with SMDPM, the NB classifier has the highest recall rate of 0.890. When using dimensionality reduction and/or 10-best feature techniques, combining the NB classifier with the 10-best feature selection method results in the best recall rate of 0.888.

Box plots of the recall, precision, and F1-score values obtained with various random state values for all of our experiments (SMM, DPM, and SMDPM) are displayed in Figures 4.2, 4.3 and 4.4. All of our models have low precision numbers. Some features can make models even more successful. Therefore, the objective of this thesis is not to find the best model, but to examine the influence of the developer profile on vulnerability prediction. Nonetheless, based on Figure 4.3, we may conclude that the SMDPM model delivers the best values. We determine that the method and various strategies do not increase the performance of the DPM model, for which we utilize the developer profile's knowledge. In comparison to SMM, we might claim that DPM is insufficient. The SMDPM model, however, shows improved performance when the two feature sets are combined. This suggests that profile information aids in the prediction of vulnerabilities inducing commits.

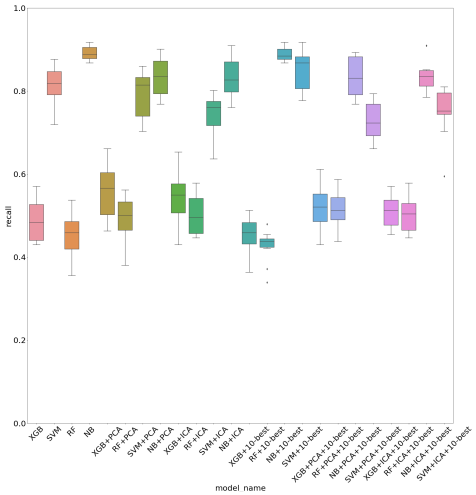
We achieve results with a total of 24 distinct performance metrics using combinations of classifiers, dimensionality reduction and/or 10-best feature selection methods. SMDPM scores better than SMM on the basis of the recall measure in 11 of these 24 possible combinations. The outcomes of further combinations are equally encouraging and demonstrate that the developer profile could enhance vulnerability prediction. We can say that our profiling method can be improved to make it more effective, and it can



((a)) SMM

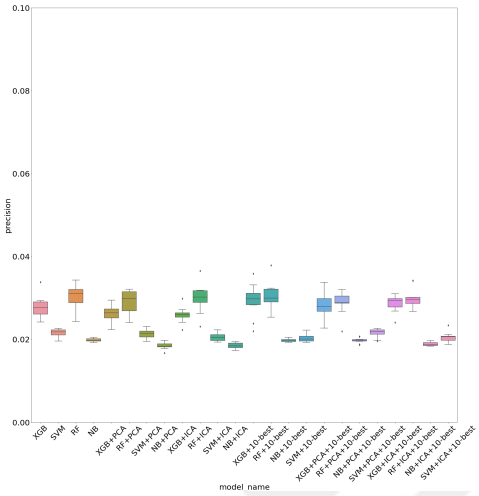


((b)) DPM

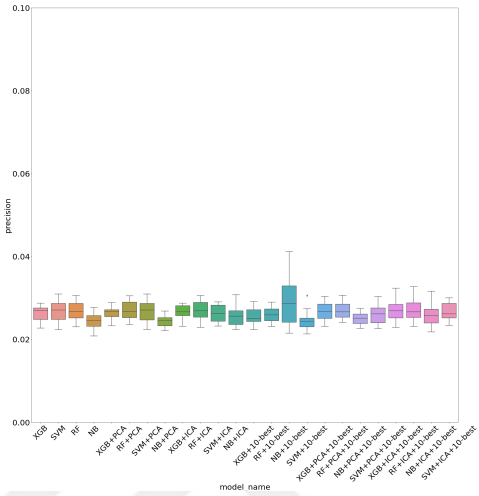


((c)) SMDPM

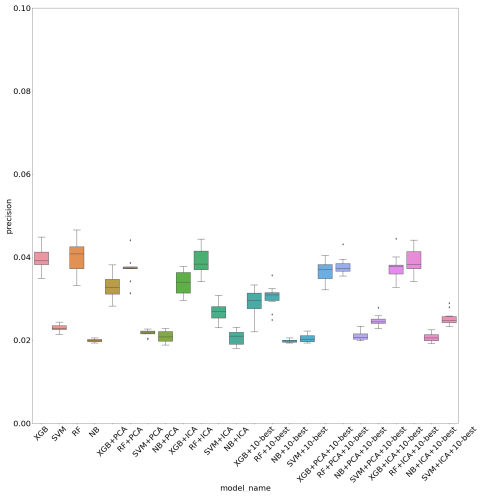
Figure 4.2 : Recall Results.



((a)) SMM

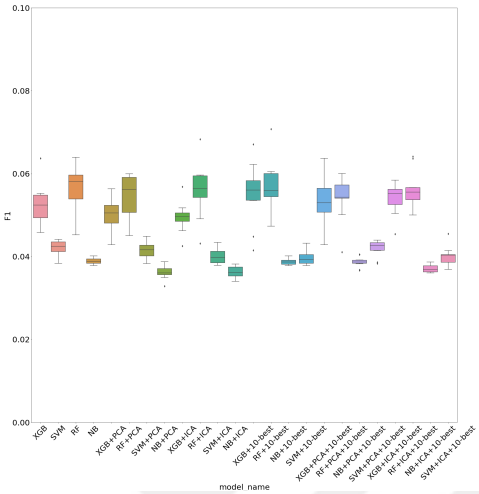


((b)) DPM

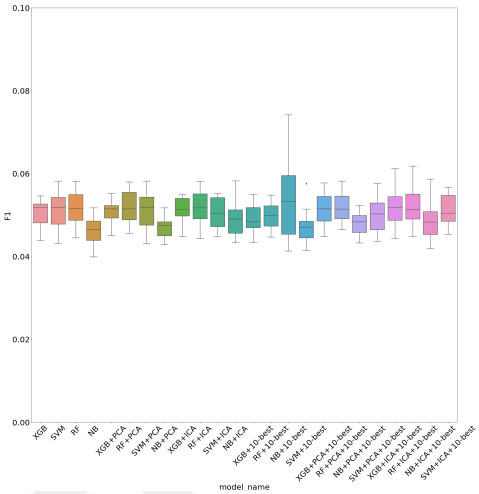


((c)) SMDPM

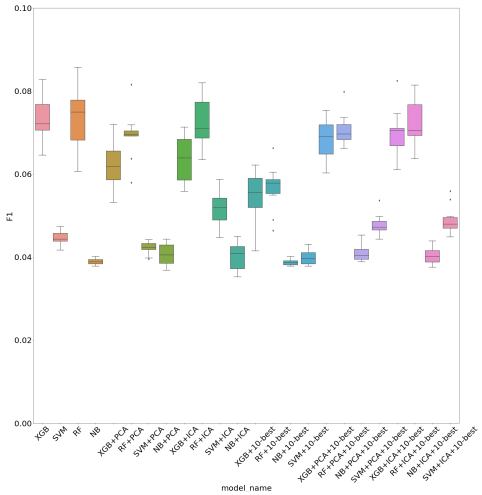
Figure 4.3 : Precision Results.



((a)) SMM



((b)) DPM



((c)) SMDPM

Figure 4.4 : F1-Score Results.

also have a positive contribution when used with other metrics. As a result, we may respond to our RQ by noting that there is empirical evidence to support that developers' coding behaviors that induce vulnerabilities can be used to predict vulnerabilities. More research should be conducted to make generalizations.

In this thesis, we use the RWR method on the HIN we created to determine the developer profile according to the concept of vulnerabilities. When analyzing the effect of the developer profile on vulnerability prediction, developer profiling may be incorporated into various methods. By including more nodes linked to vulnerability creation into our HIN, our technique may also be improved. It is also important to keep in mind how much developers alter projects as time passes. In other words, vulnerability-inducing behaviors change over time on the project that each developer is involved with. We built the data set for this thesis using the actions occurring during specific periods in the software life cycle. Therefore, these coding practices and habits may evolve based on the developer's experience. This implies that scores and rankings for developers' vulnerabilities may alter over time. As a result, the model has to be updated often to reflect the developers' newly acquired abilities and actions.

5. THREATS TO VALIDITY

Internal Validity Using different random state values, we conducted the trials ten times to generate several 70% and 30% splits on the dataset, which was divided into training and testing parts. This approach allowed us to mitigate the potential bias introduced by a single random split. Finally, the average of the experiment's outcomes was presented. Furthermore, we make sure that the data is adequately stratified. Also, we used a statistical test to confirm the differences in each performance evaluation metric across the models. This rigorous analysis helped us establish internal validity and provided a solid foundation for drawing meaningful conclusions from our experiments.

External Validity The SmartSHARK dataset's open-source projects are the only ones that this thesis covers. The dataset has been used in numerous studies [77]–[79], and it is a very comprehensive data source with lots of commitments, pull requests, and emails. One aspect to consider is the lack of a comparative analysis between our thesis results and those of previous research. Since the prediction of vulnerabilities using the SmartSHARK dataset is a relatively novel approach, we were unable to directly compare our findings with prior studies in this specific domain. Nonetheless, we mitigated this limitation by selecting the four most extensive and susceptible Java projects for our analysis, aiming to introduce diversity and increase the robustness of our conclusions. In summary, while the SmartSHARK dataset serves as a valuable resource for our research, the external validity of our conclusions beyond the covered open-source projects is yet to be determined. Future studies should explore the applicability of our findings to business projects, additional open-source projects, and projects developed with different programming languages to establish a broader understanding of vulnerability prediction in software engineering.

Construct Validity In this thesis, we suggest a vulnerability-based profile, however developers may also be profiled in a variety of ways. The profile data may alter

as Vulnerability IDs change. The specific metrics used in our chosen profile are influenced by the nature of the vulnerabilities identified and their associated vulnerability IDs. Consequently, our findings are limited by the selected profile metrics, and alternative metrics may provide additional insights into developer characteristics. To enhance the construct validity of our conclusions and ensure their generalizability, future experiments can be conducted while considering potential changes in profile metrics. By accounting for evolving vulnerability IDs and adapting the profile metrics accordingly, researchers can draw more comprehensive and nuanced conclusions. Additionally, it is worth noting that the SmartSHARK dataset includes software measurements obtained from the SourceMeter tool. This tool captures popular metrics widely used in software engineering analysis. However, it is important to acknowledge that the SourceMeter tool may exclude certain metrics if it cannot recognize them. Consequently, the availability and coverage of software metrics within the SmartSHARK dataset are dependent on the capabilities of the SourceMeter tool. This limitation should be considered when interpreting and generalizing the results based on the dataset. Further research and exploration of alternative profiling approaches and comprehensive metric coverage can contribute to a more robust understanding of developer characteristics and their association with vulnerabilities.

Conclusion Validity Although our findings are restricted to a few projects, we used a variety of classifiers, feature selection and dimensionality reduction, techniques to deliver thorough outcomes.

6. CONCLUSION AND FUTURE WORK

In addition to introducing defects into the system, low-quality codes also provide vulnerabilities that open the approaches to illicit attacks. A number of standards and programming techniques are investigated and recommended for developers in order to produce systems of the highest quality feasible while avoiding software bugs and vulnerabilities. This thesis proposes a developer profile using a HIN based on previous coding activities. By utilizing RWR, we are able to transform the developers' profile into a collection of characteristics that could be inputs into a vulnerability prediction model. Our experimental findings demonstrate that the developer profile is a valuable metric for vulnerability prediction and that, when paired with other indicators, can produce more effective models.

Moreover, our findings demonstrate that developers are more likely to introduce new vulnerabilities into the system if they have a high vulnerability-inducing score for one particular vulnerability type. In other words, the developers' coding practices and skill levels are tied to the possibility of introducing vulnerabilities. As a result, under our methodology, a developer's overall score will be impacted by a high-inducing score for one type. By doing so, we are able to avoid the cold start issue by rating the untested vulnerability types. The method still has a cold start issue with regard to developers, in any case. Since scores are based on previous commits and patterns that lead to vulnerabilities, it is not feasible to provide a score to a new developer that joins the team. As another example, the quantity of time that each developer contributes to the project is not equal, and this can impact the score.

This thesis might be enhanced by enlarging the created HIN and including additional nodes to represent other crucial information about the developers and the source code. For instance, a node that indicates technical debt activities could be included to collect more data about the developer's coding proficiency. Relationships between nodes can be created using other development life cycle data, and weight values can be

determined using a variety of techniques. Activities on open source projects and public platforms might also be utilized to more effectively fight the cold start issue. Also, it is possible to identify the ideal developer for a certain task, project, or team using the obtained developer profile and obtained metrics. It is crucial in applications like team creation, task assignment, and code repair.

REFERENCES

[1] *Common weakness enumeration*, `https://cwe.mitre.org/`.

[2] **McGraw, G.** (2008). Automated code review tools for security, *Computer*, *41*(12), 108–111.

[3] **Telang, R. and Wattal, S.** (2007). An empirical analysis of the impact of software vulnerability announcements on firm stock price, *IEEE Transactions on Software engineering*, *33*(8), 544–557.

[4] **Abelson, H.**, **Anderson, R.**, **Bellovin, S.M.**, **Benaloh, J.**, **Blaze, M.**, **Diffie, W.**, **Gilmore, J.**, **Green, M.**, **Landau, S.**, **Neumann, P.G.** *et al.* (2015). Keys under doormats: mandating insecurity by requiring government access to all data and communications, *Journal of Cybersecurity*, *1*(1), 69–79.

[5] **Shivani, T.**, **Ramakrishna, H. and Nagashree, N.** (2021). Vulnerability Management using Machine Learning Techniques, *2021 IEEE International Conference on Mobile Networks and Wireless Communications (ICMNWC)*, IEEE, pp.1–4.

[6] **Walden, J.**, **Stuckman, J. and Scandariato, R.** (2014). Predicting vulnerable components: Software metrics vs text mining, *2014 IEEE 25th international symposium on software reliability engineering*, IEEE, pp.23–33.

[7] **Shin, Y.**, **Meneely, A.**, **Williams, L. and Osborne, J.A.** (2010). Evaluating complexity, code churn, and developer activity metrics as indicators of software vulnerabilities, *IEEE transactions on software engineering*, *37*(6), 772–787.

[8] **Li, Z. and Shao, Y.** (2019). A Survey of Feature Selection for Vulnerability Prediction Using Feature-based Machine Learning, *ICMLC '19*.

[9] **Shin, Y. and Williams, L.A.** (2008). An empirical model to predict security vulnerabilities using code complexity metrics, *ESEM '08*.

[10] **Nord, R.L.**, **Ozkaya, I.**, **Schwartz, E.J.**, **Shull, F. and Kazman, R.** (2016). Can knowledge of technical debt help identify software vulnerabilities?, *9th Workshop on Cyber Security Experimentation and Test (CSET 16)*.

[11] **Sultana, K.Z.**, **Codabux, Z. and Williams, B.** (2020). Examining the relationship of code and architectural smells with software vulnerabilities, *2020 27th Asia-Pacific Software Engineering Conference (APSEC)*, IEEE, pp.31–40.

[12] **Mumtaz, H.**, **Alshayeb, M.**, **Mahmood, S. and Niazi, M.** (2018). An empirical study to improve software security through the application of code refactoring, *Information and Software Technology*, *96*, 112–125.

[13] **Di Nucci, D.**, **Palomba, F.**, **Siravo, S.**, **Bavota, G.**, **Oliveto, R. and De Lucia, A.** (2015). On the role of developer's scattered changes in bug prediction, *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, IEEE, pp.241–250.

[14] **Di Nucci, D.**, **Palomba, F.**, **De Rosa, G.**, **Bavota, G.**, **Oliveto, R. and De Lucia, A.** (2017). A developer centered bug prediction model, *IEEE Transactions on Software Engineering*, *44*(1), 5–24.

[15] **Hassan, A.E.** (2009). Predicting faults using the complexity of code changes, *2009 IEEE 31st international conference on software engineering*, IEEE, pp.78–88.

[16] **Ostrand, T.J.**, **Weyuker, E.J. and Bell, R.M.** (2010). Programmer-based fault prediction, *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, pp.1–10.

[17] **Bell, R.M.**, **Ostrand, T.J. and Weyuker, E.J.** (2013). The limited impact of individual developer data on software defect prediction, *Empirical Software Engineering*, *18*, 478–505.

[18] **Meneely, A.**, **Williams, L.**, **Snipes, W. and Osborne, J.** (2008). Predicting failures with developer networks and social network analysis, *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, pp.13–23.

[19] **Lee, T.**, **Nam, J.**, **Han, D.**, **Kim, S. and In, H.P.** (2016). Developer micro interaction metrics for software defect prediction, *IEEE Transactions on Software Engineering*, *42*(11), 1015–1035.

[20] **Vishnubhotla, S.D.**, **Mendes, E. and Lundberg, L.** (2018). An insight into the capabilities of professionals and teams in agile software development: A systematic literature review, *Proceedings of the 2018 7th international conference on software and computer applications*, pp.10–19.

[21] **Júnior, R.S.C. and Carneiro, G.d.F.** (2021). An Extended Secondary Study to Characterize the Influence of Developers Sentiments on Practices and Artifacts in Open Source Software Projects, *Enterprise Information Systems: 22nd International Conference, ICEIS 2020, Virtual Event, May 5–7, 2020, Revised Selected Papers*, Springer, pp.339–359.

[22] **Matturro, G.**, **Raschetti, F. and Fontán, C.** (2019). A Systematic Mapping Study on Soft Skills in Software Engineering., *J. Univers. Comput. Sci.*, *25*(1), 16–41.

[23] **Zhao, S.**, **Shen, B.**, **Chen, Y. and Zhong, H.** (2015). Towards Effective Developer Recommendation in Software Crowdsourcing., *SEKE*, pp.326–329.

[24] **Meyer, A.N.**, **Zimmermann, T. and Fritz, T.** (2017). Characterizing software developers by perceptions of productivity, *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, IEEE, pp.105–110.

[25] **Oliveira, E.**, **Conte, T.**, **Cristo, M. and Mendes, E.** (2016). Software project managers' perceptions of productivity factors: findings from a qualitative study, *Proceedings of the 10th ACM/IEEE international symposium on empirical software engineering and measurement*, pp.1–6.

[26] **Ouni, A.**, **Kula, R.G. and Inoue, K.** (2016). Search-based peer reviewers recommendation in modern code review, *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, IEEE, pp.367–377.

[27] **Arias, M.**, **Munoz-Gama, J. and Sepúlveda, M.** (2016). A multi-criteria approach for team recommendation, *International Conference on Business Process Management*, Springer, pp.384–396.

[28] **Mani, S.**, **Padhye, R. and Sinha, V.S.** (2016). Mining API Expertise Profiles with Partial Program Analysis, *Proceedings of the 9th India Software Engineering Conference*, pp.109–118.

[29] **Yan, J.**, **Sun, H.**, **Wang, X.**, **Liu, X. and Song, X.** (2018). Profiling developer expertise across software communities with heterogeneous information network analysis, *Proceedings of the Tenth Asia-Pacific Symposium on Internetware*, pp.1–9.

[30] **Li, J.**, **Bao, L.**, **Xing, Z.**, **Wang, X. and Zhou, B.** (2016). BPMiner: mining developers' behavior patterns from screen-captured task videos, *Proceedings of the 31st Annual ACM Symposium on Applied Computing*, pp.1371–1377.

[31] **Xu, W.**, **Sun, X.**, **Xia, X. and Chen, X.** (2017). Scalable relevant project recommendation on GitHub, *Proceedings of the 9th Asia-Pacific Symposium on Internetware*, pp.1–10.

[32] **Alshammri, M. and Qin, S.** (2017). A hybrid simulation model of individual and team performance in software project environment, *The 22nd International Congress on Modelling and Simulation (MODSIM2017).*

[33] **Ruiz, S.**, **Escudero, D.**, **Cervantes, J. and Trueba, A.** (2017). Assigning-Tasks Method for Developers in Software Projects Using up Similarity Coefficients, *Workshop on Engineering Applications*, Springer, pp.119–128.

[34] **Ye, L.**, **Sun, H.**, **Wang, X. and Wang, J.** (2018). Personalized teammate recommendation for crowdsourced software developers, *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, pp.808–813.

[35] **Santos, K.d.F.**, **Guerrero, D.D. and De Figueiredo, J.C.** (2015). Using Developers Contributions on Software Vocabularies to Identify Experts, *2015 12th International Conference on Information Technology-New Generations*, IEEE, pp.451–456.

[36] **Çetin, H.A. and Tüzün, E.** (2020). Identifying key developers using artifact traceability graphs, *Proceedings of the 16th ACM international conference on predictive models and data analytics in software engineering*, pp.51–60.

[37] **Calefato, F.**, **Lanubile, F. and Vasilescu, B.** (2019). A large-scale, in-depth analysis of developers' personalities in the Apache ecosystem, *Information and Software Technology*, *114*, 1–20.

[38] **Vishnubhotla, S.D.**, **Mendes, E. and Lundberg, L.** (2021). Understanding the perceived relevance of capability measures: A survey of Agile Software Development practitioners, *Journal of Systems and Software*, *180*, 111013.

[39] *Common weakness enumeration: SQL Injection*, `https://cwe.mitre.org/data/definitions/564.html`.

[40] **Bilgin, Z.**, **Ersoy, M.A.**, **Soykan, E.U.**, **Tomur, E.**, **Çomak, P. and Karaçay, L.** (2020). Vulnerability prediction from source code using machine learning, *IEEE Access*, *8*, 150672–150684.

[41] **Meneely, A. and Williams, L.** (2009). Secure open source collaboration: an empirical study of linus' law, *Proceedings of the 16th ACM conference on Computer and communications security*, pp.453–462.

[42] **Codabux, Z. and Dutchyn, C.** (2020). Profiling Developers Through the Lens of Technical Debt, *Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pp.1–6.

[43] **Shin, Y. and Williams, L.** (2008). Is complexity really the enemy of software security?, *Proceedings of the 4th ACM workshop on Quality of protection*, pp.47–50.

[44] **Shin, Y. and Williams, L.** (2008). An empirical model to predict security vulnerabilities using code complexity metrics, *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, pp.315–317.

[45] **Chowdhury, I. and Zulkernine, M.** (2011). Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities, *Journal of Systems Architecture*, *57*(3), 294–313.

[46] **Shin, Y. and Williams, L.** (2013). Can traditional fault prediction models be used for vulnerability prediction?, *Empirical Software Engineering*, *18*(1), 25–59.

[47] **Zhang, Y.**, **Lo, D.**, **Xia, X.**, **Xu, B.**, **Sun, J. and Li, S.** (2015). Combining software metrics and text features for vulnerable file prediction, *2015 20th International Conference on Engineering of Complex Computer Systems (ICECCS)*, IEEE, pp.40–49.

[48] **Çaglayan, B. and Bener, A.B.** (2016). Effect of developer collaboration activity on software quality in two large scale projects, *Journal of Systems and Software*, *118*, 288–296.

[49] **Moradi Dakhel, A.**, **C. Desmarais, M. and Khomh, F.**, (2021). Assessing developer expertise from the statistical distribution of programming syntax patterns, Evaluation and Assessment in Software Engineering, pp.90–99.

[50] **Kortum, F.**, **Karras, O.**, **Klünder, J. and Schneider, K.** (2019). Towards a Better Understanding of Team-Driven Dynamics in Agile Software Projects: A Characterization and Visualization Support in JIRA, *Product-Focused Software Process Improvement: 20th International Conference, PROFES 2019, Barcelona, Spain, November 27–29, 2019, Proceedings 20*, Springer, pp.725–740.

[51] **Costa, A.A.M.**, **Ramos, F.B.A.**, **Perkusich, M.**, **Freire, A.S.**, **Almeida, H.O. and Perkusich, A.** (2018). A Search-based Software Engineering Approach to Support Multiple Team Formation for Scrum Projects., *SEKE*, pp.474–473.

[52] **Zanjani, M.B.** (2016). Effective assignment and assistance to software developers and reviewers, *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp.1091–1093.

[53] **Costa, C.**, **Figueiredo, J.**, **Murta, L. and Sarma, A.** (2016). TIPMerge: recommending experts for integrating changes across branches, *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pp.523–534.

[54] **Sun, X.**, **Yang, H.**, **Xia, X. and Li, B.** (2017). Enhancing developer recommendation with supplementary information via mining historical commits, *Journal of Systems and Software*, *134*, 355–368.

[55] **Bhat, M.**, **Shumaiev, K.**, **Koch, K.**, **Hohenstein, U.**, **Biesdorf, A. and Matthes, F.** (2018). An expert recommendation system for design decision making: Who should be involved in making a design decision?, *2018 IEEE International Conference on Software Architecture (ICSA)*, IEEE, pp.85–8509.

[56] **Khatun, A. and Sakib, K.** (2017). A team allocation technique ensuring bug assignment to existing and new developers using their recency and expertise, *SOFTENG 2017: The Third International Conference on Advances and Trends in Software Engineering)*.

[57] **Soares, D.M.**, **de Lima Júnior, M.L.**, **Plastino, A. and Murta, L.** (2018). What factors influence the reviewer assignment to pull requests?, *Information and Software Technology*, *98*, 32–43.

[58] **Zhao, X.**, **Yang, Y. and Yin, M.** (2020). MHRWR: Prediction of lncRNA-disease associations based on multiple heterogeneous networks, *IEEE/ACM Transactions on Computational Biology and Bioinformatics*.

[59] **Huang, W.**, **Mo, W.**, **Shen, B.**, **Yang, Y. and Li, N.** (2016). CPDScorer: Modeling and Evaluating Developer Programming Ability across Software Communities., *SEKE*, pp.87–92.

[60] **Alexander Trautsch, Fabian Trautsch, S.H.** (2021). *The SmartSHARK Repository Mining Data*, `2102.11540`.

[61] **Ponta, S.E.**, **Plate, H.**, **Sabetta, A.**, **Bezzi, M. and Dangremont, C.** (2019). A manually-curated dataset of fixes to vulnerabilities of open-source software, *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, IEEE, pp.383–387.

[62] **Sliwerski, J.**, **Zimmermann, T. and Zeller, A.** (2005). When do changes induce fixes?, *ACM SIGSOFT Software Engineering Notes*, *30*, 1 – 5.

[63] (2022). *Vulnerabilty Prediction - Google Drive*, `https://drive.google.com/drive/folders/1s1HLh8t7Yk0z1SVLQ20e8kK_37BJi_Rd`, (Accessed on 21/04/2022).

[64] **Shi, C.**, **Li, Y.**, **Zhang, J.**, **Sun, Y. and Yu, P.S.** (2017). A Survey of Heterogeneous Information Network Analysis, *IEEE Transactions on Knowledge and Data Engineering*, *29*, 17–37.

[65] **Shi, C.**, **Li, Y.**, **Yu, P.S. and Wu, B.** (2016). Constrained-meta-path-based ranking in heterogeneous information network, *Knowledge and Information Systems*, *49*(2), 719–747.

[66] **Yao, Y.**, **Ji, B.**, **Lv, Y.**, **Li, L.**, **Xiang, J.**, **Liao, B. and Gao, W.** (2021). Predicting LncRNA–Disease Association by a Random Walk With Restart on Multiplex and Heterogeneous Networks, *Frontiers in Genetics*, *12*.

[67] **Yao, Y.**, **Ji, B.**, **Lv, Y.**, **Li, L.**, **Xiang, J.**, **Liao, B. and Gao, W.** (2021). Predicting LncRNA–Disease Association by a Random Walk With Restart on Multiplex and Heterogeneous Networks, *Frontiers in Genetics*, 1483.

[68] **Wang, L.**, **Shang, M.**, **Dai, Q. and He, P.a.** (2022). Prediction of lncRNA-disease association based on a Laplace normalized random walk with restart algorithm on heterogeneous networks, *BMC bioinformatics*, *23*(1), 1–20.

[69] **Li, L.**, **Wang, Y.**, **An, L.**, **Kong, X. and Huang, T.** (2017). A network-based method using a random walk with restart algorithm and screening tests to identify novel genes associated with Menière's disease, *PLoS One*, *12*(8), e0182592.

[70] **Lovász, L.** (1993). Random walks on graphs, *Combinatorics, Paul erdos is eighty*, *2*(1-46), 4.

[71] **Page, L.**, **Brin, S.**, **Motwani, R. and Winograd, T.** (1999). The PageRank Citation Ranking : Bringing Order to the Web, *WWW 1999*.

[72] **Wen, Y.**, **Song, X.**, **Yan, B.**, **Yang, X.**, **Wu, L.**, **Leng, D.**, **He, S. and Bo, X.** (2021). Multi-dimensional data integration algorithm based on random walk with restart, *BMC bioinformatics*, *22*(1), 1–22.

[73] **Chen, T. and Guestrin, C.** (2016). XGBoost: A Scalable Tree Boosting System, *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

[74] **Suykens, J.A. and Vandewalle, J.** (1999). Least squares support vector machine classifiers, *Neural processing letters*, *9*(3), 293–300.

[75] **Belgiu, M. and Drăguţ, L.D.** (2016). Random forest in remote sensing: A review of applications and future directions, *Isprs Journal of Photogrammetry and Remote Sensing*, *114*, 24–31.

[76] **Watson, T.J.** (2001). An empirical study of the naive Bayes classifier.

[77] **Moharil, A.**, **Orlov, D.**, **Jameel, S.**, **Trouwen, T.**, **Cassee, N. and Serebrenik, A.**, (2022). Between JIRA and GitHub: ASFBot and its Influence on Human Comments in Issue Trackers, Mining Software Repositories.

[78] **Bagheri, A. and HegedHus, P.** (2022). Is Refactoring Always a Good Egg? Exploring the Interconnection Between Bugs and Refactorings, *2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR)*, IEEE, pp.117–121.

[79] **Khoshnoud, F.**, **Nasab, A.R.**, **Toudeji, Z. and Sami, A.** (2022). Which bugs are missed in code reviews: An empirical study on SmartSHARK dataset, *2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR)*, IEEE, pp.01–05.

CURRICULUM VITAE

Name SURNAME: Tuğçe COŞKUN

EDUCATION:

- **B.Sc.:** 2017, Beykent University, Faculty of Architecture and Engineering, Computer Engineering
- **M.Sc.:** 2023, Istanbul Technical University, Faculty of Computer and Informatics Engineering, Computer Engineering

PROFESSIONAL EXPERIENCE AND REWARDS:

- January 2019–November2020 Ericsson, Software Developer
- March 2018–January 2019 Derindere Fleet Leasing, Software Developer

PUBLICATIONS, PRESENTATIONS AND PATENTS ON THE THESIS:

- **Coskun T.** , Halepmollasi R., Hanifi K., Fouladi R.F., Comak De Cnudde P., Tosun A. (2022). Profiling Developers to Predict Vulnerable Code Changes, *In Proceedings of the 18th International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE '22),*