Mühendislik Bilimleri ve Tasarım Dergisi
10(2), 561 – 573, 2022
e-ISSN: 1308-6693

Journal of Engineering Sciences and Design
DOI: 10.21923/jesd.1031774

Araştırma Makalesi Research Article

C PROGRAMLAMA DİLİNDE KAYNAK KOD GÜVENLİĞİ: SECUREC

Meltem KURT PEHLİVANOĞLU[1*], Sinan ÇALIŞIR[2], Ceren GENÇ[1], Duygu Evrim ODABAŞ[1], Berkehan ÖZTÜRK[1]

[1] Kocaeli Üniversitesi, Mühendislik Fakültesi, Bilgisayar Mühendisliği Bölümü, Kocaeli, Türkiye
[2] Revinate, Amsterdam, Hollanda

Anahtar Kelimeler

Yazılım Güvenliği, Yazılım Güvenlik Açığı, Doğal Dil İşleme, Makine Öğrenmesi, Yapay Zekâ.

Öz

Yazılım güvenliğinin temel amacı, yazılımı kötü niyetli siber saldırılara karşı dayanıklı olacak şekilde tasarlamaktır. Yazılım güvenliği, yazılım yaşam döngüsünün her aşamasında ele alınmazsa, birçok işletme, kuruluş ve hatta hükümetler, yazılım sistemlerindeki güvenlik açıkları nedeniyle sömürülür. Güvenli yazılım sistemleri geliştirmede kaynak kodun analiz edilmesi önemli bir adımdır. Güvenli olmayan kaynak kodu bölümlerinin belirlenmesi, yazılımdaki güvenlik açıklarının azaltılmasına veya kaldırılmasına yardımcı olur. C programlama dili, en yaygın kullanılan programlama dillerinden biridir, ancak güvenli değildir ve kod yerleştirme saldırılarına karşı savunmasızdır. C programlama dilinin en yaygın güvenlik açıkları arabellek taşmaları, giriş doğrulamaları ve kaynak yönetimi hatalarıdır. Bu çalışmada, C programlama dilindeki güvensiz kaynak kodlarını tespit edebilen SecureC adlı bir yazılım aracı geliştirilmiştir. İlk olarak düzenli ifadeler kullanılarak yeni kod şablonları çıkarılmış, ardından bu yeni şablonlar kullanılarak veri setindeki kaynak kodlar işaretlenmiştir. Daha sonra yüzbinlerce satıra sahip kaynak koddan güvensiz kısımları tespit ederken işaretli veri kümesi ile işaretsiz veri kümesi arasında performans farkı olup olmadığına odaklanılmıştır. Sonuç olarak, işaretli bir veri kümesinin kullanılması hem ikili hem de çoklu sınıflandırmada performans artışı sağlamıştır.

SOURCE CODE SECURITY IN C PROGRAMMING LANGUAGE: SECUREC

Keywords

Software Security, Software Vulnerability, Natural Language Processing, Machine Learning, Artificial Intelligence.

Abstract

The principal goal of software security is to design software to be resistant to malicious cyber-attacks. If software security is not handled at every stage of the life cycle of software, many businesses, organizations, and even governments are exploited due to security vulnerabilities in their software systems. Analyzing the source code in developing secure software systems is an important step. Identifying the insecure source code parts helps to reduce or remove vulnerabilities in software. The C programming language is one of the most-commonly used programming languages, but it is insecure and vulnerable to code injection attacks. C programming language's most common security vulnerabilities are buffer overflows, input validations, and resource management errors. In this paper, a software tool called SecureC has been developed, which can detect insecure source codes in the C programming language. Firstly, the new source code templates using the Regular expressions (Regex) were extracted, then the source codes in the dataset by using these new templates were marked. Then, it has been focused on finding whether there is any performance difference between the marked dataset and the unmarked dataset while detecting insecure parts from the source code that has hundreds of thousands of lines. As a result, the usage of a marked dataset ensured the performance improvement in both binary and multiple classification.

Alıntı / Cite

Kurt Pehlivanoğlu, M., Çalışır, S., Genç, C., Odabaş, D.E., Öztürk, B. (2022). C Programlama Dilinde Kaynak Kod Güvenliği: SecureC, Mühendislik Bilimleri ve Tasarım Dergisi, 10(2), 561-573.

[*] İlgili yazar / Corresponding author: meltem.kurt@kocaeli.edu.tr, +90-262-303-3571

Yazar Kimliği / Author ID (ORCID Number)	Makale Süreci / Article Process	
M. Kurt Pehlivanoğlu, 0000-0002-7581-9390	**Başvuru Tarihi / Submission Date**	02.12.2021
S. Çalışır, 0000-0001-7600-3708	**Revizyon Tarihi / Revision Date**	10.03.2022
C. Genç, 0000-0003-4110-7018	**Kabul Tarihi / Accepted Date**	17.03.2022
D. E. Odabaş, 0000-0002-4093-1366	**Yayım Tarihi / Published Date**	30.06.2022
B. Öztürk, 0000-0001-6602-1347		

1. Giriş (Introduction)

Yazılım yaşam döngüsünün temel aşamaları; planlama, analiz, tasarım, gerçekleştirim (kodlama ve test) ve bakımdır. Temelinde bu adımları içeren farklı yazılım yaşam döngüsü modelleri kullanılırken, yazılımın saldırılara karşı dirençli ve kusursuz tasarlanması beklenir. Yazılım güvenliğinin önemsenmediği durumlarda; kişiler, işletmeler ve hatta hükümetler güvenlik zafiyeti kaynaklı siber saldırılarla karşı karşıya kalmaktadır. Gerçekleşen birçok siber saldırının sebebi uygulama düzeyindeki güvenlik açıklarından kaynaklanmaktadır. Bu saldırıları engellemenin yöntemlerinden biri de özellikle kaynak kodlarındaki güvenlik zafiyetlerinin analiz edilmesi ve bu zafiyetlerin ortadan kaldırılmasıdır.

Yazılım zafiyetlerinin otomatik olarak tespit edilmesi, önemli bir araştırma problemidir. Ancak, bu sorunu çözmeye yönelik yazılım zafiyet tespit araçları kullanılmadığı durumlarda, kaynak kodunun güvenlik analizi insan gücüne bağlı kalmaktadır. Bu nedenle birçok güvenlik zafiyeti (özel uzmanlık alanı gerektirmesi, insan kaynaklı zafiyetlerin gözden kaçması vb. durumlar kaynaklı) tespit edilememektedir. En temel yazılım dillerinden biri olan C programlama dili, derlendiği zaman statik kod ortaya çıkardığı için güvenlik zafiyetlerine açıktır. Bu çalışma kapsamında, C programlama dili kaynak kodlarındaki güvenlik zafiyetlerinin tespiti amaçlı doğal dil işleme ve derin öğrenme yöntemleri kullanılarak, mevcut çözümlerdeki insana bağlılığı ortadan kaldırmak amaçlı SecureC adı verilen bir araç geliştirilmiştir.

Literatürde yer alan kaynak kod güvenlik zafiyetlerinin tespiti amaçlı yapılmış çalışmalar incelendiğinde, (Xu vd. 2018) modelin başarısını arttırmak amaçlı veri kümesinde yer alan kaynak kodların belirli şablonlara göre işaretlendiği görülmüştür. Bu kapsamda bu çalışmada, C programlama dili kaynak kodları için, detaylı işaretleme yapılabilmesi amaçlı düzenli ifadeler kullanılarak, kaynak kod üzerinde ortak şablonlar çıkarılmıştır. Veri kümesi olarak kullanılan C kaynak kodları, çıkarılan ortak şablonlara oturtturularak, yapay zekâ modeline girdi olarak verilebilecek ortak bir yapı (form) elde edilmiştir. Veri kümesi üzerinde zararlı ve zararsız kaynak kodlarının tespiti amaçlı yapay bir Yinelemeli Sinir Ağı (Recurrent Neural Network-RNN) mimarisi olan Uzun Kısa Vadeli Hafıza Ağları (Long-Short Term Memory-LSTM) modeli kullanılmıştır. Çalışmada, işaretli veri kümesi ve işaretsiz veri kümesi üzerinde ikili (zararlı/zararsız) ve çoklu sınıflandırma (zararsız/zararlı (farklı zafiyet tiplerini içermektedir)) gerçekleştirilmiştir. Literatürde yer alan çalışmalar incelendiğinde Kim ve diğerlerinin (Kim vd. 2019) yaptıkları çalışmada veri kümesinin eksikliği sıkıntısından bahsedilmiştir. SecureC modelinde kullanılan veri kümesi daha başarılı sonuçlar elde etmek için genişletilmiştir.

1.1. Motivasyon ve Katkı (Motivation and Contribution)

C programlama dili güvenlik zafiyeti oluşturan ancak günümüzde birçok problemde kullanılan bir programlama dilidir. Bu kapsamda özellikle C programlama dili özelinde kaynak kod güvenliği kontrolünü yapabilecek yazılım araçlarının geliştirilmesi önemlidir. Literatürde yapılan çalışmalar incelendiğinde, LSTM modellerinden başarılı sonuçlar elde edildiği görülmüş ve veri kümesinin yetersizliği açık bir problem olarak bırakılmıştır. Bu çalışmada, düzenli ifadeler kullanılarak kaynak kodlara ait ortak şablonlar çıkarıp, LSTM tabanlı zararlı ve zararsız kaynak kod analizi yapabilen SecureC olarak adlandırılan yeni bir yazılım aracı geliştirilmiştir.

SecureC modeli ile ikili sınıflandırma yapılırken farklı veri kümeleri birleştirilmiş ve daha başarılı sonuçlar alınması hedeflenmiştir. Kelime vektörlerinin oluşturulması aşamasında Github üzerinden açık kaynak kodlar alınarak hem veri kümesinin çeşitliliği arttırılmış hem de sabit bir veri kümesine bağlı kalmadan Github üzerinde yer alan bu kodlardaki güvenlik açıkları tespit edilmiştir. Bu çalışmalar ile literatürde yer alan veri kümesinin yetersizliği sorununun giderilmesi hedeflenmiştir.

Çalışma bu yönleriyle özgün olup, çalışmaya ait kaynak kodları erişime açık olarak paylaşıldığından diğer çalışmalar için yol gösterici, geliştirmeye açık bir model olacaktır.

1.1. Organizasyon (Organization)

Çalışmanın ikinci bölümünde, kaynak kod güvenlik zafiyetlerinin tespitinde literatürde yapılan çalışmalar verilerek açık problemler karşılaştırmalı olarak ele alınmıştır. Üçüncü bölümde, çalışma kapsamında önerilen

SecureC modeline ait kapsamlı bilgiler verilmiş olup, dördüncü bölümde elde edilen deneysel sonuçlar karşılaştırılmıştır. Son bölümde ise elde edilen sonuçlar tartışılarak ileriki çalışmalar verilmiştir.

2. İlgili Çalışmalar (Related Works)

Literatürde yer alan kaynak kod güvenliği analizi ile ilgili çalışmalar araştırılarak, bu çalışmalara ait kapsamlı bilgiler aşağıda verilmiştir.

Kaur ve arkadaşları çalışmalarında (Kaur vd. 2020), kaynak kod güvenlik zafiyetlerinin tespitinde statik kod analizi için otomatik kod tarama araçlarının kullanımı üzerine çalışmışlardır. Çalışmada veri kümesi olarak, C ve C++ programlama dilleri için Juliet, Java programlama dili için ise APACHE Tomcat kullanılmıştır. Kaynak kod güvenlik zafiyetlerinin tespiti problemi ikili sınıflandırma problemine indirgenmiştir. Statik kod analizinde C ve C++ programlama dilleri için FLAWFINDER, RATS, CPPCHECK, Java programlama dili için ise SPOTBUGS, PMD araçları kullanılmıştır. Güvenlik zafiyetinin bulunmasında en yüksek performans Soyut Sözdizimi Ağacı (Abstract Syntax Tree) yöntemi ile elde edilmiştir. Çalışma sonucunda kullanılan araçların bazı güvenlik zafiyetlerini tespit edemediği görülmüştür.

Li ve arkadaşları çalışmalarında (Li vd. 2020), otomatikleştirilmiş akıllı güvenlik açığı tespit etmek için kaynak kod temelli minimum Ara Temsil (IR) öğrenim modeli geliştirmişlerdir. SARD ve NVD veri kümelerinden CWE-119 ve CWE-399 hatalarının örneklerini, savunmasız programlama örüntülerini bulmak için kullanmışlardır. Derin öğrenme yöntemleri için; Ardışık Evrişimsel Sinir Ağları (Convolutional Neural Network-CNN) %75.9, Ardışık LSTM %69.1, Çift Yönlü Uzun Kısa Vadeli Hafıza Ağları (Bi-LSTM) %75.7, CNN+Bi-LSTM %84.9, Birleştirilmiş CNN %89.5 F1 skorları elde edilmiştir. Sınıflandırma algoritmaları için; Lineer Regresyon % 90.5, Naive Bayes %72.8, SVM %92.6, Çok Katmanlı Algılayıcı (MLP) %89.5, Gradyan Artıran Karar Ağacı (GBDT) %92.2, Rastgele Orman %93.0 F1 skorlarını elde etmişlerdir.

Verma ve arkadaşları çalışmalarında (Verma vd. 2020), doğal dil işleme yöntemleri kullanılarak kimlik avı e-postaları sınıflandırmışlardır. Çalışmada, K-En Yakın Komşu (K-Nearest Neighbors-KNN), Rastgele Orman, Lojistik Regrasyon, SGD, Naive Bayes, Destek Vektör Makinesi (Support Vector Machine-SVM) ve Doğrusal Sınıflandırma modelleri üzerinde sırasıyla 94.75, 97.55, 98.42, 98.56, 98.34, 98.70, 98.77 doğruluk oranları elde edilmiştir.

Suneja ve arkadaşları çalışmalarında (Suneja vd. 2019), C programlama dili kaynak kodları üzerinde güvenlik zafiyetlerinin tespiti amaçlı Graf Sinir Ağı (Graph Neural Network-GNN) uygulanabilirliğini araştırılmıştır. AI4VA olarak adlandırdıkları boru hattı üzerinden, GNN çıktısı alınıp, alınan çıktının anlamsal bilgisi korunarak vektörleştirilmiştir. AI4VA boru hattı yardımı ile C fonksiyonları zararlı ve zararsız olarak sınıflandırılmıştır. Çalışmada Juliet, Draper ve sbAbI veri kümeleri üzerinde Evrişimli Sinir Ağları (CNN) ve RNN derin öğrenme modelleri kullanılarak sınıflandırma yapılmıştır. Draper veri kümesi üzerinde CNN modeli ile 0.53 F1 skoru elde etmişlerdir.

Kim ve arkadaşları çalışmalarında (Kim vd. 2019), kaynak kodunun modellenmesi için makine öğrenimi yaklaşımlarından faydalanmışlardır. Sözdizimi, semantik ve kaynak akışından yararlanarak büyük kod parçalarının üstesinden gelmek ve mevcut statik yazılım analiz araçlarının kaçırdığı olası güvenlik zafiyetlerinin tespit edilebilmesi hedeflenmiştir. Çalışmada FFmpeg, LibPNG ve LibTiff veri kümeleri üzerinde Bi-LSTM modeli ve sınıflandırma yöntemi olarak '1-vs-Rest Layer' stratejisi ile çalışılmıştır. Çalışmanın kalitesinin ve performansının arttırılabilmesi için daha fazla açık kaynak kodunun toplanması ve hiper parametrelerin çoğaltılması gerektiği vurgulanmıştır.

Li ve arkadaşları çalışmalarında (Li vd. 2019), kaynak kod güvenlik zafiyetlerinin tespitinde derin sinir ağı yöntemi kullanmışlardır. Büyük ölçekli açık kaynak kodlarındaki zayıf fonksiyonları ortaya çıkarmak için semantik özellikler olarak fonksiyon adlarını kullanmışlardır. Fonksiyon isimlerini çıkarıp, zararlı ve zararsız olarak gruplandırmışlardır. Fonksiyon adlarını birbirinden ayırmak için CWE güvenlik zafiyetlerine odaklanmışlardır. Kullandıkları derin sinir ağı tabanlı yöntemden elde edilen deneysel sonuçlara göre C/C++ ve Python programlama dillerinde sırasıyla 0.91 ve 0.915 F2 skoru elde etmişlerdir.

Larrucea ve arkadaşları çalışmalarında (Larrucea vd. 2019), OpenNCP platformunun güvenlik analizini sağlama ve platformun kaynak kodunda güvenli kalıpları belirleme üzerine çalışmışlardır. Analizin ana sonuçlarından biri, güvenlik ile ilgili OpenNCP modüllerinin en fazla güvenlik açığına sahip olanlar olmasıdır. Bu nedenle, bir üye devlet, ulusal sağlık sistemi hizmetlerini geri kalan sistemlerle birleştirmek için bu platformu kullanmayı hedefliyorsa, ek önlemler gerektiği sonucuna varmışlardır. Ayrıca en yaygın iki zayıflığı sırasıyla CWE-493 ve CWE-798 olarak belirlemişlerdir.

Xu ve arkadaşları çalışmalarında (Xu vd. 2018), kaynak kod güvenlik zafiyetlerinin tespitinde Bağlamsal Uzun Kısa Süreli Bellek (CLSTM) modelinin uygulanabilirliği araştırılmıştır. Çalışmalarını, C ve C++ özelinde doğal dil işleme yöntemlerini kullanarak yapmışlardır. Kod içerisinde değişken isimleri var1, var2 vb., fonksiyon isimleri ise func1, func2 vb. şekilde standartlaştırılmıştır. Çalışmalar SARD veri kümesi ile sadece CWE-119 ve CWE-399 zafiyetleri üzerinde yapılmıştır. Farklı modellerle yapılan çalışmalar sonucunda CNN modeli için 0.95821, LSTM modeli için 0.95902, Bağlamsal Uzun Kısa Vadeli Hafıza Ağları (Contextual Long-Short Term Memory-CLSTM) modeli içinse 0.96711 doğruluk oranlarını elde edilmiştir. Yapılan deneysel çalışmalar ile CLSTM modelinin zafiyet tespiti için en iyi model olduğu sonucuna varılmıştır.

Xiaomeng ve arkadaşları çalışmalarında (Xiaomeng vd. 2018), zararlı kaynak kodu incelemeleri için derin öğrenme yöntemlerinin ve temel kod özelliği grafiğinin kullanımı üzerine çalışmalar yapılmıştır. Çalışmada, Kod Özellikli Graf (Code Property Graph-GPG), NLP ve derin öğrenme yöntemleri karışık olarak kullanılmıştır. SARD veri kümesi üzerinde yapılan çalışmalarda, CPGVA ile CNN, Zamansal Evrişimli Sinir Ağı (TCNN), LSTM ve LeNet modellerinin kullanılması ile sırasıyla: 91.1, 90.7, 90.7, 88.9 F1 skorları elde edilmiştir. Vudeepecker modeli ile ise 86.6 F1 skoru elde edilmiştir. Çalışma sonucunda CNN tabanlı yöntemlerin RNN tabanlı yöntemlerden daha iyi performans gösterdiği sonucuna varılmıştır.

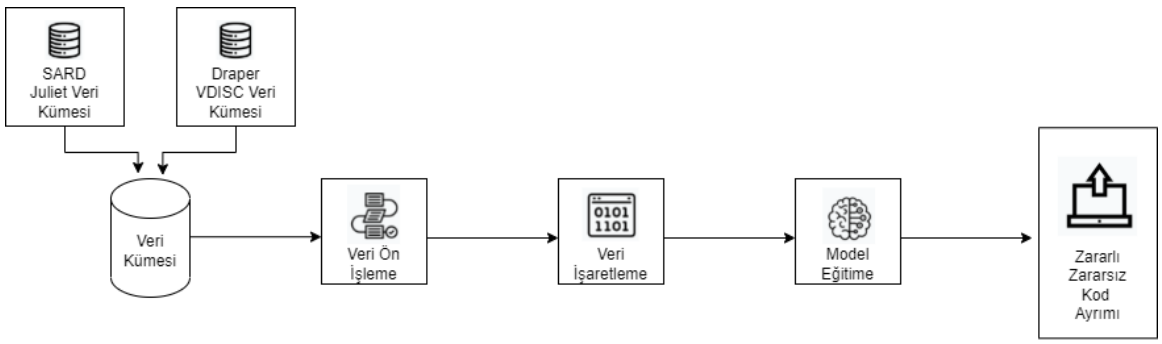
Tian ve arkadaşları çalışmalarında (Tian vd. 2020), C ve C++ dillleri için kütüphane ve API method çağırımlarının olduğu İkili Kapılı Tekrarlayan Birim (Binary Gated Recurrent Unit-BGRU) ağ modeli temelli BV-Detector adını verdikleri bir model geliştirmişlerdir. Oluşturdukları model ile ikili (binary) programlarda zafiyet tespiti yapmışlardır. Çalışmada Software Assurance Reference Dataset (SARD) (SARD, 2021) veri kümesi kullanılmıştır. Veriler 3 test kümesi olarak ele alınmıştır. Toplamda 17,466 güvenlik açığı olan, 66,093 güvenlik açığı olmayan veri üzerinde çalışmışlardır. Verilerin %80'ini eğitim, %20'sini ise test için kullanmışlardır. Kodlardan belirli parçaları çıkarabilmek için Angr isimli ikili analiz araçları kullanılmıştır. Kelime vektörlerinin oluşturulması için Word2Vec kullanarak toplamda 92,568 adet token elde edilmiştir. BGRU modelinde en iyi F1 skor değeri 89.9 iken, LSTM modelinde ise bu değer 85.9 F1 olarak elde edilmiştir.

Cao ve arkadaşları çalışmalarında (Cao vd. 2021), C ve C++ dillerinde zafiyet tespiti için BGNN4VD adını verdikleri bir model geliştirmişlerdir. Bu modelde kodun semantik özellikleri ve sözdizimini anlayabilmek için Soyut Sözdizimi Ağacı, Kontrol Akış Grafiği ve Veri Akış Grafiği kullanılırken, kelime vektörlerinin temsili için Word2Vec kullanılmıştır. Toplamda 2,149 adet güvenlik açığı olan veriyi NVD (National Vulnerability Database) ve Github açık kaynak kodları kullanarak toplamışlardır. Graf yapısını bozabileceği için kod içerisindeki tüm açıklama satırları kaldırılarak, manuel etiketleme gerçekleştirmişlerdir. Etiketleme tutarlılığını karşılaştırmak için işaretlenen her %10 veride bir Cohen'in Kappa katsayısı kullanılmıştır. 3,867 method zafiyetli, 92,058 method güvenli olarak işaretlenmiştir. Veri kümelerini herkesin erişimine açık olarak github üzerinden paylaşmışlardır. Modelde %74.4 doğruluk, %76.3 duyarlılık, %77.3 kesinlik ve %76.8 F1 skor değerleri elde edilmiştir.

Yukarıda verilen, kaynak kod kaynaklı zafiyetlerin önlenmesi amaçlı literatürde yer alan çalışmalar incelendiğinde özellikle LSTM modellerinin başarılı sonuçlar elde ettiği görülmüştür. Bu kapsamda bu çalışmada, zararlı ve zararsız kaynak kodların sınıflandırılması için derin öğrenme tabanlı LSTM modeli kullanılmıştır.

3. Önerilen Model: SecureC (Proposed Method: SecureC)

Çalışma kapsamında önerilen ve C programlama dili için zararlı kaynak kodların tespit edilmesini sağlayan SecureC yapay zekâ modeline ait sistem mimarisi Şekil 1'de verilmiştir.



Şekil 1. Sistem mimarisi (System architecture)

Şekil 1'de verilen SecureC'ye ait sistem mimarisi incelendiğinde, zararlı ve zararsız kaynak kodlarının sınıflandırılacağı LSTM tabanlı model eğitilmeden önce ilk adım veri kümesinin oluşturulması adımıdır. Çalışmada

SARD veri kümesi altında verilen test kümelerinden (SARD Test Suites) Juliet veri kümesi (Juliet Test Suite for C/C++, 2021) ile Draper VDISC (Draper VDISC Dataset, 2021) veri kümeleri birleştirilerek hibrit bir veri kümesi oluşturulmuştur. İkinci adımda ise, veri kümesinde yer alan C kaynak kodları veri ön işlemeden geçirilmiş, daha sonra bu kaynak kodları üzerinde ortak bir şablon oluşturulmak amaçlı düzenli ifadeler kullanılarak işaretlemeler gerçekleştirilmiştir. Bu işlemlerin yanı sıra hem işaretli hem de işaretsiz kelime temsil modelleri oluşturulmuştur. İşaretsiz gözlemlere ait LSTM modelinin eğitimi için işaretsiz verilerden oluşturulan kelime temsil modeli kullanılırken, işaretli gözlemlerin ait modelin eğitimi için ise işaretlenmiş verilerden oluşan kelime temsil modeli kullanılmıştır. Daha sonra LSTM modeli eğitilerek, zararlı ve zararsız kaynak kodlarının tespiti gerçekleştirilmiştir.

3.1. Veri Kümesi (Dataset)

Çalışma kapsamında, SARD-Juliet veri kümesi, Draper VDISC veri kümesi ve Github repoları üzerinden toplanan C kaynak kodları birleştirilerek hibrit bir veri kümesi oluşturulmuştur.

3.1.1. SARD-Juliet Veri Kümesi (SARD-Juliet Dataset)

SARD veri kümesi, Ulusal Standartlar ve Teknoloji Enstitüsü (National Institute of Standards and Technology-NIST) tarafından oluşturulan, farklı programlama dillerinde yazılan kod parçacıkları üzerinde zafiyete sebep olacak kod parçalarının etiketli olarak verildiği veri kümesidir. Bu veri kümesi içinde C, C++, Java, PHP ve C# programlama dillerinde yazılmış sırasıyla 170,000'in üzerinde farklı kaynak kodu ve bu kodlar içerisinde yer alan 150'den fazla güvenlik zafiyeti sınıfları etiketlenmiştir. Veri kümesi içinde her bir zayıflık Ortak Zayıflık Numaralandırması (Common Weakness Enumeration-CWE) kullanılarak (CWE-ID) gösterimiyle temsil edilmiştir. Veri kümesi içerisinde her bir programlama dili için ayrı örneklemler yer almaktadır. Bu çalışmada C programlama dili özelinde (Juliet Test Suite for C/C++) veri kümesi içinden sadece C programlama dili örneklemleri indirilmiştir. Bu veri kümesi içinde 109 farklı CWE türü yer almaktadır.

3.1.2. Draper VDISC Veri Kümesi (Draper VDISC Dataset)

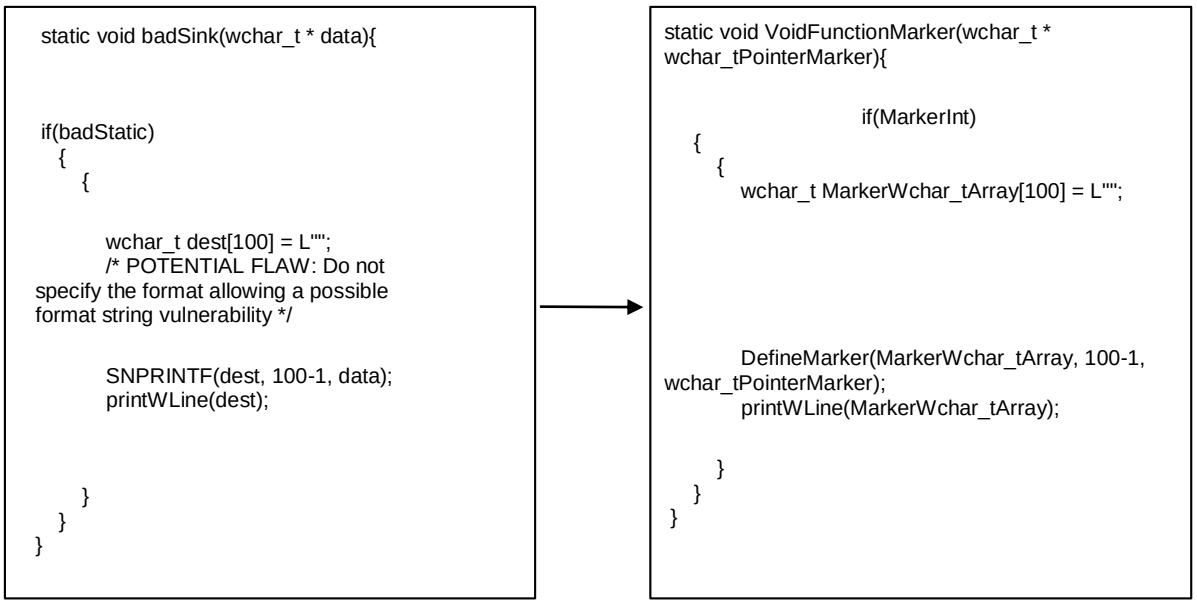
Draper VDISC veri kümesi açık kaynak projelerinden çekilmiş, statik analizlerde kullanılmak amaçlı C programlama dili için benzersiz 1,27 milyon veri içermektedir. Bu veriler, fonksiyon ismi de dahil olmak üzere fonksiyon alanını içermektedir. Veri kümesi içerisinde bulunan veriler, %80 eğitim, %80 validasyon ve %20 test olacak şekilde üç farklı HDF5 uzantılı dosyaya ayrılmıştır. Mevcut kaynak kodları, en yaygın 5 CWE türü (CWE-120 (fonksiyonların %3.7'si), CWE-119 (fonksiyonların %1.9'u), CWE-469 (fonksiyonların %0.95'i), CWE-476 (fonksiyonların %0.21'i)) ve diğerleri (CWE-other (fonksiyonların 2.7%'i)) olarak 5 sütun altında, zararlı olanlar 0 zararlı olmayanlar 1 olacak şekilde işaretlenmiştir.

3.2. Veri Önişleme (Data Preprocessing)

Bu çalışmada önerilen SecureC modeli için veri ön işleme adımları aşağıda kapsamlı olarak sunulmuştur. Bu adımlarda veri kümesi olarak kullanılan C programlama dili kaynak kodları doğal dil işleme yöntemleri kullanılarak ortak bir kod şablonuna oturtturulmuş, böylece yapay zekâ modeline girdi olarak verilebilecek genel bir forma dönüştürülmüştür. Kod kalıplarının belirli kurallar çerçevesinde standartlaştırılması diğer bir ifadeyle genel kod şablonunun çıkarılması, kod semantiğinin daha iyi anlaşılmasını sağlar.

C programlama dili kaynak kodlarının içeriğine ait genel bir kod şablonunun oluşturulabilmesi için düzenli ifadeler kullanılmıştır. Çalışmada, standartlaştırma sürecinde düzenli ifadelerin kullanımı ile genel şablonun oluşturulabilmesi için kullanılan ifadelerin daha esnek ve anlaşılabilir hale getirilebilmesi mümkün olmuştur.

Çalışmada, düzenli ifadeler kullanarak kod parçacıklarının ortak bir şablonda ifade edilmesi sağlanmıştır. Düzenli ifadelerin kullanımıyla elde edilen faydalar şu şekilde özetlenebilir; 1) toplu olarak değiştirilmek istenilen kod parçacıklarını kolay bir şekilde tespit edilmesi, 2) karmaşık ve yüksek boyutlu bir veri kümesindeki kod parçacığının daha hızlı tespit edilmesi, 3) tespit edilen kod parçacığının oluşturulan kod şablonuna uygun şekilde değiştirilebilmesi.



Şekil 2. Düzenli ifadelerle C kaynak kodlarının işaretlenmesi (Marking of C source codes with regex)

Şekil 2'de düzenli ifadeler kullanılarak işaretlenen bir kod parçası örneği verilmiştir. Şekil 2'deki örnekte; fonksiyon ismi fonksiyonun dönüş tipine göre, fonksiyonun aldığı parametreler ise parametrenin tipine göre işaretlenmiştir. Fonksiyonun kapsamı içinde bulunan "badStatic" değişkeni, fonksiyon kapsamı dışında tamsayı (integer) tipinde tanımlanmış bir değişkendir. Bu nedenle "MarkerInt" olarak işaretlenmiştir. Fonksiyon kapsamında bulunan "wchar_t" tipindekindeki "dest" isimli değişken, tipine uygun olarak "MarkerWchar_tArray" şeklinde işaretlenmiş ve fonksiyon kapsamı boyunca kullanıldığı yerlerde "MarkerWchar_tArray" olarak değiştirilmiştir. Yorum satırları modelin başarısını olumsuz etkileyeceği düşünülerek işaretli kod parçasından atılmıştır. "SNPRINTF", define ön işlemci komutu ile dosyanın başında tanımlanmıştır. Define ön işlemci komutu ile tanımlanan tüm değerler "DefineMarker" olarak işaretlenmiştir.

İşaretleme sırasında karşılaşılan dört farklı probleme ait uygulanan çözüm yaklaşımları aşağıda verilmiştir.

- **Aynı isimli fakat farklı tipte değişkenlerin tespit edilmesi problemi:** Aynı isimli fakat farklı değişken tipine ait değişken isimlerinin ayrı tiplere göre işaretlenebilmesi için, karşılaşılan değişken isimlerinin ilgili dosya içerisindeki başlangıç indekslerinin tutulması gerekmektedir. Belirlenen işaret ile değişken isminin değiştirilmesi ile ilgili adımda, karşılaşılan ilk değişken isminin indeksinden başlanarak, aynı isimli değişkenin görüldüğü diğer en küçük indekse kadar olan kısımda sadece belirlenen işaret ile değişim gerçekleştirilmiştir.
- **İndex kayması problemi:** İlk adımda verilen problem çözümünden sonra değişken isimleri, farklı bir değişken ismiyle işaretlenerek değiştirildiği için iki tanımlama arasındaki karakter farkından dolayı index kayması problemi ortaya çıkmıştır. Özellikle çok fazla sayıda kaynak kod içeren büyük veri kümelerinde index kayması nedeniyle kod parçacıklarının içeriği bozulacaktır. Bu problemin önüne geçmek amacıyla, her değişken ismi işaretleme işleminde, değişimin yapılmaya başlandığı indeksten büyük indeks değerine sahip değişkenlerin başlangıçta tutulan indeksleri, değişim esnasında oluşan karakter sayısı farkı kadar arttırılmıştır. Bu sayede işaretlenecek değişkenin, işaretlenmeden önceki karakter sayısı ile işaretlendikten sonraki karakter sayısı arasındaki fark yüzünden oluşan indeks kayması ortadan kaldırılmıştır.
- **Fonksiyon alanının belirlenmesi problemi:** Bu problemde, C dilinde fonksiyon alanının başlangıcını temsil etmek için kullanılan "{" karakteri ile fonksiyon alanının bitişini temsil eden "}" karakterinin ayrı ayrı sayılması gerekmektedir. "{" karakterinden sonra "}" karakterinin gelişine kadar arada farklı amaçlar ile kullanılan "{" ve "}" karakterlerinin sayısı da sayma işlemine dahil edilmiştir. "{" karakterinin sayısı ile "}" karakterinin sayısının eşit olması durumunda bir fonksiyon alanının tamamlandığı anlaşılır ve fonksiyon aralığını temsil ettiği anlaşılan kod parçacığı ayrı olarak işaretlenmiştir. İlgili fonksiyon aralığı, kod satırlarındaki kaymayı engellemek amacıyla işaretleme işlemi yapılmadan önce, veri kümesinden çıkarılmıştır. Daha sonra işaretlenen fonksiyon aralığı veri kümesi içerisine işaretlenmiş haliyle tekrar eklenmiştir. Bu adımda uygulanan yöntem sonucunda, veri kümesindeki fonksiyon aralıkları çıkarılan kod şablona uygun hale gelmiştir.
- **Farklı struct tanımlama problemi:** C programlama dilinde "struct" veri yapısı farklı şekillerde tanımlanabilir. Bu nedenle çalışma kapsamında, "struct" tanımlamalarına ait ortak bir şablon

oluşturulması amaçlı, her kod parçacığının "struct" içerip içermediğini kontrol eden, eğer "struct" içeriyorsa uygun şekilde işaretleme yapan fonksiyon yazılmıştır. Her bir "struct" tanımlama yöntemi, ayrı ayrı değerlendirilerek tanımlama tiplerine uygun düzenli ifadeleri içeren işlemler gerçekleştirilmiştir. Bu sayede "struct" tanımlamaları için yazılmış düzenli ifade komutları, veri kümesi içerisinde bulunan diğer kod parçacıklarını etkilememiştir.

Çalışma kapsamında, kaynak kodların düzenli ifadeler içeren ortak şablonlara oturturulması sırasında karşılaşılan dört farklı probleme ait yukarıda verilen çözümler uygulanarak C kaynak koduna ait genel bir kod şablonu oluşturulmuştur. Bu sayede veri kümesinde kullanılan tüm kaynak kodlar ortak ve genel bir kod şablonu ile temsil edilmiştir.

3.3. Kelime Temsillerinin Çıkarılması (Extracting Word Embeddings)

Çalışma kapsamında, veri kümesi içerisinde bulunan kaynak kod parçaları Byte Çifti Kodlaması (Byte pair encoding-BPE) ile simgelere (tokens) ayrılmıştır. Bu işlemden sonra kelime temsillerinin oluşturulması için CBOW ve Skip-gram temelli Word2Vec modeli oluşturulmuştur.

Word2Vec modeli girdi olarak her bir kaynak kod parçası için oluşturulan simgeleri liste olarak alır. Çalışma kapsamında kullanılan SARD-Juliet ve Draper VDISC veri kümelerindeki her bir kaynak kodun içerisindeki tokenlar (BPE'den gelen) Word2Vec modeliyle kelime vektörlerine dönüştürülmüştür. Daha fazla kelime temsilinin oluşturulması amaçlı, SARD-Juliet ve Draper VDISC veri kümelerine ek olarak Github repoları üzerinden de C kaynak kodları bu adımda kullanılmıştır. Github repolarından toplamda 72,784 farklı C kaynak kodu toplanmıştır. Bu kaynak kodlarının 58,283 tanesi işaretlenerek işaretli veri kümesine ait kelime vektörlerini temsil etmesi amaçlı kullanılmıştır.

Word2Vec modeli oluşturma işlemi, hem işaretlenmiş (düzenli ifadeler kullanılarak), hem de işaretlenmemiş veri kümesi için ayrı ayrı yapılmıştır. LSTM modelinde verilerin eğitimi için hem işaretsiz hem de işaretli veriler kullanılarak iki farklı model oluşturulduğundan; işaretsiz verilerin eğitimi için işaretsiz gözlemlerden oluşturulan Word2Vec modeli, işaretli verilere ait modelin eğitimi için ise işaretlenmiş verilerden oluşan Word2Vec modeli kullanılmıştır. Kelime temsillerinin çıkarılması adımında kullanılan örnek sayıları Tablo 1'de verilmiştir. Tablodan da görüleceği gibi SARD-Juliet ve Draper VDISC veri kümelerinde yer alan tüm gözlemler kelime temsillerinin oluşturulması için Word2Vec modeline girdi verilmiştir. Bunun yanı sıra Github repolarından toplanan 72,784 farklı C kaynak koduna ait veri kümesinin içinden 58,283 tanesi başarılı olarak etiketlenip işaretli verilere ait modelin eğitimi için kullanılmıştır.

Tablo 1. Kelime temsillerinin çıkarılmasında kullanılan örnek sayıları
(Number of samples used in extracting word embeddings)

Word2Vec VERİ KAYNAĞI	İŞARETSİZ VERİ ÖRNEK SAYISI	İŞARETLİ VERİ ÖRNEK SAYISI
SARD-Juliet Veri Kümesi	46,827	46,827
Draper VDISC Veri Kümesi	1,015,681	1,015,681
Github	72,784	58,283
Toplam	1,135,292	1,120,791

4. Deneysel Sonuçlar (Experimental Results)

Çalışma kapsamında veri kümesinin oluşturulması ve veri ön işleme adımlarından sonra önerilen yöntem ikili sınıflandırma ve çoklu sınıflandırma için ayrı olarak ele alınmıştır. İkili sınıflandırma için SARD-Juliet ve Draper VDISC veri kümeleri birleştirilmiştir. Draper VDISC üzerinde zararsız ve zararlı C kaynak kodları bulunmaktadır. SARD-Juliet üzerinde ise sadece zararlı C kaynak kodları bulunmaktadır. Bahsedilen iki veri kümesi birleştirilerek daha özgün bir veriyle çalışılması amaçlanmış ve ikili sınıflandırma yöntemlerine geçilmiştir. Çoklu sınıflandırma için sadece SARD-Juliet içerisinde bulunan kaynak kodlar kullanılmıştır. Sınıflandırma başarımlarının karşılaştırılması amaçlı F1-skor, F2-skor, kesinlik, duyarlılık, doğruluk değerlendirme ölçütleri kullanılmıştır.

4.1. SARD-Juliet ve Draper VDISC Veri Kümeleri Üzerinde İkili Sınıflandırma (Binary Classification on SARD-Juliet Dataset and Draper VDISC Dataset)

Çalışma kapsamında zararlı/zararsız C kaynak kodu analizi için geliştirilen SecureC aracında, ikili sınıflandırma için LSTM modeli kullanılmıştır. SARD-Juliet veri kümesinde sadece zararlı kaynak kodlarına ait etiketlenmiş

veriler yer alırken, Draper VDISC veri kümesi üzerinde ise zararlı ve zararsız iki sınıf yer almaktadır. Bu nedenle ikili sınıflandırma için, bu iki veri kümesi birleştirilerek veri kümesinin boyutu arttırılmıştır. Bu iki veri kümesi birleştirildiğinde toplamda 1,231,256 farklı gözlem içerir. SecureC modelinde bu gözlemlerin yaklaşık %20'si (212,500 farklı gözlem eğitim veri kümesi için, 26,769 farklı gözlem validasyon veri kümesi için) modelin geliştirilmesi aşamasında kullanılmıştır.

Güvenlik zafiyeti bulunan zararlı kaynak kodları "1", güvenlik zafiyeti bulunmayan zararsız kaynak kodları ise "0" olarak etiketlenmiştir. Tablo 2'de ikili sınıflandırma için kullanılan eğitim ve validasyon kümesinde yer alan gözlem sayıları ayrıntılı olarak verilmiştir. İşaretli ve işaretsiz veri kümelerindeki gözlem sayıları aynı tutulmuştur.

Tablo 2. İkili sınıflandırma için eğitim ve validasyon kümesindeki gözlem sayıları
(Number of observations in the training set and validation set for binary classification)

TÜR		EĞİTİM VERİ KÜMESİ GÖZLEM SAYISI	VALİDASYON KÜMESİ GÖZLEM SAYISI
Zararlı	CWE-119	3,814	481
	CWE-120	7,556	951
	CWE-469	386	1,608
	CWE-476	2,012	239
	CWE-OTHERS	14,815	2,055
Zararsız		190,041	23,773
Toplam		212,500	26,769

Kaynak kod parçasında bulunan güvenlik zafiyetlerinin ikili sınıflandırılabilmesi için işaretli ve işaretsiz veri kümesi ayrı ayrı denenmiştir. İşaretli veri kümesinde alınan sonuçlar ile, işaretsiz veri kümesinde alınan sonuçlar aşağıdaki alt başlıklarda karşılaştırılmıştır.

4.1.1. İşaretsiz Veri Kümesi ile İkili Sınıflandırma (Binary Classification with Unmarked Dataset)

İşaretsiz veri kümesi üzerinde ikili sınıflandırma amaçlı, SecureC'de kullanılan LSTM modelinde hiper parametreler "hidden_size = 32, 64, 128, 192, num_layers = 1, 2, 3, dropout_p = 0.3, 0.5, 0.8" olarak belirlenmiştir. Test edilen hiper parametreler içerisinden hidden_size için "128", num_layers için "2" ve drop_out için "0.3" değerleri 5 epoch için en iyi sonuçlar elde edildiği için seçilmiştir. Seçilen hiper parametrelere göre modelin başarım değerleri Tablo 3'te verilmiştir.

Tablo 3. İşaretsiz veri kümesi ile ikili sınıflandırma sonuçları (Binary classification results with unmarked dataset)

DEĞERLENDİRME ÖLÇÜTÜ	SKOR BİLGİSİ
F1-skor	0.8345
F2-skor	0.8169
Kesinlik	0.8761
Duyarlılık	0.8033
Doğruluk	0.9404

4.1.2. İşaretli Veri Kümesi ile İkili Sınıflandırma (Binary Classification with Marked Dataset)

İşaretli veri kümesi üzerinde ikili sınıflandırma amaçlı, SecureC'de kullanılan LSTM modelinde hiper parametreler "hidden_size = 32, 64, 128, 192, num_layers = 1, 2, 3, dropout_p = 0.3, 0.5, 0.8" olarak belirlenmiştir. Test edilen hiper parametreler içerisinden hidden_size için "64", num_layers için "1" ve drop_out için "0.3" değerleri 5 epoch için en iyi sonuçlar elde edildiği için seçilmiştir. Seçilen hiper parametrelere göre modelin başarım değerleri Tablo 4'te verilmiştir.

Tablo 4. İşaretli veri kümesi ile ikili sınıflandırma sonuçları (Binary classification results with marked dataset)

DEĞERLENDİRME ÖLÇÜTÜ	SKOR BİLGİSİ
F1-skor	0.8161
F2-skor	0.7894
Kesinlik	0.9057
Duyarlılık	0.7648
Doğruluk	0.9392

İşaretli ve işaretsiz veri kümeleri üzerinde yapılan ikili sınıflandırmalar ele alındığında, düzenli ifadeler ile işaretlenmiş işaretli veri kümesi üzerinde kesinlik değerinin 0.9057 olduğu görülürken, işaretsiz veri kümesi üzerinde elde edilen kesinlik değeri 0.8761'dir. Kesinlik değeri, modelin pozitif olarak tahmin ettiği değerlerin gerçekten kaç tanesinin pozitif olduğunu veren bir ölçüttür. Çalışma kapsamında pozitif olarak işaretlenen değerler zararlı kaynak kodlarını temsil etmektedir, bu noktada zararlı kaynak kodu olarak etiketlenen değerlerin gerçekte de zararlı olması bu problem için önemlidir. Bu da veri kümesinin işaretlenmesinin ikili sınıflandırmada önemli bir yarar sağladığını gösterir.

4.2. SARD-Juliet ve Draper VDISC Veri Kümeleri Üzerinde Çoklu Sınıflandırma (Multi Classification on SARD-Juliet Dataset and Draper VDISC Dataset)

Çalışma kapsamında zararlı/zararsız C kaynak kodu analizi için geliştirilen SecureC aracında, çoklu sınıflandırma için de LSTM modeli kullanılmıştır. Draper VDISC veri kümesi üzerinde bulunan kaynak kod parçaları sadece dört adet CWE türü içerdiği için çalışmaya katılmamıştır. Çoklu sınıflandırma için, SARD-Juliet veri kümesi içerisinde yer alan 40 CWE türü kullanılmıştır. Diğer CWE türlerine ait gözlem sayıları az olduğundan bu sınıflar dahil edilmemiştir. Kod parçasındaki güvenlik zafiyetlerinin çoklu sınıflandırılabilmesi için işaretli ve işaretsiz veri kümesi ayrı ayrı denenmiştir. Tablo 5'te CWE türlerine göre eğitim ve validasyon kümelerinde yer alan gözlem sayıları verilmiştir.

Tablo 5. CWE türlerine göre eğitim ve validasyon kümesi gözlem sayıları
(Number of observations in the training set and validation set for each CWE)

CWE TÜRÜ	EĞİTİM KÜMESİ GÖZLEM SAYISI	VALİDASYON KÜMESİ GÖZLEM SAYISI
CWE-114	522	126
CWE-121	4,531	1,172
CWE-122	2,816	715
CWE-124	1,497	327
CWE-126	1,066	266
CWE-127	1,461	363
CWE-134	2,599	641
CWE-190	3,872	988
CWE-191	2,995	731
CWE-194	1,035	272
CWE-195	1,034	274
CWE-197	787	192
CWE-252	494	136
CWE-253	546	138
CWE-272	200	52
CWE-284	175	41
CWE-319	166	50
CWE-369	762	210
CWE-398	151	30

Tablo 5. Devam (Continued)

CWE-400	661	149
CWE-401	953	235
CWE-404	352	84
CWE-415	268	56
CWE-426	181	35
CWE-427	420	120
CWE-457	514	102
CWE-476	288	72
CWE-563	297	96
CWE-590	695	169
CWE-606	438	102
CWE-617	282	60
CWE-665	171	49
CWE-675	174	42
CWE-680	267	60
CWE-690	869	222
CWE-758	296	69
CWE-761	509	139
CWE-78	4,285	1,115
CWE-789	454	86
CWE-90	422	118
Toplam	39,505	9,877

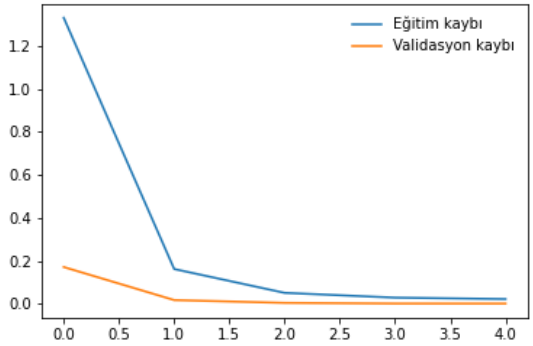
4.2.1. İşaretsiz Veri Kümesi ile Çoklu Sınıflandırma (Multi Classification with Unmarked Dataset)

İşaretsiz veri kümesi üzerinde çoklu sınıflandırma amaçlı, SecureC'de kullanılan LSTM modelinde hiper parametreler "hidden_size = 32, 64, 128, 192, num_layers = 1, 2, 3, dropout_p = 0.3, 0.5, 0.8" olarak belirlenmiştir. Test edilen hiper parametreler içerisinden hidden_size için "128", num_layers için "1" ve drop_out için "0.3" değerleri 5 epoch için en iyi sonuçlar elde edildiği için seçilmiştir. Seçilen hiper parametrelere göre modelin başarım değerleri Tablo 6'da verilmiştir.

Tablo 6. İşaretsiz veri kümesi ile çoklu sınıflandırma sonuçları (Multi classification results with unmarked dataset)

DEĞERLENDİRME ÖLÇÜTÜ	SKOR BİLGİSİ
F1-skor	0.9983
F2-skor	0.9984
Kesinlik	0.9982
Duyarlılık	0.9985
Doğruluk	0.9994

İşaretsiz veri kümesinde çoklu sınıflandırmaya ait eğitim ve validasyon kaybına ait grafik Şekil 3'te verilmiştir.



Şekil 3. İşaretsiz veri kümesinde çoklu sınıflandırma için eğitim-validasyon kaybı grafiği
(Training loss and validation loss graph for multi classification with unmarked dataset)

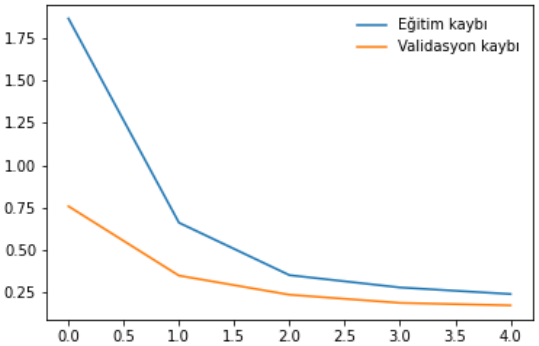
4.2.2. İşaretli Veri Kümesi ile Çoklu Sınıflandırma (Multi Classification with Marked Dataset)

İşaretli veri kümesi üzerinde çoklu sınıflandırma amaçlı, SecureC'de kullanılan LSTM modelinde hiper parametreler "hidden_size = 32, 64, 128, 192, num_layers = 1, 2, 3, dropout_p = 0.3, 0.5, 0.8" olarak belirlenmiştir. Test edilen hiper parametreler içerisinden hidden_size için "128", num_layers için "3" ve drop_out için "0.3" değerleri 5 epoch için en iyi sonuçlar elde edildiği için seçilmiştir. Seçilen hiper parametrelere göre modelin başarım değerleri Tablo 7'de verilmiştir.

Tablo 7. İşaretli veri kümesi ile çoklu sınıflandırma sonuçları (Multi classification results with marked dataset)

DEĞERLENDİRME ÖLÇÜTÜ	SKOR BİLGİSİ
F1-skor	0.9610
F2-skor	0.9570
Kesinlik	0.9736
Duyarlılık	0.9529
Doğruluk	0.9437

İşaretli veri kümesinde çoklu sınıflandırmaya ait eğitim ve validasyon kaybı değerleri Şekil 4'te verilmiştir.



Şekil 4. İşaretli veri kümesinde çoklu sınıflandırma için eğitim-validasyon kaybı grafiği
(Training loss and validation loss graph for multi classification with marked dataset)

İşaretli ve işaretsiz veri kümeleri üzerinde yapılan çoklu sınıflandırmalar ele alındığında, işaretsiz veri kümesi üzerinde geliştirilen modelin tüm değerlendirme metrikleri için daha başarılı sonuçlar elde ettiği görülmüştür.

SecureC ile işaretli ve işaretsiz veri kümeleri üzerinde ikili ve çoklu sınıflandırma için elde edilen sonuçlar, literatürde yer alan diğer çalışmalarla karşılaştırıldığında SecureC modelinin başarısı açıktır. Tablo 8'de literatürde yer alan çalışmalar ile SecureC modeli F1 skor metriği açısından karşılaştırılmıştır. Tablodan da görüleceği gibi özellikle işaretsiz veriler üzerinde eğitilen SecureC modelinde (çoklu sınıflandırma) elde edilen 0.9983 F1 skor değeri literatürdeki en iyi sonuçtur.

Tablo 8. Deneysel sonuçların mevcut literatür ile karşılaştırılması
(Comparison of the experimental results with existing literature)

ÇALIŞMA	YÖNTEM	F1 SKORU
(Li vd. 2020)	Rastgele Orman	0.93
(Suneja vd. 2019)	CNN	0.53
(Xu vd. 2018)	CLSTM	0.96711
Vudeepecker Modeli (Xiaomeng vd. 2018)	CNN	0.91
BV-Detector ((Tian vd. 2020)	BGRU (İkili-Sınıflandırma)	0.899
BGNN4VD (Cao vd. 2021)	BGNN (İkili-Sınıflandırma)	0.768
SecureC	LSTM (İşaretsiz-Çoklu Sınıflandırma)	0.9983
SecureC	LSTM (İşaretsiz-İkili Sınıflandırma)	0.8345

5. Sonuç ve Tartışma (Result and Discussion)

Bu çalışmada C programlama dili özelinde güvensiz kaynak kodlarını tespit edebilen SecureC olarak adlandırılan bir yazılım aracı geliştirilmiştir. Çalışmada, düzenli ifadeler kullanılarak yeni kod şablonları çıkarılmış, ardından bu yeni şablonlar kullanılarak veri setindeki kaynak kodlar işaretlenmiştir. Bunun yanı sıra kelime temsillerinin çıkarılmasını sağlayan, işaretli ve işaretsiz veriler kullanılarak iki farklı Word2Vec modeli geliştirilmiştir. Daha sonra yüzbinlerce satıra sahip kaynak koddan güvensiz kısımları tespit ederken, önerilen modelin işaretli veri kümesi ile işaretsiz veri kümesi üzerindeki başarımları araştırılmıştır.

SecureC, LSTM modeli kullanılarak geliştirilmiştir, bu model C programlama dilindeki kaynak kodları içeren SARD-Juliet veri kümesi, Draper VDISC veri kümesi birleştirilerek oluşturulan veri kümesini girdi olarak alır. Veri kümesinde yer alan kod parçacıkları C programlama dilinin yapısı dikkate alınarak düzenli ifadelerle ortak şablonlara oturtturulmuştur. Daha sonra kullanılan hibrit veri kümesi üzerindeki her bir kaynak kodu hem işaretli hem işaretsiz gözlemler için Word2Vec modeliyle kelime vektörlerine dönüştürülmüştür. Sonrasında bu vektörler model için özellikleri temsil eden girdiler olarak LSTM modeliyle işlenmiştir.

Çalışmada modelin hem işaretli hem de işaretsiz veri kümeleri üzerindeki başarımları değerlendirildiğinde, düzenli ifadeler ile işaretlenmiş veri kümesi üzerinde kesinlik değerinin 0.9057 olduğu görülürken, işaretsiz veri kümesi üzerinde elde edilen kesinlik değeri 0.8761 olarak elde edilmiştir. Çalışma kapsamında pozitif olarak işaretlenen değerler zararlı kaynak kodlarını temsil eder, bu noktada zararlı kaynak kodu olarak etiketlenen değerlerin gerçekte de zararlı olması, diğer bir ifadeyle kesinlik değerinin yüksek olması bu çalışmadaki araştırma problemi için oldukça önemlidir. Bu da özellikle ikili sınıflandırma için işaretleme işleminin önemini vurgular.

İkili sınıflandırmalar için yapılan çalışmalarla karşılaştırıldığında, BGNN4VD modeline göre LSTM tabanlı SecureC modelinin daha başarılı sonuçlar elde edildiği görülmektedir. BV-Detector modelinin geliştirmesi aşamasında denenen LSTM modeli ile elde edilen %85,9 F1 skor değeri geçilememiştir.

SecureC'de çoklu sınıflandırma ele alındığında, işaretsiz veri kümesi üzerinde modelin daha başarılı olduğu gözlemlenmiştir. SecureC ile işaretli ve işaretsiz veri kümeleri üzerinde ikili ve çoklu sınıflandırma için elde edilen sonuçlar, literatürde yer alan diğer çalışmalarla karşılaştırıldığında SecureC'nin başarısı açıktır.

İleriki çalışmalarda, LSTM modelinin yanı sıra farklı derin öğrenme modelleri kullanılarak önerilen yöntemin başarısı arttırılabilir. Bunun yanı sıra özellikle çoklu sınıflandırma için kullanılan veri kümesinde yer alan gözlem sayılarının ve sınıf sayısının arttırılmasının modelin başarımında katkısı olacaktır.

Teşekkür (Acknowledgement)

Bu çalışma TÜBİTAK 2209-A Üniversite Öğrencileri Araştırma Projeleri Destekleme Programı 1919B012004582 Başvuru Numaralı proje kapsamında desteklenmiştir.

Çıkar Çatışması (Conflict of Interest)

Yazarlar tarafından herhangi bir çıkar çatışması beyan edilmemiştir. No conflict of interest was declared by the authors.

Kaynaklar (References)

Cao, S., Sun, X., Bo, L., Wei, Y., Li, B. 2021. BGNN4VD: Constructing Bidirectional Graph Neural-Network for Vulnerability Detection, Information and Software Technology, 136, Doi: 10.1016/j.infsof.2021.106576.

Draper VDISC Dataset, https://osf.io/d45bw/, 2021.

Edgescan, 2020. Vulnerability Statistics Report. https://cdn2.hubspot.net/hubfs/4118561/BCC030%20Vulnerability%20Stats%20Report%20(2020)_WEB.pdf (Erişim tarihi: Kasım 2021).

Juliet Test Suite for C/C++ , https://samate.nist.gov/SARD/around.php#juliet_documents, 2021.

Kaur, A. Nayyar, R. 2020. A Comparative Study of Static Code Analysis tools for Vulnerability Detection in C/C++ and JAVA Source Code, Procedia, 171, 2023-2029.

Kim, J. Hubczenko, D., Montague, P. 2019. Towards Attention Based Vulnerability Discovery Using Source Code Representation, Artificial Neural Networks and Machine Learning – ICANN 2019: Text and Time Series, 731-746.

Larrucea. X., Santamaria, I., Colomo-Palacios, R. 2019. Assessing source code vulnerabilities in a cloud-based system for health systems:OpenNCP, IET Journals, 13, 195-202.

Li, X., Wang, Lu., Xin, Y., Yang, Y., Chen, Y. 2020. Automated Vulnerability Detection in Source Code Using Minimum Intermediate Representation Learning, 10, Doi: 10.3390/app10051692.

Li, R., Feng, C., Zhang, X., Tang, C. 2019. A Lightweight Assisted Vulnerability Discovery Method Using Deep Neural Networks, IEEE, 7, 80079 – 80092.

Samate NIST, Juliet Documents NIST, 2006. Resources from the Software Assurance Reference Dataset, Kasım 2017-2021. https://samate.nist.gov/SRD/around.php#juliet_documents (erişim tarihi: Kasım 2021).

SARD, https://samate.nist.gov/SARD/, 2021.

Suneja, S., Zheng, Y., Zhuang, Y., Laredo, J., Morari A. 2019. Learning to map source code to software vulnerability using code-as-a-graph, ArXiv abs/2006.08614.

Tian, J., Xing, W., Li, Z. 2020. BVDetector: A program slice-based binary code vulnerability intelligent detection system, Information and Software Technology, 123, Doi: 10.1016/j.infsof.2020.106289.

Verma, P., Goyal, A., Gigras, Y. 2020. Email phishing: Text classification using natural language processing, Computer Science and Information Technologies, 1, 1-12.

Xiaomeng, W., Tao, Z., Runpu, W., Wei, X., Changyu, H. 2018. 10th International Conference on Advanced Infocomm Technology (ICAIT), IEEE, Doi: 10.1109/ICAIT.2018.8686548.

Xu, A., Dai, T., Chen, H., Ming, Z., Li, W. 2018. Vulnerability Detection for Source Code Using Contextual LSTM, ICSAI, Doi: 10.1109/ICSAI.2018.8599360.