Journal of the Faculty of Engineering and Architecture of Gazi University 34:4 (2019) 1695-1704

Mühendislik Mimarlık Fakültesi Dergisi
Journal of The Faculty of Engineering and Architecture of Gazi University
Elektronik / Online ISSN : 1304 - 4915
Basılı / Printed ISSN : 1300 - 1884

A novel approach for analyzing buffer overflow vulnerabilities in binary executables by using machine learning techniques

Gürsoy Durmuş[1*], İbrahim Soğukpınar[2]
[1]HAVELSAN Naval Warfare Management Systems Technology Center., İstanbul, 34890, Turkey
[2]Department of Computer Engineering, Gebze Technical University, Kocaeli, 41400, Turkey

Highlights:

- A new method for analysis of buffer overflow vulnerability in binary executables
- The use of machine learning techniques in buffer overflow vulnerability analysis
- High performance values

Keywords:

- Software security
- Software vulnerability
- Machine learning
- Buffer overflow

Article Info:

Research Article
Received: 29.12.2017
Accepted: 21.12.2018

DOI:

10.17341/gazimmfd.571485

Correspondence:

Author: Gürsoy Durmuş
e-mail: gdurmus@yahoo.com
phone: +90 532 478 9458

Graphical/Tabular Abstract

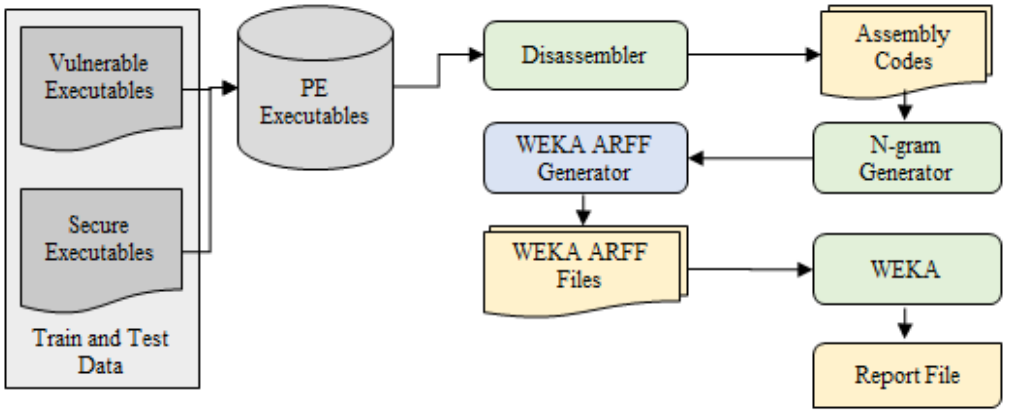


Figure A. System architecture

Purpose:
While evaluating whether a software is secure or vulnerable with traditional methods; examination of security requirements, source code analysis and software security testing activities can be performed. In many cases, these activities cannot be performed by the end user due to not exist documentation of security related requirements, absence of source codes and need to expert security testing teams. When the software is in binary executable file format, we need expert systems, which accept just only binary executables as inputs to enable security analysis on end-user side.

Theory and Methods:
In this study, we present a new method and its success, which is developed by using machine learning techniques to be used in the buffer overflow vulnerability analysis of binary executable formatted software applications. The main theory of the study as follows:

H-1: In binary executable file format software with buffer overflow vulnerabilities, there are similarities between the distributions and sequences of binary operation codes used for branching, looping, register value update, memory value update, stack operations, and system calls.

Results:
Experimental results on existing data sets indicate that PE formatted executables can be classified as vulnerable or secure by machine learning techniques based on opcode sequences and distributions. In particular, the high performance rates (95% TPR) obtained by KNN, Naïve Bayes, Hoeffding Tree, K-Star and Random Tree algorithms indicate that these algorithms are appropriate classification algorithms for vulnerability analysis.

Conclusion:
The study shows that the machine learning approach can be used to perform buffer overflow vulnerability analysis in binary executable files.

Journal of the Faculty of Engineering and Architecture of Gazi University 34:4 (2019) 1695-1704

Mühendislik Mimarlık Fakültesi Dergisi
Journal of The Faculty of Engineering and Architecture of Gazi University

Elektronik / Online ISSN : 1304 - 4915
Basılı / Printed ISSN : 1300 - 1884

Makine öğrenmesi teknikleri ile ikili yürütülebilir dosyalarda arabellek taşması zayıflığı analizi için yeni bir yaklaşım

Gürsoy Durmuş[1]*, İbrahim Soğukpınar[2]

[1]HAVELSAN Deniz Savaş Yönetim Teknolojileri Merkezi, P.K.: 11 34890, Pendik, İstanbul, Türkiye
[2]Gebze Teknik Üniversitesi, Bilgisayar Mühendisliği Bölümü, 41400, Gebze, Kocaeli, Türkiye

ÖNEÇIKANLAR

- İkili yürütülebilir dosyalarda arabellek taşması zayıflığı analizi için yeni bir yöntem
- Arabellek taşması zayıflığı analizinde makine öğrenmesi tekniklerinin kullanımı
- Yüksek başarım değerleri

Makale Bilgileri

Araştırma Makalesi
Geliş: 29.12.2017
Kabul: 21.12.2018

DOI:
10.17341/gazimmfd.571485

Anahtar Kelimeler:
Yazılım güvenliği, yazılım güvenliği zayıflığı, makine öğrenmesi, arabellek taşması

ÖZET

Bir yazılım öğesinin güvenlik zayıflığı içerip içermediğinin analizi yazılımın güvenli olup olamadığına karar vermede belirleyici bir unsurdur. Geleneksel yöntemler ile bir yazılımın güvenli olup olmadığı değerlendirilirken; yazılıma ait güvenlik gereksinimlerinin incelenmesi, kaynak kod analizi ve yazılım güvenlik test faaliyetleri icra edilebilir. Bu faaliyetler sırası ile yazılım ile ilgili gereksinimlerin belgelendirilmiş olması, yazılım kaynak kodlarının mevcut olması ve yazılım güvenliği konusunda uzman test ekiplerine olan bağımlılıkları nedeni ile çoğu zaman son kullanıcı tarafından tekrarlanamayan faaliyetlerdir. Söz konusu yazılım ikili yürütülebilir dosya formatında olduğunda, son kullanıcı tarafında güvenlik analizlerinin yapılabilmesi için sadece ikili yürütülebilir dosyanın girdi olarak kullanılabileceği sistemlere ihtiyaç duyulmaktadır. Bu çalışmada, ikili yürütülebilir dosya formatlı yazılımların arabellek taşması güvenlik zayıflığı analizinde kullanılmak üzere makine öğrenmesi teknikleri kullanılarak geliştirilen yeni bir yöntem ve başarımı sunulmuştur

A novel approach for analyzing buffer overflow vulnerabilities in binary executables by using machine learning techniques

HIGHLIGHTS

- A new method for analysis of buffer overflow vulnerability in binary executables
- The use of machine learning techniques in buffer overflow vulnerability analysis
- High performance values

Article Info

Research Article
Received: 29.12.2017
Accepted: 21.12.2018

DOI:
10.17341/gazimmfd.571485

Keywords:
Software security, software vulnerability, machine learning, buffer overflow

ABSTRACT

While evaluating whether a software is secure or vulnerable with traditional methods; examination of security requirements, source code analysis and software security testing activities can be performed. In many cases, these activities cannot be performed by the end user due to not exist documentation of security related requirements, absence of source codes and need to expert security testing teams. When the software is in binary executable file format, we need expert systems, which accept just only binary executables as inputs to enable end-user side security analysis. In this study, we present a new method and its success, which is developed by using machine learning techniques to be used in the buffer overflow vulnerability analysis of binary executable formatted software applications.

*Sorumlu Yazar/Corresponding Author: gdurmus@havelsan.com.tr, ispinar@bilmuh.gyte.edu.tr / Tel: +90 216 494 3314

1. GİRİŞ (INTRODUCTION)

Siber saldırıların her geçen gün daha da yoğun yaşandığı günümüzde, yazılımlar birer silah–hedef olarak kullanılmaya başlandı. Farklı kazanımlar peşinde olan bireysel, ekip veya kurumsal profilli saldırganlar öncelikle hedef yazılımlardaki zayıflıkları tespit etmeğe ardından da ilgili zayıflıkları sömürerek amaçlarına ulaşmaya çalışırlar. Siber saldırılar en masum hali ile eğlence amaçlı gerçekleştirilmiş olsa da sonuçları itibari ile bireysel, kurumsal ve hatta ulusal maddi zararlarla ve itibar kayıplarına neden olmaktadır. Yazılımlardaki zayıflıkların erken tespiti, hem siber saldırıların başarım oranını düşürecek hem de doğacak zararların azalmasını sağlayacaktır.

Yazılım zayıflık analizi için geleneksel yöntemlerden; yazılıma ait güvenlik gereksinimlerinin incelenmesi, kaynak kod analizi ve yazılım güvenlik test faaliyetleri tercih edilebilir. Ancak yöntemlerin uygulanabilmesi için gerekli ön koşullar çoğu kez sağlanmamaktadır. Örneğin; analiz edilecek yazılıma ait güvenlik gereksinimlerine, yazılım kaynak kodlarına ve uzman güvenlik test ekiplerine olan bağımlılıklar nedeni birçok yazılım için güvenlik analizleri üstünkörü yapılmaktadır. Yaygın kullanılan yazılımlar için, zayıflık veri tabanları ilgili bilişim güvenliği uzmanları tarafından taranmakta ve yayınlanan yamalarla güvenlik sağlanmaya çalışılmaktadır. Bu şekilde yürütülen bilgi güvenliği faaliyetleri sıfır gün saldırıları bir yana, ilgili yamalar uygulanıncaya kadar sömürülmeye açık halde bilgi güvenliğini riske etmektedir. (Şekil 1)

Gerek kamu gerekse kurumsal yazılım temininde, ilgili yazılımlar için güvenlik analizlerinin yapılması ve yazılım türüne özgü en çok bilinen ve sömürülen güvenlik zayıflıkları analiz edilmelidir. Söz konusu yazılım ikili yürütülebilir dosya formatında olduğunda, sadece ikili yürütülebilir dosya girdisi ile güvenlik analizlerini yapabilecek uzman sistemlere ihtiyaç duyulmaktadır.

İkili yürütülebilir dosya formatlı yazılımlarda en sık sömürülen güvenlik zayıflıkları:

- Yığın arabellek taşma zayıflığı,
- Çalışma esnasında kötücül komut enjeksiyon zayıflığı ve
- Yürütülebilir yetkiye sahip veri alanı zayıflığı olarak sıralanabilir.

İkili yürütülebilir dosya formatlı yazılımlar işletim sistemi mimarisine bağlı olarak farklı yapılara sahiptirler. Windows tabanlı işletim sistemleri "Taşınabilir-Yürütülebilir" (PE: Portable Executable) formatını desteklerken, Unix/Linux türevi işletim sistemleri "Yürütülebilir ve Bağlanabilir Biçimli" (ELF: Executable and Linkable Format) formatlı yazılımları desteklemekte ve çalıştırabilmektedirler.

Bu çalışmada; PE formatlı yazılımların "yığın arabellek taşma" zayıflığına karşı güvenli veya zayıf olarak sınıflandırılması için makine öğrenmesi tekniklerine dayalı yeni bir yöntem ve başarımı sunulmuştur.

Makalenin geri kalan kısmı şu şekilde düzenlenmiştir: Bölüm 2'de mevcut çalışmalar özetlenmiştir. Bölüm 3'te önerilen yöntem açıklanmıştır. Bölüm 4'te yöntemin gerçeklemesi ve başarım değerlendirmesi yapılmıştır. Bölüm 5'te sonuçlar ve öneriler paylaşılmıştır.

2. İLGİLİ ÇALIŞMALAR (RELATED WORKS)

Yazılım güvenliği için farklı tanımlar yapılmış olmasına karşın, en genel anlamda; "yazılımın saldırılara ve art niyetli kullanımlara karşı korunaklı/dirençli olması için sahip olması gereken gereksinimler bütünü" olarak tanımlamak mümkündür [1]. Yazılım güvenliği; yazılım geliştirme süreç başlangıcı diyebileceğimiz gereksinimlerin belirlenmesi safhasından itibaren göz önüne alınması gereken ve yazılım geliştirme yaşam döngüsünün her bir safhasında korunması gereken bir özellik olarak karşımıza çıkmaktadır.

Hiç kuşkusuz geliştirilen her yazılımda fark edilmiş ya da fark edilmemiş/edilememiş birçok yazılım hatası olabilir. Bu hatalara neden olan birinci unsur tasarım yanlışlıkları, ikinci unsur ise hatalı kodlamalardır. Bir yazılım hatası, bilgi güvenliği esaslarının ihlaline sebebiyet vermesi durumunda (sunulan hizmeti engelleme, yetkisiz erişim sağlama veya veri bütünlüğünü bozma) güvenlik zayıflığı olarak nitelendirilebilir [2]. Bunların dışında, yazılıma dışarıdan müdahale ile içeriğinde zayıflığa neden olacak güncellemeler de zayıflık kaynağı olarak gösterilebilir. Bu tür zayıflıklar kötücül yazılım tespit yöntemleri ile analiz edilmektedir.

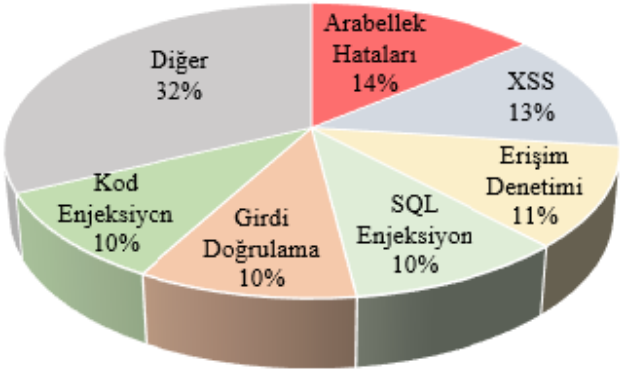
Yazılımlarda en sık rastlanan, güvenlik zayıflığına neden olan hatalar sınıflandırılmış [1, 2] ve her biri için çözüm yolları önerilmiştir. Genel olarak, güvenlik zayıflığı yaratan yazılım hataları:

- Arabellek taşmaları,
- Girdi verilerinin denetlenmemesi,
- Veri yarışları olarak karşımıza çıkmaktadır.

CVE (Common Vulnerabilities and Exposures) ve NVD (National Vulnerability Database) zayıflık veri tabanlarındaki zayıflıklar ve zayıflıkların önem derecesine göre dağılımları incelenmiş, arabellek taşmalarının çok yaygın olduğu ve önemli derecede yazılım güvenliği zayıflıkları doğurduğu bildirilmiştir [3].

Arabellek taşmaları, başta programın istemsiz bir şekilde sonlandırılmasına neden olmakla birlikte program akışının değiştirilmesine, hatalı işlem yapmasına neden olabilmektedir. Ayrıca, kod enjeksiyon zayıflığı da arabellek taşmaları nedeni ile ortaya çıkan önemli bir zayıflıktır. Özellikte veri sekmesinde yürütme yetkisi olan bir yazılımda, saldırgan arabellek taşma zayıflığını tespit ettikten sonra enjekte ettiği makine kodu komutları ile

program akışını değiştirebilmekte ve yazılımın yetkileri ile işlem yapabilmektedir [4]. 2001 yılında "Code Red", 2003 yılında "Zotob" ve 2004 yılında etkili olan "Sasser" solucanları arabellek taşma zayıflıklarını sömürerek hızla yayılmış ve ciddi zararlar vermişlerdir. Arabellek taşmalarının engellenmesi için kaynak kodun analiz edilmesi, programlama diline özgü zayıflık yaratacak işaretçi aritmetiği kullanan fonksiyonların kullanımından kaçınılması önerilmektedir [5].



Şekil 1. Yazılım güvenlik zayıflıkları dağılımı (Software vulnerability distribution)

Veri yarışları, çok iş parçacıklı (multithreaded) yazılımların servis hizmetinin aksamasına ve veri bütünlüğünün bozulmasına neden olabilmektedir. Bu tip hatalardan kaçınmak için kaynak kod analizlerinin yapılması, yazılımının kaynak kod üzerinden kontrollü çalıştırılması önerilmektedir. Olası veri yarışlarının çok ciddi sorunlara neden olacağı değerlendiriliyorsa çalışma anında kontrol altına alınmaya çalışılması ve/veya kaçınılması önerilmektedir [6, 7].

Yazılım geliştirme yaşam döngüsü sürecinin en önemli safhalarından biri olan "kodlama" aşamasında kaynak kodlarının "statik kod analiz" araçları ile incelenmesi yazılım güvenliğini artırıcı yöntemlerden biridir. Yapılan çalışmalar, kaynak kod satır sayısının artmasına paralel olarak güvenlik zayıflıklarının da arttığını göstermektedir [8, 9]. Kaynak kod denetiminin, kod gözden geçirme aktivitesi olarak gözlemsel olarak yapılması hem proje süresini uzatma hem de gözden kaçırma risklerini beraberinde getirmektedir. Bu nedenle, kod analizinin otomatik yapılabilmesi için gerek ticari gerekse açık kaynak kodlu birçok ürün geliştirilmiş ve kullanıma sunulmuştur. Kaliteli bir statik kod analiz aracı, yazılımdaki güvenlik zayıflıklarının ortaya çıkarılmasında ve giderilmesinde kolaylık sağlamaktadır [10]. Statik kod analizleri aynı zamanda kod kalitesini iyileştirme faaliyetleri içinde çıktı üretmektedir. Yapılan çalışmalar, tasarım ve kod kalitesi kötü olan yazılımlarda güvenlik açığının kod kalitesi iyi olan yazılımlara oranla çok daha yüksek olduğunu ortaya koymaktadır [11]. Kaynak koduna erişim sağlanamayan PE formatlı yazılımların analizinde statik ve dinamik analiz yöntemleri kullanılabilir. Statik analiz yönteminde, yazılım bileşenleri yapısal incelenerek olası zayıflık ve anomali tespitleri yapılmaya çalışılır. Dinamik analiz yöntemleri ile uygulama gerçek veya sembolik çalıştırılarak olası çalışma yolları çıkarılır ve hafıza erişimleri gözlemlenir. Dinamik analizler, uygulama davranışlarının gözlemlenmesi için izole bir ortamda uygulamanın yürütülmesini gerektirmektedir [12].

Tevis vd. [13, 14] yaptıkları çalışmada, PE formatlı yazılımların statik analizlerini yaparak PE dosyalarında; tablo büyüklüklerindeki tutarsızlıkları, sıfır ile doldurulmuş geniş alanları, hem yürütülebilir hem de yazılabilir bölümlerin tespitini ve arabellek taşmalarına neden olacak C/C++ kütüphane fonksiyonlarının kullanımlarının tespitini sağlayacak yöntem geliştirmişlerdir. FindSSV yazılım aracını geliştirerek yöntemlerini 2700'den fazla 6 farklı kategorideki (kurulum dosyaları, yazılım geliştirme dosyaları, Windows XP işletim sistemi dosyaları, Microsoft uygulama dosyaları, güvenlik uygulamaları) PE yazılım bileşeni üzerinde sınamışlardır. Çalışmalarında PE dosyasında yer alan her bir bölümün ve öğenin (DOS header, MS-DOS stub, file header, optional header, section table, symbol table, string table, import table) güvenlik zayıflığı ve anomali tespitinde kullanılıp kullanılamayacağını değerlendirmişlerdir. Du Varney vd. [15], ELF formatlı yazılımların güvenlik analizlerini kolaylaştıracak yeni alanların ELF formatına eklenmesini önermişlerdir. Kaynak kodun DEBUG kipinde derlenmesi durumunda güvenlik analizlerine yardımcı bilgiler (veri tipleri, veri büyüklükleri, fonksiyonların adresleri gibi) ELF dosyası içerisine yazılmaktadır. Ancak, yazılımlar dağıtılırken RELEASE kipinde derlenmekte ve bu bilgiler ELF dosyalarına aktarılmamaktadır. Güvenlik analizlerinde özellikle verilerin tipleri, büyüklükleri gibi bilgiler analiz süreçlerini kolaylaştırdıklarından bu bilgilerin belli bir formatta RELEASE kipinde derlenmiş ELF dosyalarına yazılmalarının faydalı olacağı belirtilmiştir. Bunun için derleyiciler üzerinde değişiklik yapılması gerekliliği vurgulanmıştır. Analizcilerin işine yarayacak bu bilgilerin saldırganlara dayeni fırsatlar sunacağını değerlendirmekteyiz. Ayrıca derleyicilerin bu yönteme göre güncellenmesi gerekliliği teorinin pratiğe yansıtılmasında bir engel olarak karşımıza çıkmaktadır.

Cova vd. [16] yaptıkları çalışmada, x86 işlemci mimarisinde ELF formatlı ikili yürütülebilir dosyalarda zayıflık analizlerini hem statik hem de sembolik çalıştırma yöntemlerini uygulayarak tespit etmeğe çalışmışlardır. Cova ve arkadaşları, "gördüğün (kod), çalıştırdığın değildir" [17] prensibinden yola çıkarak kaynak kod ile birlikte ikili yürütülebilir dosyaların da güvenlik zayıflığı analizlerinin yapılması gerektiğini vurgulamışlardır. Çalışmalarında C/C++ da güvenlik zayıflığına neden olan "system()" ve "popen()" fonksiyonlarının kullanımının tespit etmeği amaçlamışlardır. Yöntemlerini, geliştirdikleri yazılım aracı ile gerçekleştirerek kullanıma sunmuşlardır. Ancak, geliştirilen yöntem yüksek yanlış-pozitif sonuçlar vermektedir. Carnegie Melon Üniversitesinden araştırmacı Cha vd. [18], içerisinde DEBUG bilgisi bulunmayan ikili yürütülebilir dosyalarda güvenlik zayıflığı ve bu zayıflığı ortaya çıkaracak olan girdilerin belirlenmesi için yeni bir

yöntem geliştirmişlerdir. Yöntem, PE ve ELF formatlı yazılımlar üzerinde sınanmış ve 29 adet zayıflık tespit edilmiştir. Bu zayıflıklardan 2 tanesi daha önceden ifşa edilmemiş zayıflıklardır. Yöntemde, kullanıcı girişlerine bağlı hafıza işlemlerinin (load ve store) yapıldığı çalışma yollarının tespit edilmesi ve zayıflığa neden olan girdinin üretilmesi çözülmeye çalışılmıştır. Çalışma kapsamında daha önce geliştirilen sembolik çalıştırma motorları (CUTE, KLEE, SAGE, Mc Veto, AEG ve S2E) tasarım kararları doğrultusunda değerlendirilmiş ve ihtiyaçlar belirlenmiştir. Özellikle performans açısından etkili olan "daha önce yapılan işlerin sonraki işlerde tekrar edilmemesi, tekrar kullanılabilir olması" tasarım kararını hiçbir sembolik çalıştırma motorunun sağlamadığı; yine birçok sembolik çalıştırma motorunun da sembolik hafıza yönetimi konusunda yetersiz olduğu ifade edilmiştir. Çevrimiçi (olası bütün yürütme yolları kullanılarak) ve çevrimdışı (sadece bir yürütme yolu üzerinden) sembolik çalıştırma yöntemlerinin güçlü yanları esas alınarak hibrit bir sembolik çalıştırma yöntemi benimsenmiştir. Yöntemde çevrimiçi ve çevrimdışı sembolik çalıştırma yöntemleri durumsal değerlendirilip geçiş yapıldığı için yürütme yollarının belirlenmesinde yaşanan hafıza tüketme sorunu giderilmeye çalışılmıştır. Araştırmacıların daha önceki çalışmalarını baz alarak geliştirmiş oldukları yöntem hem zayıflık tespiti hem de zayıflığa neden olabilecek girdinin üretilmesi açısından oldukça başarılı bir çalışmadır.

Yapılan çalışmalar genel olarak değerlendirildiğinde, önerilen yöntemlerde sınırlılıkların fazla olduğunu görmekteyiz. Programlama dili, derleme kipi, geliştirme ve kurulum ortamı, kullanılan çerçeveler, ağ protokolleri gibi yazılımın karakteristiğini belirleyen her unsur, yazılım güvenliği zayıflık analizinin yapılmasını zorlaştırmakta ve karmaşıklığını üstel olarak artırmaktadır. Bu karmaşıklıktan kurtulmak için çalışmalar belirli kısıtlamalarla yürütülmüştür. Çalışmalardaki kısıtlamalar geliştirilen yöntemlerin diğer durumlarda uygulanabilirliğini azaltmaktadır.

3. ÖNERİLEN YÖNTEM (PROPOSED METHOD)

3.1. Yöntemin Kurgulanması (Construction of The Method)

Bu çalışmada, ikili yürütülebilir dosya formatlı yazılımlarda ikili işlem kod dağılımları ve dizilimleri makine öğrenmesi teknikleri ile analiz edilerek yazılımın güvenli veya zayıf olarak sınıflandırılması için yeni bir statik analiz yöntemi geliştirilmiştir. Yöntem, aşağıda önerilen H-1 hipotezi üzerine geliştirilmiştir. H-1: Güvenlik zayıflığı içeren ikili yürütülebilir dosya formatındaki yazılımlarda dallanma, döngü, yazmaç değer güncelleme, hafıza değer güncelleme, yığın işlemleri ve sistem çağrıları için kullanılan ikili işlem kodların dağılımları ve dizilimleri arasında benzerlikler vardır.

H-1 hipotezinin sınanması için kurgulanan yöntem ve yöntemin uygulandığı prototip sisteme ait süreçler ve sistem mimarisi şekil 2'de sunulmuştur.

3.2. Yöntemin Matematiksel Modeli (Mathematical Model of The Method)

İşlemci mimarisinin desteklediği ikili işlem kodları Eş. 1 ile ifade edilir.

$$C=\{c_1, c_2, c_3, \ldots, c_c\} \quad (1)$$

Güvenlik zayıflığı içeren zayıf yazılım veri seti Eş. 2 ile ifade edilsin.

$$V=\{v_1, v_2, v_3, \ldots, v_v\} \quad (2)$$

Güvenlik zayıflığı içermeyen güvenli yazılım veri seti Eş. 3 ile ifade edilsin.

$$S=\{s_1, s_2, s_3, \ldots, s_s\} \quad (3)$$

Veri setinin tamamı, Eş. 1 ve Eş. 2'nin bileşkesi olarak Eş. 4 ile ifade edilebilir.

$$VS=V \cup S \quad (4)$$

$\forall E \in VS$ ve $\forall c_i \in C$ olmak üzere; Eş. 5ile bir yazılım öğesinin ikili işlem kod dizilimlerinden oluştuğunu ifade edebiliriz.

$$E = A[c_i] \quad (5)$$

$\forall C_i \in C$ olmak üzere; Eş. 6 ile ikili işlem kodlarının olası tüm n-gramlık dizilimlerini içerdiğini belirtebiliriz.

$$C^n= C_1 \text{ X } C_2 \text{ X } \ldots. \text{ X } C_n \quad (6)$$

$g(E, n)$: $E \rightarrow C^n$, yazılımın n-gramlık ikili işlem kod dizilimlerini üreten fonksiyon olmak üzere; $P(E):E \rightarrow [0\text{-}1]$ olasılık fonksiyonu, verilen bir yazılımın n-gramlık ikili işlem kod dizilimlerini kullanarak arabellek taşması zayıflığını içerme olasılığını makine öğrenmesi algoritmaları ile tahmin etsin.

Veri setinde güvenli ve zayıf olarak sınıflandırılmış yazılım birimleri, geliştirilen yöntemin sınanmasında hem eğitim hem de test verisi olarak kullanılarak yöntemin başarımı hesaplanır. Yöntemde kullanılacak olan öğreticili makine öğrenmesi algoritmaları ayrık veya birlikte koşturularak yazılım biriminin güvenli-zayıf sınıflandırması yapılır.

4. GERÇEKLEME VE BAŞARIM SONUÇLARI (IMPLEMENTATION AND ACHIEVEMENTRESULTS)

4.1. Yöntemin Süreçleri (Processes of The Method)

Yöntemde izlenen süreçler aşağıda sıralanmıştır:

Veri setinin hazırlanması

- NVD (National Vulnerability Database), CVE(Common Vulnerabilities and Exposures) ve OSVDB (Open Source Vulnerability Database) gibi yazılım zayıflık veri

tabanlarında zayıflıklar tanımlı olmasına karşın veri seti olarak kullanılacak yazılım bileşenlerinin paylaşılmamış olması nedeni ile veri seti kendi çalışmalarımızla oluşturulmuştur.

- Veri setindeki yazılımlar, statik kod analiz araçlarının karşılaştırılması için NIST tarafından üretilen "yığın arabellek taşma zayıflığı" içeren test senaryoları [19] esas alınarak hazırlanmıştır. Veri setindeki her bir yazılım öğesinin güvenli ve zayıf sürümleri "Code: Blok" yazılım geliştirme ortamında, C/C++ programlama dilinde geliştirilmiştir. Veri setimiz, 15 adet yığın arabellek taşması zayıflığı içeren zayıf yazılım bileşeni ve bu zayıflıkların giderildiği 15 adet güvenli yazılım bileşeni olmak üzere toplam 30 yazılım bileşeninden oluşmaktadır.

Veri setinin sınıflandırılması:

- Veri setindeki yazılımlar güvenli ve zayıf olarak sınıflandırılarak veri setinde işaretlenmiştir.
- Veri setinde zayıf veya güvenli olarak sınıflandırılmış her bir yazılım öğesi kara kutu test yöntemleri ile yığın arabellek taşması zayıflığını ortaya çıkarma amaçlı sınanarak zayıf veya güvenli olduğu doğrulanmıştır.

İkili işlem kod dizilimlerinin üretilmesi:

- Veri setindeki her bir yazılım öğesinin ikili işlem kod dizilimleri Python programlama dilinde Distorm disassembler kütüphanesi kullanılarak üretilmiştir.

N-gramlık ikili işlem kod dizilimlerinin üretilmesi:

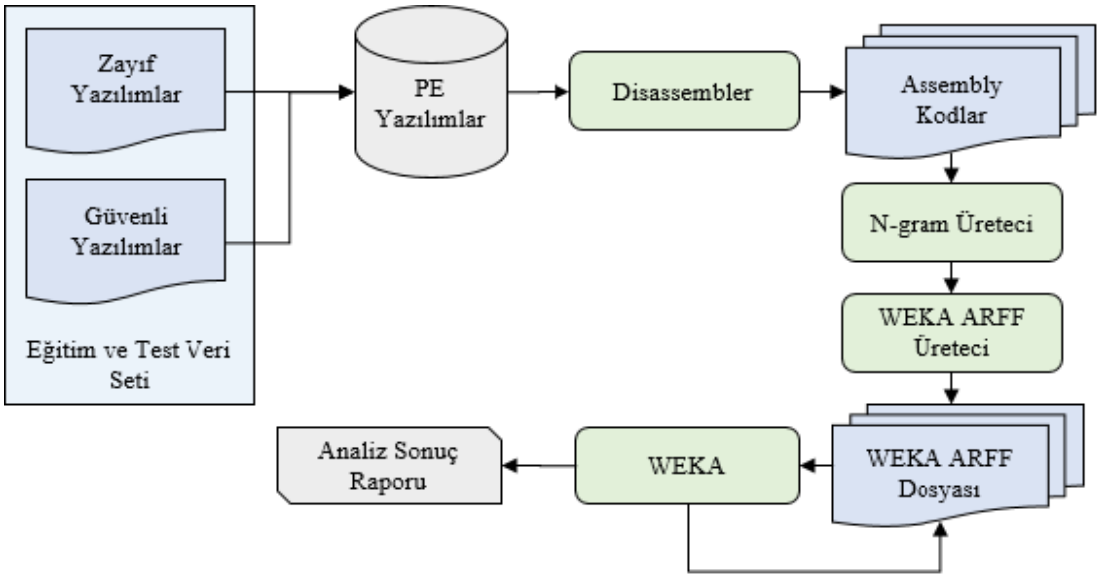
- Geliştirilen yardımcı yazılımlar ile ikili işlem kod dizilimlerinden n-gramlık ikili işlem kod dizilimleri üretilmiştir. Tablo 1 ile örnek bir yazılım bileşeninden türetilen n-gramlık ikili işlem kod dizilimleri gösterilmiştir.

WEKA nitelik-ilişki dosyasının üretilmesi:

- n-gramlık ikili işlem kod dizilimlerinin WEKA [20] aracı ile analiz edilebilmesi için ARFF (Attribute-Relation File Format) dosyaları üretilmiştir.

WEKA ile yazılımların güvenli/zayıf olarak sınıflandırılması:

- Veri temizliği: ARFF dosyasında özellik olarak yer alan ancak veri analizi sırasında öznitelik olarak kullanılmayacak olan özellikler temizlenmiştir.



Şekil 2. Sistem mimarisi (System architecture)

Tablo 1.N-gram ikili işlem kod dizilimleri(n-gram binary opcode sequences)

İkili işlem kodları	2-gram	3-gram	4-gram
dec	dec pop	dec pop nop	dec pop nop add
pop	pop nop	pop nop add	pop nop add inc
nop	nop add	nop add inc	nop add inc mov
add	add inc	add inc mov	add inc mov int
inc	inc mov	inc mov int	inc mov int push
mov	mov int	mov int push	mov int push jb
int	int push	int push jb	int push jb ins
push	push jb	push jb ins	
jb	jb ins		
ins			

- Veri dönüştürme: ARFF dosyasında özellik olarak yer alan n-gramlık ikili işlem kod dizilimleri kelime vektörlerine dönüştürülerek veri setindeki bütün n-gramlık ikili işlem kod dizilimleri öznitelik olarak belirlenmiştir.
- Sınıflandırma algoritmalarının belirlenmesi: Sınıflandırma algoritması olarak; KNN (K En Yakın Komşu), Naïve Bayes, Destek Vektör Makinesi, Bayes Multinomial, Hoeffding Tree, J48, K-Star, Random Forest, Random Tree, Decision Table ve ZeroR algoritmaları kullanılmıştır.
- Eğitim ve test modelinin belirlenmesi: Eğitim ve test modeli olarak WEKA`nın çapraz doğrulama yöntemi kullanılmıştır. Katlama değeri varsayılan değer 10 olarak kullanılmıştır.
- Modelin eğitimi ve sınanması: Veri setindeki veriler 10 gruba bölünerek 9 grup ile model eğitilmiş ve 1 grup ile sınanmıştır. Bu işlem 10 kez tekrar edilerek, çapraz doğrulama yönteminin kuralı olan veri setindeki her bir öğenin hem eğitim hem de test verisi olarak kullanılması sağlanmıştır.

Yöntemin otomatikleştirilmesi:

- Yöntemin tekrarlanabilirliğini sağlamak ve analiz süreçlerini kısaltmak için WEKA`nın bilgi-akış yeteneği kullanılmıştır. Hazırlanan bilgi-akış diyagramı ile üretilen ARFF dosyalarının analizi otomatikleştirilmiştir.

Yöntemin başarım oranı (TPR) yazılımın güvenlik zayıflığı içerme olasılığı ile ilişkilendirilmiştir. Örneğin; %90 TPR oranı ile uygulamalar sınıflandırıldığında, bir yazılım bileşeni "zayıf" olarak sınıflandırılmışsa o yazılımın zayıflık içerme olasılığını %90 olarak kabul edebiliriz.

4.2.Yöntemin Sınanması ve Değerlendirilmesi
(Testing and Evaluation of The Method)

Yöntemin sınanmasında kullanılan donanım konfigürasyonu aşağıda verilmiştir:

- İşlemci: Intel Core i5
- Sistem belleği: 8 GB
- İşletim sistemi: Windows 7 Professional

Geliştirilen yöntem başarımının değerlendirilmesi için kullanılan karışıklık matrisi aşağıdaki gibi oluşturulmuştur.

Tablo 2. Karışıklık matrisi (Confusion matrix)

Mevcut Durum	Tespit	
	Zayıf	Güvenli
Zayıf	TP Doğru-Pozitif (True Positive)	FN Yanlış- Negatif (False Negative)
Güvenli	FP Yanlış-Pozitif (False Positive)	TN Doğru-Negatif (True Negative)

Karışıklık matrisi, veri setindeki mevcut durumun ve sınıflandırma sonuçlarının Doğru-Pozitif (TP), Yanlış-Pozitif (FP), Doğru-Negatif (TN) ve Yanlış-Negatif (FN) değerleri ile ifade edilmesinde yardımcı olur. Sınıflandırma yöntemlerinin başarımları; Doğru-Pozitif Oranı(TPR: TP Rate), Yanlış-Pozitif Oranı (FPR: FPRate) ve Kesinlik (P: Precision) performans değerleri ile kıyaslanmıştır.

TPR, zayıf yazılımların ne oranda doğru tespit edildiğini yüzdesel olarak ifade eder.

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP+FN}} * 100 \tag{7}$$

FPR, güvenli yazılımların ne oranda yanlışlıkla zayıf olarak sınıflandırıldığını yüzdesel olarak ifade eder.

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP+TN}} * 100 \tag{8}$$

Kesinlik (P), zayıf olarak sınıflandırılan yazılımların ne oranda doğru sınıflandırıldığını yüzdesel olarak ifade eder.

$$\mathrm{P} = \frac{\mathrm{TP}}{\mathrm{TP+FP}} * 100 \tag{9}$$

Yöntem, n-gramlık ikili işlem kod dizilimlerini belirlemede kullanılan "n" parametresi ile geliştirilen prototip sistem kullanılarak sınanmıştır. "n" parametresi [1,200] aralığında ardışık olarak sisteme girilmiş ve başarım değerleri TPR (Eş. 7), FPR (Eş. 8) ve P (Eş. 9) denklemleri ile hesaplanmıştır. Elde edilen başarım değerleri Tablo 3 ve Şekil 3'te sunulmuştur.

4.3. Diğer Yöntemler İle Karşılaştırma
(Comparison With The Other Methods)

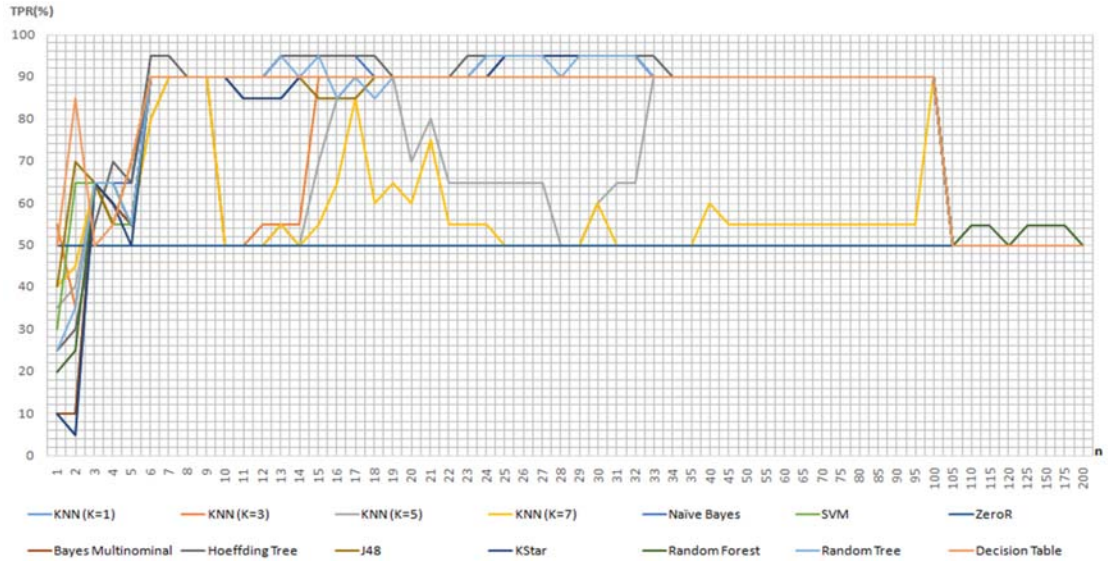
Geliştirilen yöntemin diğer yöntemlerden ayırt edici en belirgin özelliği, H-1 hipotezi ile sunulan ikili işlem kod dizilimlerinin yazılım güvenliği zayıflık analizinde kullanılabileceğini göstermesidir. Ayrıca, makine öğrenmesi tekniklerinin yazılım güvenliği analizinde kullanımı ve yönteme ait gerek yazılım kaynak kodlarının gerekse veri setinin paylaşılmış olması çalışmayı mevcut çalışmalardan farklı kılmaktadır.

Geliştirilen yöntemin diğer yöntemlerle olan karşılaştırması Tablo 4'te sunulmuştur. Mevcut yöntemler, bilinen bir zayıflığı ortaya çıkarmaya yönelik çözümler üretmekte ve başarım sonuçlarını TPR, FPR değerleri ile değil zayıflığın yakalanması ile ifade etmektedirler. Geliştirilen yöntemin başarım oranı TPR, FPR ve P değerleri ile ölçülmüştür. Diğer yöntemlerde olduğu gibi geliştirilen yöntem de problem karmaşıklığını azaltmak için belirli kısıtlamalar (veri setindeki yazılımların formatı, işlemci mimarisi, derleme kipi gibi) altında geliştirilmiştir.

Mevcut çalışmalarda kullanılan veri setleri paylaşılmadığından dolayı geliştirilen yöntemin başarımı diğer yöntemlerde kullanılan veri setleri ile sınanamamıştır. Ancak başarımın karşılaştırılabilmesi için, diğer araştırmacılar ile iletişime geçilerek kendi veri setimizin diğer yöntemler ile analiz edilmesi talep edilmiştir. Tevis [13, 14] veri setimizi kendi yöntemleri ve geliştirdikleri

Tablo 3. Sınıflandırma yöntemlerinin başarım sonuçları (Performance results of classification methods)

Algoritma	*n*	*TPR (%)*	*FPR (%)*	*P (%)*
KNN (K=1)	[25,32]	95,00	5,00	95,50
KNN (K=3)	[15,100]	90,00	10,00	91,70
KNN (K=5)	[33,100]	90,00	10,00	91,70
KNN (K=7)	[7,9]	90,00	10,00	91,70
Naïve Bayes	[13,17]	95,00	5,00	95,50
SVM	[6,100]	90,00	10,00	91,70
ZeroR	[1,200]	50,00	50,00	25,00
Bayes Multinomial	[6,100]	90,00	10,00	91,70
Hoeffding Tree	[13,18]	95,00	5,00	95,50
J48	[18,100]	90,00	10,00	91,70
K-Star	[25,32]	95,00	5,00	95,50
Random Forest	[6,100]	90,00	10,00	91,70
Random Tree	[24,27]	95,00	5,00	95,50
Decision Table	[6,100]	90,00	10,00	91,70



Şekil 3. Sınıflandırma yöntemlerinin başarım oranları (Performance results of classification methods)

FindSSV yazılımı ile analiz ederek sonuçları paylaşmıştır. FindSSV sonuç raporlarına göre;

- Veri setinde yer alan 15 adet güvenli uygulamadan 12 tanesi, "strncpy" C/C++ kütüphane fonksiyonunu kullandığı gerekçesi ile "düşük riskli" yazılım olarak değerlendirmiştir. Geriye kalan 3 adet uygulamanın analizinde FindSSV uygulaması beklenmedik şekilde sonlanarak analizi tamamlayamamıştır. Düşük riskli olarak değerlendirilen yazılımları güvenli kabul edebiliriz.

- Veri setinde yer alan ve arabellek taşması zayıflığı içeren 15 adet uygulamadan 10 tanesi, "strcpy" ve "gets" C/C++ kütüphane fonksiyonlarını kullandığı için "çok yüksek riskli" yazılım olarak değerlendirmiştir. Geriye kalan 5 adet uygulamanın analizinde FindSSV uygulaması beklenmedik şekilde sonlanarak analizi tamamlayamamıştır.

Geliştirilen yöntemin başarımı Tevis vd. [13, 14] ile karşılaştırıldığında; veri setinde yer alan güvenli ve zayıf uygulamaların çoğunlukla her 2 yöntemde de doğru sınıflandırıldığını görmekteyiz. Bu durum, geliştirilen veri setindeki yazılım bileşenlerinin doğru sınıflandırıldıklarını ve diğer yöntemler tarafından da kullanılabilir olduğunu göstermektedir. FindSSV tarafından analizi tamamlanamayan uygulamaların güvenli veya zayıf olarak sınıflandırması yapılamadığından yöntemin başarımı TPR, FPR ve P değerleri ile verilememiştir. Sezgisel olarak analizi tamamlanamayan yazılımları güvenli kabul ettiğimizde başarımı (TPR: %66,66; FPR: %0 ve P: %100), zayıf kabul ettiğimizde başarımı (TPR: %100; FPR: %20 ve P: %83,33) ve ortalama başarımı (TPR: %83,33; FPR: %10; P: %91,66) hesaplanabilir (*). Geliştirilen yöntemde elde edilen yüksek başarım (TPR: %95; FPR:%5 ve P: %95,5) değerleri yöntemin başarısını ve uygulanabilirliğini göstermektedir.

Tablo 4. Diğer yöntemler ile karşılaştırma(Comparison with the other methods)

Kriterler	Yöntemler				
	Önerilen yöntem	Tevis vd. [13, 14]	Du Varney vd. [15]	Cova vd. [16]	Cha vd. [18]
Analiz yöntemi	Statik	Statik	Statik	Statik	Dinamik
Uygulanan metodoloji	Makine öğrenmesi	Dosya formatı analizi	Derleyici ve dosya formatı güncellemesi	PE ve kaynak kodun birlikte analizi	Sembolik çalıştırma ve akış kontrol denetimi
Girdi türü	PE	PE	ELF	ELF	PE, ELF
Kaynak kod gereksinimi var mı?	Hayır	Hayır	Evet	Evet	Hayır
Kaynak kod ve veri seti paylaşımı var mı?	Evet	Hayır	Hayır	Hayır	Hayır
Başarım	TPR: %95 FPR: %5 P:%95,5	TPR: %83,33 FPR: %10 P:%91,66 (*)	-	-	-

Geliştirilen yöntemi diğer çalışmalardan ayırt edici özellikleri şöyle sıralanabilir:

- Yüksek başarım sonuçları (TPR: %95; FPR:%5 ve P: %95,5),
- Makine öğrenmesi tekniklerinin yığın arabellek taşması zayıflık analizinde kullanılması,
- Veri setinin ve prototip sistem kaynak kodlarının paylaşılması,
- Açık kaynak kodlu mimari üzerine geliştirilmesi.

5. SİMGELER (SYMBOLS)

FP : Yanlış-Pozitif (False-Positive)
FPR : Yanlış-Pozitif Oranı (False-Positive Rate)
P : Kesinlik (Precision)
TP : Doğru-Pozitif (True-Positive)
TPR : Doğru-Pozitif Oranı (True-Positive Rate)

6. SONUÇLAR (CONCLUSIONS)

Tablo 3ve Şekil 3'te verilen yüksek başarım değerleri, H-1 hipotezinde önerildiği gibi PE formatlı yazılımların ikili işlem kod dizilimleri ve dağılımları kullanılarak zayıf veya güvenli olarak sınıflandırılabileceğini göstermektedir. Özellikle KNN, Naïve Bayes, Hoeffding Tree, K-Star ve Random Treealgoritmaları ile elde edilen yüksek başarım oranları bu algoritmaların zayıflık analizi için uygun sınıflandırma algoritmaları olduğunu göstermektedir. Aynı zamanda çoğunluk sınıflandırıcı olarak da bilinen ZeroR algoritması diğer sınıflandırıcıların performansını ölçmek için temel sınıflandırıcı olarak kullanılmıştır [21]. ZeroR algoritması, sınıflandırmada eğitim verisindeki örneklerin sayılarını esas aldığından doğru-pozitif oranı olarak %50 başarım değerini vererek veri setinde kullanılan zayıf ve güvenli yazılım sayısının eşit olduğunu göstermiştir. KNN algoritması kullanıldığında K komşu sayısı değeri arttıkça başarımın düştüğü gözlemlenmiştir. Yöntemin doğru-pozitif oranı (TPR) başarım değeri ile yazılımın güvenlik zayıflığı içerme olasılığının ilişkilendirilebileceği gözlemlenmiştir. Yöntemin gerçeklendiği prototip sistem ile gerek yazılım geliştirme safhasında gerekse son kullanıcı tarafında arabellek taşması zayıflık analizlerinin yapılabileceği değerlendirilmektedir.

Yapılan çalışmada ikili yürütülebilir dosya formatlı yazılımların güvenlik zayıflığı analizi için temel teşkil edecek veri seti oluşturulmuştur. Veri setinin genişletilmesine yönelik yapılacak çalışmalar ile yazılım güvenliği zayıflık analizinde kullanılmak üzere zayıflık veri tabanı oluşturulabilir. ImageNet benzeri oluşturulacak veri tabanına zayıflık türüne göre sınıflandırılmış yazılım bileşenlerinin kaynak kodları, ikili yürütülebilir sürümleri ve zayıflığı ortaya çıkaran girdileri ile eklenmesi bu alanda yapılacak çalışmalar için çok büyük fayda ve ivme sağlayacaktır.

Geliştirilen yöntem ve prototip sisteme ait kaynak kodlar ve veri seti GitHub [22] platformu üzerinden ilgili akademisyen ve araştırmacılar ile paylaşılmıştır. Yöntem başarımının artırılması için sınıflandırmada belirleyici olan n-gramlık ikili işlem kod dizilimlerinin tespitine ve geliştirilen prototip sistemin kullanılabilirliğinin artırılmasına yönelik çalışmalarımız devam etmektedir. Ayrıca, veri setini gerek kendi geliştireceğimiz gerekse diğer çalışmalardan temin edebileceğimiz yazılımlar ile genişletmek için çalışmalarımızı sürdürmekteyiz.

KAYNAKLAR (REFERENCES)

1. McGraw, G., Software Security, IEEE Security &Privacy, 2 (2), 80-83, 2004.
2. Baca, D.,Developing Secure Software in an Agile Process, Doctoral Dissertation in Computer Science, Blekinge Institute of Technology, 2012.

3. Younan, Y., 25 Years of Vulnerabilities: 1988-2012, Research Report, Sourcefire Crop, 2013.

4. Younan, Y., Joosen, W., Piessens F., Code Injection in C and C++ : A Survey of Vulnerabilities and Countermeasures, Report CW386, July 2004.

5. Akgün, F., Buluş, E., Buluş, H.N., Yazılım Mühendisliği Açısından Uygulamalardaki Ara Bellek Taşması Zafiyetinin İncelenmesi, Elektrik-Elektronik-Bilgisayar Mühendisliği 11. Ulusal Kongresi ve Fuarı, İstanbul-TÜRKİYE, 2005.

6. Sidiroglou, S., Locasto, M.E., Boyd, S.W., Keromytis, A. D., Building a Reactive Immune System for Software Services, USENIX Annual Technical Conference, 2005.

7. Wang, Y., Kelly, T., Kudlur, M., Lafortune, S., Mahlke, S.A., Gadara: Dynamic Deadlock Avoidance for Multithreaded Programs, USENIX Symposium on Operating Systems Design and Implementation, 2008.

8. Alhazmi, O.H., Malaiya, Y.K., Ray I., Measuring, Analyzing and Predicting Security Vulnerabilities in Software Systems, Computers & Security (2006), doi:10.1016/j.cose.2006.10.002, 2006.

9. Ozment, A., Schechter, S.E., Milk or Wine: Does Software Security Improve with Age?, In the proceedings of The Fifteenth Usenix Security Symposium, July 31 - August 4 2006: Vancouver, BC, Canada, 2004.

10. Chess, B., McGraw, G., Static Analysis for Security, IEEE Security &, Privacy, 2 (6), 76-79., 2004.

11. Halkidis, S.T., Tsantalis, N., Chatzigeorgiou, A., Stephanides, G., Architectural Risk Analysis of Software Systems Based on Security Patterns, IEEE Transactions on Dependable and Secure Computing, 5, 3, 2008.

12. Utku A., Doğru İ.A., Permission based detection system for android malware, Journal of the Faculty of Engineering and Architecture of Gazi University, 32 (4), 1015-1024, 2017.

13. Jay-Evan J. Tevis, Automatic Detection of Software Security Vulnerabilities in Executable Program Files, Doctoral Dissertation in Computer Science, Auburn University, 2005.

14. Tevis, Jay-Evan J. et al., Static Analysis of Anomalies and Security Vulnerabilities in Executable Files, ACM SE'06, Mar. 10-12, 2006.

15. D.C. DuVarney, V.N. Venkatakrishnan and S. Bhatkar, SELF: A Transparent Security Extension for ELF Binaries, Proc. New Security Paradigms Workshop, 2003.

16. Cova, M., Felmetsger, V., Banks, G., Vigna, G., Static Detection of Vulnerabilities in x86 Executables, Annual Computer Security Applications Conference (ACSAC), Miami, FL, December, 2006.

17. Balakrishnan, G.,WYSINWYX: What You See Is Not What You eXecute, PhD Thesis, Computer Science Department, University of Wisconsin at Madison, August 2007.

18. Cha, S. K., Avgerinos, T., Rebert A., Brumley, D., Unleashing MAYHEM on Binary Code, Proceedings of the 2012 IEEE Symposium on Security and Privacy, p.380-394, May 20-25, 2012.

19. NIST Software Assurance Reference Dataset, https://samate.nist.gov/SRD/testsuite.php, Erişim tarihi: 11/12/2017.

20. Weka 3 - Data Mining with Open Source Machine Learning Software in Java, https://www.cs.waikato.ac.nz/ml/weka, Erişim tarihi: 11/12/2017.

21. Aydın F., Aslan Z., Diagnosis of neuro degenerative diseases using machine learning methods and wavelet transform, Journal of the Faculty of Engineering and Architecture of Gazi University, 32 (3), 749-766, 2017.

22. Durmus, G., Çalışma Kaynak Kod ve Sonuçları, http://github.com/gdurmus/, Son Erişim Tarihi: 05/04/2018.