İkili Yürütülebilir Uygulamalarda Arabellek Taşması Zayıflığına Neden Olan Şüpheli İkili İşlem Kod Dizilimlerinin Tespiti

Detection of Suspicious OpCode Sequences Causing Buffer Overflow Vulnerabilities in Binary Executable Applications

Gürsoy DURMUŞ
HAVELSAN A.Ş.
gdurmus@havelsan.com.tr
ORCID: 0000-0002-2116-1618

İbrahim SOĞUKPINAR
Gebze Teknik Üniversitesi
ispinar@gtu.edu.tr
ORCID: 0000-0002-0408-0277

Öz

Günlük hayatımızda kullandığımız birçok akıllı cihazı kişisel ve çevresel verilerimizle besliyor, daha işlevsel-yararlı hale gelmelerini sağlıyoruz. İçerdikleri yazılımlar ile "akıllı" davranan bu cihazlar çoğu zaman karar alma aşamasında bizleri yönlendirebiliyor. Günlük hayatımızda artan "yazılım kullanma" ihtiyacı aynı zamanda saldırı yüzeyini artırmakta ve bilinçli kullanıcılarda tedirginlik yaratabilmektedir. Bu durum yazılımların işlevselliği kadar güvenliğini de ön plana çıkarmaktadır. Kullandığımız yazılımın güvenlik zayıflığı içerdiğini öğrendiğimizde hızlıca ya tamamen kullanmaktan vazgeçiyor ya da güvenlik açığı giderilmiş sürümünün yayımlanmasını bekliyoruz. Yazılımlardaki güvenlik zayıflıklarının kullanım öncesi tespiti hem müşteri memnuniyetini artırmakta hem de geliştiricinin saygınlığını artırmaktadır. Bu çalışmada, ikili yürütülebilir yazılımlarda sıkça rastlanan arabellek taşması zayıflığına neden olan şüpheli ikili işlem kod dizilimlerinin tespitine yönelik geliştirilen yöntem ve deneysel sonuçlar paylaşılmıştır.

Anahtar Sözcükler: *Yazılım güvenliği, yazılım güvenliği zayıflığı, makine öğrenmesi, arabellek taşması*

Gönderme ve kabul tarihi: 04.10.2019 - 19.11.2019
Makale türü: Araştırma

Abstract

We feed many smart devices that we use in our daily lives with our personal and environmental data and make them more functional and useful. These devices, which are smart with the software they contain, can often guide us in decision-making. The increasing "need to use a software" in our daily lives also increases the attack surface and can create doubtfulness for privacy-conscious users. This makes the software security as important as functionality. When we find out that the software we use contains security vulnerability we either quickly stop using it completely or we are waiting for the new release of the secure version. Detection of security weaknesses in software before use increases customer satisfaction and developer reputation. In this study, we present a new method and its success for detecting suspicious opcode sequences which cause buffer overflow vulnerabilities in binary executables.

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

1. Giriş

Bir yazılımdan şüphesiz ilk beklenen, asli işlevini eksiksiz ve tutarlı bir şekilde yerine getirmesidir. Ancak bu özellik tek başına bir yazılımı kaliteli olarak nitelendirmek için yeterli değildir. Bir yazılımın kaliteli olarak nitelendirilebilmesi için işlevsel olmasının yanı sıra güvenilir, kolay

kullanılabilir, verimli, bakımı kolay ve taşınabilir olması gerekmektedir. Sıraladığımız bu kalite metrikleri, endüstrinin ihtiyaçları göz önüne alınarak belirlenmiş ve 1991 yılında yayınlanan, 2004 yılında revize edilen ISO 9126-4 belgesinde [1] tanımlanmıştır.

Bilişim dünyasında yaşanan hızlı gelişimler şüphesiz yazılım kalitesine bakış açısını da etkilemiş ve yeni kalite metriklerinin tanımlanmasını zaruri kılmıştır. 2011 yılında yayınlanan ISO-25010 [2] standardında, ISO-9126'da tanımlı yazılım kalite metriklerine ilave olarak uyumluluk (compatibility) ve güvenlik (security) metrikleri de eklenmiştir.

Bir yazılımda, güvenlik harici diğer kalite gereksinimlerinin yeterli düzeyde karşılanamamış olması kullanıcı memnuniyetsizliğine ve hizmet aksamalarına neden olurken, olası bir güvenlik zayıflığı; kişisel, kurumsal ya da ulusal bilgilerin sızdırılmasına, bilgi bütünlüğünün bozulmasına, maddi ve manevi kayıplara neden olabileceği gibi görev kritik sistemlerde can kaybına bile neden olabilmektedir.

Bireylerin, kurumların ve hatta milletlerin geleceklerini tehdit edecek sonuçlar doğuran siber saldırıların yaşandığı günümüzde; yazılımlar adeta bir silah-hedef olarak kullanılmaya başlandı. Bu durum bireysel ve kurumsal alanlarda kullanılacak yazılımların işlevselliği kadar güvenliğinin de önemsenmesine neden olmuştur. Gelinen noktada; "yazılımlarınız güvenli ise güvendesiniz, değilse değilsiniz" ifadesini çok rahat söyleyebiliriz.

Bilinçli bireysel kullanıcılar; yazılımların güncellemelerini takip ediyor özellikle güvenlik açığına dair yamalar varsa hemen uygulayıp "güvenli" tarafa geçiyor. Güvenlik zayıflığı içeren yazılım kullanıldığı süre zarfında ne gibi saldırılara maruz kalındığı ne tür risklerin kabullenildiği hep sonradan fark ediliyor. Kurumsal kullanıcılarda bu süreç biraz daha zaman alıcı olabiliyor. Bilişim güvenliği uzmanlarınca zayıflık veri tabanları takip ediliyor ve kullanımda olan yazılımda teyit edilen güvenlik zayıflığı için güncelleme talebi oluşturuluyor. Yazılım güncellemelerinin diğer paydaşlara olan etkilerinin araştırılması, mevcut sistemle uyumlu olup olmadığının kontrol edilmesi ve nihayetinde karar vericilerin onaylama süreci zayıflığı bilerek ve kabullenerek sürdürmeye neden olabiliyor. Bu şekilde yürütülen bilgi güvenliği faaliyetleri sıfır gün saldırıları bir yana, ilgili güncellemeler uygulanıncaya kadar sömürülmeye açık halde bilgi güvenliğini riske atmaktadır. Bu ve benzer birçok sebeple, yazılım güvenliği analizi konusunda reaktif çözümlerin yerlerini proaktif çözümler almaya başlamıştır.

Proaktif bir yaklaşımla; kullanım öncesi yazılımlardaki zayıflıkların analizi hem siber saldırıların başarım oranını düşürecek hem de doğacak zararların azalmasını sağlayarak bizi daha güvende hissettirecektir. En yalın hali ile proaktif çözüm olarak; 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. Analiz edilecek yazılıma ait güvenlik gereksinimlerine, yazılım kaynak kodlarına ve uzman güvenlik test ekiplerine olan ihtiyaç gibi bağımlılıklar nedeniyle birçok yazılım için güvenlik analizleri üstünkörü yapılmaktadır [3].

Bilimsel bir çalışma alanı doğuran bu durum akademik alanda uzun süredir çalışmaların kesintisiz devam etmesine vesile olmuştur. Gerek teknolojik gelişmeler, gerekse "yazılım" ve "platform" çeşitliliği ve her bir yazılım türüne özgü bilinen veya sonradan ortaya çıkarılan güvenlik zayıflıkları çalışma alanını genişletmekte ve güncel kılmaktadır.

Bu çalışmada; ikili yürütülebilir yazılımlarda sıkça gözlemlenen arabellek taşması zayıflığına neden olan şüpheli ikili işlem kod dizilimlerinin tespiti için makine öğrenmesi tabanlı bir yöntem önerilerek sınanmıştır. Önerilen yöntemin deneysel sonuçları ve başarımı sunulmuştur.

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

2. İlgili Çalışmalar

ISO/IEC 25010 kalite modeli, ürün kalite değerlendirme sisteminin temel taşıdır. Kalite modeli, bir yazılım ürününün özelliklerini değerlendirirken hangi kalite özelliklerinin dikkate alınacağını belirler. ISO standartlarına göre yazılım güvenliği; yazılımın gizlilik, bütünlük, inkâr edilememe ve hesap verebilirlik gibi bilgi güvenliği unsurlarının göz önüne alınmasını ve korunmasını esas alır [2]. Bununla birlikte, yazılım güvenliği

kapsamında yazılımın saldırılara ve art niyetli kullanımlara karşı korunaklı/dirençli olmasına da dikkat edilir [4]. Yazılım güvenliği; yazılım geliştirme yaşam döngüsünün her bir safhasında dikkate alınması ve korunması gereken bir özellik olarak karşımıza çıkmaktadır.

Geliştirilen her yazılımda fark edilmiş ya da fark edilmemiş/edilememiş birçok yazılım hatası olabilir. Bir yazılım hatası, bilgi güvenliği unsurlarının ihlaline sebebiyet vermesi durumunda (sunulan hizmeti engelleme, yetkisiz erişim sağlama veya veri bütünlüğünü bozma) yazılım güvenliği zayıflığı olarak nitelendirilebilir [5].

Yaygın kullanılan yazılımlar için tespit edilen yazılım güvenliği zayıflıkları EDB (Exploit Database), CVE (Common Vulnerabilities and Exposures), NVD (National Vulnerability Database) ve OSVDB (Open Sourced Vulnerability Database) gibi zayıflık veri tabanlarına kaydedilmekte ve takip edilmektedir. CVE ve NVD zayıflık veri tabanlarındaki zayıflıklar ve zayıflıkların önem derecesine göre dağılımları incelenmiş, arabellek taşmalarının çok yaygın olduğu ve önemli derecede zayıflıklar doğurduğu bildirilmiştir [6].

Çizelge-1: Yazılım güvenliği zayıflıkları ve önem derecesine göre dağılımı

Zayıflık	Dağılım (%)	Zayıflık Önceliği (%)
Arabellek taşması	14	23
Girdi doğrulama	10	7
Kod enjeksiyonu	10	10
SQL enjeksiyonu	10	20
Erişim denetimi	11	10
XSS	13	1
Diğer	32	29

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. Özellikle 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 [7]. 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 [8].

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 [9,10]. 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 [11]. 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 [12].

Kaynak koduna erişim sağlanamayan PE (Portable Executable) 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 [13].

Tevis ve arkadaşları [14,15] 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.

DuVarney ve arkadaşları [16], ELF (Executable and Linkable Format) 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 ve arkadaşları [17], x86 işlemci mimarisinde ELF formatlı ikili yürütülebilir yazılımlarda zayıflık analizlerini hem statik hem de sembolik çalıştırma yöntemlerini uygulayarak tespit etmeye çalışmışlardır. Cova ve arkadaşları, "gördüğün (kod), çalıştırdığın değildir" [18] prensibinden yola çıkarak kaynak kod ile birlikte ikili yürütülebilir yazılımların da güvenlik zayıflığı analizlerinin yapılması gerektiğini vurgulamışlardır. Çalışmalarında; C/C++ programlama dilinde hatalı kullanılması durumunda zayıflığa neden olabilecek kütüphane fonksiyonlarını ("system" ve "popen") belirtmişler ve bu fonksiyonların veri setlerindeki yazılımlarda kullanımlarını yüksek FPR (False-Positive Rate) başarım oranı ile tespit etmişlerdir.

Carnegie Melon Üniversitesinden araştırmacı Cha ve arkadaşları [19], içerisinde DEBUG bilgisi bulunmayan ikili yürütülebilir yazılımlarda güvenlik zayıflığı ve bu zayıflığı ortaya çıkaracak olan girdilerin belirlenmesi için yeni bir yöntem geliştirmişlerdir. Çalışmada, sembolik çalıştırma motorlarının (CUTE, KLEE, SAGE, McVeto, AEG ve S2E) çevrimiçi (olası bütün yürütme yolları kullanılarak) ve çevrimdışı (sadece bir yürütme yolu üzerinden) çalıştırma yöntemlerinin güçlü yanları esas alınarak hibrit bir sembolik çalıştırma yöntemi benimsenmiştir. Araştırmacıların daha önceki çalışmalarını esas alarak geliştirmiş oldukları yöntem hem zayıflık tespiti hem de zayıflığa neden olabilecek girdinin üretilmesi açısından oldukça başarılı bir çalışmadır.

Durmuş ve Soğukpınar [3], PE formatlı ikili yürütülebilir yazılımlarda sıkça rastlanan arabellek taşması zayıflığının tespitinde makine öğrenmesi yöntemlerini kullanarak yazılımların zayıf-güvenli olarak sınıflandırılabileceğini göstermişlerdir. Çalışma kapsamında kullanılan veri kümesi ve geliştirilen yazılımlar kaynak kodları ile birlikte GitHub platformu üzerinden paylaşılarak gelecek çalışmalar için altyapı oluşturmuşlardır.

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 artırmaktadır. Bu karmaşıklıktan kurtulmak için çalışmalar belirli kısıtlamalarla yürütülmüştür. Çalışmalardaki kısıtlamalar geliştirilen yöntemlerin diğer durumlarda uygulanabilirliğini azaltmaktadır.

3. Önerilen Yöntem ve Geçerleme

3.1. Yöntemin Kurgulanması

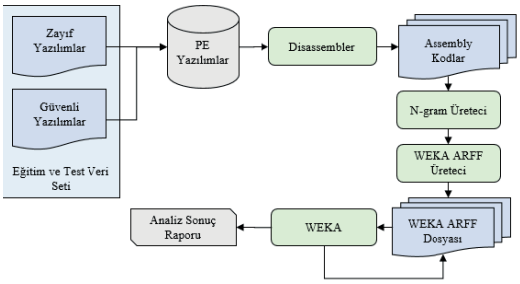
Bu çalışma, kaynak [3]'de verilen "makine öğrenmesi teknikleri ile ikili yürütülebilir dosyalarda arabellek taşması zayıflığı analizi için yeni bir yaklaşım" başlıklı çalışmanın geliştirilmesi üzerine kurgulanmıştır. Çalışmada öne sürülen H-1 hipotezi makine öğrenmesi teknikleri kullanılarak sınanmış ve kullanılan veri kümesi kapsamında yüksek başarım değerleri {TPR (True-Positive Rate): %95; FPR (False-Positive Rate):%5 ve P (Precision): %95,5} ile doğrulanmıştır.

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.

Bu çalışmada; ikili yürütülebilir yazılımlarda sıkça gözlemlenen arabellek taşması zayıflığına neden olan şüpheli ikili işlem kod dizilimlerinin tespiti için geliştirilen yöntem ve öncül çalışmamıza [3] olan etkileri değerlendirilmiştir. Çalışma kapsamında önerilen H-2 hipotezi şöyledir:

H-2: İkili yürütülebilir yazılımlarda "arabellek taşması zayıflığı" belirli ikili işlem kod dizilimleri ile tespit edilebilir.

H-2 hipotezinin sınanması için geliştirilen prototip sisteme ait bileşenler ve sistem mimarisi aşağıda sunulmuştur.



Şekil-1: Sistem mimarisi

3.2. Matematiksel Model

Güvenlik zayıflığı içeren uygulamalar Eş.1 ile ifade edilsin.

$$V=\{v_1, v_2, v_3, \ldots, v_n\} \quad (1)$$

Güvenlik zayıflığı içermeyen uygulamaları da Eş.2 ile ifade edelim.

$$S=\{s_1, s_2, s_3, \ldots, s_n\} \quad (2)$$

Çalışmada kullanılacak olan uygulama veri kümesi Eş.3 ile ifade edilebilir.

$$D=V \cup S \quad (3)$$

İşlemci mimarisinin desteklediği ikili işlem kodları Eş.4 ile ifade edilebilir.

$$C=\{c_1, c_2, c_3, \ldots, c_n\} \quad (4)$$

$\forall E \in D$ olmak üzere; d() fonksiyonu E'nin ikili işlem kodlarını üretsin.

$$d:E \rightarrow O[j] \quad (5)$$

Tanımlanacak g() fonksiyonu, verilecek "n" parametresi ile üretilen ikili işlem kodlarından n-gramlık ikili işlem kod dizilimlerini üretsin.

$$g:(n, O[j]) \rightarrow O'[k] \quad (6)$$

Üretilen n-gramlık ikili işlem kod dizilimlerinden tanımlanacak bir f() seçici fonksiyonu ile zayıflık tespitinde kullanılabilecek ayırt edici özelliğe sahip olanları seçilsin.

$$f: O'[k] \rightarrow O''[m] \quad (7)$$

P() fonksiyonu verilen n-gramlık ikili işlem kod dizilimlerini kullanarak yazılımı "0:zayıf"veya "1:güvenli"olarak tahmin etsin.

$$P: O'[x] \rightarrow \{0, 1\} \quad (8)$$

$\forall E \in D$ için; Eş.9'un sağlanması durumunda, şüpheli ikili işlem kod dizilimlerini belirlemek için kullanılacak öznitelik seçim fonksiyonun Eş.7 de belirtildiği üzere doğru çalıştığı söylenebilir.

$$P(g(n,d(E))) = P(f(g(n,d(E)))) \quad (9)$$

3.3. Sistem Bileşenleri

3.3.1 Veri Kümesi

Çalışma kapsamında, [3]'de kullanılan veri kümeleri genişletilerek yazılım bileşen sayısı 30'dan 112'ye çıkarılmıştır. Veri kümesindeki yazılımlar NIST tarafından üretilen "yığın arabellek taşma zayıflığı" içeren test senaryoları [20] esas alınarak hazırlanmıştır. Veri kümesindeki her bir yazılım öğesinin güvenli ve zayıf sürümleri "Code::Blok" yazılım geliştirme ortamında, C/C++ programlama dilinde geliştirilmiştir. Veri kümesinde zayıf veya güvenli olarak etiketlendirilmiş her bir yazılım öğesi, kara kutu test tekniği olan "sınır değer analizi" yöntemi ile herhangi bir yardımcı test aracı kullanılmadan elle sınanarak zayıf veya güvenli olduğu doğrulanmıştır.

3.3.2 Veri Dönüştürücü

Veri kümesindeki her bir yazılım öğesinin ikili işlem kod dizilimleri Python programlama dilinde Distorm disassembler kütüphanesi kullanılarak üretilmiştir. Geliştirilen yardımcı yazılımlar ile ikili işlem kod dizilimlerinden n-gramlık ikili işlem kod dizilimleri üretilmiştir. Çizelge-2 ile örnek bir yazılım bileşeninden türetilen n-gramlık ikili işlem kod dizilimleri gösterilmiştir.

Çizelge 2: N-gram ikili işlem kod dizilimleri

İkili işlem kodları	2-gram	3-gram
dec pop nop inc …..	dec pop pop nop nop inc …..	dec pop nop pop nop inc …..

3.3.3 ARFF Üreteci

n-gramlık ikili işlem kod dizilimlerinin WEKA [21] aracı ile analiz edilebilmesi için ARFF (Attribute-Relation File Format) dosyaları üretilmiştir. Üretilen ARFF dosya formatı aşağıdaki gibidir:

```
@relation SW_SEC_DATA_STRUCTURE
@attribute FILE string
@attribute NGRAM string
@attribute CLASS {Vulnerable,Secure}

@data
```

Şekil-2: ARFF dosya formatı

3.3.4 WEKA Aracı

WEKA'nın sağladığı önişlem yetenekleri ile ARFF dosyası sınıflandırma aşamasına hazır hale getirilmiştir. Yapılan önişlemler aşağıda belirtilmiştir.

Üretilen ARFF dosyasında yer alan ikili işlem kod dizilimleri (@NGRAM) WEKA'nın "StringToWordVector" fonksiyonu ile özniteliğe dönüştürülmüştür. "StringToWordVector" fonksiyonu için verilen parametre şöyledir:

- weka.filters.unsupervised.attribute.StringToWordVector -R first-last -W 1000 -prune-rate -1.0 -N 0 -stemmer weka.core.stemmers.NullStemmer -stopwords-handler weka.core.stopwords.Null -M 1 -tokenizer "weka.core.tokenizers.WordTokenizer -delimiters \" \\r\\n\\t.,;:\\'\\\"()?!\""

ARFF dosyasında nitelik olarak yer alan ancak veri analizi sırasında öznitelik olarak kullanılmayacak olan @FILE özniteliği WEKA'nın "öznitelik silme" yeteneği ile temizlenmiştir.

4. Deneysel Sonuçlar

Çalışmanın bu kısmında, kaynak [3]'de en iyi başarımı sağlayan 6-gram'lık ikili işlem kod dizilimleri referans alınarak yaptığımız analizler detaylı olarak anlatılacak ve kaynak [3]'deki başarım oranları ile karşılaştırılacaktır.

4.1 Öznitelik Seçimi

Kaynak [3]'deki çalışmada, üretilen ARFF dosyalarında yer alan ikili işlem kod dizilimlerine ait bütün değerler sınıflandırma aşamasında birer öznitelik olarak kullanılmıştır. H-2 hipotezinde belirtilen "belirli ikili işlem kod dizilimlerinin" tespiti için WEKA'nın "CfsSubsetEval" (Correlation-based Feature Subset Selection) fonksiyonu kullanılmıştır. "CfsSubsetEval" fonksiyonu verilen özniteliklerden her birinin tek başına sınıflandırmada ayırt edici özelliğini dikkate alarak sınıflandırmada kullanılacak daha küçük bir öznitelik kümesinin elde edilmesini sağlar [21].

6-gram'lık ARFF dosyasında, "CfsSubsetEval" fonksiyonu @CLASS özniteliğine göre sınıflandırma yapacak şekilde "BestFirst" algoritması ile aşağıdaki parametre ile koşturulmuştur:

- weka.attributeSelection.CfsSubsetEval -P 1 -E 1
- weka.attributeSelection.BestFirst -D 1 -N 5

Veri kümesinde yer alan 112 adet ikili yürütülebilir dosya için, üretilen 6-gram'lık ikili işlem kodlarından toplamda 3714 adet farklı öznitelik üretilmiştir. "CfsSubsetEval" fonksiyonu; 3714 öznitelikten yüksek korelasyon ve ayırt edici özelliğe sahip 4 özniteliği ön plana çıkarmaktadır. Bu öznitelikler Çizelge-3'te sunulmuştur.

Çizelge 3: Ayırt edici özelliğe sahip öznitelikler

İkili işlem kod dizilimi	Ortalama	St. Sapma
add_add_or_add_add_push	0,97	0,16
add_arpl_jae_xor_pop_push	0,96	0,18
inc_xor_pop_pop_imul_jb	0,97	0,16
pop_ins_popa_push_jb_outs	0,57	0,49

Belirlenen 4 özniteliğin veri kümesi üzerindeki ayrıştırıcı özellikleri Şekil-3,4,5 ve 6'da sunulduğu gibi görselleştirilmiştir.

Şekil-3:add_add_or_add_add_push dağılımı

Şekil-4: add_arpl_jae_xor_pop_push dağılımı

Şekil-5: inc_xor_pop_pop_imul_jb dağılımı

Şekil-6:pop_ins_popa_push_jb_outs dağılımı

Dağılımlar, belirlenen 4 özniteliğin sınıflandırmada kullanılabileceğini işaret etmektedir. Bunun için, ilgili ARFF dosyasında seçilen 4 öznitelik ve @CLASS özniteliği hariç diğer bütün öznitelikler WEKA'nın "öznitelik silme" özelliği kullanılarak silinmiştir. Bu sayede analiz edilecek ARFF dosyasının boyutu da büyük bir oranda (1/960 oranında) azaltılmıştır.

4.2 Yöntemin Sınanması ve Değerlendirilmesi

Sınıflandırma algoritması olarak yüksek başarım değerleri sağlayan KNN, Naïve Bayes ve J48 (Decision Tree) algoritmaları seçilmiştir. Sınıflandırma algoritmaları, hem seçilen öznitelikler hem de diğer bütün öznitelikler ile koşturulmuştur.

Eğitim ve test modeli olarak WEKA'nın çapraz doğrulama yöntemi kullanılmıştır. Katlama değeri varsayılan değer 10 olarak kullanılmıştır. Veri kümesindeki 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 kümesindeki her bir öğenin hem eğitim hem de test verisi olarak kullanılması sağlanmıştır.

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

Çizelge 4: Karışıklık matrisi

Mevcut Durum	**Tespit**	
	Zayıf	***Güvenli***
Zayıf	**TP** Doğru-Pozitif	**FN** Yanlış- Negatif
Güvenli	**FP** Yanlış-Pozitif	**TN** Doğru-Negatif

Karışıklık matrisi, veri kümesindeki mevcut durumun ve sınıflandırma sonuçlarının Doğru-Pozitif (TP: True Positive), Yanlış-Pozitif (FP: False Positive), Doğru-Negatif (TN: True Negative) ve Yanlış-Negatif (FN: False Negative) değerleri ile ifade edilmesinde yardımcı olur.

Sınıflandırma yöntemlerinin başarımları; Doğru-Pozitif Oranı(TPR: True Positive Rate), Yanlış-Pozitif Oranı (FPR: False Positive Rate) ve Kesinlik (P: Precision) performans değerleri ile kıyaslanmıştır.

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

$$\boldsymbol{TPR} = \frac{TP}{TP + FN} * 100 \quad (10)$$

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

$$\boldsymbol{FPR} = \frac{FP}{FP + TN} * 100 \quad (11)$$

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

$$\boldsymbol{P} = \frac{TP}{TP + FP} * 100 \quad (12)$$

Sınıflandırma algoritmaları Çizelge-5'de verilen parametreler ile koşturulmuş ve başarım değerleri Çizelge-6'da sunulmuştur.

Çizelge-5: Sınıflandırma algoritmaları ve koşum parametreleri

Sınıflandırıcı	Parametreler
KNN (K=1)	weka.classifiers.lazy.IBk -K 1 -W 0 -A "weka.core.neighboursearch.LinearNNSearch -A \"weka.core.EuclideanDistance -R first-last\""
Naïve Bayes	weka.classifiers.bayes.NaiveBayes -batchSize 100 -useKernelEstimator False
J48	weka.classifiers.trees.J48 -C 0.25 -M 2

Çizelge-6: Sınıflandırma algoritmalarının başarımları

ARFF File	Başarım (%)	Sınıflandırıcı		
		KNN	N.B.	J48
Orijinal	TPR	96,4	96,4	96,4
	FPR	3,6	3,6	3,6
	P	96,7	96,7	96,7
Azaltılmış	TPR	96,4	96,4	96,4
	FPR	3,6	3,6	3,6
	P	96,7	96,7	96,7

5. Sonuç ve Değerlendirme

Çizelge-6'da verilen başarım değerleri; öznitelik seçimi sonrası sistemin kararlılığını koruduğunu, başarım değerinde herhangi bir kayıp olmadığını göstermektedir. ARFF dosyasının büyüklüğünü de önemli ölçüde azaltan öznitelik seçimi sınıflandırma algoritmalarının daha hızlı sonuç vermelerini sağlamıştır. Seçilen özniteliklerin veri kümesi üzerindeki dağılımları ve sınıflandırma algoritmalarının vermiş olduğu başarımlar, H-2 hipotezinde öne sürülen arabellek taşması zayıflığının belirli ikili işlem kod dizilimleri ile tespit edilebileceğini göstermektedir. Özellikle "pop_ins_popa_push_jb_outs" ikili işlem kod diziliminin sınıflandırmada başat öznitelik olarak ön plana çıktığı ve yalnız kullanılması durumunda iyi sonuçlar (TPR: %92) verdiği tespit edilmiştir. Diğer seçilen ikili işlem kod dizilimleri yalnız kullanıldıklarında başarım oranları (TPR: %50-53 bandında) düşük olmasına karşı, başat dizilim ile birlikte kullanıldığında başarımı artırdıkları tespit edilmiştir. Geliştirilecek kural tabanlı sistemler ile belirlenen kod dizilimleri ile ikili yürütülebilir yazılımları "güvenli" veya "zayıf" olarak etiketlemek mümkün görünmektedir. Ayrıca, [3]'de verilen veri kümesinin genişletilmesi sonucu [3]'de elde edilen başarım değerlerinin (TPR: %95; FPR: %5; P:%95,5) aynı yöntemde daha iyi değerler (TPR: %96,4; FPR: %3,6; P:%96,7) elde edilmesine katkı sağladığı gözlemlenmiştir.

Makine öğrenmesi teknikleri ile farklı alanlardaki problemlere çözümlerin üretildiği günümüzde, biz de çalışmamız ile yazılım güvenliği analizinde makine öğrenmesi tekniklerinin etkin bir şekilde uygulanabileceğini gösterdik. Mevcut çalışmaların çıktıları olan kaynak kodlar, prototip sistemler ve veri setleri maalesef ticarileşme veya gelecek çalışmalar kaygısı nedeni ile paylaşılmamakta ve alandaki ilerlemeyi yavaşlatmaktadır. Bu duruma karşın, geliştirmiş olduğumuz yöntem ve prototip sisteme ait kaynak kodlar ve veri kümesi GitHub [23] platformu üzerinden ilgili araştırmacılar ile paylaşılmıştır. Geliştirilen prototip sistemin gerek yazılım geliştirme aşamasında gerekse son kullanıcı tarafında kullanımı oldukça basit ve pratiktir. Veri kümesinin genişletilmesi, mevcut yöntemin iyileştirilmesi ve kullanımı kolaylaştırmaya yönelik çalışmalara devam edilmektedir.

Kaynakça

[1] International Organization for Standardization, "ISO/IEC TR 9126-4: Software engineering -- Product quality -- Part4: Quality in use metrics", http://www.iso.org, Son erişim tarihi: 29 Ekim 2019

[2] International Organization for Standardization, "ISO/IEC 25010: Systems and software engineering -- Systems and software Quality Requirements and Evaluation (SQuaRE) -- System and software quality models", http://www.iso.org, Son erişim tarihi: 29 Ekim 2019

[3] Durmuş, G. , Soğukpınar, İ ., "Makine öğrenmesi teknikleri ile ikili yürütülebilir dosyalarda arabellek taşması zayıflığı analizi için yeni bir yaklaşım", Gazi Üniversitesi Mühendislik Mimarlık Fakültesi Dergisi , 34 (4), 1695-1704, 2019

[4] McGraw, G., "Software Security", IEEE Security &Privacy, vol. 2, no. 2, 2004, pp. 80-83, 2004

[5] Baca, D.,"Developing Secure Software in an Agile Process", Doctoral Dissertation in Computer Science, Blekinge Institute of Technology, 2012

[6] Younan, Y. , "25 Years of Vulnerabilities: 1988-2012", Research Report, Sourcefire Crop,2013

[7] Younan, Y., Joosen, W., Piessens F., "Code Injection in C and C++ : A Survey of Vulnerabilities and Countermeasures", Report CW386, 2004

[8] 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

[9] 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

[10] 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

[11] Chess, B., McGraw, G., "Static Analysis for Security", IEEE Security &, Privacy, vol. 2, no. 6, pp. 76-79., 2004

[12] 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, vol. 5, no. 3, 2008

[13] 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

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

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

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

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

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

[19] 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

[20] NIST Software Assurance Reference Dataset, https://samate.nist.gov/SRD/testsuite.php, Son erişim tarihi: 29 Ekim 2019

[21] Weka 3 - Data Mining with Open Source Machine Learning Software in Java, https://www.cs.waikato.ac.nz/ml/weka, Son erişim tarihi: 29 Ekim 2019

[22] 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

[23] Durmus, G., "Source Code and Data Set of The Study", http://github.com/gdurmus/, Son erişim tarihi: 29 Ekim 2019